University of Oxford

College of Computer Science

Elzein Salah Elzein

22/12/2021

1 Introduction

1.1 Introduction

Robotics has achieved its greatest success to date in the world of industrial manufacturing. Robot arms, or *manipulators*, comprise a 2 billion dollar industry. Bolted at its shoulder to a specific position in the assembly line, the robot arm can move with great speed and accu- racy to perform repetitive tasks such as spot welding and painting (figure 1.1). In the elec- tronics industry, manipulators place surface-mounted components with superhuman precision, making the portable telephone and laptop computer possible.

Yet, for all of their successes, these commercial robots suffer from a fundamental dis- advantage: lack of mobility. A fixed manipulator has a limited range of motion that depends

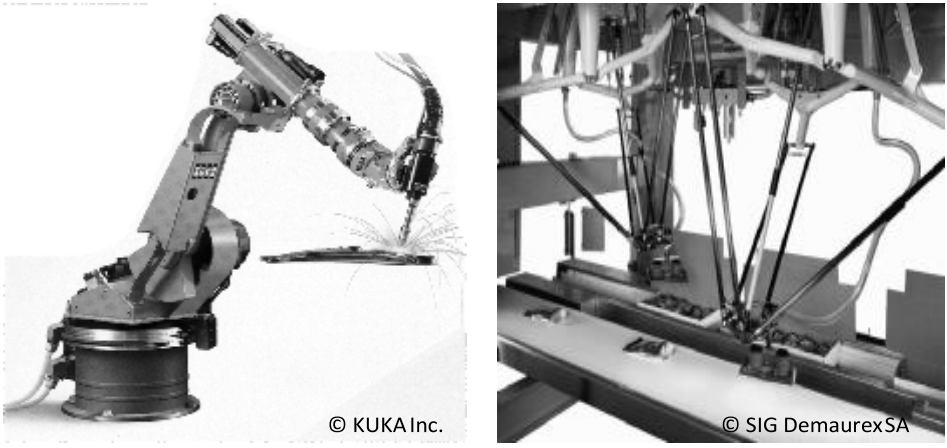


Figure 1.1

Picture of auto assembly plant-spot welding robot of KUKA and a parallel robot Delta of SIG Demau- rex SA (invented at EPFL [140]) during packaging of chocolates.

on where it is bolted down. In contrast, a mobile robot would be able to travel throughout the manufacturing plant, flexibly applying its talents wherever it is most effective.

This book focuses on the technology of mobility: how can a mobile robot move unsu- pervised through real-world environments to fulfill its tasks? The first challenge is locomo- tion itself. How should a mobile robot move, and what is it about a particular locomotion mechanism that makes it superior to alternative locomotion mechanisms?

Hostile environments such as Mars trigger even more unusual locomotion mechanisms (figure 1.2). In dangerous and inhospitable environments, even on Earth, such *teleoperated* systems have gained popularity (figures 1.3, 1.4, 1.5, 1.6). In these cases, the low-level complexities of the robot often make it impossible for a human operator to directly control its motions. The human performs localization and cognition activities, but relies on the robot's control scheme to provide motion control.

For example, Plustech's walking robot provides automatic leg coordination while the human operator chooses an overall direction of travel (figure 1.3). Figure 1.6 depicts an underwater vehicle that controls six propellers to autonomously stabilize the robot subma- rine in spite of underwater turbulence and water currents while the operator chooses posi- tion goals for the submarine to achieve.

Other commercial robots operate not where humans *cannot* go but rather share space with humans in human environments (figure 1.7). These robots are compelling not for rea- sons of mobility but because of their *autonomy*, and so their ability to maintain a sense of position and to navigate without human intervention is paramount.

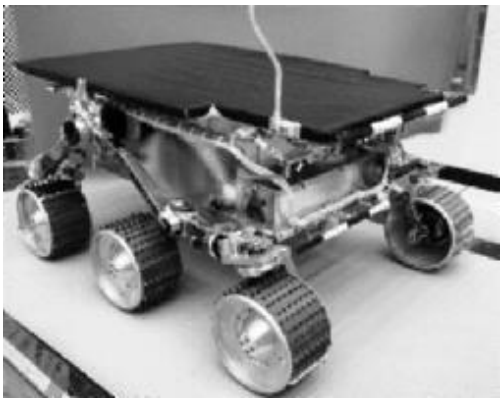


Figure 1.2

The mobile robot Sojourner was used during the Pathfinder mission to explore Mars in summer 1997. It was almost completely teleoperated from Earth. However, some on-board sensors allowed for obstacle detection. (http://ranier.oact.hq.nasa.gov/telerobotics_page/telerobotics.shtm).

© NASA/JPL



Figure 1.3

Plustech developed the first application-driven walking robot. It is designed to move wood out of the forest. The leg coordination is automated, but navigation is still done by the human operator on the robot. (http://www.plustech.fi). © Plustech.

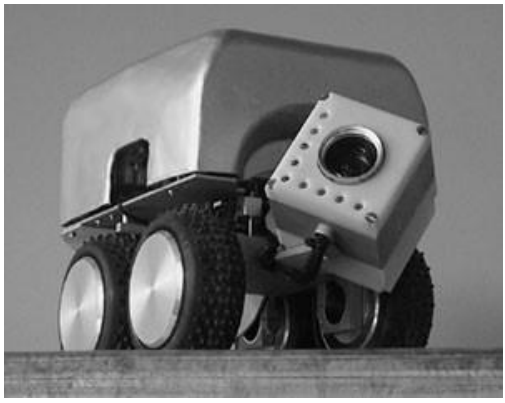


Figure 1.4

Airduct inspection robot featuring a pan-tilt camera with zoom and sensors for automatic inclination control, wall following, and intersection detection (http://asl.epfl.ch). © Sedirep / EPFL.



Figure 1.5

Picture of Pioneer, a robot designed to explore the Sarcophagus at Chernobyl. © Wide World Photos.



Figure 1.6

Picture of recovering MBARI's ALTEX AUV (autonomous underwater vehicle) onto the Icebreaker Healy following a dive beneath the Arctic ice. Todd Walsh © 2001 MBARI.

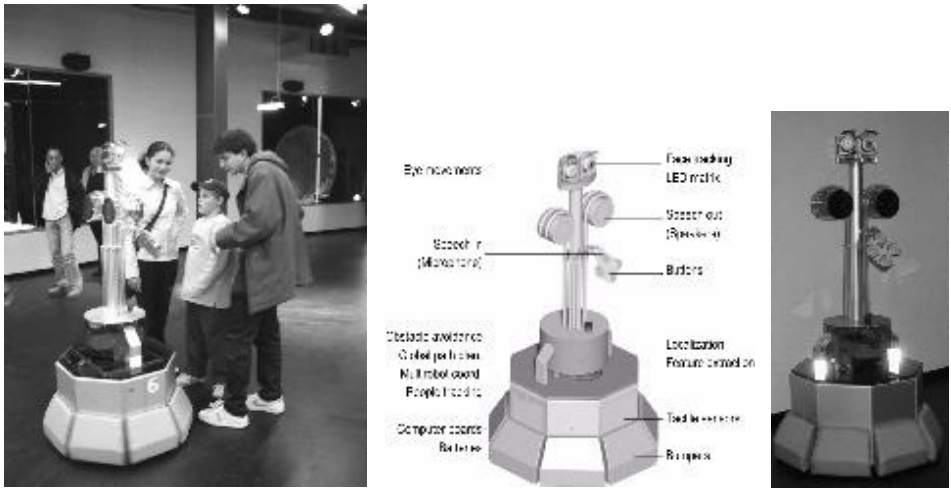


Figure 1.7

Tour-guide robots are able to interact and present exhibitions in an educational way [48, 118, 132, 143,]. Ten Roboxes have operated during 5 months at the Swiss exhibition EXPO.02, meeting hun- dreds of thousands of visitors. They were developed by EPFL [132] (http://robotics.epfl.ch) and commercialized by BlueBotics (http://www.bluebotics.ch).



Figure 1.8

Newest generation of the autonomous guided vehicle (AGV) of SWISSLOG used to transport motor blocks from one assembly station to another. It is guided by an electrical wire installed in the floor. There are thousands of AGVs transporting products in industry, warehouses, and even hospitals.

© Swisslog.

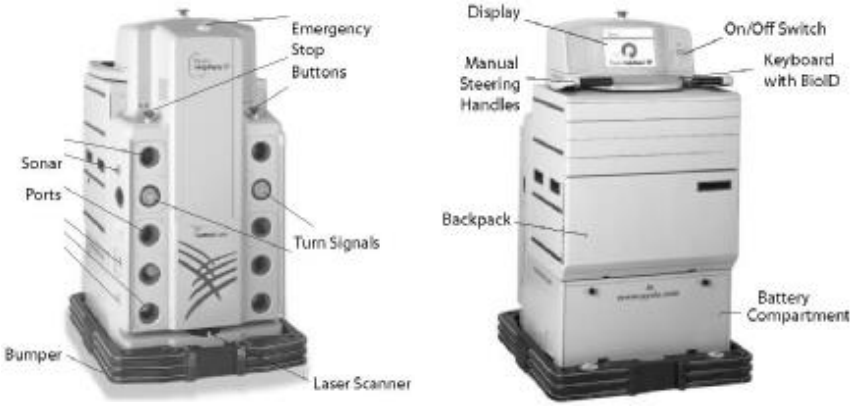


Figure 1.9

HELPMATE is a mobile robot used in hospitals for transportation tasks. It has various on-board sen- sors for autonomous navigation in the corridors. The main sensor for localization is a camera looking to the ceiling. It can detect the lamps on the ceiling as references, or landmarks (http:// www.pyxis.com). © Pyxis Corp.



Figure 1.10

BR 700 industrial cleaning robot (left) and the RoboCleaner RC 3000 consumer robot developed and sold by Alfred Kärcher GmbH & Co., Germany. The navigation system of BR 700 is based on a very sophisticated sonar system and a gyro. The RoboCleaner RC 3000 covers badly soiled areas with a special driving strategy until it is really clean. Optical sensors measure the degree of pollution of the aspirated air (http://www.karcher.de). © Alfred Kärcher GmbH & Co.

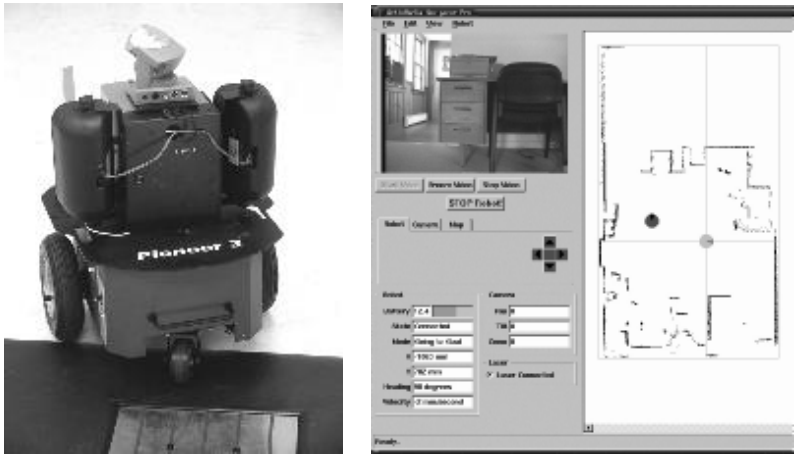


Figure 1.11

PIONEER is a modular mobile robot offering various options like a gripper or an on-board camera. It is equipped with a sophisticated navigation library developed at SRI, Stanford, CA (Reprinted with permission from ActivMedia Robotics, http://www.MobileRobots.com).

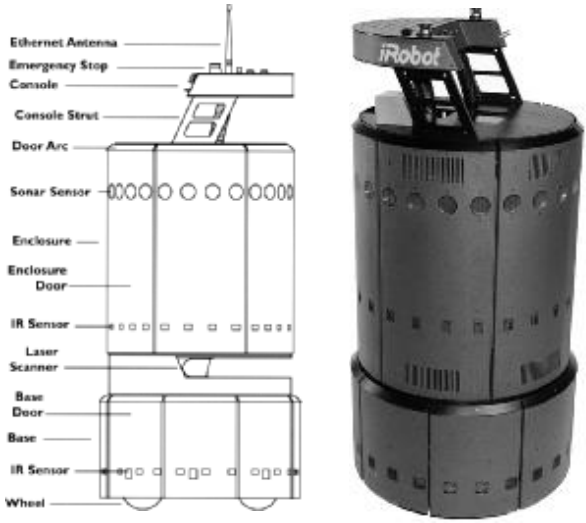


Figure 1.12

B21 of iRobot is a sophisticated mobile robot with up to three Intel Pentium processors on board. It has a large variety of sensors for high-performance navigation tasks (http://www.irobot.com/rwi/).

© iRobot Inc.

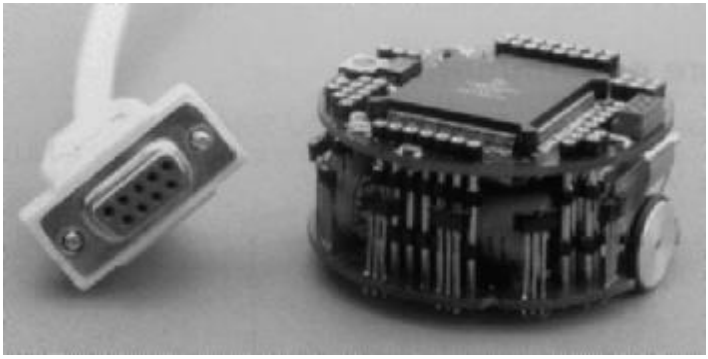


Figure 1.13

KHEPERA is a small mobile robot for research and education. It is only about 60 mm in diameter. Various additional modules such as cameras and grippers are available. More then 700 units had already been sold by the end of 1998. KHEPERA is manufactured and distributed by K-Team SA, Switzerland (http://www.k-team.com). © K-Team SA.

For example, AGV (autonomous guided vehicle) robots (figure 1.8) autonomously deliver parts between various assembly stations by following special electrical guidewires using a custom sensor. The Helpmate service robot transports food and medication throughout hospitals by tracking the position of ceiling lights, which are manually specified to the robot beforehand (figure 1.9). Several companies have developed autonomous clean- ing robots, mainly for large buildings (figure 1.10). One such cleaning robot is in use at the Paris Metro. Other specialized cleaning robots take advantage of the regular geometric pat- tern of aisles in supermarkets to facilitate the localization and navigation tasks.

Research into high-level questions of cognition, localization, and navigation can be per- formed using standard research robot platforms that are tuned to the laboratory environ- ment. This is one of the largest current markets for mobile robots. Various mobile robot platforms are available for programming, ranging in terms of size and terrain capability. The most popular research robots are those of ActivMedia Robotics, K-Team SA, and I-Robot (figures 1.11, 1.12, 1.13) and also very small robots like the Alice from EPFL (Swiss Federal Institute of Technology at Lausanne) (figure 1.14).

Although mobile robots have a broad set of applications and markets as summarized above, there is one fact that is true of virtually every successful mobile robot: its design involves the integration of many different bodies of knowledge. No mean feat, this makes mobile robotics as interdisciplinary a field as there can be. To solve locomotion problems, the mobile roboticist must understand mechanism and kinematics; dynamics and control theory. To create robust perceptual systems, the mobile roboticist must leverage the fields of signal analysis and specialized bodies of knowledge such as computer vision to properly

employ a multitude of sensor technologies. Localization and navigation demand knowl- edge of computer algorithms, information theory, artificial intelligence, and probability theory.

Figure 1.15 depicts an abstract control scheme for mobile robot systems that we will use throughout this text. This figure identifies many of the main bodies of knowledge associ- ated with mobile robotics.

This book provides an introduction to all aspects of mobile robotics, including software and hardware design considerations, related technologies, and algorithmic techniques. The intended audience is broad, including both undergraduate and graduate students in intro- ductory mobile robotics courses, as well as individuals fascinated by the field. While not absolutely required, a familiarity with matrix algebra, calculus, probability theory, and computer programming will significantly enhance the reader's experience.

Mobile robotics is a large field, and this book focuses not on robotics in general, nor on mobile robot applications, but rather on mobility itself. From mechanism and perception to localization and navigation, this book focuses on the techniques and technologies that enable robust *mobility*.

Clearly, a useful, commercially viable mobile robot does more than just move. It pol- ishes the supermarket floor, keeps guard in a factory, mows the golf course, provides tours in a museum, or provides guidance in a supermarket. The aspiring mobile roboticist will start with this book, but quickly graduate to course work and research specific to the desired application, integrating techniques from fields as disparate as human-robot interaction, computer vision, and speech understanding.

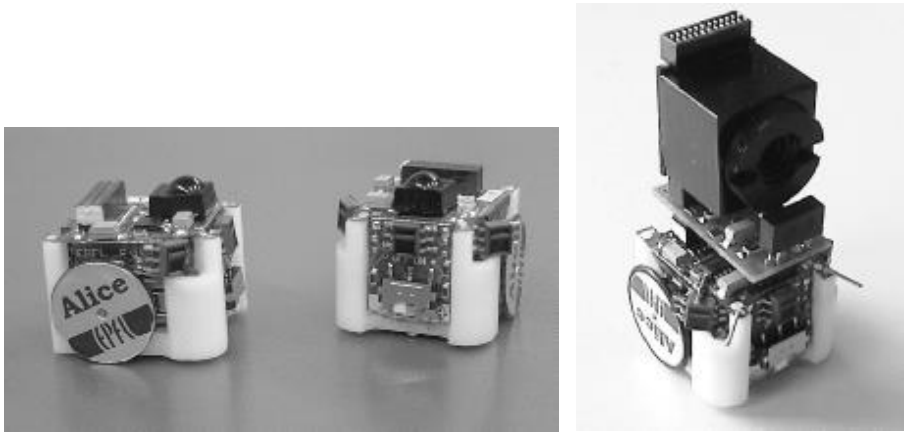


Figure 1.14

Alice is one of the smallest fully autonomous robots. It is approximately 2 x 2 x 2 cm, it has an auton- omy of about 8 hours and uses infrared distance sensors, tactile whiskers, or even a small camera for navigation [54].

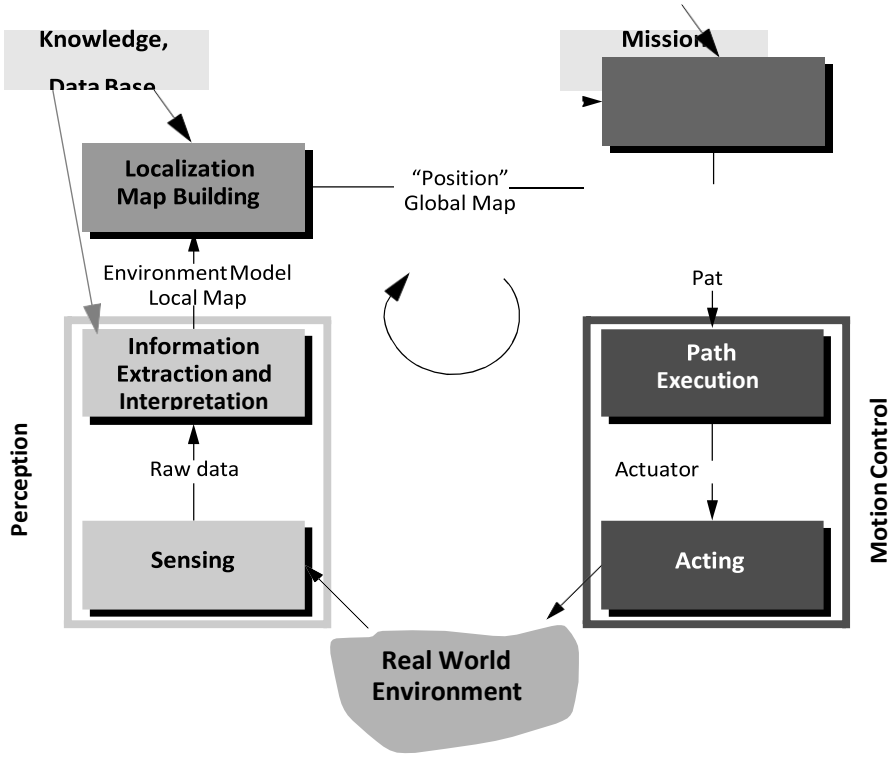


Figure 1.15

Reference control scheme for mobile robot systems used throughout this book.

1.2 An Overview of the Book

This book introduces the different aspects of a robot in modules, much like the modules shown in figure 1.15. Chapters 2 and 3 focus on the robot's low-level *locomotive ability*. Chapter 4

presents an in-depth view of *perception*. Then, Chapters 5 and 6 take us to the higher-level challenges of *localization* and even higher-level *cognition*, specifically the *ability to navigate robustly*. Each chapter builds upon previous chapters, and so the reader is encouraged to start at the beginning, even if their interest is primarily at the high level. Robotics is peculiar in that solutions to high-level challenges are most meaningful only in the context of a solid understanding of the low-level details of the system.

Chapter 2, “Locomotion”, begins with a survey of the most popular mechanisms that enable locomotion: wheels and legs. Numerous robotic examples demonstrate the particu-

lar talents of each form of locomotion. But designing a robot's locomotive system properly requires the ability to evaluate its overall motion capabilities quantitatively. Chapter 3, "Mobile Robot Kinematics", applies principles of kinematics to the whole robot, beginning with the kinematic contribution of each wheel and graduating to an analysis of robot maneuverability enabled by each mobility mechanism configuration.

The greatest single shortcoming in conventional mobile robotics is, without doubt, per- ception: mobile robots can travel across much of earth's man-made surfaces, but they cannot perceive the world nearly as well as humans and other animals. Chapter 4, "Percep- tion", begins a discussion of this challenge by presenting a clear language for describing the performance envelope of mobile robot sensors. With this language in hand, chapter 4 goes on to present many of the off-the-shelf sensors available to the mobile roboticist, describing their basic principles of operation as well as their performance limitations. The most promising sensor for the future of mobile robotics is vision, and chapter 4 includes an overview of the theory of operation and the limitations of both charged coupled device (CCD) and complementary metal oxide semiconductor (CMOS) sensors.

But perception is more than sensing. Perception is also the *interpretation* of sensed data in meaningful ways. The second half of chapter 4 describes strategies for feature extraction that have been most useful in mobile robotics applications, including extraction of geomet- ric shapes from range-based sensing data, as well as landmark and whole-image analysis using vision-based sensing.

Armed with locomotion mechanisms and outfitted with

hardware and software for per- ception, the mobile robot can move and perceive the world. The first point at which mobil- ity and sensing must meet is localization: mobile robots often need to maintain a sense of position. Chapter 5, “Mobile Robot Localization”, describes approaches that obviate the need for direct localization, then delves into fundamental ingredients of successful local- ization strategies: belief representation and map representation. Case studies demonstrate various localization schemes, including both Markov localization and Kalman filter local- ization. The final part of chapter 5 is devoted to a discussion of the challenges and most promising techniques for mobile robots to autonomously map their surroundings.

Mobile robotics is so young a discipline that it lacks a standardized architecture. There is as yet no established robot operating system. But the question of architecture is of para- mount importance when one chooses to address the higher-level competences of a mobile robot: how does a mobile robot navigate robustly from place to place, interpreting data, localizing and controlling its motion all the while? For this highest level of robot compe- tence, which we term *navigation competence*, there are numerous mobile robots that show- case particular architectural strategies. Chapter 6, “Planning and Navigation”, surveys the state of the art of robot navigation, showing that today’s various techniques are quite sim- ilar, differing primarily in the manner in which they *decompose* the problem of robot con-

trol. But first, chapter 6 addresses two skills that a competent, navigating robot usually must demonstrate: obstacle avoidance and path planning.

There is far more to know about the cross-disciplinary field of mobile robotics than can be contained in a single book. We hope, though, that this broad introduction will place the reader in the context of mobile robotics' collective wisdom. This is only the beginning, but, with luck, the first robot you program or build will have only good things to say about you.

2 Locomotion

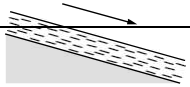
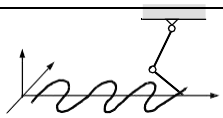
2.1 Introduction

A mobile robot needs locomotion mechanisms that enable it to move unbounded through- out its environment. But there are a large variety of possible ways to move, and so the selec- tion of a robot's approach to locomotion is an important aspect of mobile robot design. In the laboratory, there are research robots that can walk, jump, run, slide, skate, swim, fly, and, of course, roll. Most of these locomotion mechanisms have been inspired by their bio- logical counterparts (see figure 2.1).

There is, however, one exception: the actively powered wheel is a human invention that achieves extremely high efficiency on flat ground. This mechanism is not completely for- eign to biological systems. Our bipedal walking system can be approximated by a rolling polygon, with sides equal in length *d* to the span of the step (figure 2.2). As the step size decreases, the polygon approaches a circle or wheel. But nature did not develop a fully rotating, actively powered joint, which is the

technology necessary for wheeled locomo- tion.

Biological systems succeed in moving through a wide variety of harsh environments. Therefore it can be desirable to copy their selection of locomotion mechanisms. However, replicating nature in this regard is extremely difficult for several reasons. To begin with, mechanical complexity is easily achieved in biological systems through structural replica- tion. Cell division, in combination with specialization, can readily produce a millipede with several hundred legs and several tens of thousands of individually sensed cilia. In man- made structures, each part must be fabricated individually, and so no such economies of scale exist. Additionally, the cell is a microscopic building block that enables extreme min- iaturization. With very small size and weight, insects achieve a level of robustness that we have not been able to match with human fabrication techniques. Finally, the biological energy storage system and the muscular and hydraulic activation systems used by large ani- mals and insects achieve torque, response time, and conversion efficiencies that far exceed similarly scaled man-made systems.

Type of motion	Resistance to motion	Basic kinematics of motion
 Flow in a Channel 	Hydrodynamic forces	 Eddies
Crawl	Friction forces	 Longitudinal vibration
Sliding 	Friction forces	 Transverse vibration
Running	Loss of kinetic energy	Oscilla tory move ment of a multi-link pendulu m
Jumping 	Loss of kinetic energy	Oscillat ory move ment of a multi-link pendulu m
Walking 	Gravitational forces	Rolling of a polygon (see figure 2.2)

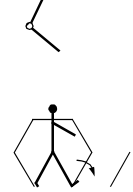


Figure 2.1

Locomotion mechanisms used in biological systems.

Owing to these limitations, mobile robots generally locomote either using wheeled mechanisms, a well-known human technology for vehicles, or using a small number of articulated legs, the simplest of the biological approaches to locomotion (see figure 2.2).

In general, legged locomotion requires higher degrees of freedom and therefore greater mechanical complexity than wheeled locomotion. Wheels, in addition to being simple, are extremely well suited to flat ground. As figure 2.3 depicts, on flat surfaces wheeled loco- motion is one to two orders of magnitude more efficient than legged locomotion. The rail- way is ideally engineered for wheeled locomotion because rolling friction is minimized on a hard and flat steel surface. But as the surface becomes soft, wheeled locomotion accumu- lates inefficiencies due to rolling friction whereas legged locomotion suffers much less because it consists only of point contacts with the ground. This is demonstrated in figure
2.3 by the dramatic loss of efficiency in the case of a tire on soft ground.

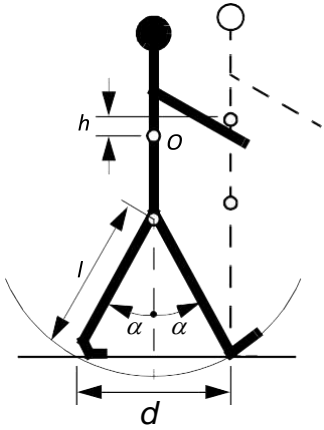
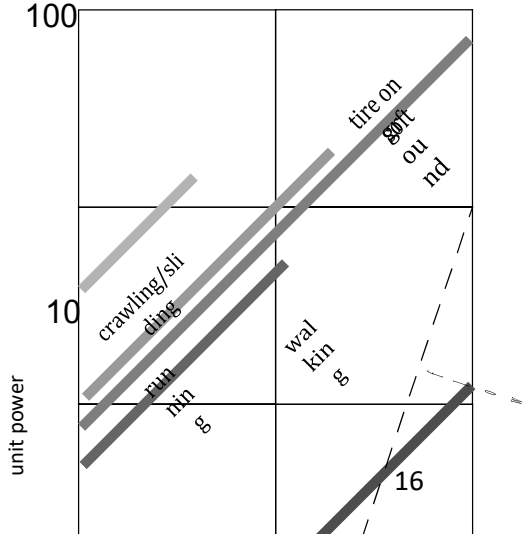


Figure 2.2

A biped walking system can be approximated by a rolling polygon, with sides equal in length *d* to the span of the step. As the step size decreases, the polygon approaches a circle or wheel with the radius *l*.



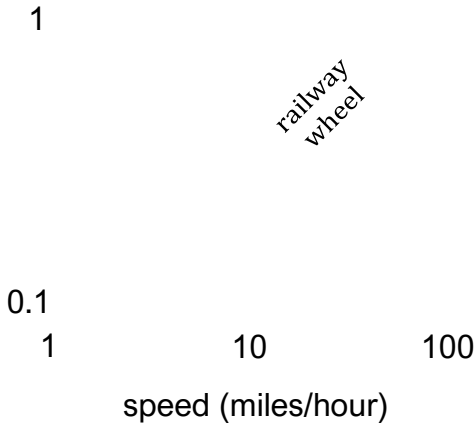


Figure 2.3

Specific power versus attainable speed of various locomotion mechanisms [33].

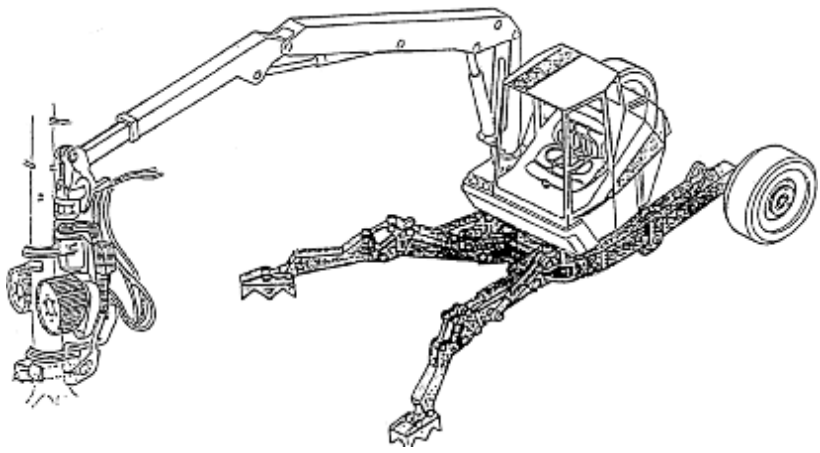


Figure 2.4

RoboTrac, a hybrid wheel-leg vehicle for rough terrain [130].

In effect, the efficiency of wheeled locomotion depends greatly on environmental qual- ities, particularly the flatness and hardness of the ground, while the efficiency of legged locomotion depends on the leg mass and body mass, both of which the robot must support at various points in a legged gait.

It is understandable therefore that nature favors legged locomotion, since locomotion systems in nature must operate on rough and unstructured terrain. For example, in the case of insects in a forest the vertical variation in ground height is often an order of magnitude greater than the total height of the insect. By the same token, the human environment fre- quently consists of engineered, smooth surfaces, both indoors and outdoors. Therefore, it is also understandable that virtually all industrial applications of mobile robotics utilize some form of wheeled locomotion. Recently, for more natural outdoor environments,

there has been some progress toward hybrid and legged industrial robots such as the forestry robot shown in figure 2.4.

In the section 2.1.1, we present general considerations that concern all forms of mobile robot locomotion. Following this, in sections 2.2 and 2.3, we present overviews of legged locomotion and wheeled locomotion techniques for mobile robots.

2.1.1 Key issues for locomotion

Locomotion is the complement of manipulation. In manipulation, the robot arm is fixed but moves objects in the workspace by imparting force to them. In locomotion, the environ- ment is fixed and the robot moves by imparting force to the environment. In both cases, the scientific basis is the study of actuators that generate interaction forces, and mechanisms

that implement desired kinematic and dynamic properties. Locomotion and manipulation thus share the same core issues of stability, contact characteristics, and environmental type:

- stability
  - number and geometry of contact points
  - center of gravity
  - static/dynamic stability
  - inclination of terrain
- characteristics of contact
  - contact point/path size and shape
  - angle of contact
  - friction
- type of environment
  - structure
  - medium, (e.g. water, air, soft or hard ground)

A theoretical analysis of locomotion begins with mechanics and physics. From this start- ing point, we can formally define and analyze all manner of mobile robot locomotion sys- tems. However, this book focuses on the mobile robot *navigation* problem, particularly stressing perception, localization, and cognition. Thus we will not delve deeply into the physical basis of locomotion. Nevertheless, the two remaining sections in this chapter present overviews of issues in legged locomotion [33] and wheeled locomotion. Then, chapter 3 presents a more detailed analysis of the kinematics and control of wheeled mobile robots.

2.2 Legged Mobile Robots

Legged locomotion is characterized by a series of point contacts

between the robot and the ground. The key advantages include adaptability and maneuverability in rough terrain. Because only a set of point contacts is required, the quality of the ground between those points does not matter so long as the robot can maintain adequate ground clearance. In addi- tion, a walking robot is capable of crossing a hole or chasm so long as its reach exceeds the width of the hole. A final advantage of legged locomotion is the potential to manipulate objects in the environment with great skill. An excellent insect example, the dung beetle, is capable of rolling a ball while locomoting by way of its dexterous front legs.

The main disadvantages of legged locomotion include power and mechanical complex- ity. The leg, which may include several degrees of freedom, must be capable of sustaining part of the robot's total weight, and in many robots must be capable of lifting and lowering the robot. Additionally, high maneuverability will only be achieved if the legs have a suf- ficient number of degrees of freedom to impart forces in a number of different directions.

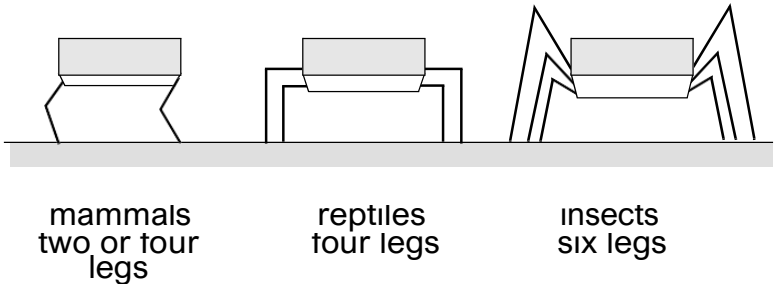


Figure 2.5

Arrangement of the legs of various animals.

2.2.1 Leg configurations and stability

Because legged robots are biologically inspired, it is instructive to examine biologically successful legged systems. A number of different leg configurations have been successful in a variety of organisms (figure 2.5). Large animals, such as mammals and reptiles, have four legs, whereas insects have six or more legs. In some mammals, the ability to walk on only two legs has been perfected. Especially in the case of humans, balance has progressed to the point that we can even jump with one leg[1]. This exceptional maneuverability comes at a price: much more complex active control to maintain balance.

In contrast, a creature with three legs can exhibit a static, stable pose provided that it can ensure that its center of gravity is within the tripod of ground contact. Static stability, dem- onstrated by a three-legged stool, means that balance is maintained with no need for motion. A small deviation from stability (e.g., gently

pushing the stool) is passively cor- rected toward the stable pose when the upsetting force stops.

But a robot must be able to lift its legs in order to walk. In order to achieve static walk- ing, a robot must have at least six legs. In such a configuration, it is possible to design a gait in which a statically stable tripod of legs is in contact with the ground at all times (figure 2.8).

Insects and spiders are immediately able to walk when born. For them, the problem of balance during walking is relatively simple. Mammals, with four legs, cannot achieve static walking, but are able to stand easily on four legs. Fauns, for example, spend several minutes attempting to stand before they are able to do so, then spend several more minutes learning to walk without falling. Humans, with two legs, cannot even stand in one place with static stability. Infants require months to stand and walk, and even longer to learn to jump, run, and stand on one leg.

---

1. In child development, one of the tests used to determine if the child is acquiring advanced loco- motion skills is the ability to jump on one leg.

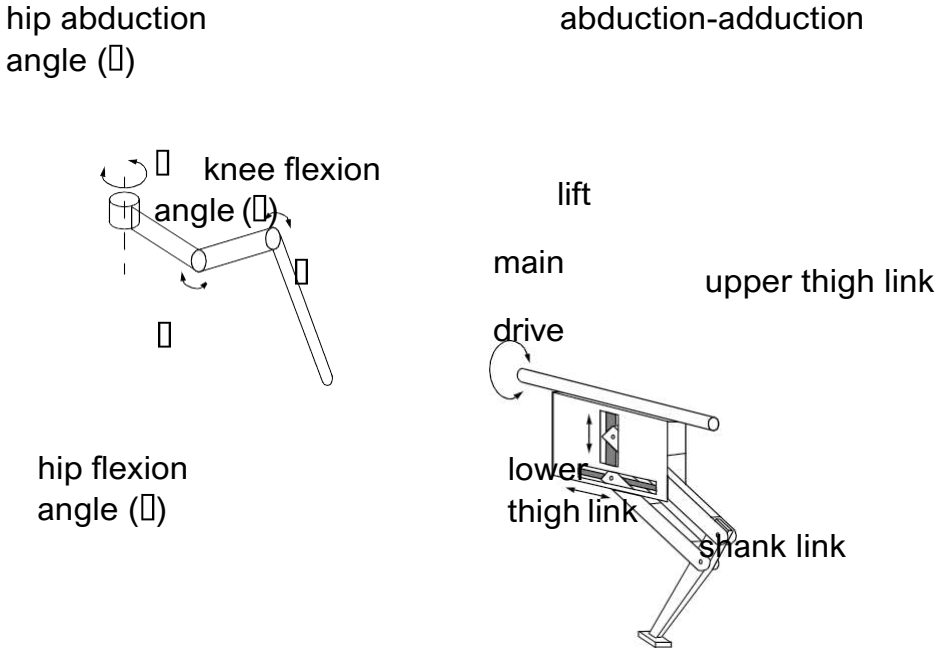


Figure 2.6

Two examples of legs with three degrees of freedom.

There is also the potential for great variety in the complexity of each individual leg. Once again, the biological world provides ample examples at both extremes. For instance, in the case of the caterpillar, each leg is extended using hydraulic pressure by constricting the body cavity and forcing an increase in pressure, and each leg is retracted longitudinally by relaxing the hydraulic pressure, then activating a single tensile muscle that pulls the leg in toward the body. Each leg has only a single degree of freedom, which is oriented longi- tudinally along the leg. Forward locomotion depends on the hydraulic pressure in the body, which

extends the distance between pairs of legs. The caterpillar leg is therefore mechani- cally very simple, using a minimal number of extrinsic muscles to achieve complex overall locomotion.

At the other extreme, the human leg has more than seven major degrees of freedom, combined with further actuation at the toes. More than fifteen muscle groups actuate eight complex joints.

In the case of legged mobile robots, a minimum of two degrees of freedom is generally required to move a leg forward by lifting the leg and swinging it forward. More common is the addition of a third degree of freedom for more complex maneuvers, resulting in legs such as those shown in figure 2.6. Recent successes in the creation of bipedal walking robots have added a fourth degree of freedom at the ankle joint. The ankle enables more consistent ground contact by actuating the pose of the sole of the foot.

In general, adding degrees of freedom to a robot leg increases the maneuverability of the robot, both augmenting the range of terrains on which it can travel and the ability of the robot to travel with a variety of gaits. The primary disadvantages of additional joints and actuators are, of course, energy, control, and mass. Additional actuators require energy and control, and they also add to leg mass, further increasing power and load requirements on existing actuators.

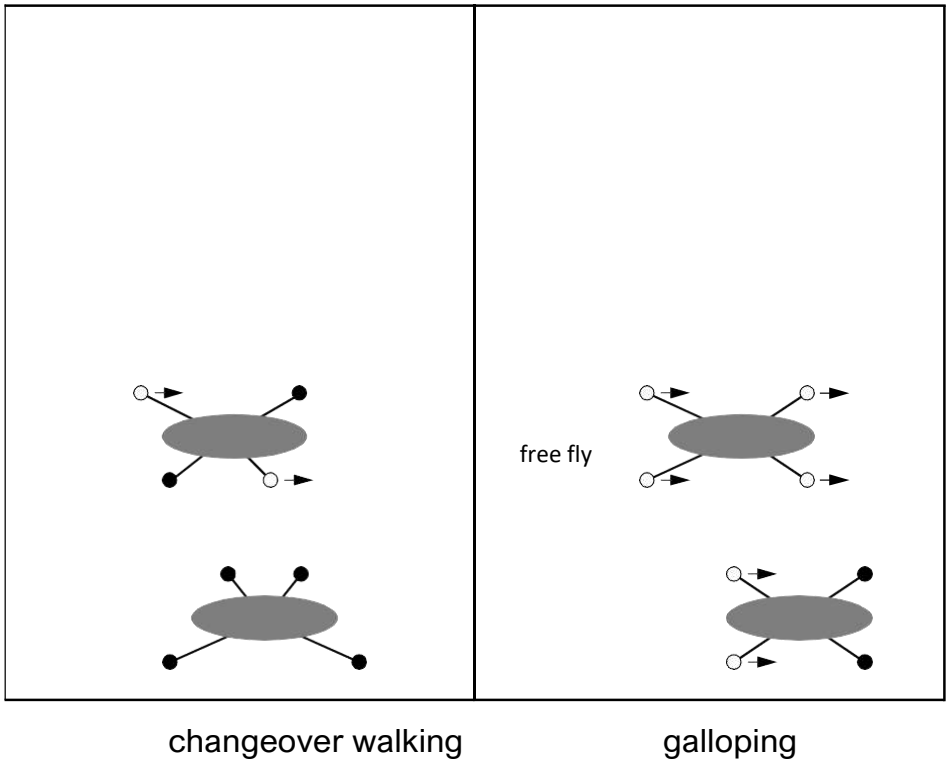


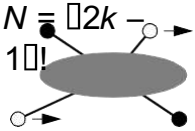
Figure 2.7

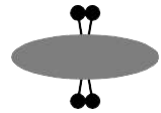
Two gaits with four legs. Because this robot has fewer than six legs, static walking is not generally possible.

In the case of a multilegged mobile robot, there is the issue of leg coordination for loco- motion, or gait control. The number of possible gaits depends on the number of legs [33]. The gait is a sequence of lift and release events for the individual legs. For a mobile robot with k legs, the total number of possible events N

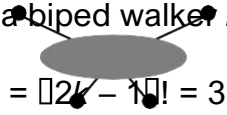
for a walking machine is

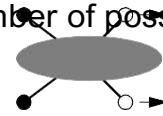
$$N = (2k - 1)! \tag{2.1}$$





For a biped walker $k = 2$ legs, the number of possible events N is





$$N = (2k - 1)! = 3! = 3 \cdot 2 \cdot 1 = 6 \tag{2.2}$$

The six different events are

1. lift right leg;
2. lift left leg;
3. release right leg;
4. release left leg;
5. lift both legs together;
6. release both legs together.

Of course, this quickly grows quite large. For example, a robot with six legs has far more gaits theoretically:

$$N = \frac{11!}{} = 39916800 \tag{2.3}$$

Figures 2.7 and 2.8 depict several four-legged gaits and the static six-legged tripod gait.

2.2.2 Examples of legged robot locomotion

Although there are no high-volume industrial applications to date, legged locomotion is an important area of long-term research. Several interesting designs are presented below, beginning with the one-legged robot and finishing with six-legged robots. For a very good overview of climbing and walking robots, see *http://www.uwe.ac.uk/clawar/*.

2.2.2.1 One leg

The minimum number of legs a legged robot can have is, of course, one. Minimizing the number of legs is beneficial for

several reasons. Body mass is particularly important to walking machines, and the single leg minimizes cumulative leg mass. Leg coordination is required when a robot has several legs, but with one leg no such coordination is needed. Perhaps most importantly, the one-legged robot maximizes the basic advantage of legged locomotion: legs have single points of contact with the ground in lieu of an entire track, as with wheels. A single-legged robot requires only a sequence of single contacts, making it amenable to the roughest terrain. Furthermore, a hopping robot can dynamically cross a gap that is larger than its stride by taking a running start, whereas a multilegged walking robot that cannot run is limited to crossing gaps that are as large as its reach.

The major challenge in creating a single-legged robot is balance. For a robot with one leg, static walking is not only impossible but static stability when stationary is also impos- sible. The robot must actively balance itself by either changing its center of gravity or by imparting corrective forces. Thus, the successful single-legged robot must be dynamically stable.

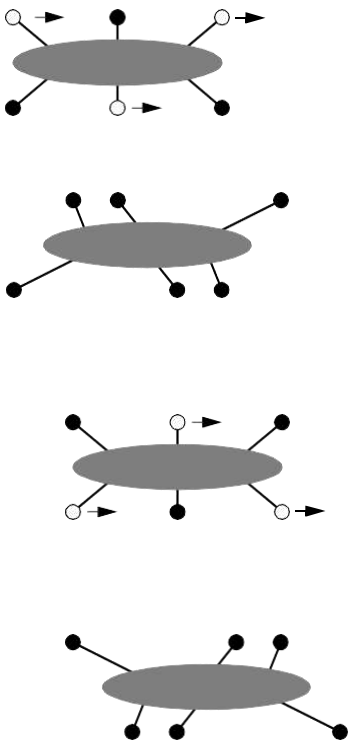


Figure 2.8

Static walking with six legs. A tripod formed by three legs always exists.

Figure 2.9 shows the Raibert hopper [28, 124], one of the most well-known single- legged hopping robots created. This robot makes continuous corrections to body attitude and to robot velocity by adjusting the leg angle with respect to the body. The actuation is hydraulic, including high-power longitudinal

extension of the leg during stance to hop back into the air. Although powerful, these actuators require a large, off-board hydraulic pump to be connected to the robot at all times.

Figure 2.10 shows a more energy-efficient design developed more recently [46]. Instead of supplying power by means of an off-board hydraulic pump, the bow leg hopper is designed to capture the kinetic energy of the robot as it lands, using an efficient bow spring leg. This spring returns approximately 85% of the energy, meaning that stable hopping requires only the addition of 15% of the required energy on each hop. This robot, which is constrained along one axis by a boom, has demonstrated continuous hopping for 20 minutes using a single set of batteries carried on board the robot. As with the Raibert hopper, the bow leg hopper controls velocity by changing the angle of the leg to the body at the hip joint.

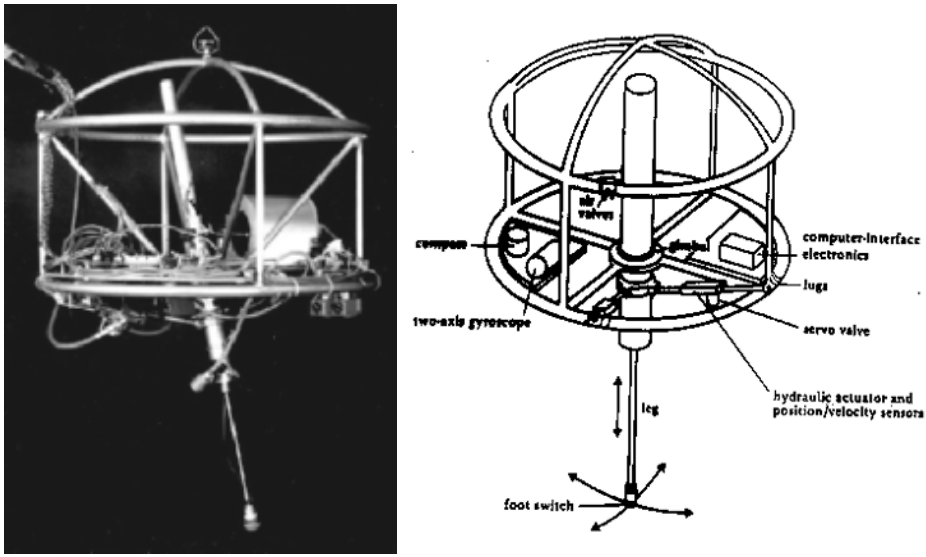


Figure 2.9

The Raibert hopper [28, 124]. Image courtesy of the LegLab and Marc Raibert. © 1983.

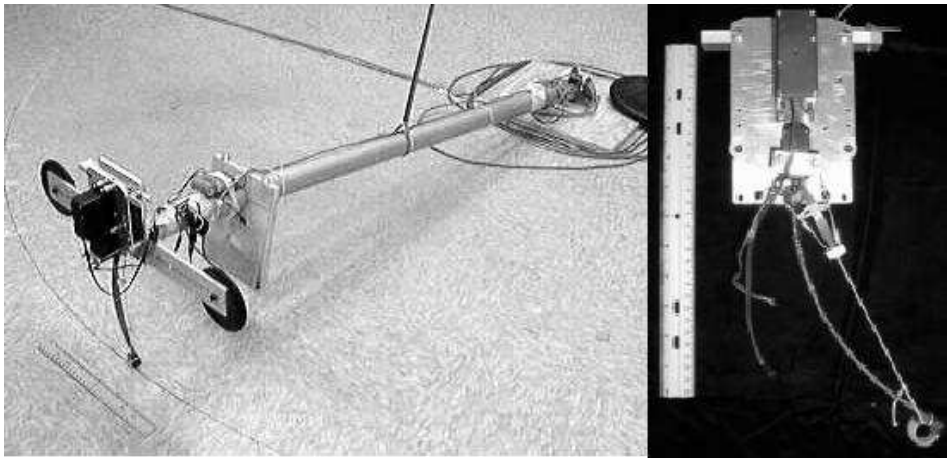


Figure 2.10

The 2D single bow leg hopper [46]. Image courtesy of H. Benjamin Brown and Garth Zeglin, CMU.



Specifications:

Weight:	7 kg
Height:	58 cm
Neck DOF:	4
Body DOF:	2
Arm DOF:	2 x 5
Legs DOF:	2 x 6 Five-finger Hands

Figure 2.11

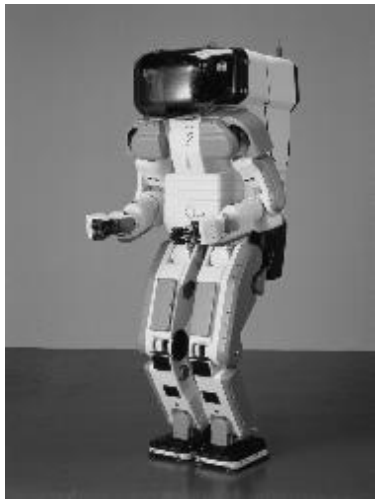
The Sony SDR-4X II, © 2003 Sony Corporation.

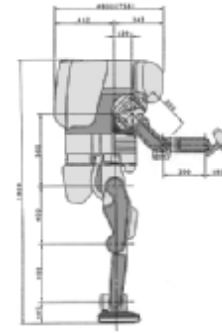
The paper of Ringrose [125] demonstrates the very important duality of mechanics and controls as applied to a single-legged hopping machine. Often clever mechanical design can perform the same operations as complex active control circuitry. In this robot, the phys- ical shape of the foot is exactly the right curve so that when the robot lands without being perfectly vertical, the proper corrective force is provided from the impact, making the robot vertical by the next landing. This robot is dynamically stable, and is furthermore passive. The correction is provided by

physical interactions between the robot and its environment, with no computer or any active control in the loop.

2.2.2.2 Two legs (biped)

A variety of successful bipedal robots have been demonstrated over the past ten years. Two legged robots have been shown to run, jump, travel up and down stairways, and even do aerial tricks such as somersaults. In the commercial sector, both Honda and Sony have made significant advances over the past decade that have enabled highly capable bipedal robots. Both companies designed small, powered joints that achieve power-to-weight per- formance unheard of in commercially available servomotors. These new “intelligent” servos provide not only strong actuation but also compliant actuation by means of torque sensing and closed-loop control.





Specifications:

Maximum speed: 2 km/h
Autonomy: 15 min
Weight: 210 kg
Height: 1.82 m
Leg DOF: 2 x 6
Arm DOF: 2 x 7

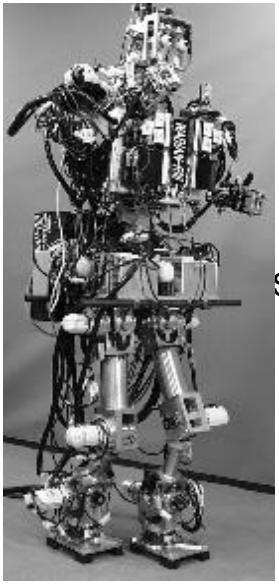
Figure 2.12

The humanoid robot P2 from Honda, Japan. © Honda Motor Corporation.

The Sony Dream Robot, model SDR-4X II, is shown in figure 2.11. This current model is the result of research begun in 1997 with the basic objective of motion entertainment and communication entertainment (i.e., dancing and singing). This robot with thirty-eight degrees of freedom has seven microphones for fine localization of sound, image-based person recognition, on-board miniature stereo depth-map reconstruction, and limited speech recognition. Given the goal of

fluid and entertaining motion, Sony spent consider- able effort designing a motion prototyping application system to enable their engineers to script dances in a straightforward manner. Note that the SDR-4X II is relatively small, standing at 58 cm and weighing only 6.5 kg.

The Honda humanoid project has a significant history but, again, has tackled the very important engineering challenge of actuation. Figure 2.12 shows model *P2*, which is an immediate predecessor to the most recent Asimo model (advanced step in innovative mobility). Note from this picture that the Honda humanoid is much larger than the SDR- 4X at 120 cm tall and 52 kg. This enables practical mobility in the human world of stairs and ledges while maintaining a nonthreatening size and posture. Perhaps the first robot to famously demonstrate biomimetic bipedal stair climbing and descending, these Honda humanoid series robots are being designed not for entertainment purposes but as human aids throughout society. Honda refers, for instance, to the height of Asimo as the minimum height which enables it to nonetheless manage operation of the human world, for instance, control of light switches.



Specifications:

Weight: 131 [kg]
Height: 1.88 [m]

DOF in total: 43
Lower Limbs: 2 x 6
Trunk: 3
Arms: 2 x 10
Neck: 4
Eyes: 2 x 2

Figure 2.13

The humanoid robot WABIAN-RIII at Waseda University in Japan [75]. Image courtesy of Atsuo Takanishi, Waseda University.

An important feature of bipedal robots is their anthropomorphic shape. They can be built to have the same approximate dimensions as humans, and this makes them excellent vehi- cles for research in human-robot interaction. WABIAN is a robot built at Waseda Univer- sities Japan (figure 2.13) for just such research [75]. WABIAN is designed to emulate human motion, and is even designed to dance like a human.

Bipedal robots can only be statically stable within some limits, and so robots such as P2 and WABIAN generally must perform

continuous balance-correcting servoing even when standing still. Furthermore, each leg must have sufficient capacity to support the full weight of the robot. In the case of four-legged robots, the balance problem is facilitated along with the load requirements of each leg. An elegant design of a biped robot is the Spring Flamingo of MIT (figure 2.14). This robot inserts springs in series with the leg actuators to achieve a more elastic gait. Combined with "kneecaps" that limit knee joint angles, the Fla- mingo achieves surprisingly biomimetic motion.

2.2.2.3 Four legs (quadruped)

Although standing still on four legs is passively stable, walking remains challenging because to remain stable the robot's center of gravity must be actively shifted during the

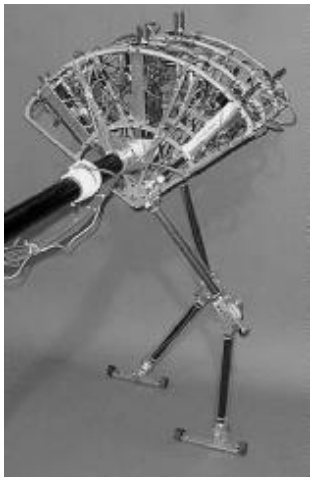


Figure 2.14

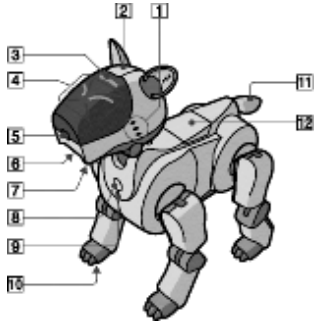
The Spring Flamingo developed at MIT [123]. Image courtesy of Jerry Pratt, MIT Leg Laboratory.

gait. Sony recently invested several million dollars to develop a four-legged robot called AIBO (figure 2.15). To create this robot, Sony produced both a new robot operating system that is near real-time and new geared servomotors that are of sufficiently high torque to sup- port the robot, yet back drivable for safety. In addition to developing custom motors and software, Sony incorporated a color vision system that enables AIBO to chase a brightly colored ball. The robot is able to function for at most one hour before requiring recharging. Early sales of the robot have been very strong, with more than 60,000 units sold in the first year. Nevertheless, the number of motors and the technology investment behind this robot dog resulted in a very high price of approximately $1500.

Four-legged robots have the potential to serve as effective artifacts for research in human-robot interaction (figure 2.16). Humans can treat the Sony robot, for example, as a pet and might develop an emotional relationship similar to that between man and dog. Fur- thermore, Sony has designed AIBO's walking style and general behavior to emulate learn- ing and maturation, resulting in dynamic behavior over time that is more interesting for the owner who can track the changing behavior. As the challenges of high energy storage and motor technology are solved, it is likely that quadruped robots much more capable than AIBO will become common throughout the human environment.

2.2.2.4 Six legs (hexapod)

Six-legged configurations have been extremely popular in mobile robotics because of their static stability during walking, thus reducing the control complexity (figures 2.17 and 1.3).



ERS-110 © 1999 Sony Corporation



ERS-210 © 2000 Sony Corporation

Figure 2.15

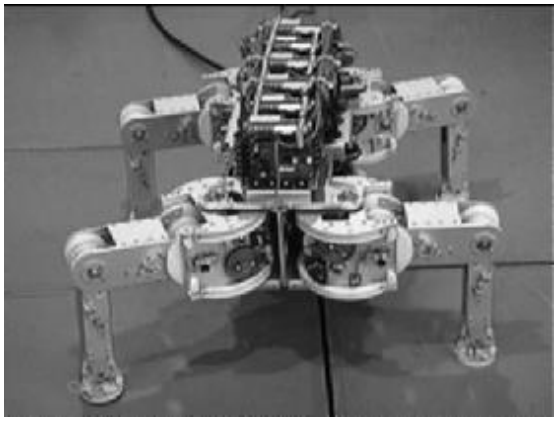
AIBO, the artificial dog from Sony, Japan.

1 Stereo microphone: Allows AIBO to pick up surrounding sounds.
2 Head sensor: Senses when a person taps or pets AIBO on the head.
3 Mode indicator: Shows AIBO's operation mode.
4 Eye lights: These light up in blue-green or red to indicate AIBO's emotional state.
5 Color camera: Allows AIBO to search for objects and recognize them by color and movement.
6 Speaker: Emits various musical tones and sound effects.
7 Chin sensor: Senses when a person touches AIBO on the chin.
8 Pause button: Press to activate AIBO or to pause AIBO.
9 Chest light: Gives information about the status of the robot.
10 Paw sensors: Located on the bottom of each paw.
11 Tail light: Lights up blue or orange to show AIBO's emotional state.
12 Back sensor: Senses when

a person touches AIBO on the back.

In most cases, each leg has three degrees of freedom, including hip flexion, knee flexion, and hip abduction (see figure 2.6). Genghis is a commercially available hobby robot that has six legs, each of which has two degrees of freedom provided by hobby servos (figure 2.18). Such a robot, which consists only of hip flexion and hip abduction, has less maneu- verability in rough terrain but performs quite well on flat ground. Because it consists of a straightforward arrangement of servomotors and straight legs, such robots can be readily built by a robot hobbyist.

Insects, which are arguably the most successful locomoting creatures on earth, excel at traversing all forms of terrain with six legs, even upside down. Currently, the gap between the capabilities of six-legged insects and artificial six-legged robots is still quite large. Interestingly, this is not due to a lack of sufficient numbers of degrees of freedom on the robots. Rather, insects combine a small number of active degrees of freedom with passive

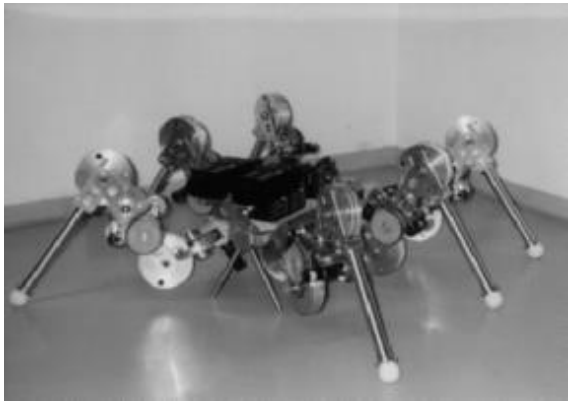


Specifications:

Weight:1 9 kg
Height: 0.25 m
DOF: 4 x 3

Figure 2.16

Titan VIII, a quadruped robot developed at Tokyo Institute of Technology. (http://mozu.mes.titech.ac.jp/research/walk/). © Tokyo Institute of Technology.



Specifications:

Maximum speed: 0.5 m/s Weight:1 6 kg
Height: 0.3 m
Length: 0.7 m
No. of legs: 6
DOF in total: 6 x

3 Power consumption:10 W

Figure 2.17

Lauron II, a hexapod platform developed at the University of Karlsruhe, Germany.

© University of Karlsruhe.

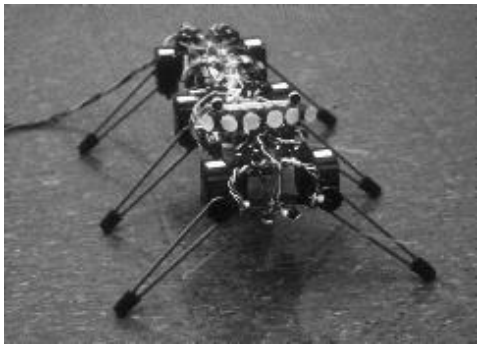


Figure 2.18

Genghis, one of the most famous walking robots from MIT, uses hobby servomotors as its actuators (http://www.ai.mit.edu/projects/genghis). © MIT AI Lab.

structures, such as microscopic barbs and textured pads, that increase the gripping strength of each leg significantly. Robotic research into such passive tip structures has only recently begun. For example, a research group is attempting to re-create the complete mechanical function of the cockroach leg [65].

It is clear from the above examples that legged robots have much progress to make before they are competitive with their biological equivalents. Nevertheless, significant gains have been realized recently, primarily due to advances in motor design. Creating actuation systems that approach the efficiency of animal muscles remains far from the reach of robotics, as does energy storage with the energy densities found in organic life forms.

2.3 Wheeled Mobile Robots

The wheel has been by far the most popular locomotion mechanism in mobile robotics and in man-made vehicles in general. It can achieve very good efficiencies, as demonstrated in figure 2.3, and does so with a relatively simple mechanical implementation.

In addition, balance is not usually a research problem in wheeled robot designs, because wheeled robots are almost always designed so that all wheels are in ground contact at all times. Thus, three wheels are sufficient to guarantee stable balance, although, as we shall see below, two-wheeled robots can also be stable. When more than three wheels are used, a suspension system is required to allow all wheels to maintain ground contact when the robot encounters uneven terrain.

Instead of worrying about balance, wheeled robot research tends to focus on the prob- lems of traction and stability, maneuverability, and control: can the robot wheels provide

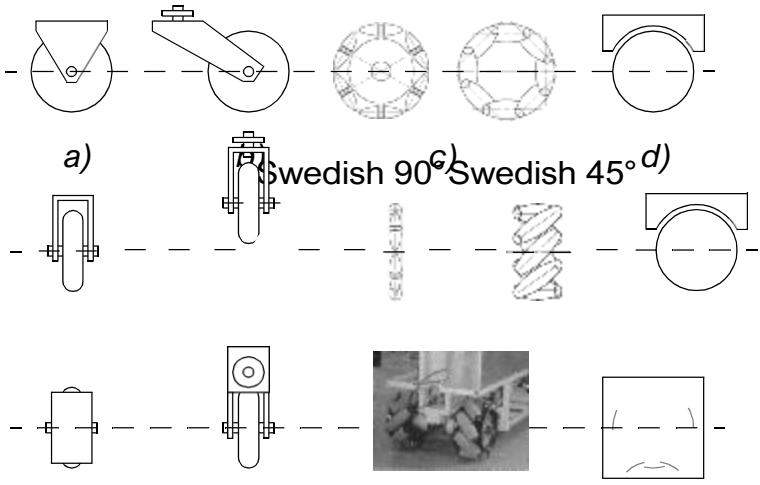


Figure 2.19

The four basic wheel types. (a) Standard wheel: two degrees of freedom; rotation around the (motor- ized) wheel axle and the contact point.(b) castor wheel: two degrees of freedom; rotation around an offset steering joint. (c) Swedish wheel: three degrees of freedom; rotation around the (motorized) wheel axle, around the rollers, and around the contact point. (d) Ball or spherical wheel: realization technically difficult.

sufficient traction and stability for the robot to cover all of the desired terrain, and does the robot's wheeled configuration enable sufficient control over the velocity of the robot?

2.3.1 Wheeled locomotion: the design space

As we shall see, there is a very large space of possible wheel configurations when one con- siders possible techniques for

mobile robot locomotion. We begin by discussing the wheel in detail, as there are a number of different wheel types with specific strengths and weak- nesses. Then, we examine complete wheel configurations that deliver particular forms of locomotion for a mobile robot.

2.3.1.1 Wheel design

There are four major wheel classes, as shown in figure 2.19. They differ widely in their kinematics, and therefore the choice of wheel type has a large effect on the overall kinemat- ics of the mobile robot. The standard wheel and the castor wheel have a primary axis of rotation and are thus highly directional. To move in a different direction, the wheel must be steered first along a vertical axis. The key difference between these two wheels is that the standard wheel can accomplish this steering motion with no side effects, as the center of rotation passes through the contact patch with the ground, whereas the castor wheel rotates around an offset axis, causing a force to be imparted to the robot chassis during steering.



Figure 2.20

Navlab I, the first autonomous highway vehicle that steers and controls the throttle using vision and radar sensors [61]. Developed at CMU.

The Swedish wheel and the spherical wheel are both designs that are less constrained by directionality than the conventional standard wheel. The Swedish wheel functions as a normal wheel, but provides low resistance in another direction as well, sometimes perpen- dicular to the conventional direction, as in the Swedish 90, and sometimes at an intermedi- ate angle, as in the Swedish 45. The small rollers attached around the circumference of the wheel are passive and the wheel's primary axis serves as the only actively powered joint. The key advantage of this design is that, although the wheel rotation is powered only along the one principal axis (through the axle), the wheel can kinematically move with very little friction along many possible trajectories, not just forward and backward.

The spherical wheel is a truly omnidirectional wheel, often designed so that it may be actively powered to spin along any direction. One mechanism for implementing this spher- ical design imitates the computer mouse, providing actively powered rollers that rest against the top surface of the sphere and impart rotational force.

Regardless of what wheel is used, in robots designed for all-terrain environments and in robots with more than three wheels, a suspension system is normally required to maintain wheel contact with the ground. One of the simplest approaches to suspension is to design flexibility into the wheel itself. For instance, in the case of some four-wheeled indoor robots that use castor wheels, manufacturers have applied a deformable tire of soft rubber to the wheel to create a primitive suspension. Of course, this limited solution cannot compete with a sophisticated suspension system in applications where the robot needs a more dynamic suspension for significantly non flat terrain.

2.3.1.2 Wheel geometry

The choice of wheel types for a mobile robot is strongly linked to the choice of wheel arrangement, or wheel geometry. The mobile robot designer must consider these two issues simultaneously when designing the locomoting mechanism of a wheeled robot. Why do wheel type and wheel geometry matter? Three fundamental characteristics of a robot are governed by these choices: maneuverability, controllability, and stability.

Unlike automobiles, which are largely designed for a highly standardized environment (the road network), mobile robots are designed for applications in a wide variety of situa- tions. Automobiles all share similar wheel configurations because there is one region in the design space that maximizes maneuverability, controllability, and stability for their stan- dard environment: the paved roadway. However, there is no single wheel configuration that maximizes these qualities for the variety of environments faced by different mobile robots. So you will see great variety in the wheel configurations of mobile robots. In fact, few robots use the Ackerman wheel configuration of the automobile because of its poor maneu- verability, with the exception of mobile robots designed for the road system (figure 2.20). Table 2.1 gives an overview of wheel configurations ordered by the number of wheels.

This table shows both the selection of particular wheel types and their geometric configu- ration on the robot chassis. Note that some of the configurations shown are of little use in mobile robot applications. For instance, the two-wheeled bicycle arrangement has moder- ate maneuverability and poor controllability. Like a single-legged hopping machine, it can never stand still. Nevertheless, this table provides an indication of the large variety

of wheel configurations that are possible in mobile robot design.

The number of variations in table 2.1 is quite large. However, there are important trends and groupings that can aid in comprehending the advantages and disadvantages of each configuration. Below, we identify some of the key trade-offs in terms of the three issues we identified earlier: stability, maneuverability, and controllability.

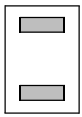
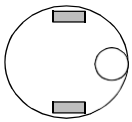
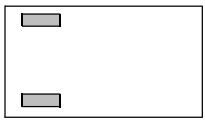
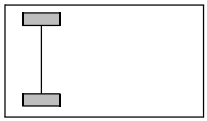
2.3.1.3 Stability

Surprisingly, the minimum number of wheels required for static stability is two. As shown above, a two-wheel differential-drive robot can achieve static stability if the center of mass is below the wheel axle. Cye is a commercial mobile robot that uses this wheel configura- tion (figure 2.21).

However, under ordinary circumstances such a solution requires wheel diameters that are impractically large. Dynamics can also cause a two-wheeled robot to strike the floor with a third point of contact, for instance, with sufficiently high motor torques from stand- still. Conventionally, static stability requires a minimum of three wheels, with the addi- tional caveat that the center of gravity must be contained within the triangle formed by the ground contact points of the wheels. Stability can be further improved by adding more wheels, although once the number of contact points exceeds three, the hyperstatic nature of the geometry will require some form of flexible suspension on uneven terrain.

Table 2.1

Wheel configurations for rolling vehicles

# of wheels	Arrangement	Description	Typical examples
2		One steering wheel in the front, one traction wheel in the rear	Bicycle, motorcycle
		Two-wheel differential drive with the center of mass (COM) below the axle	Cye personal robot
3		Two-wheel centered differen-tial drive with a third point of contact	Nomad Scout, smartRob EPFL
		Two independently driven wheels in the rear/front, 1 unpowered omnidirectional wheel in the front/rear	Many indoor robots, including the EPFL robots Pygmalion and Alice
		Two connected traction wheels (differential) in rear, 1 steered free wheel	Piaggio minitrucks

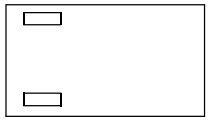
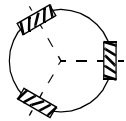
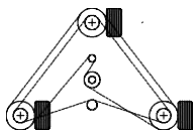
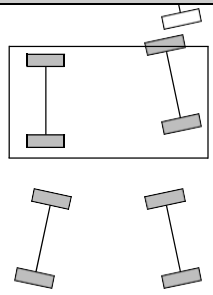
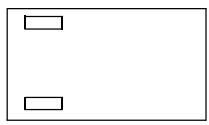
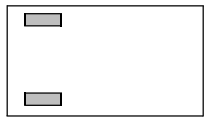
		in front	
		Two free wheels in rear, 1 steered traction wheel in front	Neptune (Carnegie Mellon University), Hero-1
		Three motorized Swedish or spherical wheels arranged in a triangle; omnidirectional move- ment is possible	Stanford wheel Tribolo EPFL, Palm Pilot Robot Kit (CMU)
		Three synchronously motorized and steered wheels; the orienta- tion is not controllable	“Synchro drive” Denning MRV-2, Geor- gia Institute of Technol- ogy, I-Robot B24, Nomad 200

Table 2.1

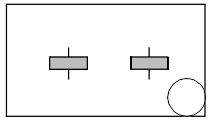
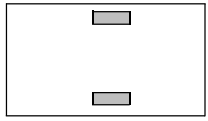
Wheel configurations for rolling vehicles

# of wheels	Arrangement	Description	Typical examples
4		Two motorized wheels in the rear, 2 steered wheels in the front; steering has to be differ- ent for the 2 wheels to avoid slipping/skidding.	Car with rear-wheel drive
		Two motorized and steered wheels in the front, 2 free wheels in the rear; steering has to be different for the 2 wheels to avoid slipping/skidding.	Car with front-wheel drive
		Four steered and motorized wheels	Four-wheel drive, four- wheel steering Hyperion (CMU)
		Two traction wheels (differen- tial) in rear/front, 2 omnidirec- tional wheels in the	Charlie (DMT-EPFL)

		front/rear	
		Four omnidirectional wheels	Carnegie Mellon Uranus
		Two-wheel differential drive with 2 additional points of con- tact	EPFL Khepera, Hyperbot Chip
		Four motorized and steered castor wheels	Nomad XR4000

Table 2.1

Wheel configurations for rolling vehicles

# of wheels	Arrangement	Description	Typical examples
6		Two motorized and steered wheels aligned in center, 1 omnidirectional wheel at each corner	First
		Two traction wheels (differen- tial) in center, 1 omnidirec- tional wheel at each corner	Terregator (Carnegie Mel- lon University)
Icons for the each wheel type are as follows:			
	unpowered omnidirectional wheel (spherical, castor, Swedish);		
	motorized Swedish wheel (Stanford wheel);		
	unpowered standard wheel;		
	motorized standard wheel;		
	motorized and steered castor wheel;		
	steered standard wheel;		

	connected wheels.

2.3.1.4 Maneuverability

Some robots are omnidirectional, meaning that they can move at any time in any direction along the ground plane (*x*, *y*) regardless of the orientation of the robot around its vertical axis. This level of maneuverability requires wheels that can move in more than just one direction, and so omnidirectional robots usually employ Swedish or spherical wheels that are powered. A good example is Uranus, shown in figure 2.24. This robot uses four Swed- ish wheels to rotate and translate independently and without constraints.

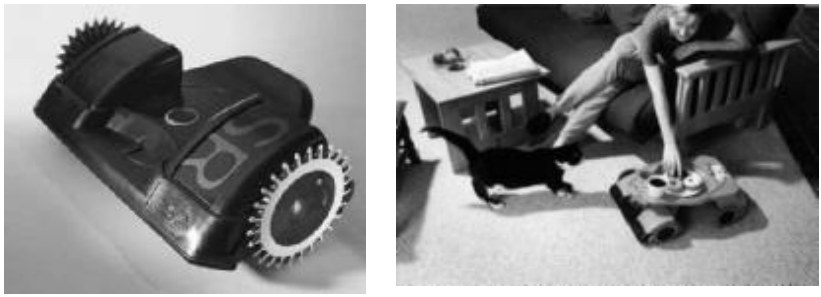


Figure 2.21

Cye, a commercially available domestic robot that can vacuum and make deliveries in the home, is built by Aethon Inc. (http://www.aethon.com). © Aethon Inc.

In general, the ground clearance of robots with Swedish and spherical wheels is some- what limited due to the mechanical constraints of constructing omnidirectional wheels. An interesting recent solution to the problem of omnidirectional navigation while solving this ground-clearance problem is the four-castor wheel configuration in which each castor wheel is actively steered and actively translated. In this configuration, the robot is truly omnidirectional because, even if the castor wheels are facing a direction perpendicular to the desired direction of travel, the robot can still move in the desired direction by steering these wheels. Because the vertical axis is offset from the ground-contact path, the result of this steering motion is robot motion.

In the research community, other classes of mobile robots are popular which achieve high maneuverability, only slightly inferior to that of the omnidirectional configurations. In such robots, motion in a particular direction may initially require a rotational

motion. With a circular chassis and an axis of rotation at the center of the robot, such a robot can spin without changing its ground footprint. The most popular such robot is the two-wheel differential-drive robot where the two wheels rotate around the center point of the robot. One or two additional ground contact points may be used for stability, based on the appli- cation specifics.

In contrast to the above configurations, consider the Ackerman steering configuration common in automobiles. Such a vehicle typically has a turning diameter that is larger than the car. Furthermore, for such a vehicle to move sideways requires a parking maneuver con- sisting of repeated changes in direction forward and backward. Nevertheless, Ackerman steering geometries have been especially popular in the hobby robotics market, where a robot can be built by starting with a remote control racecar kit and adding sensing and autonomy to the existing mechanism. In addition, the limited maneuverability of Ackerman

steering has an important advantage: its directionality and steering geometry provide it with very good lateral stability in high-speed turns.

2.3.1.5 Controllability

There is generally an inverse correlation between controllability and maneuverability. For example, the omnidirectional designs such as the four-castor wheel configuration require significant processing to convert desired rotational and translational velocities to individual wheel commands. Furthermore, such omnidirectional designs often have greater degrees of freedom at the wheel. For instance, the Swedish wheel has a set of free rollers along the wheel perimeter. These degrees of freedom cause an accumulation of slippage, tend to reduce dead-reckoning accuracy and increase the design complexity.

Controlling an omnidirectional robot for a specific direction of travel is also more diffi- cult and often less accurate when compared to less maneuverable designs. For example, an Ackerman steering vehicle can go straight simply by locking the steerable wheels and driv- ing the drive wheels. In a differential-drive vehicle, the two motors attached to the two wheels must be driven along exactly the same velocity profile, which can be challenging considering variations between wheels, motors, and environmental differences. With four- wheel omnidrive, such as the Uranus robot, which has four Swedish wheels, the problem is even harder because all four wheels must be driven at exactly the same speed for the robot to travel in a perfectly straight line.

In summary, there is no “ideal” drive configuration that simultaneously maximizes sta- bility, maneuverability, and controllability. Each mobile robot application places unique

constraints on the robot design problem, and the designer's task is to choose the most appropriate drive configuration possible from among this space of compromises.

2.3.2 Wheeled locomotion: case studies

Below we describe four specific wheel configurations, in order to demonstrate concrete applications of the concepts discussed above to mobile robots built for real-world activities.

2.3.2.1 Synchro drive

The synchro drive configuration (figure 2.22) is a popular arrangement of wheels in indoor mobile robot applications. It is an interesting configuration because, although there are three driven and steered wheels, only two motors are used in total. The one translation motor sets the speed of all three wheels together, and the one steering motor spins all the wheels together about each of their individual vertical steering axes. But note that the wheels are being steered with respect to the robot chassis, and therefore there is no direct way of reorienting the robot chassis. In fact, the chassis orientation does drift over time due to uneven tire slippage, causing rotational dead-reckoning error.

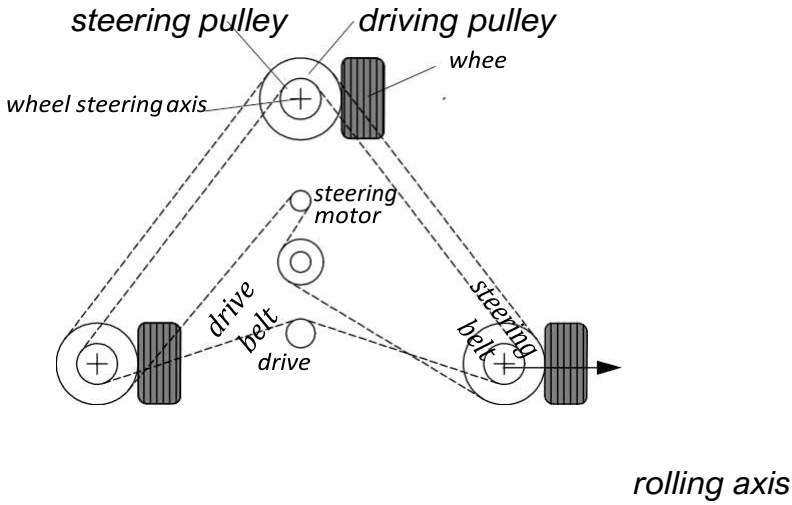


Figure 2.22

Synchro drive: The robot can move in any direction; however, the orientation of the chassis is not controllable.

Synchro drive is particularly advantageous in cases where omnidirectionality is sought. So long as each vertical steering axis is aligned with the contact path of each tire, the robot can always reorient its wheels and move along a new trajectory without changing its foot- print. Of course, if the robot chassis has directionality and the designers intend to reorient the chassis purposefully, then synchro drive is only appropriate when combined with an independently rotating turret that attaches to the wheel chassis. Commercial research robots such as the Nomadics 150 or the RWI B21r have been sold with this configuration (figure 1.12).

In terms of dead reckoning, synchro drive systems are generally superior to true omni- directional configurations but inferior to differential-drive and Ackerman steering systems. There are two main reasons for this. First and foremost, the translation motor generally drives the three wheels using a single belt. Because of to slop and backlash in the drive train, whenever the drive motor engages, the closest wheel begins spinning before the fur- thest wheel, causing a small change in the orientation of the chassis. With additional changes in motor speed, these small angular shifts accumulate to create a large error in ori- entation during dead reckoning. Second, the mobile robot has no direct control over the ori- entation of the chassis. Depending on the orientation of the chassis, the wheel thrust can be highly asymmetric, with two wheels on one side and the third wheel alone, or symmetric, with one wheel on each side and one wheel straight ahead or behind, as shown in figure 2.22. The asymmetric cases result in a variety of errors when tire-ground slippage can occur, again causing errors in dead reckoning of robot orientation.

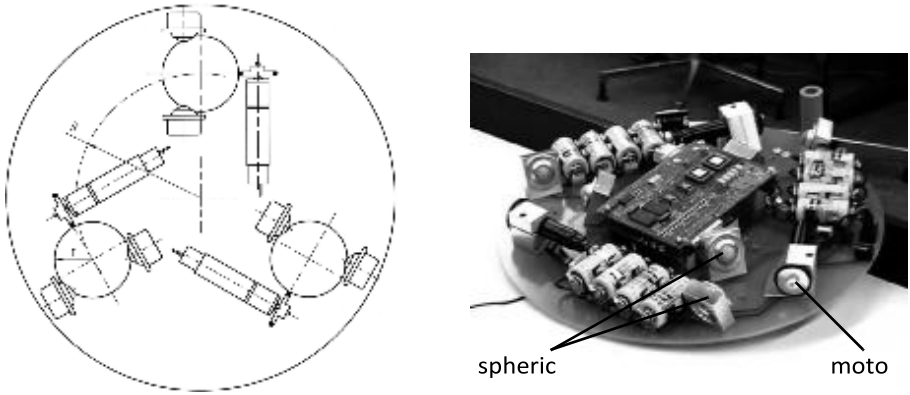


Figure 2.23

The Tribolo designed at EPFL (Swiss Federal Institute of Technology, Lausanne, Switzerland. Left: arrangement of spheric bearings and motors (bottom view). Right: Picture of the robot without the spherical wheels (bottom view).

2.3.2.2 Omnidirectional drive

As we will see later in section 3.4.2, omnidirectional movement is of great interest for com- plete maneuverability. Omnidirectional robots that are able to move in any direction (x y) at any time are also holonomic (see section 3.4.2). They can be realized by either using spherical, castor, or Swedish wheels. Three examples of such holonomic robots are presented below.

Omnidirectional locomotion with three spherical wheels. The omnidirectional robot depicted in figure 2.23 is based on three spherical wheels, each actuated by one motor. In this

design, the spherical wheels are suspended by three contact points, two given by spher- ical bearings and one by a wheel connected to the motor axle. This concept provides excel- lent maneuverability and is simple in design. However, it is limited to flat surfaces and small loads, and it is quite difficult to find round wheels with high friction coefficients.

Omnidirectional locomotion with four Swedish wheels. The omnidirectional arrange- ment depicted in figure 2.24 has been used successfully on several research robots, includ- ing the Carnegie Mellon Uranus. This configuration consists of four Swedish 45-degree wheels, each driven by a separate motor. By varying the direction of rotation and relative speeds of the four wheels, the robot can be moved along any trajectory in the plane and, even more impressively, can simultaneously spin around its vertical axis.

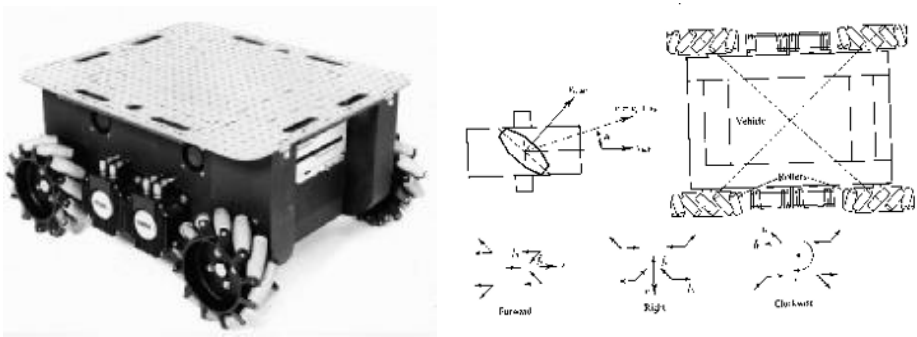


Figure 2.24

The Carnegie Mellon Uranus robot, an omnidirectional robot with four powered-swedish 45 wheels.

For example, when all four wheels spin “forward” or “backward” the robot as a whole moves in a straight line forward or backward, respectively. However, when one diagonal pair of wheels is spun in the same direction and the other diagonal pair is spun in the oppo- site direction, the robot moves laterally.

This four-wheel arrangement of Swedish wheels is not minimal in terms of control motors. Because there are only three degrees of freedom in the plane, one can build a three- wheel omnidirectional robot chassis using three Swedish 90-degree wheels as shown in table 2.1. However, existing examples such as Uranus have been designed with four wheels owing to capacity and stability considerations.

One application for which such omnidirectional designs are particularly amenable is mobile manipulation. In this case, it is desirable to reduce the degrees of freedom of the manipulator arm to save arm mass by using the mobile robot chassis motion

for gross motion. As with humans, it would be ideal if the base could move omnidirectionally with- out greatly impacting the position of the manipulator tip, and a base such as Uranus can afford precisely such capabilities.

Omnidirectional locomotion with four castor wheels and eight motors. Another solu- tion for omnidirectionality is to use castor wheels. This is done for the Nomad XR4000 from Nomadic Technologies (fig. 2.25), giving it excellent maneuverability. Unfortu- nately, Nomadic has ceased production of mobile robots.

The above three examples are drawn from table 2.1, but this is not an exhaustive list of all wheeled locomotion techniques. Hybrid approaches that combine legged and wheeled locomotion, or tracked and wheeled locomotion, can also offer particular advantages. Below are two unique designs created for specialized applications.



Figure 2.25

The Nomad XR4000 from Nomadic Technologies had an arrangement of four castor wheels for holo- nomic motion. All the castor wheels are driven and steered, thus requiring a precise synchronization and coordination to obtain a precise movement in x y and .

2.3.2.3 Tracked slip/skid locomotion

In the wheel configurations discussed above, we have made the assumption that wheels are not allowed to skid against the surface. An alternative form of steering, termed slip/skid, may be used to reorient the robot by spinning wheels that are facing the same direction at different speeds or in opposite directions. The army tank operates this way, and the Nanokhod (figure 2.26) is an example of a mobile robot based on the same concept.

Robots that make use of tread have much larger ground contact patches, and this can sig- nificantly improve their maneuverability

in loose terrain compared to conventional wheeled designs. However, due to this large ground contact patch, changing the orientation of the robot usually requires a skidding turn, wherein a large portion of the track must slide against the terrain.

The disadvantage of such configurations is coupled to the slip/skid steering. Because of the large amount of skidding during a turn, the exact center of rotation of the robot is hard to predict and the exact change in position and orientation is also subject to variations depending on the ground friction. Therefore, dead reckoning on such robots is highly inac- curate. This is the trade-off that is made in return for extremely good maneuverability and traction over rough and loose terrain. Furthermore, a slip/skid approach on a high-friction surface can quickly overcome the torque capabilities of the motors being used. In terms of power efficiency, this approach is reasonably efficient on loose terrain but extremely inef- ficient otherwise.

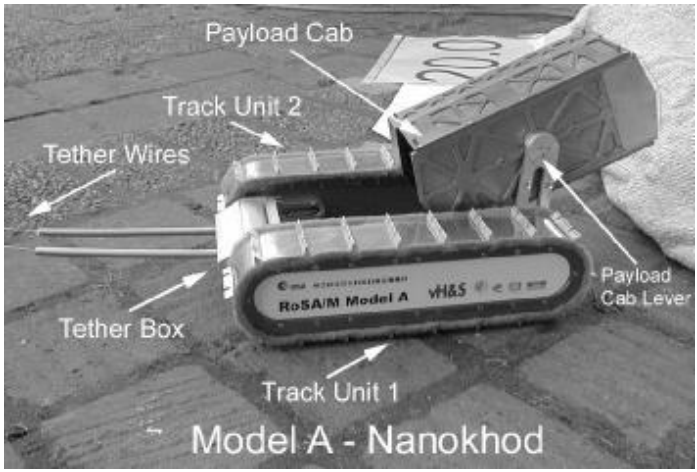


Figure 2.26

The microrover Nanokhod, developed by von Hoerner & Sulger GmbH and the Max Planck Institute, Mainz, for the European Space Agency (ESA), will probably go to Mars [138, 154].

2.3.2.4 Walking wheels

Walking robots might offer the best maneuverability in rough terrain. However, they are inefficient on flat ground and need sophisticated control. Hybrid solutions, combining the adaptability of legs with the efficiency of wheels, offer an interesting compromise. Solu- tions that passively adapt to the terrain are of particular interest for field and space robotics. The Sojourner robot of NASA/JPL (see figure 1.2) represents such a hybrid solution, able to overcome objects up to the size of the wheels. A more recent mobile robot design for similar applications has recently been produced by EPFL (figure 2.27).

This robot, called Shrimp, has six motorized wheels and is capable of climbing objects up to two times its wheel diameter [97, 133]. This enables it to climb regular stairs though the robot is even smaller than the Sojourner. Using a rhombus configuration, the Shrimp has a steering wheel in the front and the rear, and two wheels arranged on a bogie on each side. The front wheel has a spring suspension to guarantee optimal ground contact of all wheels at any time. The steering of the rover is realized by synchronizing the steering of the front and rear wheels and the speed difference of the bogie wheels. This allows for high-precision maneuvers and turning on the spot with minimum slip/skid of the four center wheels. The use of parallel articulations for the front wheel and the bogies creates a virtual center of rotation at the level of the wheel axis. This ensures maximum stability and climbing abilities even for very low friction coefficients between the wheel and the ground.

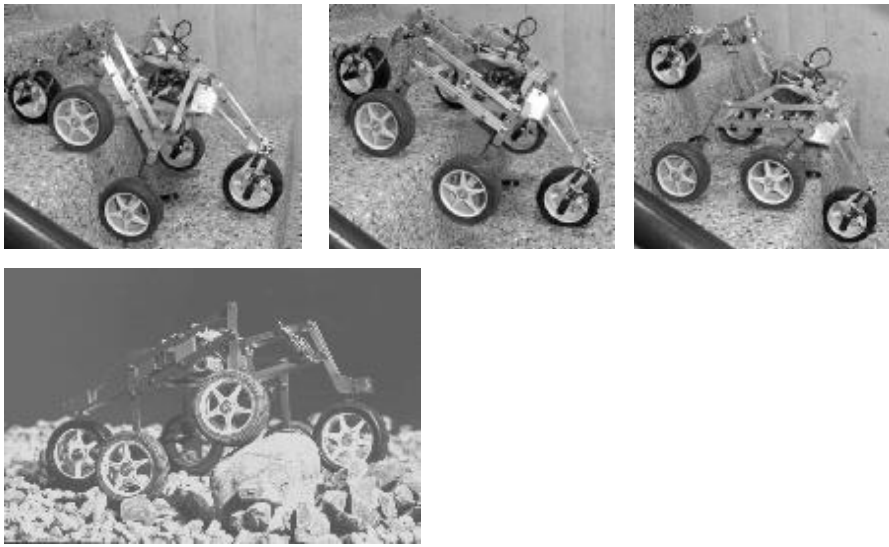


Figure 2.27

Shrimp, an all-terrain robot with outstanding passive climbing abilities (EPFL [97, 133]).

The climbing ability of the Shrimp is extraordinary in comparison to most robots of sim- ilar mechanical complexity, owing much to the specific geometry and thereby the manner in which the center of mass (COM) of the robot shifts with respect to the wheels over time. In contrast, the Personal Rover demonstrates active COM shifting to climb ledges that are also several times the diameter of its wheels, as demonstrated in figure 2.28. A majority of the weight of the Personal Rover is borne at the upper end of its swinging boom. A dedi- cated motor drives the boom to change the front/rear weight distribution in order to facili- tate step-climbing. Because this COM-shifting

scheme is active, a control loop must explicitly decide how to move the boom during a climbing scenario. In this case the Personal Rover accomplished this closed-loop control by inferring terrain based on measure- ments of current flowing to each independently driven wheel [66].

As mobile robotics research matures we find ourselves able to design more intricate mechanical systems. At the same time, the control problems of inverse kinematics and dynamics are now so readily conquered that these complex mechanics can in general be controlled. So, in the near future, we can expect to see a great number of unique, hybrid mobile robots that draw together advantages from several of the underlying locomotion mechanisms that we have discussed in this chapter. They will each be technologically impressive, and each will be designed as the expert robot for its particular environmental niche.

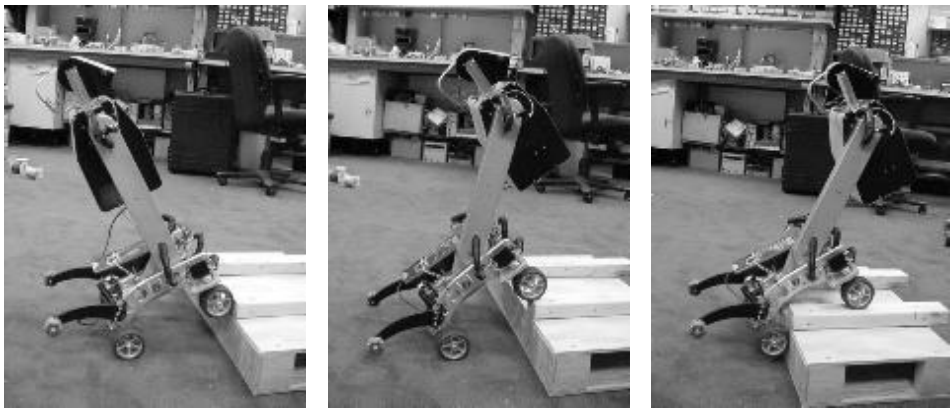


Figure 2.28

The Personal Rover, demonstrating ledge climbing using active center-of-mass shifting.

2. Literature Review

3 Mobile Robot Kinematics

3.1 Introduction

Kinematics is the most basic study of how mechanical systems behave. In mobile robotics, we need to understand the mechanical behavior of the robot both in order to design appropriate mobile robots for tasks and to understand how to create control software for an instance of mobile robot hardware.

Of course, mobile robots are not the first complex mechanical systems to require such analysis. Robot manipulators have been the subject of intensive study for more than thirty years. In some ways, manipulator robots are much more complex than early

mobile robots: a standard welding robot may have five or more joints, whereas early mobile robots were simple differential-drive machines. In recent years, the robotics community has achieved a fairly complete understanding of the kinematics and even the dynamics (i.e., relating to force and mass) of robot manipulators [11, 32].

The mobile robotics community poses many of the same kinematic questions as the robot manipulator community. A manipulator robot's workspace is crucial because it defines the range of possible positions that can be achieved by its end effector relative to its fixture to the environment. A mobile robot's workspace is equally important because it defines the range of possible poses that the mobile robot can achieve in its environment. The robot arm's controllability defines the manner in which active engagement of motors can be used to move from pose to pose in the workspace. Similarly, a mobile robot's controllability defines possible paths and trajectories in its workspace. Robot dynamics places additional constraints on workspace and trajectory due to mass and force considerations. The mobile robot is also limited by dynamics; for instance, a high center of gravity limits the practical turning radius of a fast, car-like robot because of the danger of rolling.

But the chief difference between a mobile robot and a manipulator arm also introduces a significant challenge for *position estimation*. A manipulator has one end fixed to the environment. Measuring the position of an arm's end effector is simply a matter of understand- ing the kinematics of the robot and measuring the position of all intermediate joints. The manipulator's position is thus always computable by looking at current sensor data. But a

mobile robot is a self-contained automaton that can wholly move with respect to its envi- ronment. There is no direct way to measure a mobile robot's position instantaneously. Instead, one must integrate the motion of the robot over time. Add to this the inaccuracies of *motion estimation* due to slippage and it is clear that measuring a mobile robot's position precisely is an extremely challenging task.

The process of understanding the motions of a robot begins with the process of describ- ing the contribution each wheel provides for motion. Each wheel has a role in enabling the whole robot to move. By the same token, each wheel also imposes constraints on the robot's motion; for example, refusing to skid laterally. In the following section, we intro- duce notation that allows expression of robot motion in a global reference frame as well as the robot's local reference frame. Then, using this notation, we demonstrate the construc- tion of simple forward kinematic models of motion, describing how the robot as a whole moves as a function of its geometry and individual wheel behavior. Next, we formally describe the kinematic constraints of individual wheels, and then combine these kinematic constraints to express the whole robot's kinematic constraints. With these tools, one can evaluate the paths and trajectories that define the robot's maneuverability.

3.2 Kinematic Models and Constraints

Deriving a model for the whole robot's motion is a bottom-up process. Each individual wheel contributes to the robot's motion and, at the same time, imposes constraints on robot motion. Wheels are tied together based on robot chassis geometry, and therefore their con- straints combine to form constraints on the

overall motion of the robot chassis. But the forces and constraints of each wheel must be expressed with respect to a clear and consis- tent reference frame. This is particularly important in mobile robotics because of its self- contained and mobile nature; a clear mapping between global and local frames of reference is required. We begin by defining these reference frames formally, then using the resulting formalism to annotate the kinematics of individual wheels and whole robots. Throughout this process we draw extensively on the notation and terminology presented in [52].

3.2.1 Representing robot position

Throughout this analysis we model the robot as a rigid body on wheels, operating on a hor- izontal plane. The total dimensionality of this robot chassis on the plane is three, two for position in the plane and one for orientation along the vertical axis, which is orthogonal to the plane. Of course, there are additional degrees of freedom and flexibility due to the wheel axles, wheel steering joints, and wheel castor joints. However by robot *chassis* we refer only to the rigid body of the robot, ignoring the joints and degrees of freedom internal to the robot and its wheels.

Y_I

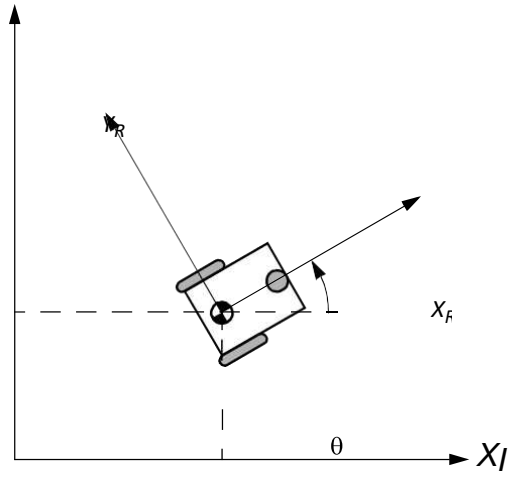


Figure 3.1

The global reference frame and the robot local reference frame.

In order to specify the position of the robot on the plane we establish a relationship between the global reference frame of the plane and the local reference frame of the robot, as in figure 3.1. The axes X_I and Y_I define an arbitrary inertial basis on the plane as the global reference frame from some origin O: $\{X_I, Y_I\}$. To specify the position of the robot, choose a point P on the robot chassis as its position reference point. The basis $\{X_R, Y_R\}$

defines two axes relative to *P* on the robot chassis and is thus the robot's local reference frame. The position of *P* in the global reference frame is specified by coordinates *x* and *y*, and the angular difference between the global and local reference frames is given by θ. We can describe the pose of the robot as a vector with these three elements. Note the use of the subscript *I* to clarify the basis of this pose as the global reference frame:

$$\xi_I = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \tag{3.1}$$

To describe robot motion in terms of component motions, it will be necessary to map motion along the axes of the global reference frame to motion along the axes of the robot's local reference frame. Of course, the mapping is a function of the current pose of the robot. This mapping is accomplished using the *orthogonal rotation matrix*:

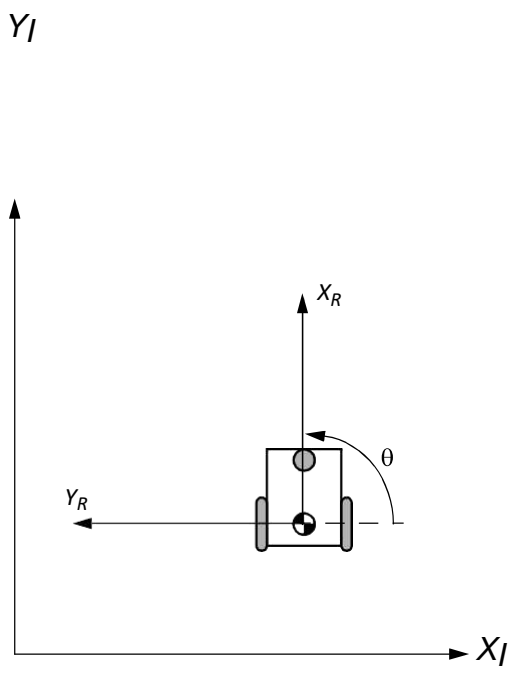


Figure 3.2

The mobile robot aligned with a global axis.

$$R(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.2}$$

This matrix can be used to map motion in the global reference frame $\{X_I, Y_I\}$ to motion in terms of the local reference frame $\{X_R, Y_R\}$. This operation is denoted by $R(\theta)^I$ because the computation of this operation depends on the value of θ:

$$\dot{\xi}_R = R(-\frac{\pi}{2})\dot{\xi}_I \tag{3.3}$$

For example, consider the robot in figure 3.2. For this robot, because $\theta = -\frac{\pi}{2}$ we can easily compute the instantaneous rotation matrix R:

$$R(-\frac{\pi}{2}) = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.4}$$

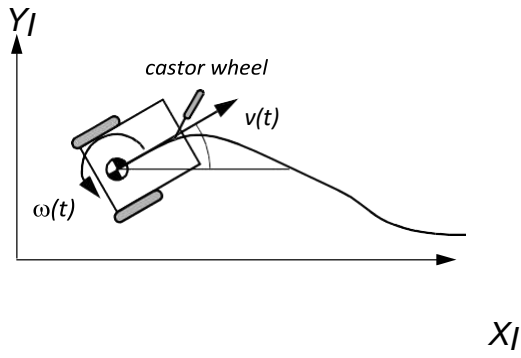


Figure 3.3

A differential-drive robot in its global reference frame.

Given some velocity ($\dot{x}$, $\dot{y}$, $\dot{\theta}$) in the global reference frame we can compute the components of motion along this robot's local axes X_R and Y_R . In this case, due to the specific angle of the robot, motion along X_R is equal to $\dot{y}$ and motion along Y_R is $-\dot{x}$:

$$\dot{\xi}_R = R(\theta)\dot{\xi} = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \dot{y} \\ -\dot{x} \\ \dot{\theta} \end{bmatrix} \quad (3.5)$$

3.2.2 Forward kinematic models

In the simplest cases, the mapping described by equation (3.3)

is sufficient to generate a formula that captures the forward kinematics of the mobile robot: how does the robot move, given its geometry and the speeds of its wheels? More formally, consider the example shown in figure 3.3.

This differential drive robot has two wheels, each with diameter r. Given a point P cen- tered between the two drive wheels, each wheel is a distance l from P. Given r, l, θ, and the spinning speed of each wheel, $\dot{\phi}_1$ and $\dot{\phi}_2$, a forward kinematic model would predict the robot's overall speed in the global reference frame:

$$\dot{\xi}_I = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = f(l, r, \theta, \dot{\phi}_1, \dot{\phi}_2) \tag{3.6}$$

From equation (3.3) we know that we can compute the robot's motion in the global ref- erence frame from motion in its local reference frame: $\dot{\xi}_I = R(\theta)^{-1}\dot{\xi}_R$. Therefore, the strat- egy will be to first compute the contribution of each of the two wheels in the local reference

frame, $\dot{\xi}_R$. For this example of a differential-drive chassis, this problem is particularly straightforward.

Suppose that the robot's local reference frame is aligned such that the robot moves for- ward along $+X_R$, as shown in figure 3.1. First consider the contribution of each wheel's spinning speed to the translation speed at P in the direction of $+X_R$. If one wheel spins while the other wheel contributes nothing and is stationary, since P is halfway between the two wheels, it will move instantaneously with half the speed: $\dot{x} = \frac{1}{2} r \dot{\theta}_1$ and $\dot{x} = \frac{1}{2} r \dot{\theta}_2$. In a differential drive robot, these two contributions can simply be added to calculate the $\dot{x}_R$ component of $\dot{\xi}_R$. Consider, for example, a differential robot in which each wheel spins with equal speed but in opposite directions. The result is a stationary, spinning robot. As expected, $\dot{x}_R$ will be zero in this case. The value of $\dot{y}_R$ is even simpler to calculate. Neither wheel can contribute to sideways motion in the robot's reference frame, and $\dot{y}_R$ is always zero. Finally, we must compute the rotational component $\dot{\theta}$ of $\dot{\xi}_R$. Once again, the contributions of each wheel can be computed independently and just added. Consider the right wheel (we will call this wheel 1). Forward spin of this wheel results in *counterclockwise* rotation at point P. Recall that if wheel 1 spins alone, the robot pivots around wheel 2. The rotation velocity $\dot{\theta}_1$ at P can be computed because the wheel is instantaneously moving along the arc of a circle of radius $2l$:

$$\dot{\theta}_1 = \frac{r\dot{\phi}_1}{2l} \tag{3.7}$$

The same calculation applies to the left wheel, with the exception that forward spin results in *clockwise* rotation at point P :

$$\dot{\phi}_2 = \frac{-r\dot{\theta}_2}{2l} \tag{3.8}$$

Combining these individual formulas yields a kinematic model for the differential-drive example robot:

$$\dot{\xi}_I = R(\theta)^{-1} \begin{bmatrix} \frac{r\dot{\theta}_1}{2} + \frac{r\dot{\theta}_2}{2} \\ 0 \\ \frac{r\dot{\theta}_1}{2l} + \frac{-r\dot{\theta}_2}{2l} \end{bmatrix} \tag{3.9}$$

We can now use this kinematic model in an example. However, we must first compute

$R(\theta)^{-1}$. In general, calculating the inverse of a matrix may be challenging. In this case, however, it is easy because it is simply a transform from θI to θR rather than vice versa:

$$R(\theta)^{-1} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.10}$$

Suppose that the robot is positioned such that $\theta = \theta 2$, $r = 1$, and $l = 1$. If the robot

engages its wheels unevenly, with speeds $\dot{\theta}_1 = 4$ and $\dot{\theta}_2 = 2$, we can compute its veloc- ity in the global reference frame:

$$\dot{\theta}_I = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 3 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 3 \\ 1 \end{bmatrix} \tag{3.11}$$

So this robot will move instantaneously along the *y*-axis of the global reference frame with speed 3 while rotating with speed 1. This approach to kinematic modeling can provide information about the motion of a robot given its component wheel speeds in straightfor- ward cases. However, we wish to determine the space of possible motions for each robot chassis design. To do this, we must go further, describing formally the constraints on robot motion imposed by each wheel. Section 3.2.3 begins this process by describing constraints for various wheel types; the

rest of this chapter provides tools for analyzing the characteristics and workspace of a robot given these constraints.

3.2.3 Wheel kinematic constraints

The first step to a kinematic model of the robot is to express constraints on the motions of individual wheels. Just as shown in section 3.2.2, the motions of individual wheels can later be combined to compute the motion of the robot as a whole. As discussed in chapter 2, there are four basic wheel types with widely varying kinematic properties. Therefore, we begin by presenting sets of constraints specific to each wheel type.

However, several important assumptions will simplify this presentation. We assume that the plane of the wheel always remains vertical and that there is in all cases one single point of contact between the wheel and the ground plane. Furthermore, we assume that there is no sliding at this single point of contact. That is, the wheel undergoes motion only under conditions of pure rolling and rotation about the vertical axis through the contact point. For a more thorough treatment of kinematics, including sliding contact, refer to [25].

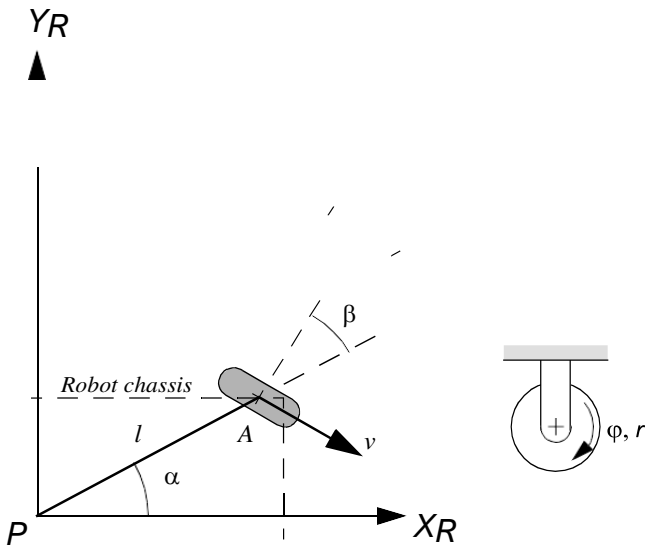


Figure 3.4

A fixed standard wheel and its parameters.

Under these assumptions, we present two constraints for every wheel type. The first con- straint enforces the concept of rolling contact – that the wheel must roll when motion takes place in the appropriate direction. The second constraint enforces the concept of no lateral slippage – that the wheel must not slide orthogonal to the wheel plane.

3.2.3.1 Fixed standard wheel

The fixed standard wheel has no vertical axis of rotation for steering. Its angle to the chassis is thus fixed, and it is limited to motion back and forth along the wheel plane and rotation around

its contact point with the ground plane. Figure 3.4 depicts a fixed standard wheel A
and indicates its position pose relative to the robot's local reference frame $\{X_R, Y_R\}$. The position of A is expressed in polar coordinates by distance l and angle α. The angle of the wheel plane relative to the chassis is denoted by β, which is fixed since the fixed standard wheel is not steerable. The wheel, which has radius r, can spin over time, and so its rota- tional position around its horizontal axle is a function of time t: $\varphi(t)$.

The rolling constraint for this wheel enforces that all motion along the direction of the wheel plane must be accompanied by the appropriate amount of wheel spin so that there is pure rolling at the contact point:

$$\begin{bmatrix} \sin(\alpha + \beta) & -\cos(\alpha + \beta) & -l\cos\beta \end{bmatrix} R(\theta) \begin{bmatrix} \dot{x}_I \\ \dot{y}_I \\ \dot{\theta} \end{bmatrix} - r\dot{\varphi} = 0 \tag{3.12}$$

The first term of the sum denotes the total motion along the wheel plane. The three elements of the vector on the left represent mappings from each of x, y, θ to their contributions for motion along the wheel plane. Note that the $R(\theta)\,\xi_I$ term is used to transform the motion parameters $\dot{\xi}_I$ that are in the global reference frame $\{X_I, Y_I, \theta\}$ into motion parameters in the local reference frame $\{X_R, Y_R, \theta\}$ as shown in example equation (3.5). This is necessary because all other parameters in the equation, β, γ, l, are in terms of the robot's local reference frame. This motion along the wheel plane must be equal, according to this constraint, to the motion accomplished by spinning the wheel, $r\dot{\varphi}$.

The sliding constraint for this wheel enforces that the component of the wheel's motion orthogonal to the wheel plane must be zero:

$$\begin{bmatrix} \cos(\alpha+\beta+\gamma) & \sin(\alpha+\beta+\gamma) & l\sin\gamma \end{bmatrix} R(\theta)\,\dot{\xi}_I = 0 \tag{3.13}$$

For example, suppose that wheel A is in a position such that $(\alpha+\beta+\gamma) = 0$, $(\alpha+\beta) = 0$, $\gamma = 0$. This would place the contact point of the wheel on X_I with the plane of the wheel oriented parallel to Y_I. If $\theta = 0$, then the sliding constraint [equation (3.13)] reduces to

$$\begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ 0 \end{bmatrix} = \tag{3.14}$$

This constrains the component of motion along X_I to be zero and since X_I and X_R are parallel in this example, the wheel is

constrained from sliding sideways, as expected.

3.2.3.2 Steered standard wheel

The steered standard wheel differs from the fixed standard wheel only in that there is an additional degree of freedom: the wheel may rotate around a vertical axis passing through the center of the wheel and the ground contact point. The equations of position for the steered standard wheel (figure 3.5) are identical to that of the fixed standard wheel shown in figure 3.4 with one exception. The orientation of the wheel to the robot chassis is no longer a single fixed value, β, but instead varies as a function of time: $\beta(t)$. The rolling and sliding constraints are

$$\begin{bmatrix} \sin(\alpha+\beta) & -\cos(\alpha+\beta) & -l\cos\beta \end{bmatrix} R(\theta)\dot{\xi}_I - r\dot{\varphi} = 0 \tag{3.15}$$

$$\begin{bmatrix} \cos(\alpha+\beta) & \sin(\alpha+\beta) & l\sin\beta \end{bmatrix} R(\theta)\dot{\xi}_I = 0 \tag{3.16}$$

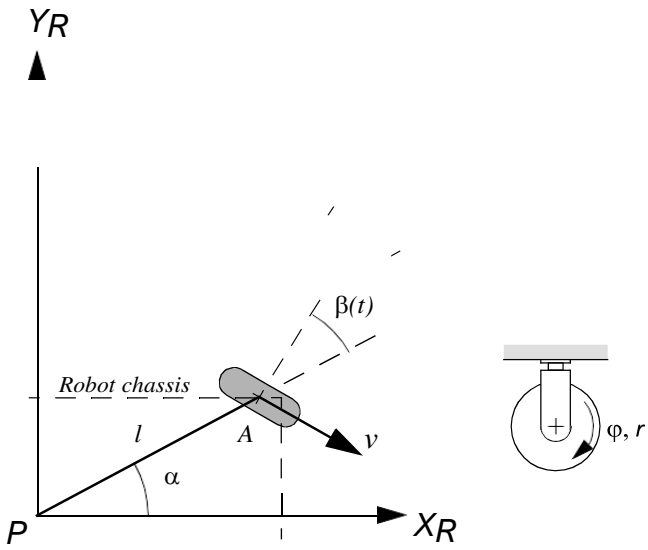


Figure 3.5

A steered standard wheel and its parameters.

These constraints are identical to those of the fixed standard wheel because, unlike [illegible],
[illegible] does not have a direct impact on the instantaneous motion constraints of a robot. It is
only by integrating over time that changes in steering angle can affect the mobility of a
vehicle. This may seem subtle, but is a very important distinction between change in steer- ing position, [illegible], and change in wheel spin, [illegible].

3.2.3.3 Castor wheel

Castor wheels are able to steer around a vertical axis. However, unlike the steered standard wheel, the vertical axis of rotation in

a castor wheel does not pass through the ground con- tact point. Figure 3.6 depicts a castor wheel, demonstrating that formal specification of the castor wheel's position requires an additional parameter.

The wheel contact point is now at position B, which is connected by a rigid rod AB of fixed length d to point A fixes the location of the vertical axis about which B steers, and this point A has a position specified in the robot's reference frame, as in figure 3.6. We assume that the plane of the wheel is aligned with AB at all times. Similar to the steered standard wheel, the castor wheel has two parameters that vary as a function of time. $\phi(t)$ represents the wheel spin over time as before. $\theta(t)$ denotes the steering angle and orienta- tion of AB over time.

For the castor wheel, the rolling constraint is identical to equation (3.15) because the offset axis plays no role during motion that is aligned with the wheel plane:

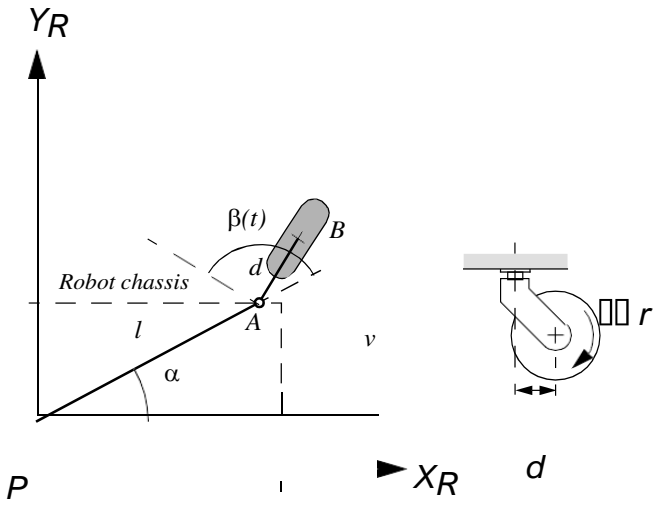


Figure 3.6

A castor wheel and its parameters.

$$\begin{bmatrix} \sin(\alpha+\beta) & -\cos(\alpha+\beta) & -l\cos\beta \\ R(\theta) & l & -r \end{bmatrix} \dot{\xi}_I = 0 \tag{3.17}$$

The castor geometry does, however, have significant impact on the sliding constraint. The critical issue is that the lateral force on the wheel occurs at point A because this is the attachment point of the wheel to the chassis. Because of the offset ground contact point rel- ative to A, the constraint that there be zero lateral movement would be wrong. Instead, the constraint is much like a rolling constraint, in that appropriate rotation of the vertical axis must take place:

$$\begin{bmatrix} \cos\beta + \gamma \sin\beta + \gamma d + l\sin\gamma \\ R(\beta)\gamma l + d\zeta \end{bmatrix} = 0 \qquad (3.18)$$

In equation (3.18), any motion orthogonal to the wheel plane must be balanced by an equivalent and opposite amount of castor steering motion. This result is critical to the success of castor wheels because by setting the value of ζ any arbitrary lateral motion can be acceptable. In a steered standard wheel, the steering action does not by itself cause a move- ment of the robot chassis. But in a castor wheel the steering action itself moves the robot chassis because of the offset between the ground contact point and the vertical axis of rota- tion.



Figure 3.7

Office chair with five castor wheels.

More concisely, it can be surmised from equations (3.17) and (3.18) that, given *any* robot chassis motion $\dot{\xi}_I$, there exists some value for spin speed $\dot{\varphi}$ and steering speed $\dot{\theta}$ such that the constraints are met. Therefore, a robot with only castor wheels can move with any velocity in the space of possible robot motions. We term such systems *omnidirectional*. A real-world example of such a system is the five-castor wheel office chair shown in figure 3.7. Assuming that all joints are able to move freely, you may select any motion vector on the plane for the chair and push it by hand. Its castor wheels will spin and steer as needed to achieve that motion without contact point sliding. By the same token, if each of the chair's castor wheels housed two motors, one for spinning and one for steering, then a control system would be able to move the chair along any trajectory in the plane. Thus, although the kinematics of castor wheels is somewhat complex, such wheels do not impose any real constraints on the kinematics of a robot chassis.

3.2.3.4 Swedish wheel

Swedish wheels have no vertical axis of rotation, yet are able to move *omnidirectionally* like the castor wheel. This is possible by adding a degree of freedom to the fixed standard wheel. Swedish wheels consist of a fixed standard wheel with rollers attached to the wheel perimeter with axes that are antiparallel to the main axis of the fixed wheel component. The exact angle between the roller axes and the main axis can vary, as shown in figure 3.8.

For example, given a Swedish 45-degree wheel, the motion vectors of the principal axis and the roller axes can be drawn as in figure 3.8. Since each axis can spin clockwise or counterclockwise, one can combine any vector along one axis with any vector along the other axis. These two axes are not necessarily independent (except in the case of the Swed- ish 90-degree wheel); however, it is visually clear that any desired direction of motion is achievable by choosing the appropriate two vectors.

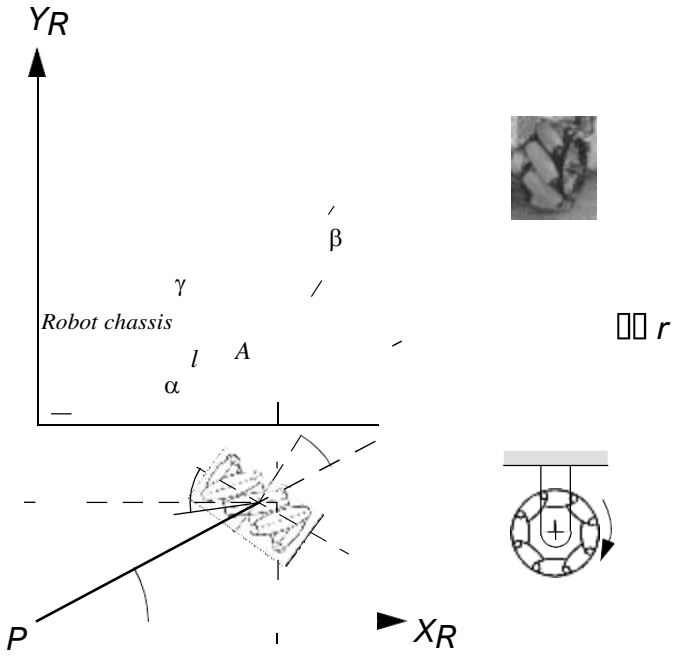


Figure 3.8

A Swedish wheel and its parameters.

The pose of a Swedish wheel is expressed exactly as in a fixed standard wheel, with the addition of a term, , representing the angle between the main wheel plane and the axis of rotation of the small circumferential rollers. This is depicted in figure 3.8 within the robot's reference frame.

Formulating the constraint for a Swedish wheel requires some subtlety. The instanta- neous constraint is due to the specific orientation of the small rollers. The axis around which these

rollers spin is a zero component of velocity at the contact point. That is, moving in that direction without spinning the main axis is not possible without sliding. The motion constraint that is derived looks identical to the rolling constraint for the fixed standard wheel in equation (3.12) except that the formula is such that the modified by adding

effective direction along which the rolling constraint holds is along this zero component rather than along the wheel plane:

$$\left[\sin(\alpha + \beta + \gamma) \quad -\cos(\alpha + \beta + \gamma) \quad -l\cos(\beta + \gamma)\right] R(\theta)\dot{\xi}_I - r\dot{\varphi} = 0 \tag{3.19}$$

Orthogonal to this direction the motion is not constrained because of the free rotation $\dot{\varphi}_{sw}$ of the small rollers.

$$\left[\cos(\alpha + \beta + \gamma) \quad \sin(\alpha + \beta + \gamma) \quad l\sin(\beta + \gamma)\right] R(\theta)\dot{\xi}_I - r\dot{\varphi} \sin\gamma - r_{sw}\dot{\varphi}_{sw} = 0 \tag{3.20}$$

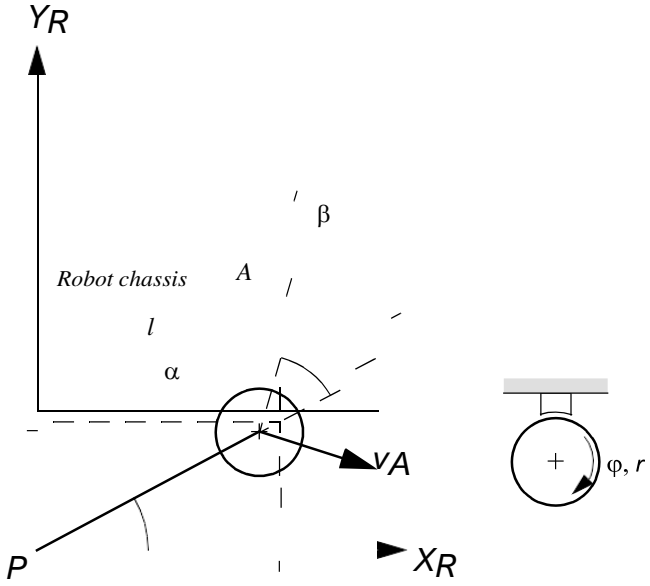


Figure 3.9

A spherical wheel and its parameters.

The behavior of this constraint and thereby the Swedish wheel changes dramatically as the value varies. Consider = 0 . This represents the swedish 90-degree wheel. In this case, the zero component of velocity is in line with the wheel plane and so equation (3.19) reduces exactly to equation (3.12), the fixed standard wheel rolling constraint. But because
of the rollers, there is no sliding constraint orthogonal to the wheel plane [see equation (3.20)]. By varying the value of , any desired motion vector can be made to satisfy equation (3.19) and therefore the wheel is omnidirectional. In fact, this special case of the Swed- ish design results in fully decoupled

motion, in that the rollers and the main wheel provide orthogonal directions of motion.

At the other extreme, consider $\gamma = \pi/2$. In this case, the rollers have axes of rotation

that are parallel to the main wheel axis of rotation. Interestingly, if this value is substituted for γ in equation (3.19) the result is the fixed standard wheel sliding constraint, equation (3.13). In other words, the rollers provide no benefit in terms of lateral freedom of motion since they are simply aligned with the main wheel. However, in this case the main wheel never needs to spin and therefore the rolling constraint disappears. This is a degenerate form of the Swedish wheel and therefore we assume in the remainder of this chapter that $\gamma \neq \pi/2$.

3.2.3.5 Spherical wheel

The final wheel type, a ball or spherical wheel, places no direct constraints on motion (fig- ure 3.9). Such a mechanism has no principal axis of rotation, and therefore no appropriate rolling or sliding constraints exist. As with castor wheels and Swedish wheels, the spherical

wheel is clearly omnidirectional and places no constraints on the robot chassis kinematics. Therefore equation (3.21) simply describes the roll rate of the ball in the direction of motion v_A of point A of the robot.

$$\begin{bmatrix} \sin(\alpha + \beta) & -\cos(\alpha + \beta) & -l\cos\beta \end{bmatrix} R(\theta)\dot{\xi}_I - r\dot{\varphi} = 0 \tag{3.21}$$

By definition the wheel rotation orthogonal to this direction is zero.

$$\begin{bmatrix} \cos(\alpha + \beta) & \sin(\alpha + \beta) & l\sin\beta \end{bmatrix} R(\theta)\dot{\xi}_I = 0 \tag{3.22}$$

As can be seen, the equations for the spherical wheel are exactly the same as for the fixed standard wheel. However, the interpretation of equation (3.22) is different. The omnidirec- tional spherical wheel can have any arbitrary direction of movement, where the motion direction given by γ is a free variable deduced from equation (3.22). Consider the case that the robot is in pure translation in the direction of Y_R. Then equation (3.22) reduces to $\sin(\alpha + \beta) = 0$, thus $\beta = -\alpha$, which makes sense for this special case.

3.2.4 Robot kinematic constraints

Given a mobile robot with M wheels we can now compute the kinematic constraints of the robot chassis. The key idea is that each wheel imposes zero or more constraints on robot motion,

and so the process is simply one of appropriately combining all of the kinematic constraints arising from all of the wheels based on the placement of those wheels on the robot chassis.

We have categorized all wheels into five categories: (1) fixed and (2)steerable standard wheels, (3) castor wheels, (4) Swedish wheels, and (5) spherical wheels. But note from the wheel kinematic constraints in equations (3.17), (3.18), and (3.19) that the castor wheel,
Swedish wheel, and spherical wheel impose *no* kinematic constraints on the robot chassis, since $\dot{\beta}_I$ can range freely in all of these cases owing to the internal wheel degrees of freedom.

Therefore only fixed standard wheels and steerable standard wheels have impact on robot chassis kinematics and therefore require consideration when computing the robot's kinematic constraints. Suppose that the robot has a total of N standard wheels, comprising
N_f fixed standard wheels and N_S steerable standard wheels. We use $\beta_S(t)$ to denote the
variable steering angles of the N_S steerable standard wheels. In contrast, β_f refers to the orientation of the N_f fixed standard wheels as depicted in figure 3.4. In the case of wheel spin, both the fixed and steerable wheels have rotational positions around the horizontal
axle that vary as a function of time. We denote the fixed and steerable cases separately as
$\beta_f(t)$ and $\beta_S(t)$, and use $\beta(t)$ as an aggregate matrix that combines both values:

$$\beta(t) = \begin{bmatrix} \beta_f(t) \\ \beta_s(t) \end{bmatrix} \tag{3.23}$$

The rolling constraints of all wheels can now be collected in a single expression:

$$J_1(\beta_s)R(\alpha)\dot{\xi}_I - J_2\dot{\beta} = 0 \tag{3.24}$$

This expression bears a strong resemblance to the rolling constraint of a single wheel, but substitutes matrices in lieu of single values, thus taking into account all wheels. J_2 is a constant diagonal $N \times N$ matrix whose entries are radii r of all standard wheels. $J_1(\beta_s)$ denotes a matrix with projections for all wheels to their motions along their individual wheel planes:

$$J_1(\beta_s) = \begin{bmatrix} J_{1f} \\ J_{1s}(\beta_s) \end{bmatrix} \tag{3.25}$$

Note that $J_1(\beta_s)$ is only a function of β_s and not β_f. This is because the orientations of steerable standard wheels vary as a function of time, whereas the orientations of fixed stan-

dard wheels are constant. J_{1f} is therefore a constant matrix of projections for all fixed stan- dard wheels. It has size ($N_f \times 3$), with each row consisting of the three terms in the three-matrix from equation (3.12) for each fixed standard wheel. $J_{1s}(\beta_s)$ is a matrix of size ($N_s \times 3$), with each row consisting of the three terms in the three-matrix from equation (3.15) for each steerable standard wheel.

In summary, equation (3.24) represents the constraint that all standard wheels must spin around their horizontal axis an appropriate amount based on their motions along the wheel plane so that rolling occurs at the ground contact point.

We use the same technique to collect the sliding constraints of all standard wheels into a single expression with the same structure as equations (3.13) and (3.16):

$$C_1(\beta_s)R(\theta)\dot{\xi}_I = 0 \tag{3.26}$$

$$C_1(\beta_s) = \begin{bmatrix} C_{1f} \\ C_{1s}(\beta_s) \end{bmatrix} \tag{3.27}$$

C_{1f} and C_{1s} are ($N_f \times 3$) and ($N_s \times 3$) matrices whose rows are the three terms in the three-matrix of equations (3.13) and (3.16) for all fixed and steerable standard wheels. Thus

equation (3.26) is a constraint over all standard wheels that their components of motion orthogonal to their wheel planes must be zero. This sliding constraint over all standard wheels has the most significant impact on defining the overall maneuverability of the robot chassis, as explained in the next section.

3.2.5 Examples: robot kinematic models and constraints

In section 3.2.2 we presented a forward kinematic solution for $\dot{\xi}_I$ in the case of a simple differential-drive robot by combining each wheel's contribution to robot motion. We can now use the tools presented above to construct the same kinematic expression by direct
application of the rolling constraints for every wheel type. We proceed with this technique applied again to the differential drive robot, enabling verification of the method as com- pared to the results of section 3.2.2. Then we proceed to the case of the three-wheeled omni- directional robot.

3.2.5.1 A differential-drive robot example

First, refer to equations (3.24) and (3.26). These equations relate robot motion to the rolling and sliding constraints $J_1(\beta_s)$ and $C_1(\beta_s)$, and the wheel spin speed of the robot's wheels, $\dot{\varphi}$. Fusing these two equations yields the following expression:

$$\begin{bmatrix} J_1(\beta_s) & R(\theta) \\ C_1(\beta_s) & \end{bmatrix} \dot{\xi}_I = \begin{bmatrix} J_2 \dot{\varphi} \\ 0 \end{bmatrix} \qquad (3.28)$$

1 *s*

Once again, consider the differential drive robot in figure 3.3. We will construct $J_1(\beta_s)$ and $C_1(\beta_s)$ directly from the rolling

constraints of each wheel. The castor is unpowered and is free to move in any direction, so we ignore this third point of contact altogether. The two remaining drive wheels are not steerable, and therefore $J_1(\beta_s)$ and $C_1(\beta_s)$ simplify to J_{1f} and C_{1f} respectively. To employ the fixed standard wheel's rolling constraint formula, equation (3.12), we must first identify each wheel's values for α and β. Suppose that the robot's local reference frame is aligned such that the robot moves forward along $+X_R$, as shown in figure 3.1. In this case, for the right wheel $\alpha = -\pi/2$, $\beta = \pi$, and for the left wheel, $\alpha = \pi/2$, $\beta = 0$. Note the value of α for the right wheel is necessary to ensure that positive spin causes motion in the $+X_R$ direction (figure 3.4). Now we can compute the J_{1f} and C_{1f} matrix using the matrix terms from equations (3.12) and (3.13). Because the two fixed standard wheels are parallel, equation (3.13) results in only one independent equation, and equation (3.28) gives

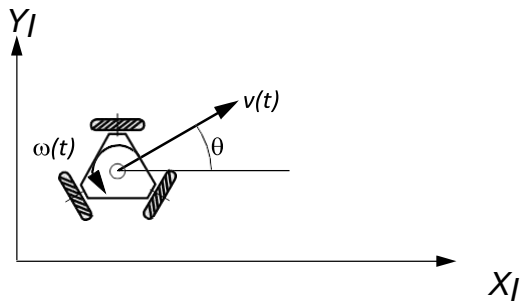




Figure 3.10

A three-wheel omnidrive robot developed by Carnegie Mellon University (www.cs.cmu.edu/~pprk).

$$\begin{bmatrix} 1 & 0 & l \\ 1 & 0 & -l \\ 0 & 1 & 0 \end{bmatrix} R(\theta) \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} J_2 \theta_1 \\ J_2 \theta_2 \\ 0 \end{bmatrix} \tag{3.29}$$

Inverting equation (3.29) yields the kinematic equation specific to our differential drive robot:

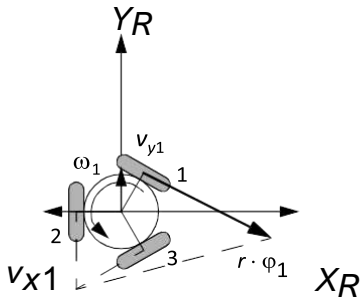
$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix}_I = R(\theta)^{-1} \begin{bmatrix} 1 & 0 & l \\ 1 & 0 & -l \\ 0 & 1 & 0 \end{bmatrix}^{-1} \begin{bmatrix} J_2 \dot{\theta}_1 \\ J_2 \dot{\theta}_2 \\ 0 \end{bmatrix} = R(\theta)^{-1} \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 \\ \frac{1}{2l} & -\frac{1}{2l} & 0 \\ -1 & 1 & J_2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ 0 \end{bmatrix} \tag{3.30}$$

This demonstrates that, for the simple differential-drive case, the combination of wheel rolling and sliding constraints describes the kinematic behavior, based on our manual cal- culation in section 3.2.2.

3.2.5.2 An omnidirectional robot example

Consider the omniwheel robot shown in figure 3.10. This robot has three Swedish 90- degree wheels, arranged radially symmetrically, with the rollers perpendicular to each main wheel.

First we must impose a specific local reference frame upon the robot. We do so by choosing point P at the center of the robot, then aligning the robot with the local reference



ICR

Figure 3.11

The local reference frame plus detailed parameters for wheel 1.

frame such that X_R is colinear with the axis of wheel 2. Figure 3.11 shows the robot and its local reference frame arranged in this manner.

We assume that the distance between each wheel and P is l, and that all three wheels have the same radius, r. Once again, the value of $\dot{\xi}_I$ can be computed as a combination of the rolling constraints of the robot's three omnidirectional wheels, as in equation (3.28). As with the differential- drive robot, since this robot has no steerable wheels, $J_1(\beta_s)$ simplifies to J_{1f}:

$$\dot{\xi}_I = R(\theta)^{-1} J_1^{-1} \begin{bmatrix} \dot{\varphi}_1 \\ \dot{\varphi}_2 \\ \dot{\varphi}_3 \end{bmatrix} \tag{3.31}$$

l $1f$ 2

We calculate J_{1f}

using the matrix elements of the rolling constraints for the Swedish wheel, given by equation (3.19). But to use these values, we must establish the values

□□ □□ □ for each wheel. Referring to figure (3.8), we can see that □= 0 for the Swedish 90- degree wheel. Note that this immediately simplifies equation (3.19) to equation (3.12), the rolling constraints of a fixed standard wheel. Given our particular placement of the local reference frame, the value of □ for each wheel is easily computed:

□□1 = □ □ 3□□ □□2 = □□□ □□3 = –□ □ □ = 0 for all wheels because the 3□ . Furthermore,

wheels are tangent to the robot's circular body. Constructing and simplifying $J_1 f$ using equation (3.12) yields

$$J_{1f} = \begin{bmatrix} \sin\frac{\pi}{3} & -\cos\frac{\pi}{3} & -l \\ 0 & -\cos\theta & -l \\ = & & \end{bmatrix} \begin{bmatrix} \frac{\sqrt{3}}{2} & -\frac{1}{2} & -l \\ 0 & 1 & -l \end{bmatrix} \quad (3.32)$$

$$\sin\frac{\pi}{3} \quad -\cos\frac{\pi}{3} \quad -l \qquad \frac{\sqrt{3}}{2} \quad \frac{1}{2} \quad -l$$

Once again, computing the value of $\dot{\theta}_I$ requires calculating the inverse, J_{1f}^{-1}, as needed in equation (3.31). One approach would be to apply rote methods for calculating the inverse of a 3 x 3 square matrix. A second approach would be to compute the contribution of each Swedish wheel to chassis motion, as shown in section 3.2.2. We leave this process as an exercise for the enthusiast. Once the $\dot{\theta}_I$ can be isolated: inverse is obtained,

$$\dot{\theta}_I = R(\theta)^{-1} \begin{bmatrix} \frac{1}{\sqrt{3}} & 0 & -\frac{1}{\sqrt{3}} \\ -\frac{1}{3} & \frac{2}{3} & -\frac{1}{3} \end{bmatrix} J_2 \quad (3.33)$$

$$-\frac{1}{3l} \quad -\frac{1}{3l} \quad -\frac{1}{3l}$$

Consider a specific omnidrive chassis with $l = 1$ and $r = 1$ for all wheels. The robot's local reference frame and global reference frame are aligned, so that $\theta = 0$. If wheels 1, 2, and 3 spin at speeds $\dot{\phi}_1 = 4$ $\dot{\phi}_2 = 1$ $\dot{\phi}_3 = 2$, what is the resulting motion of the whole robot? Using the equation above, the

answer can be calculated readily:

$$\dot{\xi}_I = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -\frac{1}{\sqrt{3}} & 0 & \frac{1}{\sqrt{3}} \\ -\frac{1}{3} & \frac{2}{3} & -\frac{1}{3} \\ -\frac{1}{3} & -\frac{1}{3} & -\frac{1}{3} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} 4 \\ 1 \\ 2 \end{bmatrix} = \begin{bmatrix} -\frac{2}{\sqrt{3}} \\ -\frac{4}{3} \\ -\frac{7}{3} \end{bmatrix} \tag{3.34}$$

So this robot will move instantaneously along the x -axis with positive speed and along the y axis with negative speed while rotating clockwise. We can see from the above exam- ples that robot motion can be predicted by combining the rolling constraints of individual wheels.

The sliding constraints comprising $C_1(\beta_s)$ can be used to go even further, enabling us to evaluate the maneuverability and workspace of the robot rather than just its predicted motion. Next, we examine methods for using the sliding constraints, sometimes in conjunc-
tion with rolling constraints, to generate powerful analyses of the maneuverability of a robot chassis.

3.3 Mobile Robot Maneuverability

The kinematic mobility of a robot chassis is its ability to directly move in the environment. The basic constraint limiting mobility is the rule that every wheel must satisfy its sliding constraint. Therefore, we can formally derive robot mobility by starting from equation (3.26).

In addition to instantaneous kinematic motion, a mobile robot is able to further manip- ulate its position, over time, by steering steerable wheels. As we will see in section 3.3.3, the overall maneuverability of a robot is thus a combination of the mobility available based on the kinematic sliding constraints of the standard wheels, plus the additional freedom contributed by steering and spinning the steerable standard wheels.

3.3.1 Degree of mobility

Equation (3.26) imposes the constraint that every wheel must avoid any lateral slip. Of course, this holds separately for each and every wheel, and so it is possible to specify this constraint separately for fixed and for steerable standard wheels:

$$C_1 {}^{f}R(\beta)\dot{\xi}_I = 0 \qquad C_1$$

$C_{1s}(\beta_s)R(\beta)\dot{\xi}_I = 0$ (3.35)

For both of these constraints to be satisfied, the motion vector $R(\beta)\dot{\xi}_I$ (3.36)

must belong to the *null space* of the projection matrix $C_1(\beta_s)$, which is simply a combination of C_{1f} and C_{1s}. Mathematically, the null space of $C_1(\beta_s)$ is the space N such that for any vector n in N, $C_1(\beta_s)n = 0$. If the kinematic constraints are to be honored, then the motion of the robot must always be within this space N. The kinematic constraints [equations (3.35) and (3.36)] can also be demonstrated geometrically using the concept of a robot's *instantaneous center of rotation* (*ICR*).

Consider a single standard wheel. It is forced by the sliding constraint to have zero lat- eral motion. This can be shown geometrically by drawing a *zero motion line* through its horizontal axis, perpendicular to the wheel plane (figure 3.12). At any given instant, wheel motion along the zero motion line must be zero. In other words, the wheel must be moving

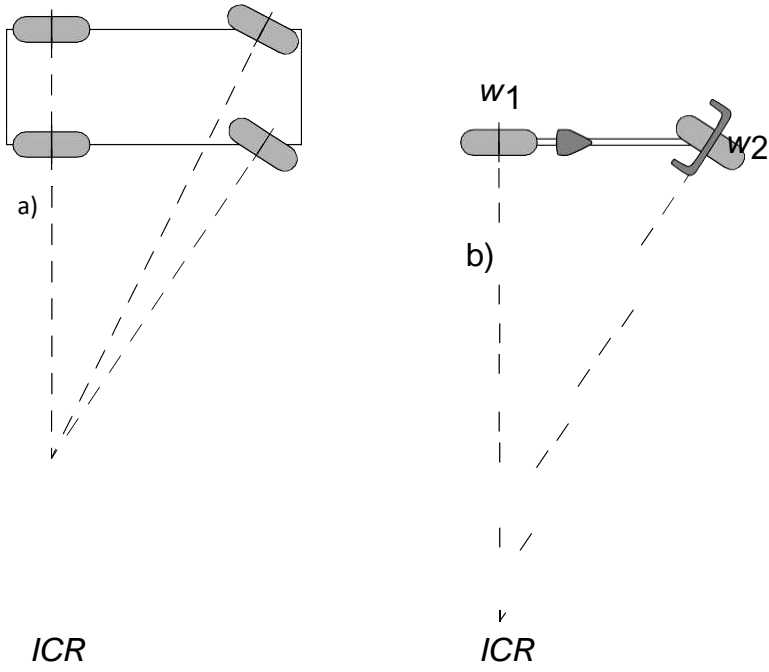


Figure 3.12

(a) Four-wheel with car-like Ackerman steering. (b) bicycle.

instantaneously along some circle of radius R such that the center of that circle is located on the zero motion line. This center point, called the instantaneous center of rotation, may lie anywhere along the zero motion line. When R is at infinity, the wheel moves in a straight line.

A robot such as the Ackerman vehicle in figure 3.12a can have several wheels, but must always have a single *ICR* . Because all of its zero motion lines meet at a single point, there is a single solution for robot motion, placing the *ICR* at this meet point.

This *ICR* geometric construction demonstrates how robot mobility is a function of the number of constraints on the robot's motion, not the number of wheels. In figure 3.12b, the bicycle shown has two wheels, w_1 and w_2 . Each wheel contributes a constraint, or a zero
motion line. Taken together the two constraints result in a single point as the only remaining
solution for the *ICR* . This is because the two constraints are independent, and thus each further constrains overall robot motion.

But in the case of the differential drive robot in figure 3.13a, the two wheels are aligned along the same horizontal axis. Therefore, the *ICR* is constrained to lie along a line, not at a specific point. In fact, the second wheel imposes no additional kinematic constraints on robot motion since its zero motion line is identical to that of the first wheel. Thus, although the bicycle and differential-drive chassis have the same number of nonomnidirectional wheels, the former has two independent kinematic constraints while the latter has only one.

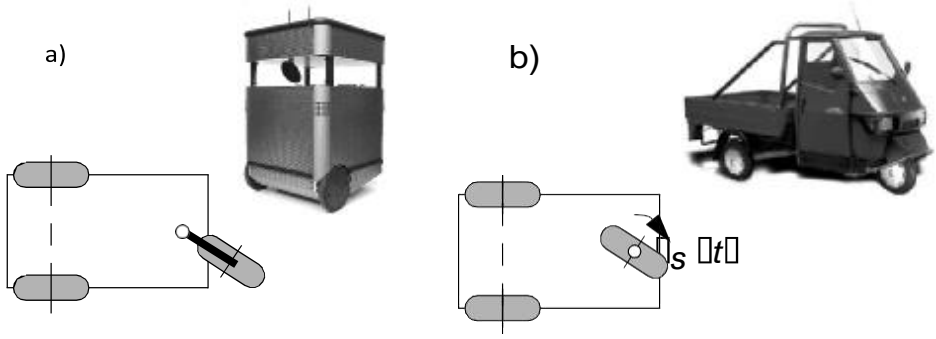


Figure 3.13

(a) Differential drive robot with two individually motorized wheels and a castor wheel, e.g., the Pyg- malion robot at EPFL. (b) Tricycle with two fixed standard wheels and one steered standard wheel,
e.g. Piaggio minitransporter.

The Ackerman vehicle of figure 3.12a demonstrates another way in which a wheel may be unable to contribute an independent constraint to the robot kinematics. This vehicle has two steerable standard wheels. Given the instantaneous position of just one of these steer- able wheels and the position of the fixed rear wheels, there is only a single solution for the *ICR* . The position of the second steerable wheel is absolutely constrained by the *ICR* . Therefore, it offers no independent constraints to robot motion.

Robot chassis kinematics is therefore a function of the set of *independent* constraints arising from all standard wheels. The mathematical interpretation of independence is related to the *rank* of a matrix. Recall that the rank of a matrix is the smallest

number of independent rows or columns. Equation (3.26) represents all sliding constraints imposed by the wheels of the mobile robot. [ ] is the number of independent Therefore *rank* $C_1(\beta_s)$ straints. con-

The greater the number of independent constraints, and therefore the greater the rank of $C_1(\beta_s)$, the more constrained is the mobility of the robot. For example, consider a robot with a single fixed standard wheel. Remember that we consider only standard wheels. This robot may be a unicycle or it may have several Swedish wheels; however, it has exactly one fixed standard wheel. The wheel is at a position relative specified by parameters

to the robot's local reference frame. $C_1(\beta_s)$ is comprised of C_{1f} and C_{1s} . However, since there are no steerable standard wheels C_{1s} is empty and therefore $C_1(\beta_s)$ contains only C_{1f} . Because there is one fixed standard wheel, this matrix has a rank of one and therefore this robot has a single independent constrain on mobility:

$$C_1(\beta_s) = C_{1f} = \begin{bmatrix} \cos\alpha + \alpha \sin\alpha + \\ \alpha l \sin\beta \end{bmatrix} \quad (3.37)$$

Now let us add an additional fixed standard wheel to create a differential-drive robot by constraining the second wheel to be aligned with the same horizontal axis as the original wheel. Without loss of generality, we can place point P at the midpoint between the centers of the two wheels. Given $\dot{\phi}_1 = \omega_1 r_1$ for wheel w_1 and $\dot{\phi}_2 = \omega_2 r_2$ for wheel w_2, it holds geometrically that $\omega_1 l_1 = l_2 \omega_2$ $\omega_1 = \dot{\phi}_2 = 0$ $\omega_1 + \omega = \dot{\phi}_2$. Therefore, in this case, the matrix $C_1(\beta_s)$ has two constraints but a rank of one:

$$C_1(\beta_s) = C_{1f} = \begin{bmatrix} \cos\alpha_1 & \sin\alpha_1 & l_1 \\ 0 & 0 & 0 \end{bmatrix} \qquad (3.38)$$

Alternatively, consider the case when w_2 is placed in the wheel plane of w_1 but with the same orientation, as in a bicycle with the steering locked in the forward position. We again place point P between the two wheel centers, and orient the wheels such that they lie on axis x_1. This geometry implies that $\omega_1 l_1 = l_2 \omega_2$ $\omega_1 = \dot{\phi}_2 = \omega_2 = 0$ $\omega_1 = 0$ $\omega_2 = \omega$ and, therefore, the matrix $C_1(\beta_s)$ retains two independent constraints and has a rank of two:

$$C_1(\beta_s) = C_{1f} = \begin{bmatrix} \cos\alpha_2 & \sin\alpha_2 & l_1 \\ \sin\alpha_2 & -\cos\alpha_2 & 0 \\ \sin\alpha_2 & -l_1 \cos\alpha_2 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ l_1 & \\ 0 & -1 \\ l_1 & \end{bmatrix} \qquad (3.39)$$

In general, if $rank\,C_{1f} < 1$ then the vehicle can, at best, only travel along a circle or along a straight line. This configuration means that the robot has two or more independent constraints due to fixed standard wheels that do not share the same horizontal axis of rota- tion. Because such configurations have only a degenerate form of mobility in the plane, we do not consider them in the remainder of this chapter. Note, however, that some degenerate configurations such as the four-wheeled slip/skid steering system are useful in certain envi- ronments, such as on loose soil and sand, even though they fail to satisfy sliding constraints. Not surprisingly, the price that must be paid for such violations of the sliding constraints is that dead reckoning based on odometry becomes less accurate and power efficiency is reduced dramatically.

In general, a robot will have zero or more fixed standard wheels and zero or more steer- able standard wheels. We can therefore identify the possible range of rank values for any robot: $0 \le rank\,C_1(\beta_s) \le 3$. Consider the case $rank\,C_1(\beta_s) = 0$. This is only possible if there are zero independent kinematic constraints in $C_1(\beta_s)$. In this case there are neither fixed nor steerable standard wheels attached to the robot frame: $N_f = N_s = 0$.

Consider the other extreme, $rank\,C_1(\beta_s) = 3$. This is the maximum possible rank since the kinematic constraints are specified along three degrees of freedom (i.e., the con- straint matrix is three columns wide). Therefore, there cannot be more than three indepen-

dent constraints. In fact, when $\left[\right]$ = 3 , then the robot is completely constrained
rank $C_1(\alpha_s)$

in all directions and is, therefore, degenerate since motion in the plane is totally impossible.

Now we are ready to formally define a robot's *degree of mobility* α_m :

$$\alpha_m = dim\,N\left[\right] = 3 - rank\left[\right] \qquad (3.40)$$
$$C_1(\alpha_s) \qquad C_1(\alpha_s)$$

$$\left[\quad \right]$$

The dimensionality of the null space (*dimN*) of matrix $C_1(\alpha_s)$ is a measure of the

number of degrees of freedom of the robot chassis that can be immediately manipulated
through changes in wheel velocity. It is logical must range between 0 therefore that α_m
and 3.

Consider an ordinary differential-drive chassis. On such a robot there are two fixed stan- dard wheels sharing a common horizontal axis. As discussed above, the second wheel adds no independent kinematic constraints to the system. Therefore, $\left[\right]$ *rank* $C_1(\alpha_s)$ = 1 and α_m = 2 . This fits with intuition: a differential drive robot can control both the rate of its change in orientation and its forward/reverse speed, *simply by manipulating wheel velocities*. In other words, its *ICR* is constrained to lie on the infinite line extending from its
wheels' horizontal axles.

In contrast, consider a bicycle chassis. This configuration consists of one fixed standard wheel and one steerable standard wheel. In this case, each wheel contributes an indepen- dent

sliding constraint to $C_1(\beta_s)$. Therefore, $\delta_m = 1$. Note that the bicycle has the same total number of nonomidirectional wheels as the differential-drive chassis, and indeed one
of its wheels is steerable. Yet it has one less degree of mobility. Upon reflection this is appropriate. A bicycle only has control over its forward/reverse speed by direct manipula- tion of wheel velocities. Only by steering can the bicycle change its *ICR* .

As expected, based on equation (3.40) any robot consisting only of omnidirectional wheels such as Swedish or spherical wheels will have the maximum mobility, $\delta_m = 3$. Such a robot can directly manipulate all three degrees of freedom.

3.3.2 Degree of steerability

The degree of mobility defined above quantifies the degrees of controllable freedom based on changes to wheel velocity. Steering can also have an eventual impact on a robot chassis pose ξ , although the impact is indirect because after changing the angle of a steerable stan- dard wheel, the robot must move for the change in steering angle to have impact on pose.

As with mobility, we care about the number of independently controllable steering parameters when defining the *degree of steerability* δ_s :

$$\delta_s = rank\begin{bmatrix} \\ C_{1s}(\beta_s) \end{bmatrix} \tag{3.41}$$

Recall that in the case of mobility, an increase in the rank of $C_1(\beta_s)$ implied more kine- matic constraints and thus a less mobile system. In the case of steerability, an increase in the rank of $C_{1s}(\beta_s)$ implies more degrees of steering freedom and thus greater eventual maneuverability. Since $C_1(\beta_s)$ includes $C_{1s}(\beta_s)$, this means that a steered standard wheel can both decrease mobility and increase steerability: its particular orientation at any instant imposes a kinematic constraint, but its ability to change that orientation can lead to addi- tional trajectories.

The range of β_s can be specified: $0 \le \beta_s \le 2$. The case $\beta_s = 0$ implies that the robot has no steerable standard wheels, $N_s = 0$. The case $\beta_s = 1$ is most common when a robot configuration includes one or more steerable standard wheels.

For example, consider an ordinary automobile. In this case $N_f = 2$ and $N_s = 2$. But the fixed wheels share a common axle and so $rank\ C_{1f} = 1$. The fixed wheels and any one of the steerable wheels constrain the *ICR* to be a point along the line extending from the rear axle. Therefore, the second steerable wheel cannot impose any independent kine- matic constraint and so $rank\ C_{1s}(\beta_s) = 1$. In this case $\beta_m = 1$ and $\beta_s = 1$.

The case $\beta_s = 2$ is only possible in robots with no fixed standard wheels: $N_f = 0$. Under these circumstances, it is possible to create a chassis with two separate steerable standard wheels, like a pseudobicycle (or the two-steer) in which both wheels are steerable. Then, orienting one wheel constrains the *ICR* to a line while the second wheel can con- strain the *ICR* to any point along that line. Interestingly, this means that the $\beta_s = 2$

implies that the robot can place its *ICR* anywhere on the ground plane.

3.3.3 Robot maneuverability

The overall degrees of freedom that a robot can manipulate, called the *degree of maneuver- ability* δ_M , can be readily defined in terms of mobility and steerability:

$$\delta_M = \delta_m + \delta_s \tag{3.42}$$

Therefore maneuverability includes both the degrees of freedom that the robot manipu- lates directly through wheel velocity and the degrees of freedom that it indirectly manipu- lates by changing the steering configuration and moving. Based on the investigations of the previous sections, one can draw the basic types of wheel configurations. They are depicted in figure 3.14

Note that two robots with the same δ_M are not necessarily equivalent. For example, dif-
ferential drive and tricycle geometries (figure 3.13) have equal maneuverability $\delta_M = 2$. In differential drive all maneuverability is the result of direct mobility because $\delta_m = 2$ and $\delta_s = 0$. In the case of a tricycle the maneuverability results from steering also: $\delta_m = 1$ and $\delta_S = 1$. Neither of these configurations allows the *ICR* to range anywhere on the plane. In both cases, the *ICR* must lie on a predefined line with respect to the robot refer-

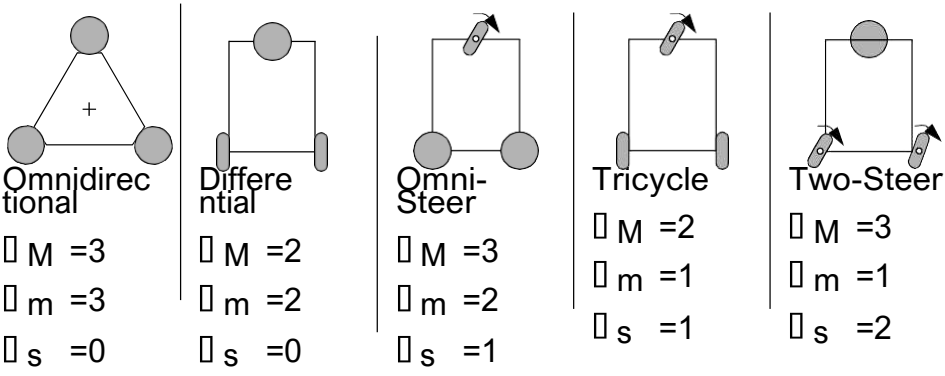


Figure 3.14

The five basic types of three-wheel configurations. The spherical wheels can be replaced by castor or Swedish wheels without influencing maneuverability. More configurations with various numbers of wheels are found in chapter 2.

ence frame. In the case of differential drive, this line extends from the common axle of the two fixed standard wheels, with the differential wheel velocities setting the *ICR* point on this line. In a tricycle, this line extends from the shared common axle of the fixed wheels, with the steerable wheel setting the *ICR* point along this line.

More generally, for any robot with $\beta_M = 2$ the *ICR* is always constrained to lie on a line and for any robot with $\beta_M = 3$ the *ICR* can be set to any point on the plane.

One final example will demonstrate the use of the tools we have developed above. One common robot configuration for indoor mobile robotics research is the *synchro drive* con- figuration

(figure 2.22). Such a robot has two motors and three wheels that are locked together. One motor provides power for spinning all three wheels while the second motor provides power for steering all three wheels. In a three-wheeled synchro drive robot
$N_f = 0$ and $N_s = 3$. Therefore $rank\ C_{1s}(\beta_s)$ can be used to determine both β_m and

β_s. The three wheels do not share a common axle, therefore two of the three contribute independent sliding constraints. The third must be dependent on these two constraints for
motion to be possible. Therefore $rank\ C_{1s}(\beta_s) = 2$ and $\beta_m = 1$. This is intuitively cor-
rect. A synchro drive robot with the steering frozen manipulates only one degree of free- dom, consisting of traveling back and forth on a straight line.

However an interesting complication occurs when considering β_s. Based on equation (3.41) the robot should have $\beta_s = 2$. Indeed, for a three-wheel-steering robot with the geo- metric configuration of a synchro drive robot this would be correct. However, we have additional information: in a synchro drive configuration a single motor steers all three
wheels using a belt drive. Therefore, although ideally, if the wheels were independently steerable, then the system would achieve $\beta_s = 2$, in the case of synchro drive the drive

system further constrains the kinematics such that in reality $\delta_S = 1$. Finally, we can com- pute maneuverability based on these values: $\delta_M = 2$ for a synchro drive robot.

This result implies that a synchro drive robot can only manipulate, in total, two degrees of freedom. In fact, if the reader reflects on the wheel configuration of a synchro drive robot it will become apparent that there is no way for the chassis orientation to change. Only the $x - y$ position of the chassis can be manipulated and so, indeed, a synchro drive robot has only two degrees of freedom, in agreement with our mathematical conclusion.

3.4 Mobile Robot Workspace

For a robot, maneuverability is equivalent to its control degrees of freedom. But the robot is situated in some environment, and the next question is to situate our analysis in the envi- ronment. We care about the ways in which the robot can use its control degrees of freedom to position itself in the environment. For instance, consider the Ackerman vehicle, or auto- mobile. The total number of control degrees of freedom for such a vehicle is $\delta_M = 2$, one for steering and the second for actuation of the drive wheels. But what is the total degrees of freedom of the vehicle in its environment? In fact it is three: the car can position itself on the plane at any x, y point and with any angle θ .

Thus identifying a robot's space of possible configurations is important because surpris- ingly it can exceed δ_M . In addition to *workspace*, we care about how the robot is able to move between various configurations: what are the types of paths that it can follow and, furthermore, what are its possible trajectories through this configuration space? In the

remainder of this discussion, we move away from inner kinematic details such as wheels and focus instead on the robot chassis pose and the chassis degrees of freedom. With this in mind, let us place the robot in the context of its workspace now.

3.4.1 Degrees of freedom

In defining the workspace of a robot, it is useful to first examine its *admissible velocity space*. Given the kinematic constraints of the robot, its velocity space describes the inde- pendent components of robot motion that the robot can control. For example, the velocity space of a unicycle can be represented with two axes, one representing the instantaneous forward speed of the unicycle and the second representing the instantaneous change in ori- entation, θ, of the unicycle.

The number of dimensions in the velocity space of a robot is the number of indepen- dently achievable velocities. This is also called the *differentiable degrees of freedom* (*DDOF*). A robot's *DDOF* is *always* equal to its degree of mobility δ_m. For example, a bicycle has the following degree of maneuverability: $\delta_M = \delta_m + \delta_s = 1 + 1 = 2$. The *DDOF* of a bicycle is indeed 1.

In contrast to a bicycle, consider an *omnibot*, a robot with three Swedish wheels. We know that in this case there are zero standard wheels and therefore $\delta_M = \delta_m + \delta_s = 3 + 0 = 3$. So, the omnibot has three differential degrees of freedom.

This is appropriate, given that because such a robot has no kinematic motion constraints, it is able to independently set all three pose variables: x, y, θ.

Given the difference in *DDOF* between a bicycle and an omnibot, consider the overall degrees of freedom in the workspace of each configuration. The omnibot can achieve any pose (x, y, θ) in its environment and can do so by directly achieving the goal positions of all three axes simultaneously because $DDOF = 3$. Clearly, it has a workspace with $DOF = 3$.

Can a bicycle achieve any pose (x, y, θ) in its environment? It can do so, but achieving some goal points may require more time and energy than an equivalent omnibot. For exam- ple, if a bicycle configuration must move laterally 1 m, the simplest successful maneuver would involve either a spiral or a back-and-forth motion similar to *parallel parking* of auto- mobiles. Nevertheless, a bicycle can achieve any (x, y, θ) and therefore the workspace of a bicycle has $DOF = 3$ as well.

Clearly, there is an inequality relation at work: $DDOF \leq \delta_M \leq DOF$. Although the dimensionality of a robot's workspace is an important attribute, it is clear from the example above that the particular paths available to a robot matter as well. Just as workspace *DOF* governs the robot's ability to achieve various poses, so the robot's *DDOF* governs its abil- ity to achieve various paths.

3.4.2 Holonomic robots

In the robotics community, when describing the path space of a mobile robot, often the con- cept of holonomy is used. The term *holonomic* has broad applicability to several mathemat- ical areas, including differential equations, functions and constraint expressions. In mobile robotics, the term refers specifically to the kinematic constraints of the robot chassis. A *holonomic robot* is a robot that has zero nonholonomic kinematic constraints. Conversely, a *nonholonomic robot* is a robot with one or more nonholonomic kinematic constraints.

A *holonomic kinematic constraint* can be expressed as an explicit function of position variables only. For example, in the case of a mobile robot with a single fixed standard wheel, a holonomic kinematic constraint would be expressible using x, y, θ only. Such a constraint may not use derivatives of these values, such as $\dot{x}$, $\dot{y}$, or $\dot{\theta}$. A *nonholonomic kinematic constraint* requires a differential relationship, such as the deriva- tive of a position variable. Furthermore, it cannot be integrated to provide a constraint in terms of the position variables only. Because of this latter point of view, nonholonomic sys- tems are often called *nonintegrable* systems.

Consider the fixed standard wheel sliding constraint:

$$\begin{bmatrix} \cos(\alpha+\beta) + \gamma\sin\beta & \sin(\alpha+\beta) - \gamma\cos\beta & l\sin\beta \end{bmatrix} R(\theta)\dot{\xi}_I = 0 \quad (3.43)$$

This constraint must use robot motion $\dot{\xi}_I$ rather than pose ξ because the point is to constrain robot motion perpendicular to the wheel plane to be zero. The constraint is noninte- grable, depending explicitly on robot motion. Therefore, the sliding constraint is a nonholonomic constraint. Consider a bicycle configuration, with one fixed standard wheel and one steerable standard wheel. Because the fixed wheel sliding constraint will be in force for such a robot, we can conclude that the bicycle is a nonholonomic robot.

But suppose that one locks the bicycle steering system, so that it becomes two fixed stan- dard wheels with separate but parallel axes. We know that $\delta_M = 1$ for such a configura- tion. Is it nonholonomic? Although it may not appear so because of the sliding and rolling constraints, the locked bicycle is actually holonomic. Consider the workspace of this locked bicycle. It consists of a single infinite line along which the bicycle can move (assum- ing the steering was frozen straight ahead). For formulaic simplicity, assume that this infinite line is aligned with X_I in the global reference frame and that $\beta_1 - \beta_2 = \pi$ $\beta_2 - \beta_1 = 0$ $\beta_2 = \pi$. In this case the sliding constraints of both wheels can be replaced with an equally complete set of constraints on the robot pose: $\Delta y = 0$ $\Delta\theta = 0$. This eliminates two nonholonomic constraints, corresponding to

the sliding constraints of the two wheels.

The only remaining nonholonomic kinematic constraints are the rolling constraints for each wheel:

$$\begin{bmatrix} -\sin(\alpha+\beta) & \cos(\alpha+\beta) & l\cos\beta \\ R(\beta) & I & r \end{bmatrix} = 0 \tag{3.44}$$

This constraint is required for each wheel to relate the speed of wheel spin to the speed of motion projected along the wheel plane. But in the case of our locked bicycle, given the initial rotational position of a wheel at the origin, ϕ_o, we can replace this constraint with one that directly relates position on the line, x, with wheel rotation angle, ϕ: $\phi = \phi x / r + \phi_o$.

The locked bicycle is an example of the first type of holonomic robot – where constraints do exist but are all holonomic kinematic constraints. This is the case for all holonomic robots with $\delta_M \geq 3$. The second type of holonomic robot exists when there are no kinematic constraints, that is, $N_f = 0$ and $N_s = 0$. Since there are no kinematic constraints, there are also no nonholonomic kinematic constraints and so such a robot is always holonomic. This is the case for all holonomic robots with $\delta_M = 3$.

An alternative way to describe a holonomic robot is based on the relationship between the differential degrees of freedom of a robot and the degrees of freedom of its workspace: *a robot is holonomic if and only if* $DDOF = DOF$. Intuitively, this is because it is only through nonholonomic constraints (imposed by steerable or fixed standard wheels) that a robot can achieve a workspace with degrees of freedom exceeding its differential degrees of freedom, $DOF > DDOF$. Examples include differential drive and bicycle/tricycle con- figurations.

In mobile robotics, useful chassis generally must achieve poses in a workspace with dimensionality 3, so in general we require $DOF = 3$ for the chassis. But the "holonomic" abilities to maneuver around obstacles without affecting orientation and to track at a target while following an arbitrary path are important additional considerations. For these rea- sons, the particular form of holonomy most relevant to mobile robotics is that of $DDOF = DOF = 3$. We define this class of robot configurations as omnidirectional: an *omnidirectional robot* is a holonomic robot with $DDOF = 3$.

3.4.3 Path and trajectory considerations

In mobile robotics, we care not only about the robot's ability to reach the required final con- figurations but also about *how* it gets there. Consider the issue of a robot's ability to follow paths: in the best case, a robot should be able to trace any path through its workspace of poses. Clearly, any omnidirectional robot can do this because it is holonomic in a three- dimensional workspace. Unfortunately, omnidirectional robots must use unconstrained wheels, limiting the choice of wheels to Swedish wheels, castor wheels, and spherical wheels. These wheels have not yet been

incorporated into designs allowing far larger amounts of ground clearance and suspensions. Although powerful from a path space point of view, they are thus much less common than fixed and steerable standard wheels, mainly because their design and fabrication are somewhat complex and expensive.

Additionally, nonholonomic constraints might drastically improve stability of move- ments. Consider an omnidirectional vehicle driving at high speed on a curve with constant diameter. During such a movement the vehicle will be exposed to a non- negligible centrip- etal force. This lateral force pushing the vehicle out of the curve has to be counteracted by the motor torque of the omnidirectional wheels. In case of motor or control failure, the vehi- cle will be thrown out of the curve. However, for a car-like robot with kinematic con- straints, the lateral forces are passively counteracted through the sliding constraints, mitigating the demands on motor torque.

But recall an earlier example of high maneuverability using standard wheels: the bicycle on which both wheels are steerable, often called the *two-steer*. This vehicle achieves a degree of steerability of 2, resulting in a high degree of maneuverability:
$\delta_M = \delta_m + \delta_s = 1 + 2 = 3$. Interestingly, this configuration is not holonomic, yet has a
high degree of maneuverability in a workspace with $DOF = 3$.

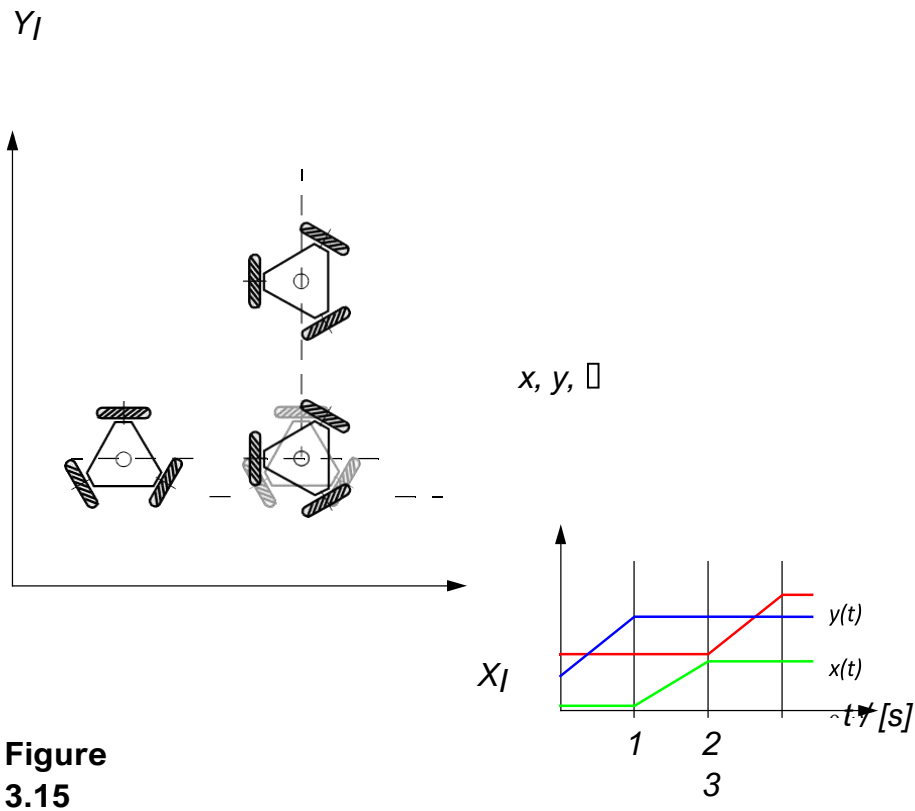


Figure 3.15

Example of robot trajectory with omnidirectional robot: move for 1 second with constant speed of 1 m/s along axis X_I ; change orientation counterclockwise 90 degree, in 1 second; move for 1 second with constant speed of 1 m/s along axis Y_I .

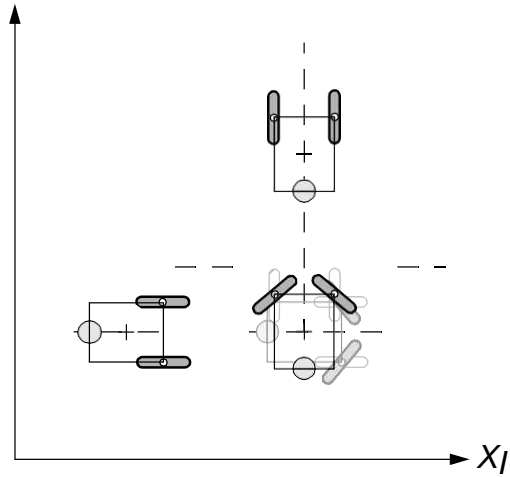
The maneuverability result, $\delta_M = 3$, means that the two-steer can select any *ICR* by appropriately steering its two wheels. So, how does this compare to an omnidirectional robot? The ability to manipulate its *ICR* in the plane means that the two-steer can

follow *any* path in its workspace. More generally, any robot with $\delta_M = 3$ can follow any path in its workspace from its initial pose to its final pose. An omnidirectional robot can also follow any path in its workspace and, not surprisingly, since $\delta_m = 3$ in an omnidirectional robot, then it must follow that $\delta_M = 3$.

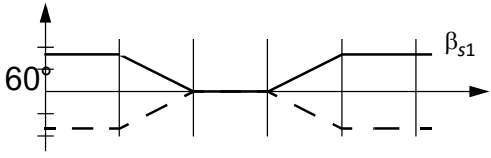
But there is still a difference between a degree of freedom granted by steering versus by direct control of wheel velocity. This difference is clear in the context of *trajectories* rather than paths. A trajectory is like a path, except that it occupies an additional dimension: time. Therefore, for an omnidirectional robot on the ground plane a path generally denotes a trace through a 3D space of pose; for the same robot a trajectory denotes a trace through the 4D space of pose plus time.

For example, consider a goal trajectory in which the robot moves along axis X_I at a con- stant speed of 1 m/s for 1 second, then changes orientation counterclockwise 90 degrees also in 1 second, then moves parallel to axis Y_I for 1 final second. The desired 3-second trajectory is shown in figure 3.15, using plots of x, y and θ in relation to time.

Y_I



β_{s1}, β_{s2}



-60°

x, *y*, β

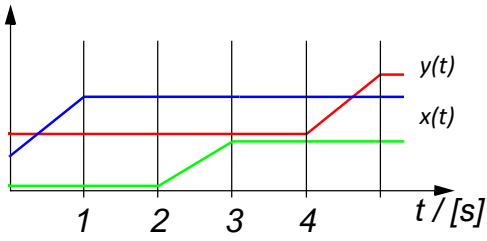


Figure 3.16

Example of robot trajectory similar to figure 3.15 with two steered wheels: move for 1 second with

constant speed of 1 m/s along axis X_I ; rotate steered wheels -50 / 50 degree respectively; change ori- entation counterclockwise 90 degree in 1 second; rotate steered wheels 50 / -50 degree respectively;

move for 1 second with constant speed of 1 m/s along axis Y_I .

Can the omnidirectional robot accomplish this trajectory? We assume that the robot can achieve some arbitrary, finite velocity at each wheel. For simplicity, we further assume that acceleration is infinite; that is, it takes zero time to reach any desired velocity. Under these assumptions, the omnidirectional robot can indeed follow the trajectory of figure 3.15. The transition between the motion of second 1 and second 2, for example, involves only changes to the wheel velocities.

Because the two-steer has $\delta_M = 3$, it must be able to follow the path that would result

from projecting this trajectory into timeless workspace. However, it cannot follow this 4D trajectory. Even if steering velocity is finite and arbitrary, although the two-steer would be able to change steering speed instantly, it would have to wait for the angle of the steerable wheels to change to the desired position before initiating a change in the robot chassis ori- entation. In short, the two-steer requires changes to internal degrees of freedom and because these changes take time, arbitrary trajectories are not attainable. Figure 3.16 depicts the most similar trajectory that a two-steer can achieve. In contrast to the desired three phases of motion, this trajectory has five phases.

3.5 Beyond Basic Kinematics

The above discussion of mobile robot kinematics is only an introduction to a far richer topic. When speed and force are also considered, as is particularly necessary in the case of high-speed mobile robots, dynamic constraints must be expressed in addition to kinematic constraints. Furthermore, many mobile robots such as tank-type chassis and four-wheel slip/skid systems violate the kinematic models above. When analyzing

such systems, it is often necessary to explicitly model the dynamics of viscous friction between the robot and the ground plane.

More significantly, the kinematic analysis of a mobile robot system provides results concerning the theoretical workspace of that mobile robot. However to effectively move in this workspace a mobile robot must have appropriate actuation of its degrees of freedom. This problem, called motorization, requires further analysis of the forces that must be actively supplied to realize the kinematic range of motion available to the robot.

In addition to motorization, there is the question of controllability: under what condi- tions can a mobile robot travel from the initial pose to the goal pose in bounded time? Answering this question requires knowledge – both knowledge of the robot kinematics and knowledge of the control systems that can be used to actuate the mobile robot. Mobile robot control is therefore a return to the practical question of designing a real-world control algo- rithm that can drive the robot from pose to pose using the trajectories demanded for the application.

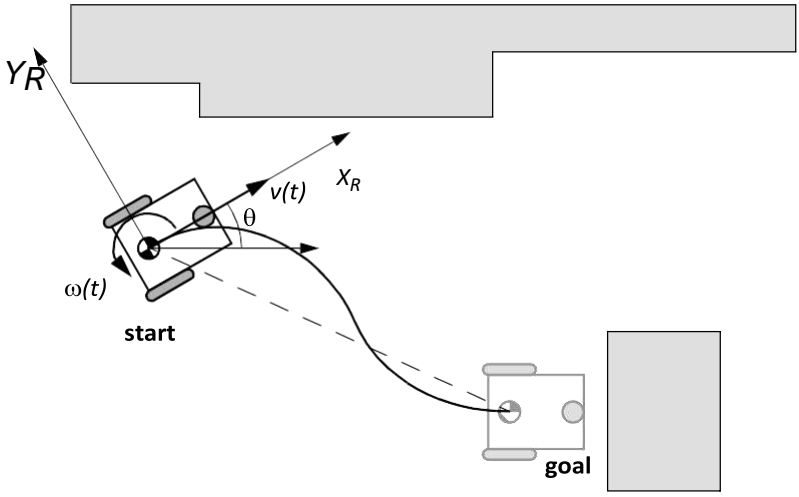


Figure 3.17

Typical situation for feedback control of a mobile robot

3.6 Motion Control (Kinematic Control)

As seen above, motion control might not be an easy task for nonholonomic systems. How- ever, it has been studied by various research groups, for example, [8, 39, 52, 53, 137] and some adequate solutions for motion control of a mobile robot system are available.

3.6.1 Open loop control (trajectory-following)

The objective of a kinematic controller is to follow a trajectory

described by its position or velocity profile as a function of time. This is often done by dividing the trajectory (path) in motion segments of clearly defined shape, for example, straight *lines* and segments of a *cir- cle*. The control problem is thus to precompute a smooth trajectory based on line and circle segments which drives the robot from the initial position to the final position (figure 3.18). This approach can be regarded as open-loop motion control, because the measured robot position is not fed back for velocity or position control. It has several disadvantages:

- It is not at all an easy task to precompute a feasible trajectory if all limitations and con- straints of the robot's velocities and accelerations have to be considered.
- The robot will not automatically adapt or correct the trajectory if dynamic changes of the environment occur.

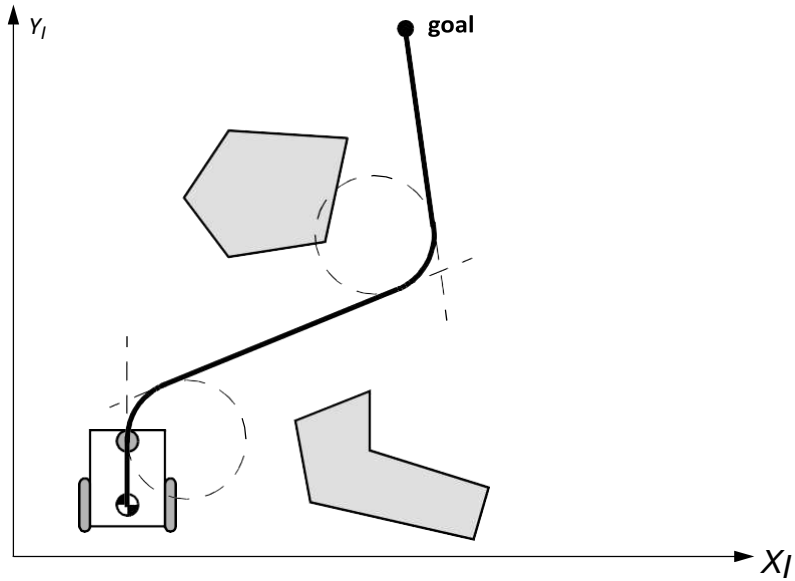


Figure 3.18

Open-loop control of a mobile robot based on straight lines and circular trajectory segments.

- The resulting trajectories are usually not smooth, because the transitions from one tra- jectory segment to another are, for most of the commonly used segments (e.g., lines and part of circles), not smooth. This means there is a discontinuity in the robot's accelera- tion.

3.6.2 Feedback control

A more appropriate approach in motion control of a mobile robot

is to use a real-state feed- back controller. With such a controller the robot's path-planning task is reduced to setting intermediate positions (subgoals) lying on the requested path. One useful solution for a sta- bilizing feedback control of differential-drive mobile robots is presented in section 3.6.2.1. It is very similar to the controllers presented in [39, 100]. Others can be found in [8, 52, 53, 137].

3.6.2.1 Problem statement

Consider the situation shown in figure 3.17, with an arbitrary position and orientation of
the robot and a predefined goal position and orientation. The actual pose error vector given in the robot reference frame X Y is $e = {}^{R}[x\ y\ \theta]^{T}$ with x y , and θ being the goal coordinates of the robot.

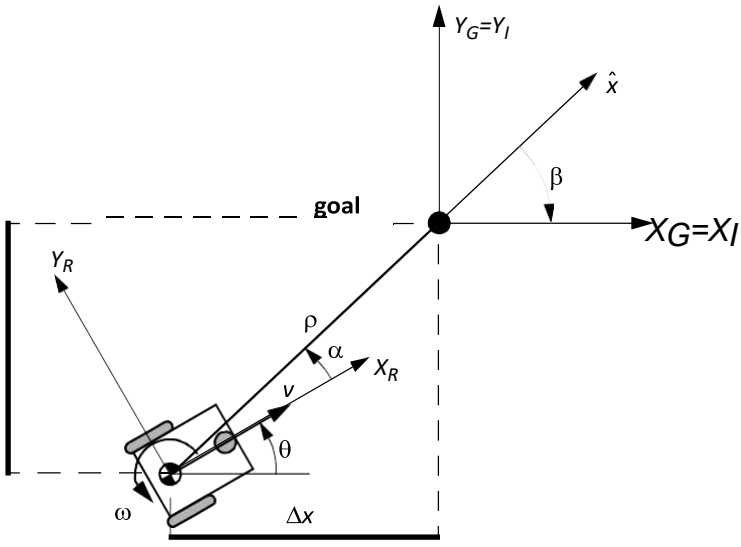


Figure 3.19

Robot kinematics and its frames of interests.

The task of the controller layout is to find a control matrix K, if it exists

$$K = \begin{bmatrix} k_{11} & k_{12} \\ k_{13} & \end{bmatrix} \quad \text{with} \quad k = e(k, t) \tag{3.45}$$

$$\begin{matrix} k_{21} & k \\ 2 & \\ 2 & \end{matrix} \quad \begin{matrix} k \\ 23 \end{matrix} \qquad ij$$

such that the control of $v(t)$ and $\omega(t)$

$$\begin{bmatrix} v(t) \\ \omega(t) \end{bmatrix} = K \cdot e = K \begin{bmatrix} x \\ y \end{bmatrix}_R \tag{3.46}$$

drives the error e toward zero.[2]

$$\lim_{t \to \infty} e(t) = 0 \tag{3.47}$$

---

2. Remember that $v(t)$ is always heading in the X_R direction of the robot's reference frame due to the nonholonomic constraint.

3.6.2.2 Kinematic model

We assume, without loss of generality, that the goal is at the origin of the inertial frame (fig- ure 3.19). In the following the position vector $[x\ y\ \theta]^T$ is always represented in the inertial frame.

The kinematics of a differential-drive mobile robot described in the inertial frame $\{X_I, Y_I, \theta\}$ is given by

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{3.48}$$

where $\dot{x}$ and $\dot{y}$ are the linear velocities in the direction of the X_I and Y_I of the inertial frame.

Let θ denote the angle between the x_R axis of the robot's reference frame and the vector $\hat{x}$ connecting the center of the axle of the wheels with the final position. If $\theta \in I_1$, where

$$I_1 = \left[-\frac{\pi}{2}, \frac{\pi}{2} \right] \tag{3.49}$$

then consider the coordinate transformation into polar coordinates with its origin at the goal position.

$$\rho = \sqrt{\Delta x^2 + \Delta y^2}$$

$$\alpha = -\theta + \operatorname{atan2}(\Delta y, \Delta x)$$

(3.50)

$$\square = - \square - \square$$

(3.51)

(3.52)

This yields a system description, in the new polar coordinates, using a matrix equation

$$\begin{bmatrix} \square \\ \square \\ \square \end{bmatrix} = \begin{bmatrix} -\cos\square & \\ 0 & \\ \frac{\sin\square}{\square} & -1 \\ -\frac{\sin\square}{\square} & 0 \end{bmatrix} \begin{bmatrix} \square \\ \square \end{bmatrix} \quad (3.53)$$

where $\square$ is the distance between the center of the robot's wheel axle and the goal position,

 denotes the angle between the X_R axis of the robot reference frame, and the X_I axis asso- ciated with the final position v and are the tangent and the angular velocity respectively.

On the other hand, if I_2 , where

$$I_2 = \begin{matrix} -\cos\beta & -\sin\beta \\ \frac{\sin\beta}{\rho} & -\frac{\cos\beta}{\rho} \end{matrix} \quad (3.54)$$

redefining the forward direction of the robot by setting $v = -v$, we obtain a system described by a matrix equation of the form

$$\begin{bmatrix} \dot{\rho} \\ \dot{\beta} \\ \dot{\alpha} \end{bmatrix} = \begin{bmatrix} \cos\beta & 0 \\ -\frac{\sin\beta}{\rho} & 1 \\ \frac{\sin\beta}{\rho} & 0 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \quad (3.55)$$

3.6.2.3 Remarks on the kinematic model in polar coordinates [eq. (3.53) and (3.55)]

- The coordinate transformation is not defined at $x = y = 0$; as in such a point the deter- minant of the Jacobian matrix of the transformation is not defined, that is unbounded.
- For I_1 the forward direction of the robot points toward the goal, for I_2 it is the reverse direction.
- By properly defining the forward direction of the robot at its initial configuration, it is always possible to have I_1 at $t = 0$. However, this does not mean that remains in I_I for all time t

. Hence, to avoid that the robot changes direction during approaching the goal, it is necessary to determine, if possible, the controller in such a way that $\alpha \leq l_1$ for all t, whenever $\alpha(0) \leq l_1$. The same applies for the reverse direction (see stability issues below).

3.6.2.4 The control law

The control signals v and ω must now be designed to drive the robot from its actual con- figuration, say (x_0, y_0, θ_0), to the goal position. It is obvious that equation (3.53) presents a discontinuity at $\rho = 0$; thus the theorem of Brockett does not obstruct smooth stabiliz-
ability.

If we consider now the linear control law

$$v = k_\rho \rho \tag{3.56}$$

$$\omega = k_\alpha \alpha + k_\beta \beta \tag{3.57}$$

we get with equation (3.53) a closed-loop system described by

$$\begin{bmatrix} \dot{\rho} \\ \dot{\alpha} \\ \dot{\beta} \end{bmatrix} = \begin{bmatrix} -k_\rho \cos\alpha \\ k_\rho \sin\alpha - k_\alpha \alpha \\ -k_\rho \sin\alpha \end{bmatrix} \quad (3.58)$$

The system does not have any singularity at $\rho = 0$ and has a unique equilibrium point at $(\rho, \alpha, \beta) = (0, 0, 0)$. Thus it will drive the robot to this point, which is the goal posi- tion.

- In the Cartesian coordinate system the control law [equation (3.57)] leads to equations which are not defined at $x = y = 0$.
- Be aware of the fact that the angles α and β have always to be expressed in the range $(-\pi, \pi]$.
- Observe that the control signal v has always a constant sign, that is, it is positive when- ever $(\alpha, \beta) \in I_1$ and it is always negative otherwise. This implies that the robot performs its parking maneuver always in a single direction and without reversing its motion.

In figure 3.20 you find the resulting paths when the robot is initially on a circle in the xy plane. All movements have smooth trajectories toward the goal in the center. The con- trol parameters for this simulation were set to

$$k = (k_\rho, k_\alpha, k_\beta) = (3, 8, -1.5). \quad (3.59)$$

3.6.2.5 Local stability issue

It can further be shown, that the closed-loop control system [equation (3.58)] is locally exponentially stable if

$$k_{\square} \square 0 ; \quad k_{\square} \square \quad k_{\square} - k_{\square} \qquad (3.60)$$
$$0 ; \qquad \square 0$$

Proof:

Linearized around the equilibrium (cos x = 1 , sin x = x) position, equation (3.58) can be written as

$$\begin{bmatrix} \square \\ \square \\ \square \end{bmatrix} = \begin{bmatrix} -k_{\square} & 0 & 0 \\ 0 & -\square k_{\square} - \\ k_{\square} \square & -k_{\square} & \end{bmatrix} \begin{bmatrix} \square \\ \square \\ \square \end{bmatrix}, \qquad (3.61)$$
$$0 \quad -k_{\square} \quad 0$$

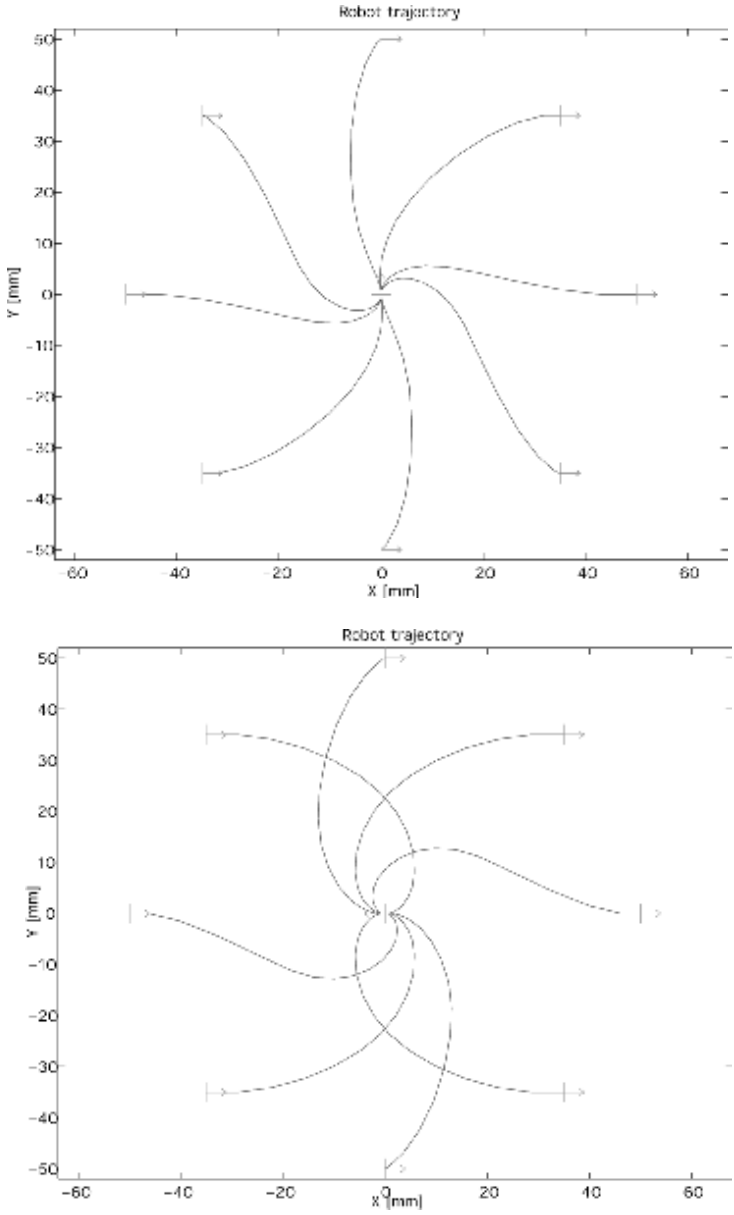


Figure 3.20

Resulting paths when the robot is initially on the unit circle in the *x*,*y* plane.

hence it is locally exponentially stable if the eigenvalues of the matrix

$$A = \begin{bmatrix} -k_{\square} & 0 & 0 \\ 0 & -\square k_{\square} - k_{\square} \square & \\ 0 & -k_{\square} & \\ & -k_{\square} & 0 \end{bmatrix} \tag{3.62}$$

all have a negative real part. The characteristic polynomial of the matrix A is

$$\square\square + k_{\square}\square\square\square^2 + \square\square k_{\square} - k_{\square}\square - k_{\square}\, k_{\square}\square \tag{3.63}$$

and all roots have negative real part if

$$k_{\square} \square 0 ; \quad -k_{\square} \square \quad k_{\square} - k_{\square} \qquad 0 ; \quad \square 0 \tag{3.64}$$

which proves the claim.

For robust position control, it might be advisable to apply the strong stability condition, which ensures that the robot does not change direction during its approach to the goal:

$$k_{\square} \square 0 ; \quad k_{\square} \square \quad k + \frac{5}{-} k - \frac{2}{--} \qquad 0 ; \quad k \square 0 \tag{3.65}$$

$$\alpha \quad 3 \beta \square \square$$

This implies that □ □ l_1 for all t, whenever □□0□ □ l_1 and □ □ l_2 for all t, whenever □□0□ □ l_2 respectively. This strong stability condition has also been verified in applica- tions.

4 Perception

One of the most important tasks of an autonomous system of any kind is to acquire knowl- edge about its environment. This is done by taking measurements using various sensors and then extracting meaningful information from those measurements.

In this chapter we present the most common sensors used in mobile robots and then dis- cuss strategies for extracting information from the sensors. For more detailed information about many of the sensors used on mobile robots, refer to the comprehensive book *Sensors for Mobile Robots* by H.R. Everett [15].

4.1 Sensors for Mobile Robots

There are a wide variety of sensors used in mobile robots (figure 4.1). Some sensors are used to measure simple values like the internal temperature of a robot's electronics or the rotational speed of the motors. Other, more sophisticated sensors can be used to acquire information about the robot's environment or even to directly measure a robot's global position. In this chapter we focus primarily on sensors used to extract information about the robot's environment. Because a mobile robot moves around, it will frequently encounter unforeseen environmental characteristics, and therefore such sensing is particularly critical. We begin with a functional classification of sensors. Then, after presenting basic tools for describing a sensor's performance, we

proceed to describe selected sensors in detail.

4.1.1 Sensor classification

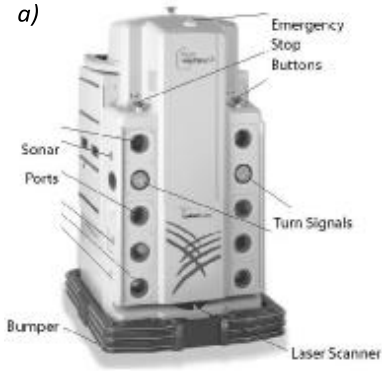
We classify sensors using two important functional axes: *proprioceptive/exteroceptive* and

passive/active.

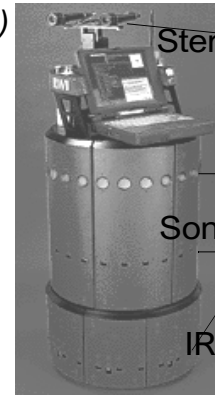
Proprioceptive sensors measure values internal to the system (robot); for example, motor speed, wheel load, robot arm joint angles, battery voltage.

Exteroceptive sensors acquire information from the robot's environment; for example, distance measurements, light intensity, sound amplitude. Hence exteroceptive sensor measurements are interpreted by the robot in order to extract meaningful environmental fea- tures.

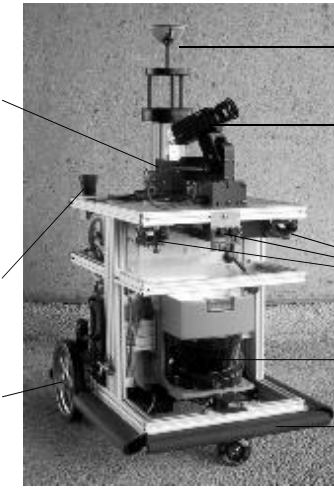
a)


Emergency
Stop
Buttons
Sonar
Ports
Turn Signals
Bumper
Laser Scanner

a)


Pan-Tilt
Stereo Camera
Sonar Sensors
IR Sensors

c)


Omnidirectional Camera
IMU
Inertial Measurement Unit
Pan-Tilt Camera
Emergency Stop Button
Wheel Encoders

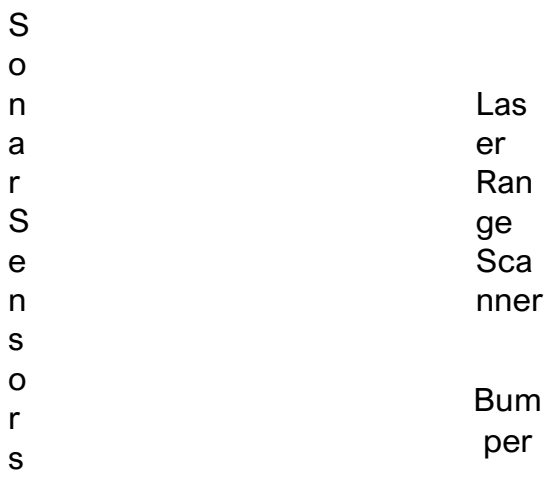


Figure 4.1

Examples of robots with multi-sensor systems: (a) HelpMate from Transition Research Corporation;

(b) B21 from Real World Interface; (c) BIBA Robot, BlueBotics SA.

Passive sensors measure ambient environmental energy entering the sensor. Examples of passive sensors include temperature probes, microphones, and CCD or CMOS cameras. *Active* sensors emit energy into the environment, then measure the environmental reac- tion. Because active sensors can manage more controlled interactions with the environ- ment, they often achieve superior performance. However, active sensing introduces several risks: the outbound energy may affect the very characteristics that the sensor is attempting to measure. Furthermore, an active sensor may suffer from interference between its signal

and those beyond its control. For example, signals emitted by other nearby robots, or sim- ilar sensors on the same robot, may influence the resulting measurements. Examples of active sensors include wheel quadrature encoders, ultrasonic sensors, and laser rangefind- ers.

Table 4.1 provides a classification of the most useful sensors for mobile robot applica- tions. The most interesting sensors are discussed in this chapter.

Table 4.1

Classification of sensors used in mobile robotics applications

General classification (typical use)	Sensor Sensor System	PC or EC	A or P
Tactile sensors (detection of physical contact or closeness; security switches)	Contact switches, bumpers Optical barriers Noncontact proximity sensors	E C E C E C	P A A
Wheel/motor sensors (wheel/motor speed and position)	Brush encoders Potentiometers Synchros, resolvers Optical encoders Magnetic encoders	P C P C P C P C P	P P A A A A A

	Inductive encoders Capacitive encoders	C P C P C	
Heading sensors (orientation of the robot in relation to a fixed reference frame)	Compass Gyroscopes Inclinometers	E C P C E C	P P A/P
Ground-based beacons (localization in a fixed reference frame)	GPS Active optical or RF beacons Active ultrasonic beacons Reflective beacons	E C E C E C E C	A A A A
Active ranging (reflectivity, time-of-flight, and geo- metric triangulation)	Reflectivity sensors Ultrasonic sensor Laser rangefinder Optical triangulation (1D) Structured light (2D)	E C E C E C E C E C	A A A A A
Motion/speed sensors (speed relative to fixed or moving objects)	Doppler radar Doppler sound	E C E C	A A

Vision-based sensors (visual ranging, whole-image analy- sis, segmentation, object recognition)	CCD/CMOS camera(s) Visual ranging packages Object tracking packages	EC	P

A, active; P, passive; P/A, passive/active; PC, proprioceptive; EC, exteroceptive.

The sensor classes in table 4.1 are arranged in ascending order of complexity and descending order of technological maturity. Tactile sensors and proprioceptive sensors are critical to virtually all mobile robots, and are well understood and easily implemented. Commercial quadrature encoders, for example, may be purchased as part of a gear-motor assembly used in a mobile robot. At the other extreme, visual interpretation by means of one or more CCD/CMOS cameras provides a broad array of potential functionalities, from obstacle avoidance and localization to human face recognition. However, commercially available sensor units that provide visual functionalities are only now beginning to emerge [90, 160].

4.1.2 Characterizing sensor performance

The sensors we describe in this chapter vary greatly in their performance characteristics. Some sensors provide extreme accuracy in well-controlled laboratory settings, but are overcome with error when subjected to real-world environmental variations. Other sensors provide narrow, high-precision data in a wide variety of settings. In order to quantify such performance characteristics, first we formally define the sensor performance terminology that will be valuable throughout the rest of this chapter.

4.1.2.1 Basic sensor response ratings

A number of sensor characteristics can be rated quantitatively in a laboratory setting. Such performance ratings will necessarily be best-case scenarios when the sensor is placed on a real-world robot, but are nevertheless useful.

Dynamic range is used to measure the spread between the lower and upper limits of input values to the sensor while

maintaining normal sensor operation. Formally, the dynamic range is the ratio of the maximum input value to the minimum measurable input value. Because this raw ratio can be unwieldy, it is usually measured in *decibels,* which are computed as ten times the common logarithm of the dynamic range. However, there is potential confusion in the calculation of decibels, which are meant to measure the ratio between *powers*, such as watts or horsepower. Suppose your sensor measures motor current and can register values from a minimum of 1 mA to 20 Amps. The dynamic range of this current sensor is defined as

$$10 \log \left[\frac{2}{0.001} \right] = 43 \text{ dB} \tag{4.1}$$

Now suppose you have a voltage sensor that measures the voltage of your robot's bat- tery, measuring any value from 1 mV to 20 V. Voltage is not a unit of power, but the square of voltage is proportional to power. Therefore, we use 20 instead of 10:

$$20 \cdot \log\left[\frac{2}{\frac{0}{0.0}}\right] = 86 \text{ dB} \quad (4.2)$$

Range is also an important rating in mobile robot applications because often robot sen- sors operate in environments where they are frequently exposed to input values beyond their working range. In such cases, it is critical to understand how the sensor will respond. For example, an optical rangefinder will have a minimum operating range and can thus pro- vide spurious data when measurements are taken with the object closer than that minimum. *Resolution* is the minimum difference between two values that can be detected by a sen- sor. Usually, the lower limit of the dynamic range of a sensor is equal to its resolution. However, in the case of digital sensors, this is not necessarily so. For example, suppose that you have a sensor that measures voltage, performs an analog-to-digital (A/D) conversion, and outputs the converted value as an 8-bit number linearly corresponding to between 0 and 5 V. If this sensor is truly linear, then it has $2^8 - 1$ total output values, or a resolution of $5\text{ V} \cdot 255 = 20\text{ mV}$.

Linearity is an important measure governing the behavior of the sensor's output signal as the input signal varies. A linear response indicates that if two inputs x and y result in the two outputs $f(x)$ and $f(y)$, then for any values a and b, $f(ax + by) = af(x) + bf(y)$. This means that a plot of the sensor's input/output response is simply a straight line.

Bandwidth or *frequency* is used to measure the speed with which a sensor can provide a stream of readings. Formally, the number of measurements per second is defined as the sen- sor's

frequency in *hertz.* Because of the dynamics of moving through their environment, mobile robots often are limited in maximum speed by the bandwidth of their obstacle detec- tion sensors. Thus, increasing the bandwidth of ranging and vision-based sensors has been a high-priority goal in the robotics community.

4.1.2.2 In situ sensor performance

The above sensor characteristics can be reasonably measured in a laboratory environment with confident extrapolation to performance in real-world deployment. However, a number of important measures cannot be reliably acquired without deep understanding of the com- plex interaction between all environmental characteristics and the sensors in question. This is most relevant to the most sophisticated sensors, including active ranging sensors and visual interpretation sensors.

Sensitivity itself is a desirable trait. This is a measure of the degree to which an incre- mental change in the target input signal changes the output signal. Formally, sensitivity is the ratio of output change to input change. Unfortunately, however, the sensitivity of exteroceptive sensors is often confounded by undesirable sensitivity and performance cou- pling to other environmental parameters.

Cross-sensitivity is the technical term for sensitivity to environmental parameters that are orthogonal to the target parameters for the sensor. For example, a flux-gate compass can demonstrate high sensitivity to magnetic north and is therefore of use for mobile robot nav- igation. However, the compass will also demonstrate high sensitivity to ferrous building materials, so much so that its cross-sensitivity often makes the sensor useless in some indoor environments. High cross-sensitivity of a sensor is generally undesirable, especially when it cannot be modeled.

Error of a sensor is defined as the difference between the sensor's output measurements and the true values being measured, within some specific operating context. Given a true value v and a measured value m, we can define *error* as $error = m - v$.

Accuracy is defined as the degree of conformity between the sensor's measurement and the true value, and is often expressed as a proportion of the true value (e.g., 97.5% accu- racy). Thus small error corresponds to high accuracy and vice versa:

$$accuracy = 1 - \left| \frac{error}{v} \right| \tag{4.3}$$

Of course, obtaining the ground truth, v , can be difficult or impossible, and so establish- ing a confident characterization of sensor accuracy can be problematic. Further, it is impor- tant to distinguish between two different sources of error:

Systematic errors are caused by factors or processes that can

in theory be modeled. These errors are, therefore, deterministic (i.e., predictable). Poor calibration of a laser rangefinder, an unmodeled slope of a hallway floor, and a bent stereo camera head due to an earlier collision are all possible causes of systematic sensor errors.

Random errors cannot be predicted using a sophisticated model nor can they be miti- gated by more precise sensor machinery. These errors can only be described in probabilistic terms (i.e., stochastically). Hue instability in a color camera, spurious rangefinding errors, and black level noise in a camera are all examples of random errors.

Precision is often confused with accuracy, and now we have the tools to clearly distin- guish these two terms. Intuitively, high precision relates to reproducibility of the sensor results. For example, one sensor taking multiple readings of the same environmental state has high precision if it produces the same output. In another example, multiple copies of this sensor taking readings of the same environmental state have high precision if their out- puts agree. Precision does not, however, have any bearing on the accuracy of the sensor's output with respect to the true value being measured. Suppose that the *random error* of a sensor is characterized by some mean value μ and a standard deviation σ. The formal def- inition of precision is the ratio of the sensor's output range to the standard deviation:

$$\text{precision} = \frac{range}{\Box} \tag{4.4}$$

Note that only $\Box$ and not $\Box$ has impact on precision. In contrast, mean error $\Box$ is

directly proportional to overall sensor error and inversely proportional to sensor accuracy.

4.1.2.3 Characterizing error: the challenges in mobile robotics

Mobile robots depend heavily on exteroceptive sensors. Many of these sensors concentrate on a central task for the robot: acquiring information on objects in the robot's immediate vicinity so that it may interpret the state of its surroundings. Of course, these "objects" sur- rounding the robot are all detected from the viewpoint of its local reference frame. Since the systems we study are mobile, their ever-changing position and their motion have a sig- nificant impact on overall sensor behavior. In this section, empowered with the terminol- ogy of the earlier discussions, we describe how dramatically the sensor error of a mobile robot disagrees with the ideal picture drawn in the previous section.

Blurring of systematic and random errors. Active ranging sensors tend to have failure modes that are triggered largely by specific relative positions of the sensor and environment targets. For example, a sonar sensor will produce specular reflections, producing grossly inaccurate measurements of range, at specific angles to a smooth sheetrock wall. During motion of the robot, such relative angles occur at stochastic intervals. This is

especially true in a mobile robot outfitted with a ring of multiple sonars. The chances of one sonar entering this error mode during robot motion is high. From the perspective of the moving robot, the sonar measurement error is a random error in this case. Yet, if the robot were to stop, becoming motionless, then a very different error modality is possible. If the robot's static position causes a particular sonar to fail in this manner, the sonar will fail consistently and will tend to return precisely the same (and incorrect!) reading time after time. Once the robot is motionless, the error appears to be systematic and of high precision.

The fundamental mechanism at work here is the cross-sensitivity of mobile robot sen- sors to robot pose and robot-environment dynamics. The models for such cross-sensitivity are not, in an underlying sense, truly random. However, these physical interrelationships are rarely modeled and therefore, from the point of view of an incomplete model, the errors appear random during motion and systematic when the robot is at rest.

Sonar is not the only sensor subject to this blurring of systematic and random error modality. Visual interpretation through the use of a CCD camera is also highly susceptible to robot motion and position because of camera dependence on lighting changes, lighting specularity (e.g., glare), and reflections. The important point is to realize that, while systematic error and random error are well-defined in a controlled setting, the mobile robot can exhibit error characteristics that bridge the gap between deterministic and stochastic error mechanisms.

Multimodal error distributions. It is common to characterize the behavior of a sensor's random error in terms of a probability distribution over various output values. In general, one knows very little about the causes of random error and therefore several simplifying assumptions are commonly used. For example, we can assume that the error is *zero-mean*, in that it symmetrically generates both positive and negative measurement error. We can go even further and assume that the probability density curve is Gaussian. Although we dis- cuss the mathematics of this in detail in section 4.2, it is important for now to recognize the fact that one frequently assumes *symmetry* as well as *unimodal distribution*. This means that measuring the correct value is most probable, and any measurement that is further away from the correct value is less likely than any measurement that is closer to the correct value. These are strong assumptions that enable powerful mathematical principles to be applied to mobile robot problems, but it is important to realize how wrong these assump- tions usually are.

Consider, for example, the sonar sensor once again. When ranging an object that reflects the sound signal well, the sonar will exhibit high accuracy, and will induce random error based on noise, for example, in the timing circuitry. This portion of its sensor behavior will exhibit error characteristics that are fairly symmetric and unimodal. However, when the sonar sensor is moving through an environment and is sometimes faced with materials that cause coherent reflection rather than returning the sound signal to the sonar sensor, then the sonar will grossly overestimate the distance to the object. In such cases, the error will be biased toward positive measurement error and will be far from the correct value. The error is not strictly systematic, and so

we are left modeling it as a probability distribution of random error. So the sonar sensor has two separate types of operational modes, one in which the signal does return and some random error is possible, and the second in which the signal returns after a multipath reflection, and gross overestimation error occurs. The probability distribution could easily be at least bimodal in this case, and since overestima- tion is more common than underestimation it will also be asymmetric.

As a second example, consider ranging via stereo vision. Once again, we can identify two modes of operation. If the stereo vision system correctly correlates two images, then the resulting random error will be caused by camera noise and will limit the measurement accuracy. But the stereo vision system can also correlate two images *incorrectly*, matching two fence posts, for example, that are not the same post in the real world. In such a case stereo vision will exhibit gross measurement error, and one can easily imagine such behav- ior violating both the unimodal and the symmetric assumptions.

The thesis of this section is that sensors in a mobile robot *may* be subject to multiple modes of operation and, when the sensor error is characterized, unimodality and symmetry may be grossly violated. Nonetheless, as we shall see, many successful mobile robot sys- tems make use of these simplifying assumptions and the resulting mathematical techniques with great empirical success.

The above sections have presented a terminology with which we can characterize the advantages and disadvantages of various mobile robot sensors. In the following sections, we do the same for a sampling of the most commonly used mobile robot sensors today.

4.1.3 Wheel/motor sensors

Wheel/motor sensors are devices used to measure the internal state and dynamics of a mobile robot. These sensors have vast applications outside of mobile robotics and, as a result, mobile robotics has enjoyed the benefits of high-quality, low-cost wheel and motor sensors that offer excellent resolution. In the next section, we sample just one such sensor, the optical incremental encoder.

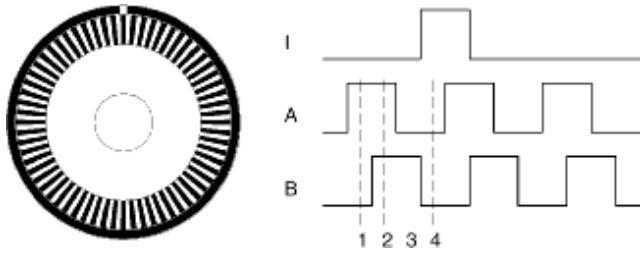
4.1.3.1 Optical encoders

Optical incremental encoders have become the most popular device for measuring angular speed and position within a motor drive or at the shaft of a wheel or steering mechanism. In mobile robotics, encoders are used to control the position or speed of wheels and other motor-driven joints. Because these sensors are *proprioceptive*, their estimate of position is best in the reference frame of the robot and, when applied to the problem of robot *localiza- tion*, significant corrections are required as, discussed in chapter 5.

An optical encoder is basically a mechanical light chopper that produces a certain number of sine or square wave pulses for each shaft revolution. It consists of an illumina- tion source, a fixed grating that masks the light, a rotor disc with a fine optical grid that rotates with the shaft, and fixed optical detectors. As the rotor moves, the amount of light striking the optical detectors

varies based on the alignment of the fixed and moving grat- ings. In robotics, the resulting sine wave is transformed into a discrete square wave using a threshold to choose between *light* and *dark* states. Resolution is measured in *cycles per rev- olution* (CPR). The minimum angular resolution can be readily computed from an encoder's CPR rating. A typical encoder in mobile robotics may have 2000 CPR, while the optical encoder industry can readily manufacture encoders with 10000 CPR. In terms of required bandwidth, it is of course critical that the encoder be sufficiently fast to count at the shaft spin speeds that are expected. Industrial optical encoders present no bandwidth limitation to mobile robot applications.

Usually in mobile robotics the *quadrature encoder* is used. In this case, a second illumi- nation and detector pair is placed 90 degrees shifted with respect to the original in terms of the rotor disc. The resulting twin square waves, shown in figure 4.2, provide significantly more information. The ordering of which square wave produces a rising edge first identifies the direction of rotation. Furthermore, the four detectably different states improve the res- olution by a factor of four with no change to the rotor disc. Thus, a 2000 CPR encoder in quadrature yields 8000 counts. Further improvement is possible by retaining the sinusoidal



State	Ch A	Ch B
s_1	high	low
s_2	high	high
s_3	low	high
s_4	low	low

Figure 4.2

Quadrature optical wheel encoder: The observed phase relationship between channel A and B pulse trains are used to determine the direction of the rotation. A single slot in the outer track generates a reference (index) pulse per revolution.

wave measured by the optical detectors and performing sophisticated interpolation. Such methods, although rare in mobile robotics, can yield 1000-fold improvements in resolution. As with most proprioceptive sensors, encoders are generally in the controlled environ- ment of a mobile robot's internal structure, and so systematic error and cross-sensitivity can be engineered away. The accuracy of optical encoders is often assumed to be 100% and, although this may not be entirely correct, any errors at the level of an optical encoder are dwarfed by errors downstream of the motor shaft.

4.1.4 Heading sensors

Heading sensors can be *proprioceptive* (gyroscope, inclinometer) or *exteroceptive* (com- pass). They are used to determine the robot's orientation and inclination. They allow us, together with appropriate velocity information, to integrate the

movement to a position esti- mate. This procedure, which has its roots in vessel and ship navigation, is called *dead reck- oning*.

4.1.4.1 Compasses

The two most common modern sensors for measuring the direction of a magnetic field are the Hall effect and flux gate compasses. Each has advantages and disadvantages, as described below.

The Hall effect describes the behavior of electric potential in a semiconductor when in the presence of a magnetic field. When a constant current is applied across the length of a semiconductor, there will be a voltage difference in the perpendicular direction, across the semiconductor's width, based on the relative orientation of the semiconductor to magnetic flux lines. In addition, the sign of the voltage potential identifies the direction of the mag- netic field. Thus, a single semiconductor provides a measurement of flux and direction along one dimension. Hall effect digital compasses are popular in mobile robotics, and con-



Figure 4.3

Digital compass: Sensors such as the digital/analog Hall effect sensor shown, available from Dins- more (http://dinsmoregroup.com/dico), enable inexpensive (< $ 15) sensing of magnetic fields.

tain two such semiconductors at right angles, providing two axes of magnetic field (thresh- olded) direction, thereby yielding one of eight possible compass directions. The instruments are inexpensive but also suffer from a range of disadvantages. Resolution of a digital Hall effect compass is poor. Internal sources of error include the nonlinearity of the basic sensor and systematic bias errors at the semiconductor level. The resulting circuitry must perform significant filtering, and this lowers the bandwidth of Hall effect compasses to values that are slow in mobile robot terms. For example, the Hall effect compass pictured in figure 4.3 needs 2.5 seconds to settle after a 90 degree spin.

The flux gate compass operates on a different principle. Two small coils are wound on ferrite cores and are fixed perpendicular to one another. When alternating current is activated in both coils, the magnetic field causes shifts in the phase depending on its relative alignment with each coil. By measuring

both phase shifts, the direction of the magnetic field in two dimensions can be computed. The flux gate compass can accurately measure the strength of a magnetic field and has improved resolution and accuracy; however, it is both larger and more expensive than a Hall effect compass.

Regardless of the type of compass used, a major drawback concerning the use of the Earth's magnetic field for mobile robot applications involves disturbance of that magnetic field by other magnetic objects and man-made structures, as well as the bandwidth limita- tions of electronic compasses and their susceptibility to vibration. Particularly in indoor environments, mobile robotics applications have often avoided the use of compasses, although a compass can conceivably provide useful *local* orientation information indoors, even in the presence of steel structures.

4.1.4.2 Gyroscopes

Gyroscopes are heading sensors which preserve their orientation in relation to a fixed ref- erence frame. Thus they provide an absolute measure for the heading of a mobile system.

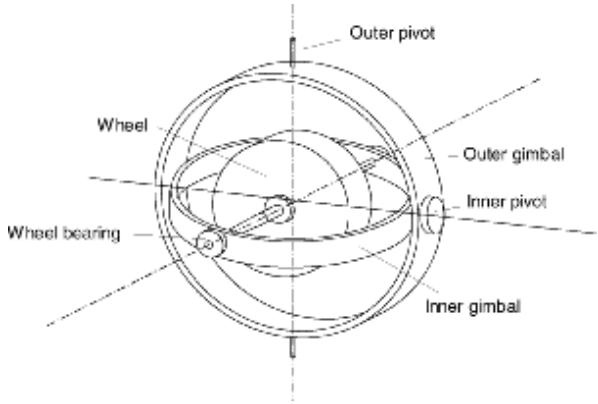


Figure 4.4

Two-axis mechanical gyroscope.

Gyroscopes can be classified in two categories, mechanical gyroscopes and optical gyro- scopes.

Mechanical gyroscopes. The concept of a mechanical gyroscope relies on the inertial properties of a fast-spinning rotor. The property of interest is known as the gyroscopic pre- cession. If you try to rotate a fast-spinning wheel around its vertical axis, you will feel a harsh reaction in the horizontal axis. This is due to the angular momentum associated with a spinning wheel and will keep the axis of the gyroscope inertially stable. The reactive torque and thus the tracking stability with the inertial frame are proportional to the spin- ning speed , the precession speed , and the wheel's inertia *I* .

$$\tau = I\omega\Omega \tag{4.5}$$

By arranging a spinning wheel, as seen in figure 4.4, no torque can be transmitted from the outer pivot to the wheel axis. The spinning axis will therefore be space-stable (i.e., fixed in an inertial reference frame). Nevertheless, the remaining friction in the bearings of the gyro axis introduce small torques, thus limiting the long-term space stability and introduc- ing small errors over time. A high quality mechanical gyroscope can cost up to $100,000 and has an angular drift of about 0.1 degrees in 6 hours.

For navigation, the spinning axis has to be initially selected. If the spinning axis is aligned with the north-south meridian, the earth's rotation has no effect on the gyro's hor- izontal axis. If it points east-west, the horizontal axis reads the earth rotation.

Rate gyros have the same basic arrangement as shown in figure 4.4 but with a slight modification. The gimbals are restrained by a torsional spring with additional viscous damping. This enables the sensor to measure angular speeds instead of absolute orientation.

Optical gyroscopes. Optical gyroscopes are a relatively new innovation. Commercial use began in the early 1980s when they were first installed in aircraft. Optical gyroscopes are angular speed sensors that use two monochromatic light beams, or lasers, emitted from the same source, instead of moving, mechanical parts. They work on the principle that the speed of light remains unchanged and, therefore, geometric change can cause light to take a varying amount of time to reach its destination. One laser beam is sent traveling clockwise through a fiber while the other travels counterclockwise. Because the laser traveling in the direction of rotation has a slightly shorter path, it will have a higher frequency. The differ- ence in frequency Δf of the two beams is a proportional to the angular velocity Ω of the cylinder. New solid-state optical gyroscopes based on the same principle are build using microfabrication technology, thereby providing heading information with resolution and bandwidth far beyond the needs of mobile robotic applications. Bandwidth, for instance, can easily exceed 100 kHz while resolution can be smaller than 0.0001 degrees/hr.

4.1.5 Ground-based beacons

One elegant approach to solving the localization problem in mobile robotics is to use active or passive beacons. Using the interaction of on-board sensors and the environmental bea- cons, the robot can identify its position precisely. Although the

general intuition is identical to that of early human navigation beacons, such as stars, mountains, and lighthouses, modern technology has enabled sensors to localize an outdoor robot with accuracies of better than 5 cm within areas that are kilometers in size.

In the following section, we describe one such beacon system, the global positioning system (GPS), which is extremely effective for outdoor ground-based and flying robots. Indoor beacon systems have been generally less successful for a number of reasons. The expense of environmental modification in an indoor setting is not amortized over an extremely large useful area, as it is, for example, in the case of the GPS. Furthermore, indoor environments offer significant challenges not seen outdoors, including multipath and environmental dynamics. A laser-based indoor beacon system, for example, must dis- ambiguate the one true laser signal from possibly tens of other powerful signals that have reflected off of walls, smooth floors, and doors. Confounding this, humans and other obsta- cles may be constantly changing the environment, for example, occluding the one true path from the beacon to the robot. In commercial applications, such as manufacturing plants, the environment can be carefully controlled to ensure success. In less structured indoor set- tings, beacons have nonetheless been used, and the problems are mitigated by careful beacon placement and the use of passive sensing modalities.

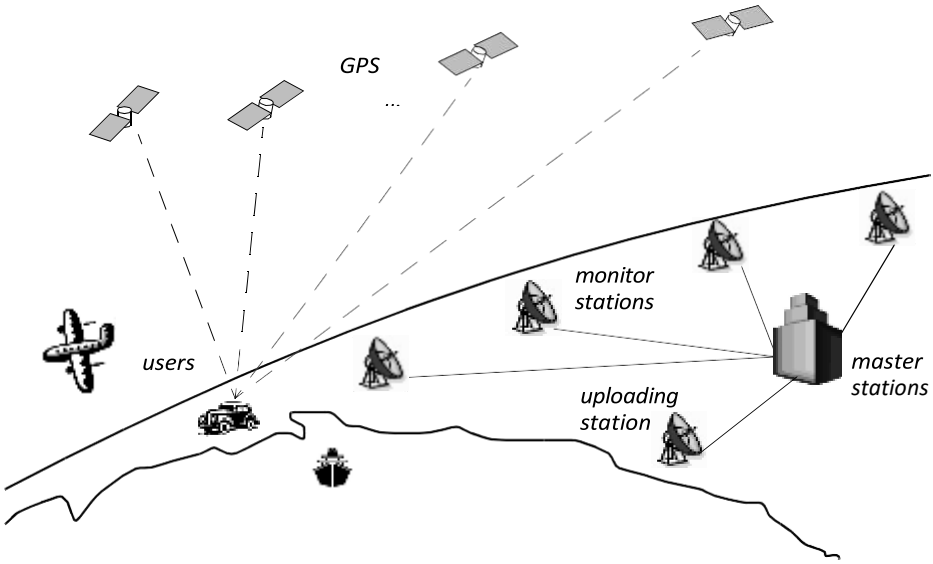


Figure 4.5

Calculation of position and heading based on GPS.

4.1.5.1 The global positioning system

The *global positioning system* (GPS) was initially developed for military use but is now freely available for civilian navigation. There are at least twenty-four operational GPS sat- ellites at all times. The satellites orbit every 12 hours at a height of 20.190 km. Four satel- lites are located in each of six planes inclined 55 degrees with respect to the plane of the earth's equator (figure 4.5).

Each satellite continuously transmits data that indicate its location and the current time. Therefore, GPS receivers are completely passive but exteroceptive sensors. The GPS sat- ellites synchronize their transmissions so that their signals are

sent at the same time. When a GPS receiver reads the transmission of two or more satellites, the arrival time differences inform the receiver as to its relative distance to each satellite. By combining information regarding the arrival time and instantaneous location of four satellites, the receiver can infer its own position. In theory, such triangulation requires only three data points. However, timing is extremely critical in the GPS application because the time intervals being mea- sured are in nanoseconds. It is, of course, mandatory that the satellites be well synchro- nized. To this end, they are updated by ground stations regularly and each satellite carries on-board atomic clocks for timing.

The GPS receiver clock is also important so that the travel time of each satellite's trans- mission can be accurately measured. But GPS receivers have a simple quartz clock. So, although three satellites would ideally provide position in three axes, the GPS receiver requires four satellites, using the additional information to solve for four variables: three position axes plus a time correction.

The fact that the GPS receiver must read the transmission of four satellites simulta- neously is a significant limitation. GPS satellite transmissions are extremely low-power, and reading them successfully requires direct line-of-sight communication with the satel- lite. Thus, in confined spaces such as city blocks with tall buildings or in dense forests, one is unlikely to receive four satellites reliably. Of course, most indoor spaces will also fail to provide sufficient visibility of the sky for a GPS receiver to function. For these reasons, the GPS has been a popular sensor in mobile robotics, but has been relegated to projects involv- ing mobile robot traversal of wide-open spaces and autonomous flying machines.

A number of factors affect the performance of a localization sensor that makes use of the GPS. First, it is important to understand that, because of the specific orbital paths of the GPS satellites, coverage is not geometrically identical in different portions of the Earth and therefore resolution is not uniform. Specifically, at the North and South Poles, the satellites are very close to the horizon and, thus, while resolution in the latitude and longitude direc- tions is good, resolution of altitude is relatively poor as compared to more equatorial loca- tions.

The second point is that GPS satellites are merely an information source. They can be employed with various

strategies in order to achieve dramatically different levels of localization resolution. The basic strategy for GPS use, called *pseudorange* and described above, generally performs at a resolution of 15 m. An extension of this method is *differen- tial GPS (DGPS)*, which makes use of a second receiver that is static and at a known exact position. A number of errors can be corrected using this reference, and so resolution improves to the order of 1 m or less. A disadvantage of this technique is that the stationary receiver must be installed, its location must be measured very carefully, and of course the moving robot must be within kilometers of this static unit in order to benefit from the DGPS technique.

A further improved strategy is to take into account the phase of the carrier signals of each received satellite transmission. There are two carriers, at 19 cm and 24 cm, and there- fore significant improvements in precision are possible when the phase difference between multiple satellites is measured successfully. Such receivers can achieve 1 cm resolution for point positions and, with the use of multiple receivers, as in DGPS, sub-1 cm resolution.

A final consideration for mobile robot applications is bandwidth.The GPS will generally offer no better than 200 to 300 ms latency, and so one can expect no better than 5 Hz GPS updates. On a fast-moving mobile robot or flying robot, this can mean that local motion integration will be required for proper control due to GPS latency limitations.

4.1.6 Active ranging

Active ranging sensors continue to be the most popular sensors in mobile robotics. Many ranging sensors have a low price point, and, most importantly, all ranging sensors provide easily interpreted outputs: direct measurements of distance from the robot to objects in its vicinity. For obstacle detection and avoidance, most mobile robots rely heavily on active ranging sensors. But the local freespace information provided by ranging sensors can also be accumulated into representations beyond the robot's current local reference frame. Thus active ranging sensors are also commonly found as part of the localization and environmen- tal modeling processes of mobile robots. It is only with the slow advent of successful visual interpretation competence that we can expect the class of active ranging sensors to gradu- ally lose their primacy as the sensor class of choice among mobile roboticists.

Below, we present two *time-of-flight* active ranging sensors: the ultrasonic sensor and the laser rangefinder. Then, we present two geometric active ranging sensors: the optical triangulation sensor and the structured light sensor.

4.1.6.1 Time-of-flight active ranging

Time-of-flight ranging makes use of the propagation speed of sound or an electromagnetic wave. In general, the travel distance of a sound of electromagnetic wave is given by

$$d = c \cdot t \tag{4.6}$$

where

d = distance traveled (usually round-trip);

c = speed of wave propagation;

t = time of flight.

It is important to point out that the propagation speed v of sound is approximately 0.3 m/ms whereas the speed of electromagnetic signals is 0.3 m/ns, which is 1 million times faster. The time of flight for a typical distance, say 3 m, is 10 ms for an ultrasonic system but only 10 ns for a laser rangefinder. It is thus evident that measuring the time of flight t with electromagnetic signals is more technologically challenging. This explains why laser range sensors have only recently become affordable and robust for use on mobile robots.

The quality of time-of-flight range sensors depends mainly on

- uncertainties in determining the exact time of arrival of the reflected signal;
- inaccuracies in the time-of-flight measurement (particularly with laser range sensors);
- the dispersal cone of the transmitted beam (mainly with ultrasonic range sensors);

- interaction with the target (e.g., surface absorption, specular reflections);
- variation of propagation speed;
- the speed of the mobile robot and target (in the case of a dynamic target);

As discussed below, each type of time-of-flight sensor is sensitive to a particular subset of the above list of factors.

The ultrasonic sensor (time-of-flight, sound). The basic principle of an ultrasonic sensor is to transmit a packet of (ultrasonic) pressure waves and to measure the time it takes for this wave packet to reflect and return to the receiver. The distance d of the object caus- ing the reflection can be calculated based on the propagation speed of sound c and the time of flight t .

$$d = \frac{c \cdot t}{2} \tag{4.7}$$

The speed of sound c in air is given by

$$c = \sqrt{\gamma R T} \tag{4.8}$$

where

γ = ratio of specific heats;

R = gas constant;

T = temperature in degrees Kelvin.

In air at standard pressure and 20° C the speed of sound is

approximately c = 343 m/s. Figure 4.6 shows the different signal output and input of an ultrasonic sensor. First, a series of sound pulses are emitted, comprising the *wave packet*. An integrator also begins to linearly climb in value, measuring the time from the transmission of these sound waves to detection of an echo. A threshold value is set for triggering an incoming sound wave as a valid echo. This threshold is often decreasing in time, because the amplitude of the expected echo decreases over time based on dispersal as it travels longer. But during trans- mission of the initial sound pulses and just afterward, the threshold is set very high to sup- press triggering the echo detector with the outgoing sound pulses. A transducer will continue to ring for up to several milliseconds after the initial transmission, and this gov- erns the *blanking time* of the sensor. Note that if, during the blanking time, the transmitted sound were to reflect off of an extremely close object and return to the ultrasonic sensor, it may fail to be detected.

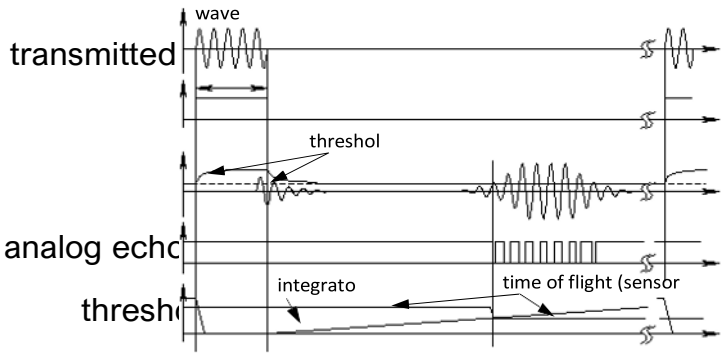


Figure 4.6

Signals of an ultrasonic sensor.

However, once the blanking interval has passed, the system will detect any above- threshold reflected sound, triggering a digital signal and producing the distance measure- ment using the integrator value.

The ultrasonic wave typically has a frequency between 40 and 180 kHz and is usually generated by a piezo or electrostatic transducer. Often the same unit is used to measure the reflected signal, although the required blanking interval can be reduced

through the use of separate output and input devices. Frequency can be used to select a useful range when choosing the appropriate ultrasonic sensor for a mobile robot. Lower frequencies corre- spond to a longer range, but with the disadvantage of longer post-transmission ringing and, therefore, the need for longer blanking intervals. Most ultrasonic sensors used by mobile robots have an effective range of roughly 12 cm to 5 m. The published accuracy of com- mercial ultrasonic sensors varies between 98% and 99.1%. In mobile robot applications, specific implementations generally achieve a resolution of approximately 2 cm.

In most cases one may want a narrow opening angle for the sound beam in order to also obtain precise directional information about objects that are encountered. This is a major limitation since sound propagates in a cone-like manner (figure 4.7) with opening angles around 20 to 40 degrees. Consequently, when using ultrasonic ranging one does not acquire depth data points but, rather, entire regions of constant depth. This means that the sensor tells us only that there is an object at a certain distance within the area of the measurement cone. The sensor readings must be plotted as segments of an arc (sphere for 3D) and not as point measurements (figure 4.8). However, recent research developments show significant improvement of the measurement quality in using sophisticated echo processing [76].

Ultrasonic sensors suffer from several additional drawbacks, namely in the areas of error, bandwidth, and cross-sensitivity. The published accuracy values for ultrasonics are

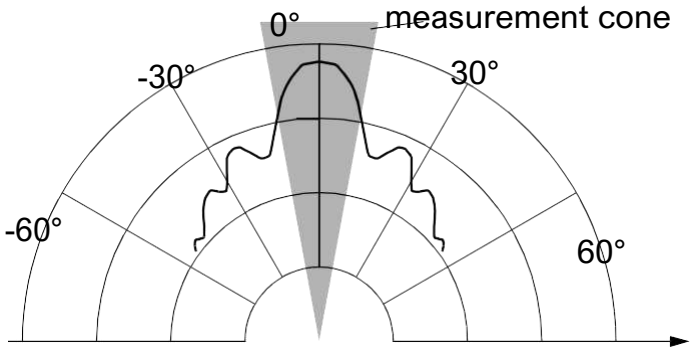


Figure 4.7

Typical intensity distribution of an ultrasonic sensor.



a) b)

Figure 4.8

Typical readings of an ultrasonic system: (a) 360 degree scan; (b) results from different geometric primitives [23]. Courtesy of John Leonard, MIT.

nominal values based on successful, perpendicular reflections of the sound wave off of an acoustically reflective material. This does not capture the effective error modality seen on a mobile robot moving through its environment. As the ultrasonic transducer's angle to the object being ranged varies away from perpendicular, the chances become good that the sound waves will coherently reflect away from the sensor, just as light at a shallow angle reflects off of a smooth surface. Therefore, the true error behavior of ultrasonic sensors is compound, with a well-understood error distribution near the true value in the case of a suc- cessful retroreflection, and a more poorly understood set of range values that are grossly larger than the true value in the case of coherent reflection. Of course, the acoustic proper- ties of the material being ranged have direct impact on the sensor's performance. Again, the impact is discrete, with one material possibly failing to produce a reflection that is suf- ficiently strong to be sensed by the unit. For example, foam, fur, and cloth can, in various circumstances, acoustically absorb the sound waves.

A final limitation of ultrasonic ranging relates to bandwidth. Particularly in moderately open spaces, a single ultrasonic sensor has a relatively slow cycle time. For example, mea- suring the distance to an object that is 3 m away will take such a sensor 20 ms, limiting its operating speed to 50 Hz. But if the robot has a ring of twenty ultrasonic sensors, each firing sequentially and measuring to minimize interference between the sensors, then the ring's cycle time becomes 0.4 seconds and the overall update frequency of any one sensor is just 2.5 Hz. For a robot conducting moderate speed motion while avoiding obstacles using ultrasonics, this update rate can have a measurable impact on the maximum speed possible while still sensing and avoiding

obstacles safely.

Laser rangefinder (time-of-flight, electromagnetic). The laser rangefinder is a time-of- flight sensor that achieves significant improvements over the ultrasonic range sensor owing to the use of laser light instead of sound. This type of sensor consists of a transmitter which illuminates a target with a collimated beam (e.g., laser), and a receiver capable of detecting the component of light which is essentially coaxial with the transmitted beam. Often referred to as optical radar or *lidar* (light detection and ranging), these devices produce a range estimate based on the time needed for the light to reach the target and return. A mechanical mechanism with a mirror sweeps the light beam to cover the required scene in a plane or even in three dimensions, using a rotating, nodding mirror.

One way to measure the time of flight for the light beam is to use a pulsed laser and then measure the elapsed time directly, just as in the ultrasonic solution described earlier. Elec- tronics capable of resolving picoseconds are required in such devices and they are therefore very expensive. A second method is to measure the beat frequency between a frequency- modulated continuous wave (FMCW) and its received reflection. Another, even easier method is to measure the phase shift of the reflected light. We describe this third approach in detail.

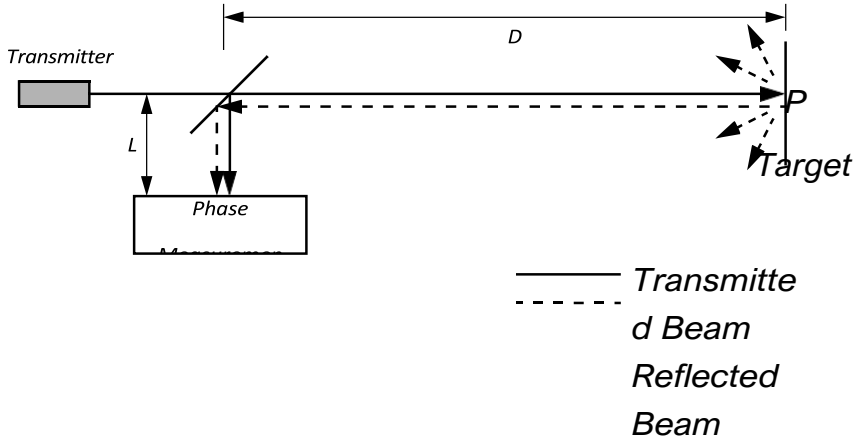


Figure 4.9

Schematic of laser rangefinding by phase-shift measurement.

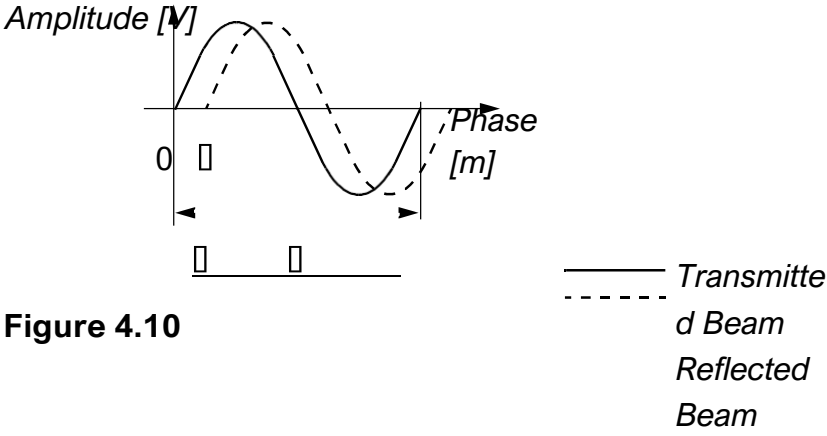


Figure 4.10

Range estimation by measuring the phase shift between transmitted and received signals.

Phase-shift measurement. Near-infrared light (from a light-emitting diode [LED] or laser) is collimated and transmitted from the transmitter in figure 4.9 and hits a point *P* in the environment. For surfaces having a roughness greater than the wavelength of the inci- dent light, diffuse reflection will occur, meaning that the light is reflected almost isotropi- cally. The wavelength of the infrared light emitted is 824 nm and so most surfaces, with the exception of only highly polished reflecting objects, will be diffuse reflectors. The compo- nent of the infrared light which falls within the receiving aperture of the sensor will return almost parallel to the transmitted beam for distant objects.

The sensor transmits 100% amplitude modulated light at a known frequency and mea- sures the phase shift between the transmitted and reflected signals. Figure 4.10 shows how this technique can be used to measure range. The wavelength of the modulating signal obeys the equation $c = f \lambda$ where c is the speed of light and f the modulating frequency. For f = 5 MHz (as in the AT&T sensor), λ = 60 m. The total distance D' covered by the emitted light is

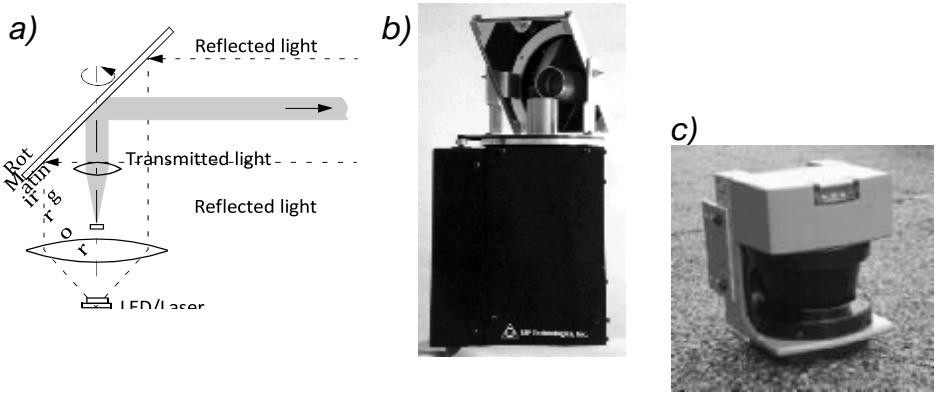


Figure 4.11

(a) Schematic drawing of laser range sensor with rotating mirror; (b) Scanning range sensor from EPS Technologies Inc.; (c) Industrial 180 degree laser range sensor from Sick Inc., Germany

$$D' = L + 2D = L + \frac{\Delta\theta}{2\theta} \tag{4.9}$$

where D and L are the distances defined in figure 4.9. The required distance D , between the beam splitter and the target, is therefore given by

$$D = -\frac{\Delta\theta}{4\theta} \tag{4.10}$$

where $\Delta\phi$ is the electronically measured phase difference between the transmitted and reflected light beams, and λ the known modulating wavelength. It can be seen that the transmission of a single frequency modulated wave can theoretically result in ambiguous range estimates since, for example, if λ = 60 m, a target at a range of 5 m would give an indistinguishable phase measurement from a target at 65 m, since each phase angle would be 360 degrees apart. We therefore define an "ambiguity interval" of λ, but in practice we note that the range of the sensor is much lower than λ due to the attenuation of the signal in air.

It can be shown that the confidence in the range (phase estimate) is inversely propor- tional to the square of the received signal amplitude, directly affecting the sensor's accu- racy. Hence dark, distant objects will not produce as good range estimates as close, bright objects.

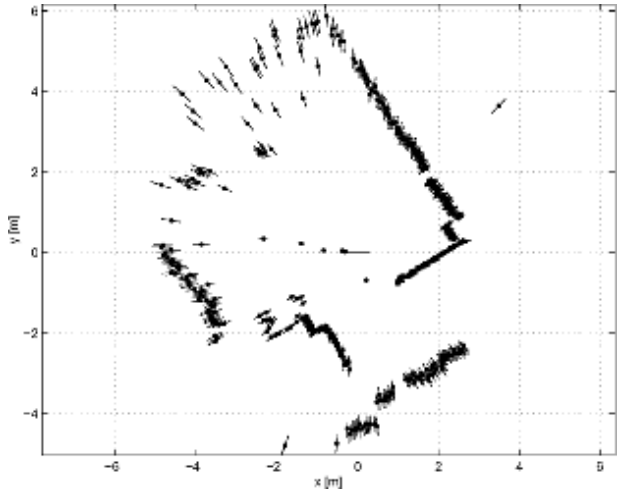


Figure 4.12

Typical range image of a 2D laser range sensor with a rotating mirror. The length of the lines through the measurement points indicate the uncertainties.

In figure 4.11 the schematic of a typical 360 degrees laser range sensor and two exam- ples are shown. figure 4.12 shows a typical range image of a 360 degrees scan taken with a laser range sensor.

As expected, the angular resolution of laser rangefinders far exceeds that of ultrasonic sensors. The Sick laser scanner shown in Figure 4.11 achieves an angular resolution of 0.5 degree. Depth resolution is approximately 5 cm, over a range from 5 cm up to 20 m or more, depending upon the brightness of the object being ranged. This device performs twenty five 180 degrees scans per second but has no mirror nodding capability

for the ver- tical dimension.

As with ultrasonic ranging sensors, an important error mode involves coherent reflection of the energy. With light, this will only occur when striking a highly polished surface. Prac- tically, a mobile robot may encounter such surfaces in the form of a polished desktop, file cabinet or, of course, a mirror. Unlike ultrasonic sensors, laser rangefinders cannot detect the presence of optically transparent materials such as glass, and this can be a significant obstacle in environments, for example, museums, where glass is commonly used.

4.1.6.2 Triangulation-based active ranging

Triangulation-based ranging sensors use geometric properties manifest in their measuring strategy to establish distance readings to objects. The simplest class of triangulation-based

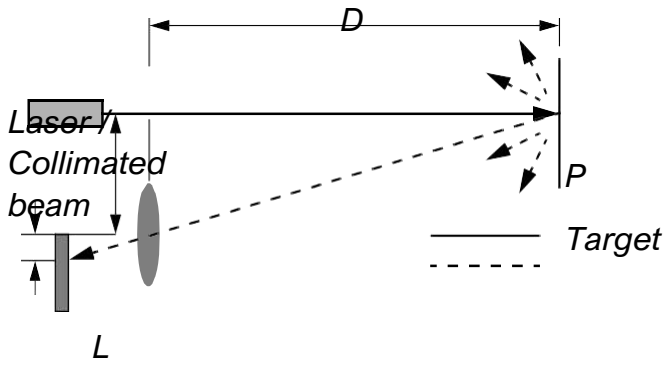


Figure 4.13

Principle of 1D laser triangulation.

rangers are *active* because they project a known light pattern (e.g., a point, a line, or a tex- ture) onto the environment. The reflection of the known pattern is captured by a receiver and, together with known geometric values, the system can use simple triangulation to establish range measurements. If the

receiver measures the position of the reflection along a single axis, we call the sensor an optical triangulation sensor in 1D. If the receiver mea- sures the position of the reflection along two orthogonal axes, we call the sensor a struc- tured light sensor. These two sensor types are described in the two sections below.

Optical triangulation (1D sensor). The principle of optical triangulation in 1D is straightforward, as depicted in figure 4.13. A collimated beam (e.g., focused infrared LED, laser beam) is transmitted toward the target. The reflected light is collected by a lens and projected onto a position-sensitive device (PSD) or linear camera. Given the geometry of figure 4.13, the distance D is given by

$$D = f \frac{L}{x} \tag{4.11}$$

The distance is proportional to $1 / x$; therefore the sensor resolution is best for close objects and becomes poor at a distance. Sensors based on this principle are used in range sensing up to 1 or 2 m, but also in high-precision industrial measurements with resolutions far below 1 μm.

Optical triangulation devices can provide relatively high accuracy with very good reso- lution (for close objects). However, the operating range of such a device is normally fairly limited by geometry. For example, the optical triangulation sensor pictured in figure 4.14

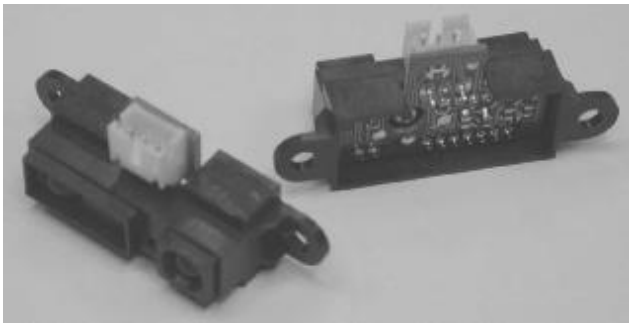


Figure 4.14

A commercially available, low-cost optical triangulation sensor: the Sharp GP series infrared rangefinders provide either analog or digital distance measures and cost only about $ 15.

operates over a distance range of between 8 and 80 cm. It is inexpensive compared to ultra- sonic and laser rangefinder sensors. Although more limited in range than sonar, the optical triangulation sensor has high bandwidth and does not suffer from cross-sensitivities that are more common in the sound domain.

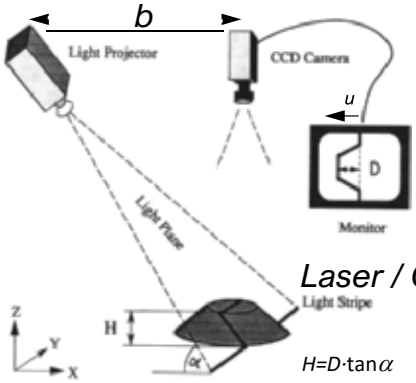
Structured light (2D sensor). If one replaces the linear camera or PSD of an optical tri- angulation sensor with a 2D receiver such as a CCD or CMOS camera, then one can recover distance to a large set of points instead of to only one point. The emitter must project a known pattern, or *structured light*, onto the environment. Many systems exist which either project light textures (figure 4.15b) or emit collimated light (possibly laser) by means of a rotating mirror. Yet another popular alternative is to project a laser stripe (figure 4.15a) by turning a laser beam into a

plane using a prism. Regardless of how it is created, the projected light has a known structure, and therefore the image taken by the CCD or CMOS receiver can be filtered to identify the pattern's reflection.

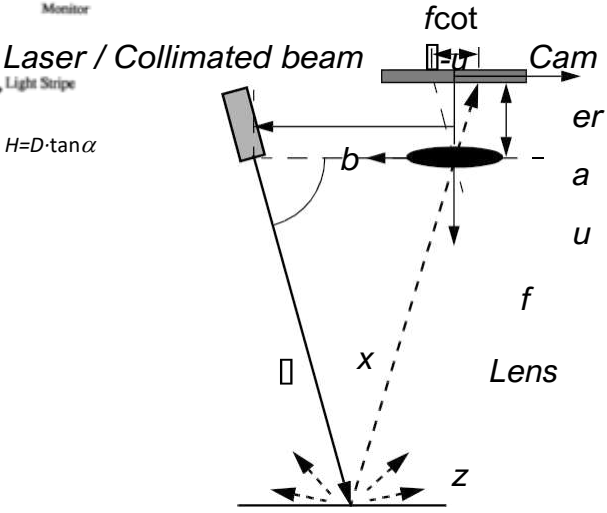
Note that the problem of recovering depth is in this case far simpler than the problem of passive image analysis. In passive image analysis, as we discuss later, existing features in the environment must be used to perform *correlation*, while the present method projects a known pattern upon the environment and thereby avoids the standard correlation problem altogether. Furthermore, the structured light sensor is an active device so it will continue to work in dark environments as well as environments in which the objects are featureless (e.g., uniformly colored and edgeless). In contrast, stereo vision would fail in such texture- free circumstances.

Figure 4.15c shows a 1D active triangulation geometry. We can examine the trade-off in the design of triangulation systems by examining the geometry in figure 4.15c. The mea-

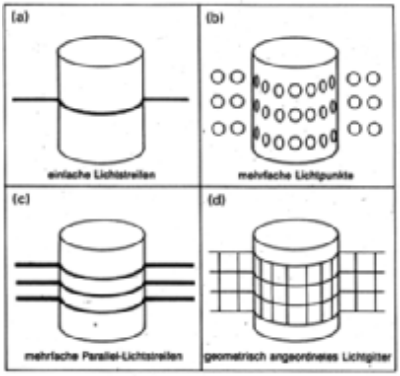
a)



$H=D\cdot\tan\alpha$



b)



(x, z) *Target*

Transmitted Beam ———

- - - - - -

c) *Reflected Beam*

Figure 4.15

(a) Principle of active two dimensional triangulation. (b) Other possible light structures. (c) 1D sche- matic of the principle. Image (a) and (b) courtesy of Albert-Jan Baerveldt, Halmstad University.

sured values in the system are and *u,* the distance of the illuminated point from the origin in the imaging sensor. (Note the imaging sensor here can be a camera or an array of photo diodes of a position-sensitive device (e.g., a 2D PSD).

From figure 4.15c, simple geometry shows that

$$x = \frac{b\,\square}{u}; \quad z = \frac{b}{\square \frac{f}{f\cot\square - \frac{u}{u}}} \tag{4.12}$$

where f is the distance of the lens to the imaging plane. In the limit, the ratio of image res- olution to range resolution is defined as the triangulation gain G_p and from equation (4.12) is given by

$$\frac{\delta u}{\delta z} = G_p \frac{b}{f} = \frac{b}{z^2} \tag{4.13}$$

This shows that the ranging accuracy, for a given image resolution, is proportional to source/detector separation b and focal length f, and decreases with the square of the range z. In a scanning ranging system, there is an additional effect on the ranging accuracy, caused by the measurement of the projection angle α. From equation 4.12 we see that

$$\frac{\delta \alpha}{\delta z} = G_\alpha \frac{b\sin\alpha}{z^2} \tag{4.14}$$

We can summarize the effects of the parameters on the sensor accuracy as follows:

- *Baseline length (b)*: the smaller b is, the more compact the sensor can be. The larger b is, the better the range resolution will be. Note also that although these sensors do not suffer from the correspondence problem, the disparity problem still occurs. As the base- line length b is increased, one introduces the chance that, for close objects, the illumi- nated point(s) may not be in the receiver's field of view.

- *Detector length and focal length (f)*: A larger detector length can provide either a larger field of view or an improved range resolution or partial benefits for both. Increasing the detector length, however, means a larger sensor head and worse electrical characteristics (increase in random error and reduction of bandwidth). Also, a short focal length gives a large field of view at the expense of accuracy, and vice versa.

At one time, laser stripe-based structured light sensors were common on several mobile robot bases as an inexpensive alternative to laser rangefinding devices. However, with the increasing quality of laser rangefinding sensors in the 1990s, the structured light system has become relegated largely to vision research rather than applied mobile robotics.

4.1.7 Motion/speed sensors

Some sensors measure directly the relative motion between the robot and its environment. Since such motion sensors detect relative motion, so long as an object is moving relative to the robot's reference frame, it will be detected and its speed can be estimated. There are a number of sensors that inherently measure some aspect of motion or change. For example, a pyroelectric sensor detects change in heat. When a human walks across the sensor's field

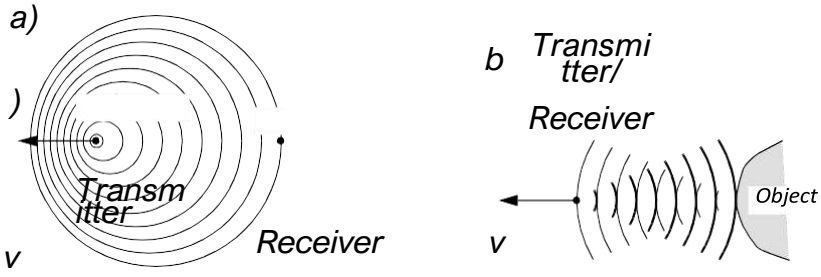


Figure 4.16

Doppler effect between two moving objects (a) or a moving and a stationary object (b).

of view, his or her motion triggers a change in heat in the sensor's reference frame. In the next section, we describe an important type of motion detector based on the Doppler effect. These sensors represent a well-known technology with decades of general applications behind them. For fast-moving mobile robots such as autonomous highway vehicles and unmanned flying vehicles, Doppler-based motion detectors are the obstacle detection sensor of choice.

4.1.7.1 Doppler effect-based sensing (radar or sound)

Anyone who has noticed the change in siren pitch that occurs when an approaching fire engine passes by and recedes is familiar with the Doppler effect.

A transmitter emits an electromagnetic or sound wave with a frequency f_t . It is either received by a receiver (figure 4.16a) or reflected from an object (figure 4.16b). The mea- sured frequency f_r at the receiver is a function of the relative speed v between transmitter
and receiver according to

$$f_r = f_t \frac{1}{1 + v / c} \tag{4.15}$$

if the transmitter is moving and

$$f_r = f_t (1 + v / c) \tag{4.16}$$

if the receiver is moving.

In the case of a reflected wave (figure 4.16b) there is a factor of 2 introduced, since any change x in relative separation affects the round-trip path length by $2x$. Furthermore, in such situations it is generally more convenient to consider the change in frequency Δf , known as the *Doppler shift*, as opposed to the *Doppler frequency* notation above.

$$\Delta f = f_t - f_r = \frac{2 f_t v \cos\theta}{c} \tag{4.17}$$

$$v = \frac{\Delta f}{\frac{c}{2 f_t \cos\theta}} \tag{4.18}$$

where

Δf = Doppler frequency shift;

θ = relative angle between direction of motion and beam axis.

The Doppler effect applies to sound and electromagnetic waves. It has a wide spectrum of applications:

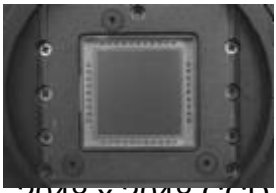
- *Sound waves*: for example, industrial process control, security, fish finding, measure of ground speed.
- *Electromagnetic waves*: for example, vibration measurement, radar systems, object tracking.

A current application area is both autonomous and manned highway vehicles. Both microwave and laser radar systems have been designed for this environment. Both systems have equivalent range, but laser can suffer when visual signals are deteriorated by environ- mental conditions such as rain, fog, and so on. Commercial microwave radar systems are already available for installation on highway trucks. These systems are called VORAD (vehicle on-board radar) and have a total range of approximately 150 m. With an accuracy of approximately 97%,

these systems report range rates from 0 to 160 km/hr with a resolu- tion of 1 km/hr. The beam is approximately 4 degrees wide and 5 degrees in elevation. One of the key limitations of radar technology is its bandwidth. Existing systems can provide information on multiple targets at approximately 2 Hz.

4.1.8 Vision-based sensors

Vision is our most powerful sense. It provides us with an enormous amount of information about the environment and enables rich, intelligent interaction in dynamic environments. It is therefore not surprising that a great deal of effort has been devoted to providing machines with sensors that mimic the capabilities of the human vision system. The first step in this process is the creation of sensing devices that capture the same raw information light that the human vision system uses. The next section describes the two current technologies for creating vision sensors: CCD and CMOS. These sensors have specific limitations in per- formance when compared to the human eye, and it is important that the reader understand these limitations. Afterward, the second and third sections describe vision-based sensors



2048 x 2048 CCD array



Sony DFW-X700

Orangemicro iBOT Firewire





Cannon IXUS 300

Figure 4.17

Commercially available CCD chips and CCD cameras. Because this technology is relatively mature, cameras are available in widely varying forms and costs (http://www.howstuffworks.com/digital- camera2.htm).

that are commercially available, like the sensors discussed previously in this chapter, along with their disadvantages and most popular applications.

4.1.8.1 CCD and CMOS sensors

CCD technology. The charged coupled device is the most popular basic ingredient of robotic vision systems today. The CCD chip (see figure 4.17) is an array of light-sensitive picture

elements, or pixels, usually with between 20,000 and several million pixels total. Each pixel can be thought of as a light-sensitive, discharging capacitor that is 5 to 25 μm in size. First, the capacitors of all pixels are charged fully, then the integration period begins. As photons of light strike each pixel, they liberate electrons, which are captured by electric fields and retained at the pixel. Over time, each pixel accumulates a varying level of charge based on the total number of photons that have struck it. After the integration period is complete, the relative charges of all pixels need to be frozen and read. In a CCD, the reading process is performed at one corner of the CCD chip. The bottom row of pixel charges is transported to this corner and read, then the rows above shift down and the pro- cess is repeated. This means that each charge must be transported across the chip, and it is critical that the value be preserved. This requires specialized control circuitry and custom fabrication techniques to ensure the stability of transported charges.

The photodiodes used in CCD chips (and CMOS chips as well) are not equally sensitive to all frequencies of light. They are sensitive to light between 400 and 1000 nm wavelength. It is important to remember that photodiodes are less sensitive to the ultraviolet end of the spectrum (e.g., blue) and are overly sensitive to the infrared portion (e.g., heat).

You can see that the basic light-measuring process is colorless: it is just measuring the total number of photons that strike each pixel in the integration period. There are two common approaches for creating *color* images. If the pixels on the CCD chip are grouped into 2 x 2 sets of four, then red, green, and blue dyes can be applied to a color filter so that each individual pixel receives only light of one color. Normally, two pixels measure green while one pixel each measures red and blue light intensity. Of course, this one-chip color CCD has a geometric resolution disadvantage. The number of pixels in the system has been effectively cut by a factor of four, and therefore the image resolution output by the CCD camera will be sacrificed.

The three-chip color camera avoids these problems by splitting the incoming light into three complete (lower intensity) copies. Three separate CCD chips receive the light, with one red, green, or blue filter over each entire chip. Thus, in parallel, each chip measures light intensity for one color, and the camera must combine the CCD chips' outputs to create a joint color image. Resolution is preserved in this solution, although the three-chip color cameras are, as one would expect, significantly more expensive and therefore more rarely used in mobile robotics.

Both three-chip and single-chip color CCD cameras suffer from the fact that photo- diodes are much more sensitive to the near-infrared end of the spectrum. This means that the overall system detects blue light much more poorly than red and green. To compensate, the gain must be increased on the blue channel, and this introduces greater absolute noise on blue than on red and green. It is not uncommon to assume at least one to two bits of addi- tional noise on the blue channel. Although there is no

satisfactory solution to this problem today, over time the processes for blue detection have been improved and we expect this positive trend to continue.

The CCD camera has several camera parameters that affect its behavior. In some cam- eras, these values are fixed. In others, the values are constantly changing based on built-in feedback loops. In higher-end cameras, the user can modify the values of these parameters via software. The *iris position* and *shutter speed* regulate the amount of light being mea- sured by the camera. The iris is simply a mechanical aperture that constricts incoming light, just as in standard 35 mm cameras. Shutter speed regulates the integration period of the chip. In higher-end cameras, the effective shutter speed can be as brief at 1/30,000 seconds and as long as 2 seconds. *Camera gain* controls the overall amplification of the analog sig- nal, prior to A/D conversion. However, it is very important to understand that, even though the image may appear brighter after setting high gain, the shutter speed and iris may not have changed at all. Thus gain merely amplifies the signal, and amplifies along with the signal all of the associated noise and error. Although useful in applications where imaging is done for human consumption (e.g., photography, television), gain is of little value to a mobile roboticist.

In color cameras, an additional control exists for *white balance*. Depending on the source of illumination in a scene (e.g., fluorescent lamps, incandescent lamps, sunlight, underwater filtered light, etc.), the relative measurements of red, green, and blue light that define pure white light will change dramatically. The human eye compensates for all such effects in ways that are not fully understood, but, the camera can demonstrate glaring incon- sistencies in which the same table looks blue in one image, taken during the night, and yellow in another image, taken during the day. White balance controls enable the user to change the relative gains for red, green, and blue in order to maintain more consistent color definitions in varying contexts.

The key disadvantages of CCD cameras are primarily in the areas of inconstancy and dynamic range. As mentioned above, a number of parameters can change the brightness and colors with which a camera creates its image. Manipulating these parameters in a way to provide consistency over time and over environments, for example, ensuring that a green shirt always looks green, and something dark gray is always dark gray, remains an open problem in the vision community. For more details on the fields of color constancy and luminosity constancy, consult [40].

The second class of disadvantages relates to the behavior of a CCD chip in environments with extreme illumination. In cases of very low illumination, each pixel will receive only a small number of photons. The longest possible integration period (i.e., shutter speed) and camera optics (i.e., pixel size, chip size, lens focal length and diameter) will determine the minimum level of light for which the signal is stronger than random error noise. In cases of very high illumination, a pixel fills its well with free electrons and,

as the well reaches its limit, the probability of trapping additional electrons falls and therefore the linearity between incoming light and electrons in the well degrades. This is termed *saturation* and can indicate the existence of a further problem related to cross-sensitivity. When a well has reached its limit, then additional light within the remainder of the integration period may cause further charge to leak into neighboring pixels, causing them to report incorrect values or even reach secondary saturation. This effect, called *blooming*, means that individual pixel values are not truly independent.

The camera parameters may be adjusted for an environment with a particular light level, but the problem remains that the dynamic range of a camera is limited by the well capacity of the individual pixels. For example, a high-quality CCD may have pixels that can hold 40,000 electrons. The noise level for reading the well may be 11 electrons, and therefore the dynamic range will be 40,000:11, or 3600:1, which is 35 dB.

CMOS technology. The complementary metal oxide semiconductor chip is a significant departure from the CCD. It too has an array of pixels, but located alongside each pixel are several transistors specific to that pixel. Just as in CCD chips, all of the pixels accumulate charge during the integration period. During the data collection step, the CMOS takes a new

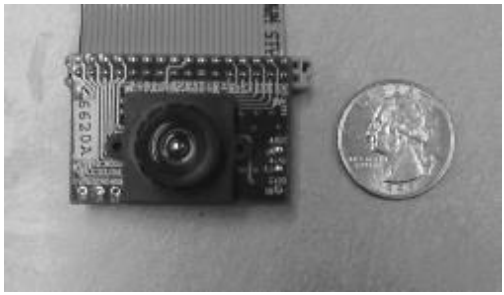


Figure 4.18

A commercially available, low-cost CMOS camera with lens attached.

approach: the pixel-specific circuitry next to every pixel measures and amplifies the pixel's signal, all in parallel for every pixel in the array. Using more traditional traces from general semiconductor chips, the resulting pixel values are all carried to their destinations.

CMOS has a number of advantages over CCD technologies. First and foremost, there is no need for the specialized clock drivers and circuitry required in the CCD to transfer each pixel's charge down all of the array columns and across all of its rows. This also means that specialized semiconductor manufacturing processes are not required to create CMOS chips. Therefore, the same production lines that create microchips can create inexpensive CMOS chips as well (see figure 4.18). The CMOS chip is so much simpler that it consumes significantly less power; incredibly, it operates with a power consumption that is one-hundredth the power consumption of a CCD chip. In a mobile robot, power is a scarce resource and therefore this is an important advantage.

On the other hand, the CMOS chip also faces several disadvantages. Most importantly, the circuitry next to each pixel consumes valuable real estate on the face of the light-detect- ing array. Many photons hit the transistors rather than the photodiode, making the CMOS chip significantly less sensitive than an equivalent CCD chip. Second, the CMOS technol- ogy is younger and, as a result, the best resolution that one can purchase in CMOS format continues to be far inferior to the best CCD chips available. Time will doubtless bring the high-end CMOS imagers closer to CCD imaging performance.

Given this summary of the mechanism behind CCD and CMOS chips, one can appreci- ate the sensitivity of any vision-based robot sensor to its environment. As compared to the human eye, these chips all have far poorer adaptation, cross-sensitivity, and dynamic range. As a result, vision sensors today continue to be fragile. Only over time, as the underlying performance of imaging chips improves, will significantly more robust vision-based sen- sors for mobile robots be available.

Camera output considerations. Although digital cameras have inherently digital output, throughout the 1980s and early 1990s, most affordable vision modules provided analog output signals, such as NTSC (National Television Standards Committee) and PAL (Phase Alternating Line). These camera systems included a D/A converter which, ironically, would be counteracted on the computer using a *framegrabber*, effectively an A/D converter board situated, for example, on a computer's bus. The D/A and A/D steps are far from noisefree, and furthermore the color depth of the analog signal in such cameras was opti- mized for human vision, not computer vision.

More recently, both CCD and CMOS technology vision systems provide digital signals that can be directly utilized by the roboticist. At the most basic level, an imaging chip pro- vides parallel digital I/O (input/output) pins that communicate discrete pixel level values. Some vision modules make use of these direct digital signals, which must be handled sub- ject to hard-time constraints governed by the imaging chip. To relieve the real-time demands, researchers often place an *image buffer chip* between the imager's digital output and the computer's digital inputs. Such chips, commonly used in webcams, capture a com-plete image snapshot and enable non real time access to the pixels, usually in a single, ordered pass.

At the highest level, a roboticist may choose instead to utilize a higher-level digital transport protocol to communicate with an imager. Most common are the IEEE 1394 (*Firewire*) standard and the USB (and USB 2.0) standards, although some older imaging modules also support serial (RS-232). To use any such high-level protocol, one must locate or create driver code both for that communication layer and for the particular implementa- tion

details of the imaging chip. Take note, however, of the distinction between lossless digital video and the standard digital video stream designed for human visual consumption. Most digital video cameras provide digital output, but often only in compressed form. For vision researchers, such compression must be avoided as it not only discards information but even introduces image detail that does not actually exist, such as MPEG (Moving Pic- ture Experts Group) discretization boundaries.

4.1.8.2 Visual ranging sensors

Range sensing is extremely important in mobile robotics as it is a basic input for successful obstacle avoidance. As we have seen earlier in this chapter, a number of sensors are popular in robotics explicitly for their ability to recover depth estimates: ultrasonic, laser rangefinder, optical rangefinder, and so on. It is natural to attempt to implement ranging functionality using vision chips as well.

However, a fundamental problem with visual images makes rangefinding relatively dif- ficult. Any vision chip collapses the 3D world into a 2D image plane, thereby losing depth information. If one can make strong assumptions regarding the size of objects in the world, or their particular color and reflectance, then one can directly interpret the appearance of the 2D image to recover depth. But such assumptions are rarely possible in real-world

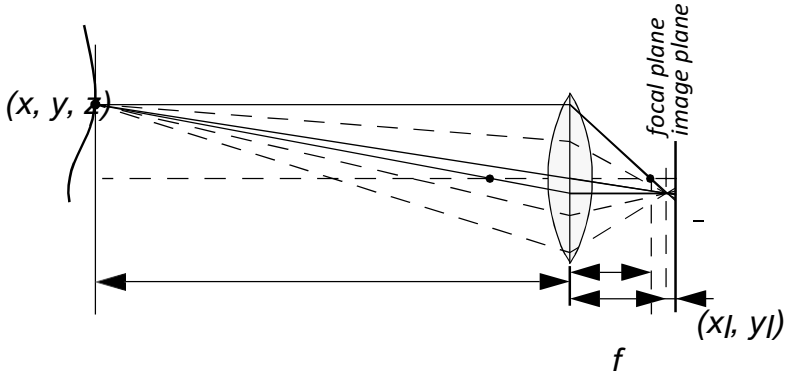


Figure 4.19

Depiction of the camera optics and its impact on the image. In order to get a sharp image, the image plane must coincide with the focal plane. Otherwise the image of the point (*x,y,z*) will be blurred in the image, as can be seen in the drawing above.

mobile robot applications. Without such assumptions, a single picture does not provide enough information to recover spatial information.

The general solution is to recover depth by looking at *several* images of the scene to gain more information, hopefully enough to at least partially recover depth. The images used must be different, so that taken together they provide additional information. They could differ in viewpoint, yielding *stereo* or *motion* algorithms. An alternative is to create differ- ent images,

not by changing the viewpoint, but by changing the camera geometry, such as the focus position or lens iris. This is the fundamental idea behind depth from focus and depth from defocus techniques.

In the next section, we outline the general approach to the depth from focus techniques because it presents a straightforward and efficient way to create a vision-based range sen- sor. Subsequently, we present details for the correspondence-based techniques of depth from stereo and motion.

Depth from focus. The depth from focus class of techniques relies on the fact that image properties not only change as a function of the scene but also as a function of the camera parameters. The relationship between camera parameters and image properties is depicted in figure 4.19.

The basic formula governing image formation relates the distance of the object from the lens, *d* in the above figure, to the distance *e* from the lens to the focal point, based on the focal length *f* of the lens:

$$\frac{1}{f} = \frac{1}{d} + \frac{1}{e} \tag{4.19}$$

If the image plane is located at distance *e* from the lens, then for the specific object

voxel depicted, all light will be focused at a single point on the image plane and the object voxel will be *focused*. However, when the image plane is not at *e* , as is depicted in figure 4.19, then the light from the object voxel will be cast on the image plane as a *blur circle*. To a first approximation, the light is homogeneously distributed throughout this blur circle, and the radius *R* of the circle can be characterized according to the equation

$$R = \frac{L\,\delta}{2e} \tag{4.20}$$

L is the diameter of the lens or aperture and δ is the displacement of the image plan from the focal point.

Given these formulas, several basic optical effects are clear. For example, if the aperture or lens is reduced to a point, as in a pinhole camera, then the radius of the blur circle approaches zero. This is consistent with the fact that decreasing the iris aperture opening causes the *depth of field* to increase until all objects are in focus. Of course, the disadvan- tage of doing so is that we are allowing less light to form the image on the image plane and so this is practical only in bright circumstances.

The second property that can be deduced from these optics

equations relates to the sen- sitivity of blurring as a function of the distance from the lens to the object. Suppose the image plane is at a fixed distance 1.2 from a lens with diameter $L = 0.2$ and focal length $f = 0.5$. We can see from equation (4.20) that the size of the blur circle R changes pro- portionally with the image plane displacement . If the object is at distance $d = 1$, then from equation (4.19) we can compute $e = 1$ and therefore = 0.2. Increase the object distance to d = 2 and as a result = 0.533. Using equation (4.20) in each case we can compute $R = 0.02$ and $R = 0.08$ respectively. This demonstrates high sensitivity for defocusing when the object is close to the lens.

In contrast, suppose the object is at $d = 10$. In this case we compute $e = 0.526$. But if the object is again moved one unit, to $d = 11$, then we compute $e = 0.524$. The resulting blur circles are $R = 0.117$ and $R = 0.129$, far less than the quadrupling in R when the obstacle is one-tenth the distance from the lens. This analysis demonstrates the fundamental limitation of depth from focus techniques: they lose sensitivity as objects move farther away (given a fixed focal length). Interestingly, this limitation will turn out to apply to vir- tually all visual ranging techniques, including depth from stereo and depth from motion.

Nevertheless, camera optics can be customized for the depth range of the intended appli- cation. For example, a zoom lens with a very large focal length f will enable range resolu-

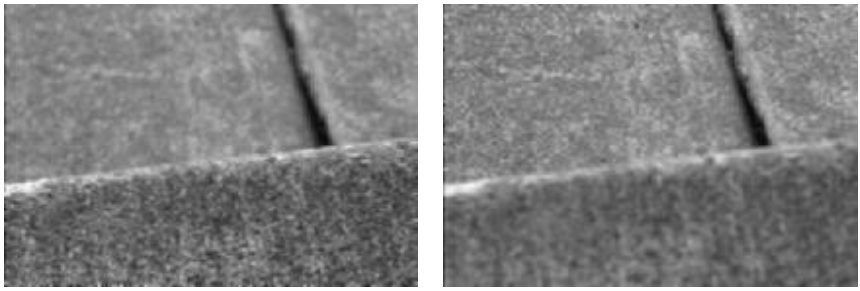


Figure 4.20

Two images of the same scene taken with a camera at two different focusing positions. Note the sig- nificant change in texture sharpness between the near surface and far surface. The scene is an outdoor concrete step.

tion at significant distances, of course at the expense of field of view. Similarly, a large lens diameter, coupled with a very fast shutter speed, will lead to larger, more detectable blur circles.

Given the physical effects summarized by the above equations, one can imagine a visual ranging sensor that makes use of multiple images in which camera optics are varied (e.g., image plane displacement □) and the same scene is captured (see figure 4.20). In fact, this approach is not a new invention. The human visual system uses an abundance of cues and techniques, and one system demonstrated in humans is depth from focus. Humans vary the focal length of their lens continuously at a rate of about 2 Hz. Such approaches, in which the lens optics are actively searched in order to maximize focus, are technically called *depth from focus*. In contrast, *depth from defocus* means that depth is recovered using a series of images that have been taken with different camera geometries.

The depth from focus method is one of the simplest visual ranging techniques. To deter- mine the range to an object, the sensor simply moves the image plane (via focusing) until maximizing the sharpness of the object. When the sharpness is maximized, the correspond- ing position of the image plane directly reports range. Some autofocus cameras and virtu- ally all autofocus video cameras use this technique. Of course, a method is required for measuring the sharpness of an image or an object within the image. The most common tech- niques are approximate measurements of the subimage *intensity* (*I*) *gradient*:

$$sharpness_1 = \sum_x \sum_y |I(x, y) - I(x-1, y)| \quad (4.21)$$

$$sharpness_2 = \sum_x \sum_y (I(x, y) - I(x-2, y-2))^2 \quad (4.22)$$



Figure 4.21

The Cheshm robot uses three monochrome cameras as its only ranging sensor for obstacle avoidance in the context of humans, static obstacles such as bushes, and convex obstacles such as ledges and steps.

A significant advantage of the horizontal sum of differences technique [equation (4.21)] is that the calculation can be implemented in analog circuitry using just a rectifier, a low- pass filter, and a high-pass filter. This is a common approach in commercial cameras and video recorders. Such systems will be sensitive to contrast along one particular axis, although in practical terms this is rarely an issue.

However depth from focus is an active search method and will be slow because it takes time to change the focusing parameters of the camera, using, for example, a servo-con- trolled focusing ring. For this reason this method has not been applied to mobile

robots.

A variation of the depth from focus technique has been applied to a mobile robot, dem- onstrating obstacle avoidance in a variety of environments, as well as avoidance of concave obstacles such as steps and ledges [117]. This robot uses three monochrome cameras placed as close together as possible with different, fixed lens focus positions (figure 4.21).

Several times each second, all three frame-synchronized cameras simultaneously cap- ture three images of the same scene. The images are each divided into five columns and three rows, or fifteen subregions. The approximate sharpness of each region is computed using a variation of equation (4.22), leading to a total of forty-five sharpness values. Note that equation (4.22) calculates sharpness along diagonals but skips one row. This is due to a subtle but important issue. Many cameras produce images in *interlaced* mode. This means

that the odd rows are captured first, then afterward the even rows are captured. When such a camera is used in dynamic environments, for example, on a moving robot, then adjacent rows show the dynamic scene at two different time points, differing by up to one-thirtieth of a second. The result is an artificial blurring due to motion and not optical defocus. By comparing only even-numbered rows we avoid this interlacing side effect.

Recall that the three images are each taken with a camera using a different focus posi- tion. Based on the focusing position, we call each image *close*, *medium* or *far*. A 5 x 3 coarse depth map of the scene is constructed quickly by simply comparing the sharpness values of each of the three corresponding regions. Thus, the depth map assigns only two bits of depth information to each region using the values *close, medium,* and *far*. The crit- ical step is to adjust the focus positions of all three cameras so that flat ground in front of the obstacle results in *medium* readings in one row of the depth map. Then, unexpected readings of either *close* or *far* will indicate convex and concave obstacles respectively, enabling basic obstacle avoidance in the vicinity of objects on the ground as well as drop-offs into the ground.

Although sufficient for obstacle avoidance, the above depth from focus algorithm pre- sents unsatisfyingly coarse range information. The alternative is *depth from defocus*, the most desirable of the focus-based vision techniques.

Depth from defocus methods take as input two or more images of the same scene, taken with different, known camera geometry. Given the images and the camera geometry set- tings, the goal is to recover the depth information of the 3D scene represented

by the images. We begin by deriving the relationship between the actual scene properties (irradi- ance and depth), camera geometry settings, and the image g that is formed at the image plane.

The *focused image* $f(x, y)$ of a scene is defined as follows. Consider a pinhole aperture ($L = 0$) in lieu of the lens. For every point p at position (x, y) on the image plane, draw a line through the pinhole aperture to the corresponding, visible point P in the actual scene. We define $f(x, y)$ as the irradiance (or light intensity) at p due to the light from P . Intu- itively, $f(x, y)$ represents the intensity image of the scene perfectly in focus.

The *point spread function* $h(x_g, y_g, x_f, y_f, R_x, y)$ is defined as the amount of irradiance from point P in the scene (corresponding to (x_f, y_f) in the focused image f that contributes to point (x_g, y_g) in the observed, defocused image g . Note that the point spread function depends not only upon the source, (x_f, y_f), and the target, (x_g, y_g), but also on R , the blur circle radius. R , in turn, depends upon the distance from point P to the lens, as can be seen by studying equations (4.19) and (4.20).

Given the assumption that the blur circle is homogeneous in intensity, we can define h as follows:

$$h(x, y, x_g, y_g) = \begin{cases} \frac{1}{\pi R^2} & \text{if } (x - x_g)^2 + (y - y_g)^2 \le R^2 \\ 0 & \text{if } (x - x_g)^2 + (y - y_g)^2 > R^2 \end{cases} \tag{4.23}$$

Intuitively, point P contributes to the image pixel (x_g, y_g) only when the blur circle of point P contains the point (x_g, y_g). Now we can write the general formula that computes the value of each pixel in the image, $f(x, y)$, as a function of the point spread function and
the focused image:

$$g(x_g, y_g) = \int_x \int_y h(x_g, y_g, x, y) f(x, y) \, dx \, dy \tag{4.24}$$

This equation relates the depth of scene points via R to the observed image g. Solving for R would provide us with the depth map. However, this function has another unknown, and that is f, the focused image. Therefore, one image alone is insufficient to solve the depth recovery problem, assuming we do *not* know how the fully focused image would look.

Given two images of the same scene, taken with varying camera geometry, in theory it will be possible to solve for g as well as R because f stays constant. There are a number of algorithms for implementing such a solution accurately and

quickly. The classic approach is known as *inverse filtering* because it attempts to directly solve for R , then extract depth information from this solution. One special case of the inverse filtering solu- tion has been demonstrated with a real sensor. Suppose that the incoming light is split and sent to two cameras, one with a large aperture and the other with a pinhole aperture [121]. The pinhole aperture results in a fully focused image, directly providing the value of f . With this approach, there remains a single equation with a single unknown, and so the solution is straightforward. Pentland [121] has demonstrated such a sensor, with several meters of range and better than 97% accuracy. Note, however, that the pinhole aperture necessitates a large amount of incoming light, and that furthermore the actual image intensities must be normalized so that the pinhole and large-diameter images have equivalent total radiosity. More recent depth from defocus methods use statistical techniques and charac- terization of the problem as a set of linear equations [64]. These matrix-based methods have recently achieved significant improvements in accuracy over all previous work.

In summary, the basic advantage of the depth from defocus method is its extremely fast speed. The equations above do not require search algorithms to find the solution, as would the correlation problem faced by depth from stereo methods. Perhaps more importantly, the depth from defocus methods also need not capture the scene at different perspectives, and are therefore unaffected by occlusions and the disappearance of objects in a second view.

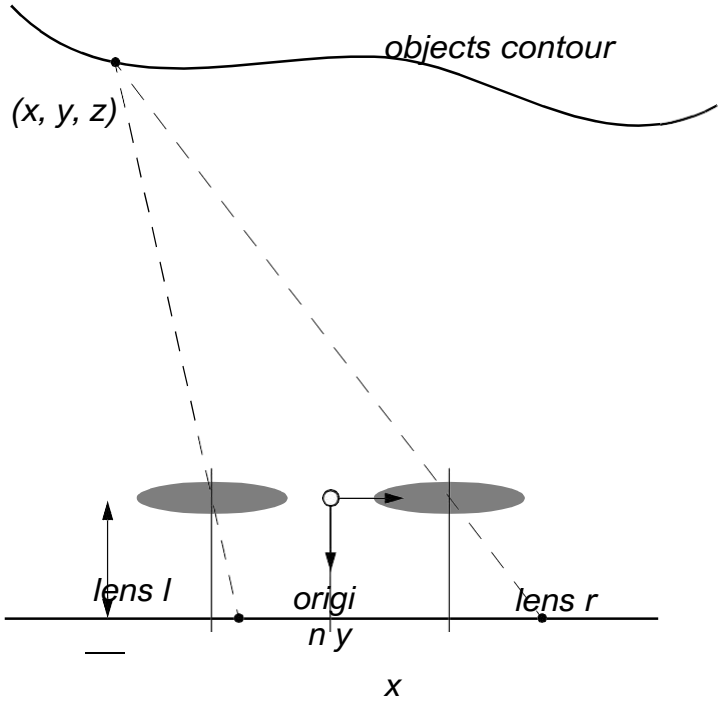


Figure 4.22

Idealized camera geometry for stereo vision.

As with all visual methods for ranging, accuracy decreases with distance. Indeed, the accuracy can be extreme; these

methods have been used in microscopy to demonstrate ranging at the micrometer level.

Stereo vision. Stereo vision is one of several techniques in which we recover depth infor- mation from two images that depict the scene from different perspectives. The theory of depth from stereo has been well understood for years, while the engineering challenge of creating a practical stereo sensor has been formidable [16, 29, 30]. Recent times have seen the first successes on this front, and so after presenting a basic formalism of stereo ranging, we describe the state-of-the-art algorithmic approach and one of the recent, commercially available stereo sensors.

First, we consider a simplified case in which two cameras are placed with their optical axes parallel, at a separation (called the *baseline*) of *b*, shown in figure 4.22.

In this figure, a point on the object is described as being with at coordinate (x, y, z)

respect to a central origin located between the two camera lenses. The position of this

point's light rays on each camera's image is depicted in camera-specific local coordinates. Thus, the origin for the coordinate frame referenced by points of the form (x_l,y_l) is located at the center of lens l .

From the figure 4.22, it can be seen that

$$\frac{x_l}{f} = \frac{x + b/2}{z} \quad \text{and} \quad \frac{x_r}{f} = \frac{x - b/2}{z} \tag{4.25}$$

and (out of the plane of the page)

$$\frac{y_l}{f} = \frac{y_r}{f} = \frac{y}{z} \tag{4.26}$$

where f is the distance of both lenses to the image plane. Note from equation (4.25) that

$$\frac{x_l - x_r}{f} = \frac{b}{z} \tag{4.27}$$

where the *difference* in the image coordinates, $x_l - x_r$ is called the *disparity.* This is an

important term in stereo vision, because it is only by measuring disparity that we can recover depth information. Using the

disparity and solving all three above equations pro- vides formulas for the three dimensions of the scene point being imaged:

$$x = \frac{b\left(x_l + \frac{x_l - x_r}{2}\right)}{x_l - x_r}; \quad y = \frac{b\left(y_l + \frac{y_l - y_r}{2}\right)}{x_l - x_r}; \quad z = b\frac{f}{x_l - x_r} \tag{4.28}$$

Observations from these equations are as follows:

- Distance is inversely proportional to disparity. The distance to near objects can therefore be measured more accurately than that to distant objects, just as with depth from focus techniques. In general, this is acceptable for mobile robotics, because for navigation and obstacle avoidance closer objects are of greater importance.
- Disparity is proportional to b . For a given disparity error, the accuracy of the depth esti- mate increases with increasing baseline b .
- As b is increased, because the physical separation between the cameras is increased, some objects may appear in one camera but not in the other. Such objects by definition will not have a disparity and therefore will not be ranged successfully.

- A point in the scene visible to both cameras produces a pair of image points (one via each lens) known as a *conjugate pair*. Given one member of the conjugate pair, we know that the other member of the pair lies somewhere along a line known as an *epipolar line*. In the case depicted by figure 4.22, because the cameras are perfectly aligned with one another, the epipolar lines are horizontal lines (i.e., along the x direction).

However, the assumption of perfectly aligned cameras is normally violated in practice. In order to optimize the range of distances that can be recovered, it is often useful to turn the cameras inward toward one another, for example. Figure 4.22 shows the *orientation* vectors that are necessary to solve this more general problem. We will express the position of a scene point P in terms of the reference frame of each camera separately. The reference frames of the cameras need not be aligned, and can indeed be at any arbitrary orientation relative to one another.

For example the position of point P will be described in terms of the left camera frame as $r'_l = [x'_l, y'_l, z'_l]$. Note that these are the coordinates of point P, not the position of its counterpart in the left camera image. P can also be described in terms of the right camera frame as $r'_r = [x'_r, y'_r, z'_r]$. If we have a rotation matrix R and translation matrix r_0 relat- ing the relative positions of cameras l and r, then we can define r'_r in terms of r'_l:

$$r'_r = R \cdot r'_l + r_0 \tag{4.29}$$

where R is a 3 x 3 rotation matrix and r_0 is an offset translation

matrix between the two cameras.

Expanding equation (4.29) yields

$$\begin{bmatrix} x'_r \\ y'_r \\ z'_r \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{21} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} x'_l \\ y'_l \\ z'_l \end{bmatrix} + \begin{bmatrix} r_0 \\ r_0 \\ r_0 \end{bmatrix} \begin{matrix} 1 \\ 2 \\ 3 \end{matrix} \qquad (4.30)$$

The above equations have two uses:

1. We could find r'_r if we knew R, r'_l and r_0 . Of course, if we knew r'_l then we would have complete information regarding the position of P relative to the left camera, and so the depth recovery problem would be solved. Note that, for perfectly aligned cameras
   as in figure 4.22, $R = I$ (the identify matrix).
2. We could calibrate the system and find r_{11}, r_{12} … given a set of conjugate pairs
   $\square\square x'_l\square\ y'_l\square\ z'_l\ \square\square\ \square x'_r\square\ y'_r\square\ z'_r\ \square\square$.

In order to carry out the calibration step of step 2 above, we must find values for twelve unknowns, requiring twelve equations. This means that calibration requires, for a given scene, four conjugate points.

The above example supposes that regular translation and rotation are all that are required to effect sufficient calibration for stereo depth recovery using two cameras. In fact, single- camera calibration is itself an active area of research, particularly when the goal includes any 3D recovery aspect. When researchers intend to use even a single camera with high pre- cision in 3D, internal errors relating to the exact placement of the imaging chip relative to the lens optical axis, as well as aberrations in the lens system itself, must be calibrated against. Such single-camera calibration involves finding solutions for the values for the exact offset of the imaging chip relative to the optical axis, both in translation and angle, and finding the relationship between distance along the imaging chip surface and external viewed surfaces. Furthermore, even without optical aberration in play, the lens is an inher- ently radial instrument, and so the image projected upon a flat imaging surface is radially distorted (i.e., parallel lines in the viewed world converge on the imaging chip).

A commonly practiced technique for such single-camera calibration is based upon acquiring multiple views of an easily analyzed planar pattern, such as a grid of black squares on a white background. The corners of such squares can easily be extracted, and using an interactive refinement algorithm the intrinsic calibration parameters of a camera can be extracted. Because modern imaging systems are capable of spatial accuracy greatly exceeding the pixel size, the payoff of such refined calibration can be significant. For fur- ther discussion of

calibration and to download and use a standard calibration program, see [158].

Assuming that the calibration step is complete, we can now formalize the range recovery problem. To begin with, we do not have the position of P available, and therefore (x'_l, y'_l, z'_l) and (x'_r, y'_r, z'_r) are unknowns. Instead, by virtue of the two cameras we have pixels on the image planes of each camera, (x_l, y_l, z_l) and (x_r, y_r, z_r). Given the focal length f of the cameras we can relate the position of P to the left camera image as follows:

$$\frac{x_l}{f} = \frac{x'_l}{z'_l} \quad \text{and} \quad \frac{y_l}{f} = \frac{y'_l}{z'_l} \tag{4.31}$$

Let us concentrate first on recovery of the values z'_l and z'_r. From equations (4.30) and (4.31) we can compute these values from any two of the following equations:

$$\frac{(r_{11}x_l + r_{12}y_l + r_{13}f)z'_l}{f} + r_{01} = \frac{x_r}{f}z'_r \tag{4.32}$$

$$x'_r = \left(r_{21}\frac{x_l}{f} + r_{22}\frac{y_l}{f} + r_{23}\right) z'_l + r_{02} = \frac{x'_r}{f} z'_r \tag{4.33}$$

$$z'_l + r_{03} = \left(r_{31}\frac{x_l}{f} + r_{32}\frac{y_l}{f} + r_{33}\right) z'_r \tag{4.34}$$

The same process can be used to identify values for x' and y', yielding complete infor- mation about the position of point P. However, using the above equations requires us to have identified conjugate pairs in the left and right camera images: image points that orig- inate at the same object point P in the scene. This fundamental challenge, identifying the conjugate pairs and thereby recovering disparity, is the *correspondence problem*. Intu- itively, the problem is, given two images of the same scene from different perspectives, how can we identify the same object points in both images? For every such identified object point, we will be able to recover its 3D position in the scene.

The correspondence problem, or the problem of matching the same object in two differ- ent inputs, has been one of the most challenging problems in the computer vision field and the artificial intelligence fields. The basic approach in nearly all proposed solutions involves converting each image in order to create more stable and more information-rich data. With more reliable data in hand, stereo algorithms *search* for the best conjugate pairs representing as many of the images' pixels as

possible.

The search process is well understood, but the quality of the resulting depth maps depends heavily upon the way in which images are treated to reduce noise and improve sta- bility. This has been the chief technology driver in stereo vision algorithms, and one par- ticular method has become widely used in commercially available systems.

The zero crossings of Laplacian of Gaussian (ZLoG). ZLoG is a strategy for identify- ing features in the left and right camera images that are stable and will match well, yielding high-quality stereo depth recovery. This approach has seen tremendous success in the field of stereo vision, having been implemented commercially in both software and hardware with good results. It has led to several commercial stereo vision systems and yet it is extremely simple. Here we summarize the approach and explain some of its advantages.

The core of ZLoG is the Laplacian transformation of an image. Intuitively, this is noth- ing more than the second derivative. Formally, the Laplacian $L(x, y)$ of an image with intensities $I(x, y)$ is defined as

$$L(x, y) = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2} \tag{4.35}$$

So the Laplacian represents the second derivative of the image, and is computed along both axes. Such a transformation, called a *convolution*, must be computed over the discrete space of image pixel values, and therefore an approximation of equation (4.35) is required for application:

$$L = P \otimes I \tag{4.36}$$

We depict a discrete operator P, called a *kernel*, that approximates the second derivative operation along both axes as a 3 x 3 table:

$$\begin{matrix} 0 \\ 1 \\ 0 \end{matrix}\begin{bmatrix} 1 & 0 \\ -4 & 1 \\ 1 & 0 \end{bmatrix} \tag{4.37}$$

Application of the kernel P to convolve an image is straightforward. The kernel defines the contribution of each pixel in the image to the corresponding pixel in the target as well as its neighbors. For example, if a pixel (5,5) in the image I has value $I[5][5] = 10$, then application of the kernel depicted by equation (4.37) causes pixel $I[5][5]$ to make the fol- lowing contributions to the target image L :

$L[5][5]$ += -40;

$L[4][5]$ += 10;

$L[6][5]$ += 10;

L[5][4][+= 10;

L[5][6][+= 10.

Now consider the graphic example of a step function, representing a pixel row in which the intensities are dark, then suddenly there is a jump to very bright intensities. The second derivative will have a sharp positive peak followed by a sharp negative peak, as depicted in figure 4.23. The Laplacian is used because of this extreme sensitivity to changes in the image. But the second derivative is in fact oversensitive. We would like the Laplacian to trigger large peaks due to real changes in the scene's intensities, but we would like to keep signal noise from triggering false peaks.

For the purpose of removing noise due to sensor error, the ZLoG algorithm applies Gaussian smoothing first, then executes the Laplacian convolution. Such smoothing can be effected via convolution with a 3 [3 table that approximates Gaussian smoothing:

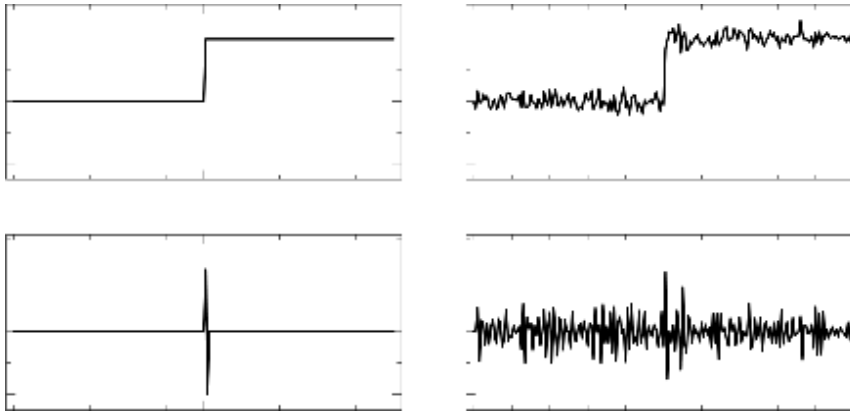


Figure 4.23

Step function example of second derivative shape and the impact of noise.

$$\begin{bmatrix} \frac{1}{16} & \frac{2}{16} & \frac{1}{16} \\ \frac{2}{16} & \frac{4}{16} & \frac{2}{16} \\ \frac{1}{16} & \frac{2}{16} & \frac{1}{16} \end{bmatrix} \tag{4.38}$$

Gaussian smoothing does not really remove error; it merely distributes image variations over larger areas. This should seem

familiar. In fact, Gaussian smoothing is almost identical to the blurring caused by defocused optics. It is, nonetheless, very effective at removing high-frequency noise, just as blurring removes fine-grained detail. Note that, like defocus- ing, this kernel does not change the total illumination but merely redistributes it (by virtue of the divisor 16).

The result of Laplacian of Gaussian (LoG) image filtering is a target array with sharp positive and negative spikes identifying boundaries of change in the original image. For example, a sharp edge in the image will result in both a positive spike and a negative spike, located on either side of the edge.

To solve the correspondence problem, we would like to identify specific *features* in LoG that are amenable to matching between the left camera and right camera filtered images. A very effective feature has been to identify each *zero crossing* of the LoG as such a feature.

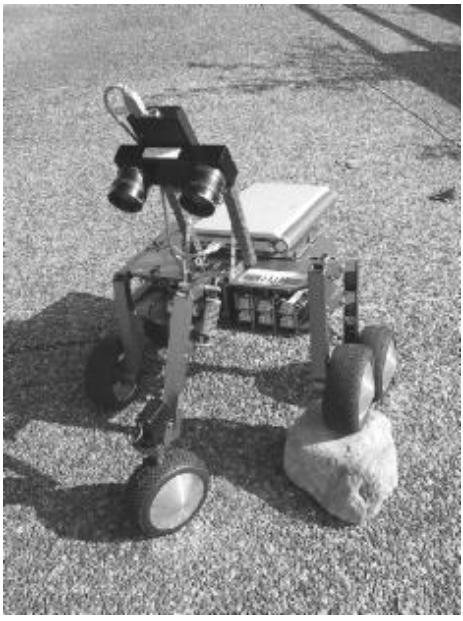


Figure 4.25

The SVM module mounted on EPFL's Shrimp robot.

Many zero crossings do lie at edges in images, but their occurrence is somewhat broader than that. An interesting characteristic of zero crossings is that they are very sharply defined, covering just one "pixel" width in the filtered image. The accuracy can even be further enhanced by using interpolation to establish the position of the zero crossing with subpixel accuracy. All told, the accuracy of the zero crossing features in ZLoG have made them the preferred features in state-of-the-art stereo depth recovery algorithms.

Figure 4.24 shows on an example the various steps required to extract depth information from a stereo image.

Several commercial stereo vision depth recovery sensors have been available for researchers over the past 10 years. A popular unit in mobile robotics today is the digital stereo head (or SVM) from Videre Design shown in figure 4.25.

The SVM uses the LoG operator, following it by tessellating the resulting array into sub- regions within which the sum of absolute values is computed. The correspondence problem is solved at the level of these subregions, a process called *area correlation,* and after cor- respondence is solved the results are interpolated to one-fourth pixel precision. An impor- tant feature of the SVM is that it produces not just a depth map but distinct measures of

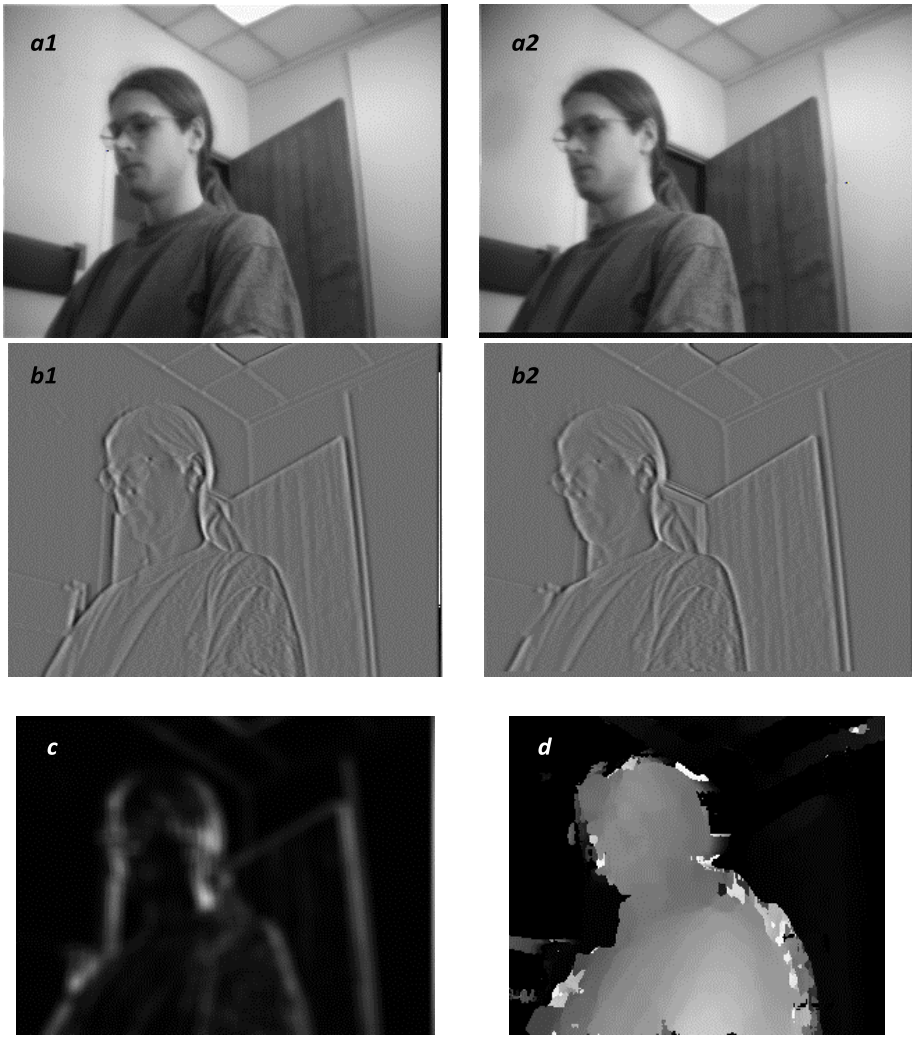


Figure 4.24

Extracting depth information from a stereo image. (a1 and a2) Left and right image. (b1 and b2) Ver- tical edge filtered left and right image: filter = [1 2 4 -2 -10 -2 4 2 1]. (c) Confidence image: bright = high confidence (good texture); dark = low confidence

(no texture). (d) Depth image (dispar- ity): bright = close; dark = far.

match quality for each pixel. This is valuable because such additional information can be used over time to eliminate spurious, incorrect stereo matches that have poor match quality. The performance of SVM provides a good representative of the state of the art in stereo ranging today. The SVM consists of sensor hardware, including two CMOS cameras and DSP (Digital Signal Processor) hardware. In addition, the SVM includes stereo vision soft- ware that makes use of a standard computer (e.g., a Pentium processor). On a 320 x 240 pixel image pair, the SVM assigns one of thirty-two discrete levels of disparity (i.e., depth) to every pixel at a rate of twelve frames per second (based on the speed of a 233 MHz Pen- tium II). This compares favorably to both laser rangefinding and ultrasonics, particularly when one appreciates that ranging information with stereo is being computed for not just one target point, but all target points in the image.

It is important to note that the SVM uses CMOS chips rather than CCD chips, demon- strating that resolution sufficient for stereo vision algorithms is readily available using the less expensive, power efficient CMOS technology.

The resolution of a vision-based ranging system will depend upon the range to the object, as we have stated before. It is instructive to observe the published resolution values for the SVM sensor. Although highly dependent on the camera optics, using a standard 6 mm focal length lens pair, the SVM claims a resolution of 10 mm at 3 m range, and a res- olution of 60 mm at 10 m range. These values are based on ideal circumstances, but never- theless exemplify the rapid loss in resolution that will accompany vision-based ranging.

4.1.8.3 Motion and optical flow

A great deal of information can be recovered by recording time-varying images from a fixed (or moving) camera. First, we distinguish between the motion field and optical flow:

- Motion field: this assigns a velocity vector to every point in an image. If a point in the environment moves with velocity v_0 , then this induces a velocity v_i in the image plane. It is possible to determine mathematically the relationship between v_i and v_0 .
- Optical flow: it can also be true that brightness patterns in the image move as the object that causes them moves (light source). Optical flow is the apparent motion of these brightness patterns.

In our analysis here we assume that the optical flow pattern will correspond to the motion field, although this is not always true in practice. This is illustrated in figure 4.26a where a sphere exhibits spatial variation of brightness, or shading, in the image of the sphere since its surface is curved. If the surface moves, however, this shading pattern will not move hence the optical flow is zero everywhere even though the motion field is not zero. In figure 4.26b, the opposite occurs. Here we have a fixed sphere with a moving light source. The shading in the image will change as the source moves. In this case the optical

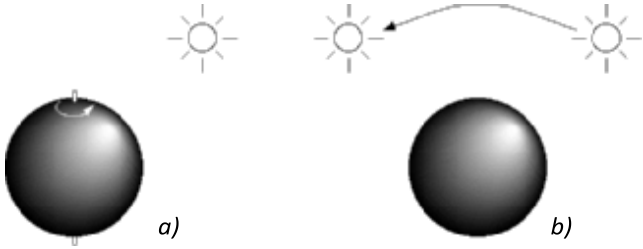


Figure 4.26

Motion of the sphere or the light source here demonstrates that optical flow is not always the same as the motion field.

flow is nonzero but the motion field is zero. If the only information accessible to us is the optical flow and we depend on this, we will obtain incorrect results in both cases.

Optical Flow. There are a number of techniques for attempting to measure optical flow and thereby obtain the scene's motion field. Most algorithms use local information, attempting to find the motion of a local patch in two consecutive images. In some cases, global information regarding smoothness and consistency can help to further disambiguate such *matching* processes. Below we present details for the optical flow constraint equation method. For more details on this and other methods refer to [41, 77, 146].

Suppose first that the time interval between successive snapshots is so fast that we can assume that the measured intensity of a portion of the same object is effectively constant. Mathematically, let $E(x, y, t)$ be the image irradiance at time t at the image point (x, y). If $u(x, y)$ and $v(x, y)$ are the x and

y components of the optical flow vector at that point, we need to search a new image for a point where the irradiance will be the same at time $t + \delta t$, that is, at point $x + \delta t\, y + \delta t$, where $\delta x = u\delta t$ and $\delta y = v\delta t$. That is,

$$E(x + u\delta t, y + v\delta t, t + \delta t) = E(x, y, t) \tag{4.39}$$

for a small time interval, δt . This will capture the motion of a constant-intensity *patch* through time. If we further assume that the brightness of the image varies smoothly, then we can expand the left hand side of equation (4.39) as a Taylor series to obtain

$$E(x, y, t) + \delta x \frac{\partial E}{\partial x} + \delta y \frac{\partial E}{\partial y} + \delta t \frac{\partial E}{\partial t} + e = E(x, y, t) \tag{4.40}$$

where e contains second- and higher-order terms in δx , and so on. In the limit as δt tends to zero we obtain

$$\frac{\partial E}{\partial x}\frac{dx}{dt} + \frac{\partial E}{\partial y}\frac{dy}{dt} + \frac{\partial E}{\partial t} = 0 \tag{4.41}$$

from which we can abbreviate

$$u = \frac{dx}{dt}; \quad v = \frac{dy}{dt} \tag{4.42}$$

and

$$E_x = \frac{\partial E}{\partial x}; \quad E_y = \frac{\partial E}{\partial y}; \quad E_t = \frac{\partial E}{\partial t} \tag{4.43}$$

so that we obtain

$$E_x u + E_y v + E_t = 0 \tag{4.44}$$

The derivative E_t represents how quickly the intensity changes with time while the derivatives E_x and E_y represent the spatial rates of intensity change (how quickly intensity changes across the image). Altogether, equation (4.44) is known as the *optical flow constraint equation* and the three derivatives can be estimated for

each pixel given successive
images.

We need to calculate both *u* and *v* for each pixel, but the optical flow constraint equation only provides one equation per pixel, and so this is insufficient. The ambiguity is intuitively clear when one considers that a number of equal-intensity pixels can be inherently ambig- uous – it may be unclear which pixel is the resulting location for an equal-intensity origi- nating pixel in the prior image.

The solution to this ambiguity requires an additional constraint. We assume that in gen- eral the motion of adjacent pixels will be similar, and that therefore the overall optical flow of all pixels will be smooth. This constraint is interesting in that we know it will be violated to *some* degree, but we enforce the constraint nonetheless in order to make the optical flow computation tractable. Specifically, this constraint will be violated precisely when different objects in the scene are moving in different directions with respect to the vision system. Of course, such situations will tend to include edges, and so this may introduce a useful visual cue.

Because we know that this smoothness constraint will be somewhat incorrect, we can mathematically define the degree to which we violate this constraint by evaluating the for- mula

$$e_S = \iint (u^2 + v^2)\,dx\,dy \tag{4.45}$$

which is the integral of the square of the magnitude of the gradient of the optical flow. We also determine the error in the optical flow constraint equation (which in practice will not quite be zero).

$$e_C = \iint (E_x u + E_y v + E_t)^2\,dx\,dy \tag{4.46}$$

Both of these equations should be as small as possible so we want to minimize $e_S + \alpha e_C$, where α is a parameter that weights the error in the image motion equation relative to the departure from smoothness. A large parameter should be used if the brightness measurements are accurate and small if they are noisy. In practice the parameter α is adjusted manually and interactively to achieve the best performance.

The resulting problem then amounts to the calculus of variations, and the Euler equations yield

$$\nabla^2 u = \alpha^2 (E_x u + E_y v + E_t) E_x \tag{4.47}$$

$$\nabla^2 v = \alpha^2 (E_x u + E_y v + E_t) E_y \tag{4.48}$$

where

$$\underline{\nabla}^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \tag{4.49}$$

which is the Laplacian operator.

Equations (4.47) and (4.48) form a pair of elliptical second-order partial differential equations which can be solved iteratively.

Where silhouettes (one object occluding another) occur, discontinuities in the optical flow will occur. This of course violates the smoothness constraint. One possibility is to try and find edges that are indicative of such occlusions, excluding the pixels near such edges from the optical flow computation so that smoothness is a more realistic assumption. Another possibility is to opportunistically make use of these distinctive edges. In fact, cor- ners can be especially easy to *pattern-match* across subsequent images and thus can serve as fiducial markers for optical flow computation in their own right.

Optical flow promises to be an important ingredient in future vision algorithms that combine cues across multiple algorithms. However, obstacle avoidance and navigation

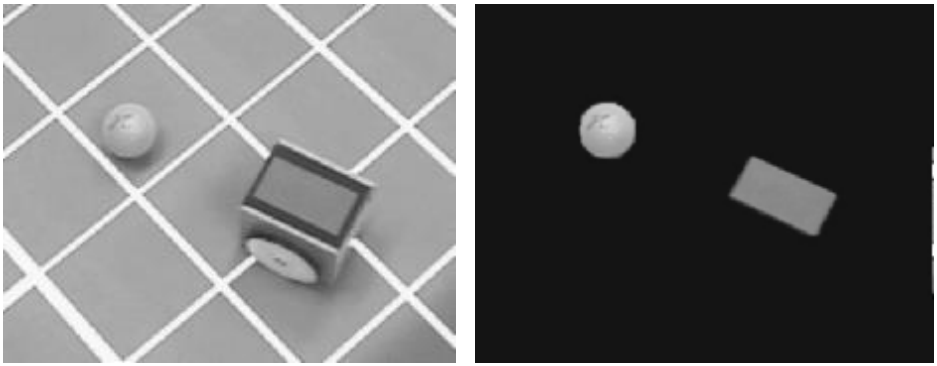


Figure 4.27

Color markers on the top of EPFL's STeam Engine soccer robots enable a color-tracking sensor to locate the robots and the ball in the soccer field.

control systems for mobile robots exclusively using optical flow have not yet proved to be broadly effective.

4.1.8.4 Color-tracking sensors

Although depth from stereo will doubtless prove to be a popular application of vision-based methods to mobile robotics, it mimics the functionality of existing sensors, including ultra- sonic, laser, and optical rangefinders. An important aspect of vision-based sensing is that the vision chip can provide sensing modalities and cues that no other mobile robot sensor provides. One such novel sensing modality is detecting and tracking color in the environment.

Color represents an environmental characteristic that is orthogonal to range, and it rep- resents both a natural cue and an artificial cue that can provide new information to a mobile robot.

For example, the annual robot soccer events make extensive use of color both for environmental marking and for robot localization (see figure 4.27).

Color sensing has two important advantages. First, detection of color is a straightfor- ward function of a single image, therefore no correspondence problem need be solved in such algorithms. Second, because color sensing provides a new, independent environmen- tal cue, if it is combined (i.e., *sensor fusion*) with existing cues, such as data from stereo vision or laser rangefinding, we can expect significant information gains.

Efficient color-tracking sensors are now available commercially. Below, we briefly describe two commercial, hardware-based color-tracking sensors, as well as a publicly available software-based solution.

Cognachrome color-tracking system. The Cognachrome Vision System form Newton Research Labs is a color-tracking hardware-based sensor capable of extremely fast color tracking on a dedicated processor [162]. The system will detect color blobs based on three user-defined colors at a rate of 60 Hz. The Cognachrome system can detect and report on a maximum of twenty-five objects per frame, providing centroid, bounding box, area, aspect ratio, and principal axis orientation information for each object independently.

This sensor uses a technique called *constant thresholding* to identify each color. In *RGB* (red, green and blue) space, the user defines for each of *R , G* , and *B* a minimum and maximum value. The 3D box defined by these six constraints forms a color bounding box, and any pixel with *RGB* values that are all within this bounding box is identified as a target. Target pixels are merged into larger objects that are then reported to the user.

The Cognachrome sensor achieves a position resolution of one pixel for the centroid of each object in a field that is 200 x 250 pixels in size. The key advantage of this sensor, just as with laser rangefinding and ultrasonics, is that there is no load on the mobile robot's main processor due to the sensing modality. All processing is performed on sensor-specific hardware (i.e., a Motorola 68332 processor and a mated framegrabber). The Cognachrome system costs several thousand dollars, but is being superseded by higher-performance hard- ware vision processors at Newton Labs, Inc.

CMUcam robotic vision sensor. Recent advances in chip manufacturing, both in terms of CMOS imaging sensors and high-speed, readily available microprocessors at the 50+ MHz

range, have made it possible to manufacture low-overhead intelligent vision sensors with functionality similar to Cognachrome for a fraction of the cost. The CMUcam sensor is a recent system that mates a low-cost microprocessor with a consumer CMOS imaging chip to yield an intelligent, self-contained vision sensor for $100, as shown in figure 4.29. This sensor is designed to provide high-level information extracted from the camera image to an external processor that may, for example, control a mobile robot. An external processor configures the sensor's streaming data mode, for instance, specifying tracking mode for a bounded *RGB* or *YUV* value set. Then, the vision sensor processes the data in real time and outputs high-level information to the external consumer. At less than 150 mA of current draw, this sensor provides image color statistics and color-tracking services at approximately twenty frames per second at a resolution of 80 x 143 [126].

Figure 4.29 demonstrates the color-based object tracking service as provided by CMUcam once the sensor is trained on a human hand. The approximate shape of the object is extracted as well as its bounding box and approximate center of mass.

CMVision color tracking software library. Because of the rapid speedup of processors in recent times, there has been a trend toward executing basic vision processing on a main



Figure 4.28

The CMUcam sensor consists of three chips: a CMOS imaging chip, a SX28 microprocessor, and a Maxim RS232 level shifter [126].



Figure 4.29

Color-based object extraction as applied to a human hand.

processor within the mobile robot. Intel Corporation's computer

vision library is an opti- mized library for just such processing [160]. In this spirit, the CMVision color-tracking software represents a state-of-the-art software solution for color tracking in dynamic envi- ronments [47]. CMVision can track up to thirty-two colors at 30 Hz on a standard 200 MHz Pentium computer.

The basic algorithm this sensor uses is constant thresholding, as with Cognachrome, with the chief difference that the *YUV* color space is used instead of the *RGB* color space when defining a six-constraint bounding box for each color. While *R*, *G*, and *B* values encode the intensity of each color, *YUV* separates the color (or *chrominance*) measure from the brightness (or *luminosity*) measure. *Y* represents the image's luminosity while *U*

and *V* together capture its chrominance. Thus, a bounding box expressed in *YUV* space can achieve greater stability with respect to changes in illumination than is possible in *RGB* space.

The CMVision color sensor achieves a resolution of 160 x 120 and returns, for each object detected, a bounding box and a centroid. The software for CMVision is available freely with a Gnu Public License at [161].

Key performance bottlenecks for both the CMVision software, the CMUcam hardware system, and the Cognachrome hardware system continue to be the quality of imaging chips and available computational speed. As significant advances are made on these frontiers one can expect packaged vision systems to witness tremendous performance improvements.

4.2 Representing Uncertainty

In section 4.1.2 we presented a terminology for describing the performance characteristics of a sensor. As mentioned there, sensors are imperfect devices with errors of both system- atic and random nature. Random errors, in particular, cannot be corrected, and so they rep- resent atomic levels of sensor uncertainty.

But when you build a mobile robot, you combine information from many sensors, even using the same sensors repeatedly, over time, to possibly build a model of the environment. How can we scale up, from characterizing the uncertainty of a single sensor to the uncer- tainty of the resulting robot system?

We begin by presenting a statistical representation for the random error associated with an individual sensor [12]. With a quantitative tool in hand, the standard Gaussian uncer- tainty model can be presented and evaluated. Finally, we present a

framework for comput- ing the uncertainty of conclusions drawn from a set of quantifiably uncertain measurements, known as the *error propagation law*.

4.2.1 Statistical representation

We have already defined *error* as the difference between a sensor measurement and the true value. From a statistical point of view, we wish to characterize the error of a sensor, not for one specific measurement but for any measurement. Let us formulate the problem of sens- ing as an estimation problem. The sensor has taken a set of n measurements with values $\square_i$. The goal is to characterize the estimate of the true value $E\square X\square$ given these measure-
ments:

$$E\square X\square = g\square\square_1\square \square_2\square \square\square \square_n\square \tag{4.50}$$

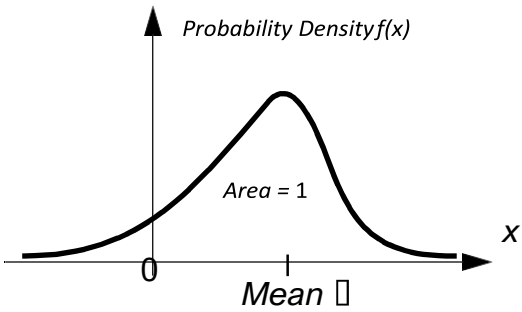


Figure 4.30

A sample probability density function, showing a single probability peak (i.e., unimodal) with asymp- totic drops in both directions.

From this perspective, the true value is represented by a random (and therefore unknown) variable X . We use a *probability density function* to characterize the statistical properties of the value of X .

In figure 4.30, the density function identifies for each possible value x of X a probabil- ity density $f(x)$ along the y -axis. The area under the curve is 1, indicating the complete chance of X having *some* value:

$$\int_{-\infty}^{\infty} f(x)\,dx = 1 \qquad (4.51)$$

The probability of the value of X falling between two limits a and b is computed as the bounded integral:

$$P(a \leq X \leq b) = \int_a^b f(x)dx \tag{4.52}$$

The probability density function is a useful way to characterize the possible values of X because it not only captures the range of X but also the comparative probability of different values for X. Using $f(x)$ we can quantitatively define the mean, variance, and standard deviation as follows.

The *mean value* μ is equivalent to the expected value $E(X)$ if we were to measure X an infinite number of times and average all of the resulting values. We can easily define $E(X)$:

$$\mu = E[X] = \int_{-\infty}^{\infty} xf(x)dx \tag{4.53}$$

Note in the above equation that calculation of $E[X]$ is identical to the weighted average of all possible values of x. In contrast, the *mean square value* is simply the weighted aver- age of the squares of all values of x:

$$E[X^2] = \int_{-\infty}^{\infty} x^2 f(x)dx \tag{4.54}$$

Characterization of the "width" of the possible values of X is a key statistical measure, and this requires first defining the *variance* σ^2:

$$Var[X] = \sigma^2 = \int_{-\infty}^{\infty} (x - \mu)^2 f(x)dx \tag{4.55}$$

Finally, the *standard deviation* σ is simply the square root of variance, and σ^2 will play important roles in our characterization of the error of a single sensor as well as the error of a model generated by combining multiple sensor readings.

4.2.1.1 Independence of random variables.

With the tools presented above, we often evaluate systems with multiple random variables. For instance, a mobile robot's laser

rangefinder may be used to measure the position of a feature on the robot's right and, later, another feature on the robot's left. The position of each feature in the real world may be treated as random variables, X_1 and X_2 .

Two random variables X_1 and X_2 are *independent* if the particular value of one has no

bearing on the particular value of the other. In this case we can draw several important con-

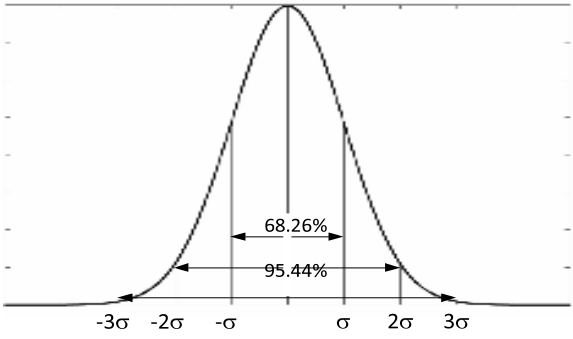
clusions about the statistical behavior of X_1 and X_2 . First, the expected value (or mean value) of the product of random variables is equal to the product of their mean values:

$$E[X_1 X_2] = E[X_1] E[X_2] \tag{4.56}$$

Second, the variance of their sums is equal to the sum of their variances:

$$Var[X_1 + X_2] = Var[X_1] + Var[X_2] \tag{4.57}$$

In mobile robotics, we often assume the independence of random variables even when this assumption is not strictly true. The simplification that results makes a number of the existing mobile robot-mapping and navigation algorithms tenable, as described in



$$f(x) = \frac{1}{\sqrt{2\pi}\,\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

Figure 4.31

The Gaussian function with $\mu = 0$ and $\sigma = 1$. We shall refer to this as the reference Gaussian. The value 2σ is often refereed to as the signal quality; 95.44% of the values are falling within $\pm 2\sigma$.

chapter 5. A further simplification, described in section 4.2.1.2, revolves around one spe- cific probability density function used more often than any other when modeling error: the Gaussian distribution.

4.2.1.2 Gaussian distribution

The Gaussian distribution, also called the *normal distribution,* is used across engineering disciplines when a well-behaved error model is required for a random variable for which no error model of greater felicity has been discovered. The Gaussian has many character- istics that make it mathematically advantageous to other ad hoc probability density func- tions. It is symmetric around the mean μ. There is no particular bias for being larger than or smaller than μ, and this makes sense when there is no

information to the contrary. The Gaussian distribution is also unimodal, with a single peak that reaches a maximum at μ (necessary for any symmetric, unimodal distribution). This distribution also has tails (the value of $f(x)$ as x approaches $-\infty$ and ∞) that only approach zero asymptotically. This means that all amounts of error are possible, although very large errors may be highly improbable. In this sense, the Gaussian is conservative. Finally, as seen in the formula for the Gaussian probability density function, the distribution depends only on two parameters:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[\frac{-(x-\mu)^2}{2\sigma^2}\right] \tag{4.58}$$

The Gaussian's basic shape is determined by the structure of this formula, and so the only two parameters required to fully specify a particular Gaussian are its mean, μ, and its

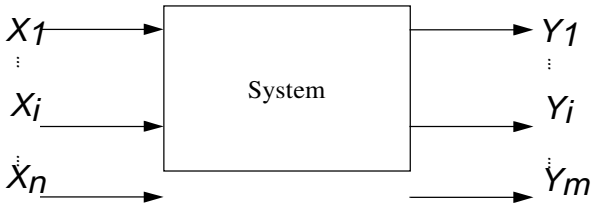


Figure 4.32

Error propagation in a multiple-input multi-output system with *n* inputs and *m* outputs.

standard deviation, σ. Figure 4.31 shows the Gaussian function with $\mu = 0$ and $\sigma = 1$. Suppose that a random variable X is modeled as a Gaussian. How does one identify the chance that the value of X is within one standard deviation of μ? In practice, this requires integration of $f(x)$, the Gaussian function to compute the area under a portion of the curve:

$$Area = \int_{\mu-\sigma}^{\mu+\sigma} f(x)\,dx \tag{4.59}$$

Unfortunately, there is no closed-form solution for the integral in equation (4.59), and so the common technique is to use a Gaussian *cumulative probability table*. Using such a table, one can compute the probability for various value ranges of X:

$P(\mu - \sigma \le X \le \mu + \sigma) = 0.68$;

$P(\mu - 2\sigma \le X \le \mu + 2\sigma) = 0.95$;

$P(\mu - 3\sigma \leq X \leq \mu + 3\sigma) = 0.997$.

For example, 95% of the values for X fall within two standard deviations of its mean. This applies to *any* Gaussian distribution. As is clear from the above progression, under the Gaussian assumption, once bounds are relaxed to 3σ, the overwhelming proportion of values (and, therefore, probability) is subsumed.

4.2.2 Error propagation: combining uncertain measurements

The probability mechanisms above may be used to describe the errors associated with a single sensor's attempts to measure a real-world value. But in mobile robotics, one often uses a series of measurements, all of them uncertain, to extract a single environmental mea- sure. For example, a series of uncertain measurements of single points can be fused to extract the position of a line (e.g., a hallway wall) in the environment (figure 4.36).

Consider the system in figure 4.32, where X_i are n input signals with a known proba- bility distribution and Y_i are m outputs. The question of interest is: what can we say about

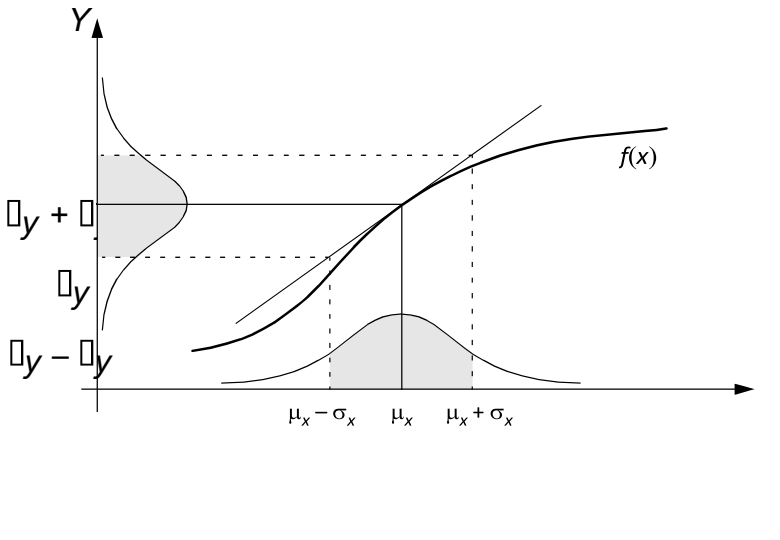


Figure 4.33

One-dimensional case of a nonlinear error propagation problem.

the probability distribution of the output signals Y_i if they depend with known functions f_i upon the input signals? Figure 4.33 depicts the 1D version of this error propagation problem as an example.

The general solution can be generated using the first order Taylor expansion of output covariance matrix C_Y is given by the error propagation law:

$$C_Y = F_X C_X F_X^T \tag{4.60}$$

where

C_X = covariance matrix representing the input uncertainties;

C_Y = covariance matrix representing the propagated uncertainties for the outputs;

F_X is the *Jacobian* matrix defined as

$$F_X = \nabla_x f(X)^T = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}^T = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_1}{\partial x_n} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}. \quad (4.61)$$

This is also the transpose of the gradient of $f(X)$.

We will not present a detailed derivation here but will use equation (4.60) to solve an example problem in section 4.3.1.1.

4.3 Feature Extraction

An autonomous mobile robot must be able to determine its relationship to the environment by making measurements with its sensors and then using those measured signals. A wide variety of sensing technologies are available, as shown in the previous section. But every sensor we have presented is imperfect: measurements always have error and, therefore, uncertainty associated with them. Therefore, sensor inputs must be used in a way that enables the robot to interact with its environment successfully in spite of measurement uncertainty.

There are two strategies for using uncertain sensor input to guide the robot's behavior. One strategy is to use each sensor measurement as a raw and individual value. Such raw sensor values could, for example, be tied directly to robot behavior, whereby the robot's actions are a function of its sensor inputs. Alternatively, the raw sensor values could be used to update an intermediate model, with the robot's actions being triggered as a function of this model rather than the individual sensor measurements.

The second strategy is to extract information from one or more sensor readings first, generating a higher-level *percept* that can then be used to inform the robot's model and per- haps the robot's actions directly. We call this process *feature extraction*, and it is this next, optional step in the perceptual interpretation pipeline (figure 4.34) that we will now discuss. In practical terms, mobile robots do not necessarily use feature extraction and scene interpretation for every activity. Instead,

robots will interpret sensors to varying degrees depending on each specific functionality. For example, in order to guarantee emergency stops in the face of immediate obstacles, the robot may make direct use of raw forward- facing range readings to stop its drive motors. For local obstacle avoidance, raw ranging sensor strikes may be combined in an occupancy grid model, enabling smooth avoidance of obstacles meters away. For map-building and precise navigation, the range sensor values and even vision sensor measurements may pass through the complete perceptual pipeline, being subjected to feature extraction followed by scene interpretation to minimize the impact of individual sensor uncertainty on the robustness of the robot's mapmaking and navigation skills. The pattern that thus emerges is that, as one moves into more sophisti- cated, long-term perceptual tasks, the feature extraction and scene interpretation aspects of the perceptual pipeline become essential.

Feature definition. Features are recognizable structures of elements in the environment. They usually can be extracted from measurements and mathematically described. Good features are always perceivable and easily detectable from the environment. We distinguish

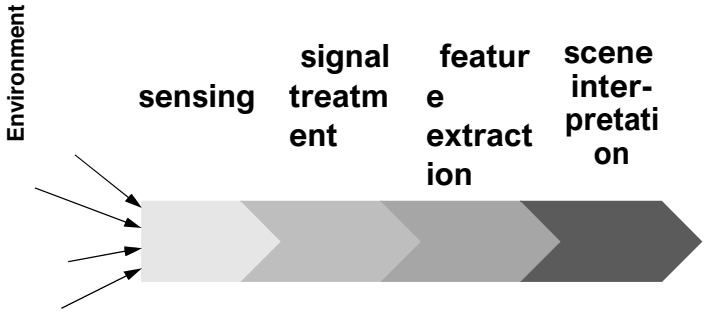


Figure 4.34

The perceptual pipeline: from sensor readings to knowledge models.

between *low-level features* (*geometric primitives*) like lines, circles, or polygons, and *high- level features* (*objects*) such as edges, doors, tables, or a trash can. At one extreme, raw sensor data provide a large volume of data, but with low distinctiveness of each individual quantum of data. Making use of raw data has the potential advantage that every bit of infor- mation is fully used, and thus there is a high conservation of information. Low-level fea- tures are abstractions of raw data, and as such provide a lower volume of data while increasing the distinctiveness of each feature. The hope, when one incorporates low-level features, is that the features are filtering out poor or useless data, but of course it is also likely that some valid information will be lost as a result of the feature extraction process. High-level features provide maximum abstraction from the raw data, thereby reducing the volume of data as much as possible while providing highly distinctive resulting features. Once again, the abstraction process has the risk of filtering away important information,

potentially lowering data utilization.

Although features must have some spatial locality, their geometric extent can range widely. For example, a corner feature inhabits a specific coordinate location in the geomet- ric world. In contrast, a visual “fingerprint” identifying a specific room in an office building applies to the entire room, but has a location that is spatially limited to the one particular room.

In mobile robotics, features play an especially important role in the creation of environ- mental models. They enable more compact and robust descriptions of the environment, helping a mobile robot during both map-building and localization. When designing a mobile robot, a critical decision revolves around choosing the appropriate features for the robot to use. A number of factors are essential to this decision:

Target environment. For geometric features to be useful, the target geometries must be readily detected in the actual environment. For example, line features are extremely useful in office building environments due to the abundance of straight wall segments, while the same features are virtually useless when navigating Mars.

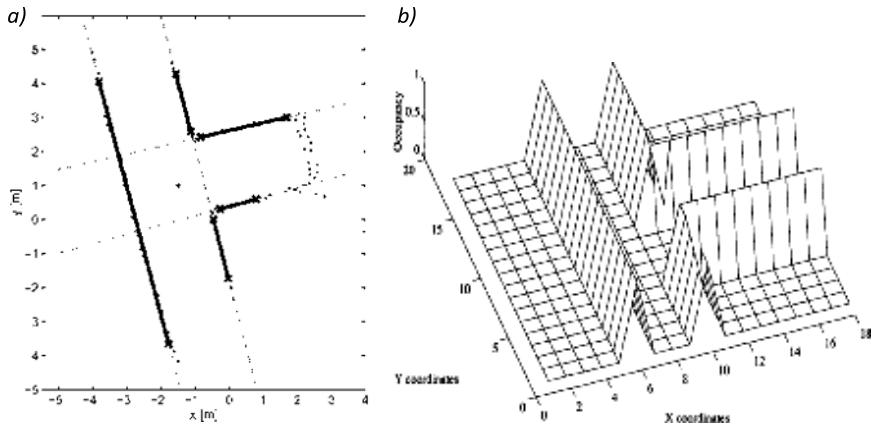


Figure 4.35

Environment representation and modeling: (a) feature based (continuous metric); (b) occupancy grid (discrete metric). Courtesy of Sjur Vestli.

Available sensors. Obviously, the specific sensors and sensor uncertainty of the robot impacts the appropriateness of various features. Armed with a laser rangefinder, a robot is well qualified to use geometrically detailed features such as corner features owing to the high-quality angular and depth resolution of the laser scanner. In contrast, a sonar-equipped robot may not have the appropriate tools for corner feature extraction.

Computational power. Vision-based feature extraction can effect a significant computa- tional cost, particularly in robots where the vision sensor processing is performed by one of the

robot's main processors.

Environment representation. Feature extraction is an important step toward scene inter- pretation, and by this token the features extracted must provide information that is conso- nant with the representation used for the environmental model. For example, nongeometric vision-based features are of little value in purely geometric environmental models but can be of great value in topological models of the environment. Figure 4.35 shows the applica- tion of two different representations to the task of modeling an office building hallway. Each approach has advantages and disadvantages, but extraction of line and corner features has much more relevance to the representation on the left. Refer to chapter 5, section 5.5 for a close look at map representations and their relative trade-offs.

In the following two sections, we present specific feature extraction techniques based on the two most popular sensing modalities of mobile robotics: range sensing and visual appearance-based sensing.

4.3.1 Feature extraction based on range data (laser, ultrasonic, vision-based ranging)

Most of today's features extracted from ranging sensors are geometric primitives such as line segments or circles. The main reason for this is that for most other geometric primitives the parametric description of the features becomes too complex and no closed-form solu- tion exists. Here we describe line extraction in detail, demonstrating how the uncertainty models presented above can be applied to the problem of combining multiple sensor mea- surements. Afterward, we briefly present another very successful feature of indoor mobile robots, the corner feature, and demonstrate how these features can be combined in a single representation.

4.3.1.1 Line extraction

Geometric feature extraction is usually the process of comparing and matching measured sensor data against a predefined description, or template, of the expect feature. Usually, the system is overdetermined in that the number of sensor measurements exceeds the number of feature parameters to be estimated. Since the sensor measurements all have some error, there is no perfectly consistent solution and, instead, the problem is one of optimization. One can, for example, extract the feature that minimizes the discrepancy with all sensor measurements used (e.g,. least-squares estimation).

In this section we present an optimization-based solution to the

problem of extracting a line feature from a set of uncertain sensor measurements. For greater detail than is pre- sented below, refer to [14, pp. 15 and 221].

Probabilistic line extraction from uncertain range sensor data. Our goal is to extract a line feature based on a set of sensor measurements as shown in figure 4.36. There is uncer- tainty associated with each of the noisy range sensor measurements, and so there is no single line that passes through the set. Instead, we wish to select the best possible match, given some optimization criterion.

More formally, suppose n ranging measurement points in polar coordinates $x_i = \square\square_i\square\square_i\square$ are produced by the robot's sensors. We know that there is uncertainty asso- ciated with each measurement, and so we can model each measurement using two random variables $X_i = \square P_i \square Q_i \square$. In this analysis we assume that uncertainty with respect to the actual value of P and Q is independent. Based on equation (4.56) we can state this for- mally:

$$E\square P_i \square P_j\square = E\square P_i \square E\square P_j \square \quad \square i\square j = 1\square\square\square n \tag{4.62}$$

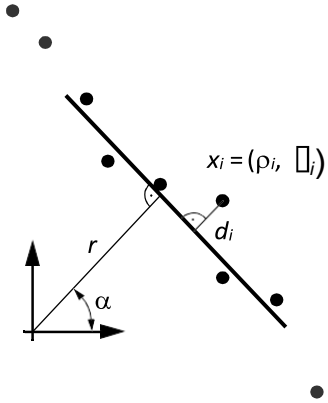


Figure 4.36

Estimating a line in the least-squares sense. The model parameters y (length of the perpendicular) and

 (its angle to the abscissa) uniquely describe a line.

$$E(Q_i - Q_j) = E(Q_i) - E(Q_j) = 1 - 1 = 0 \quad (4.63)$$

$$E(P_i - Q_j) = E(P_i) - E(Q_j) = \rho_i - \rho_j = 1 - 1 = 0 \quad (4.64)$$

Furthermore, we assume that each random variable is subject to a Gaussian probability density curve, with a mean at the true value and with some specified variance:

$$P_2 = N(\quad) \tag{4.65}$$

$$i \qquad i \quad _i$$

$$Q_2 = N(\quad) \tag{4.66}$$

$$i \qquad i \quad _i$$

Given some measurement point ( ), we can calculate the corresponding Euclidean coordinates as $x =$ cos and $y =$ sin . If there were no error, we would want to find a line for which all measurements lie on that line:

$$\begin{aligned} &\cos(\cos + \sin\sin) - r = \cos \\ &( ) - ( ) - r = 0 \end{aligned} \tag{4.67}$$

Of course there is measurement error, and so this quantity will not be zero. When it is nonzero, this is a measure of the error between the measurement point ( ) and the line, specifically in terms of the minimum orthogonal distance between the point and the line. It is always important to understand how the error that shall be minimized is being measured. For example a number of line extraction techniques do not minimize this orthogonal point-

line distance, but instead the distance parallel to the *y*-axis between the point and the line. A good illustration of the variety of optimization criteria is available in [17] where several algorithms for fitting circles and ellipses are presented which minimize algebraic and geo- metric distances.

For each specific $\square\square_i\square\ \square_i\ \square$, we can write the orthogonal distance d_i between $\square\square_i\square\ \square_i\ \square$ and the line as

$$\square_i \cos\square\square_i - \square\square - r = d_i \,. \tag{4.68}$$

If we consider each measurement to be equally uncertain, we can sum the square of all errors together, for all measurement points, to quantify an overall fit between the line and all of the measurements:

$$S = \square d_i^2 = \square\square\square_i \cos\square\square_i - \square\square - r\square^2 \tag{4.69}$$

i i

Our goal is to minimize S when selecting the line parameters $\square\square\square\ r\square$. We can do so by solving the nonlinear equation system

$$\frac{\square S}{\square} = 0 \quad \square = 0 \,. \tag{4.70}$$

$\underline{S}$

$\square\square$

$\square$

r

The above formalism is considered an *unweighted least-squares* solution because no distinction is made from among the measurements. In reality, each sensor measurement may have its own, unique uncertainty based on the geometry of the robot and environment when the measurement was recorded. For example, we know with regard to vision-based stereo ranging that uncertainty and, therefore, variance increases as a square of the distance between the robot and the object. To make use of the variance σ^2 that models the uncer- tainty regarding distance ρ_i of a particular sensor measurement, we compute an individual weight w_i for each measurement using the formula

$$w = 1 / \sigma^2. \tag{4.71}$$

Then, equation (4.69) becomes

2. The issue of determining an adequate weight when σ_i is given (and perhaps some additional information) is complex in general and beyond the scope of this text. See [9] for a careful treatment.

$$S = \sum w_i d_i^2 = \sum w_i (\rho_i \cos(\theta_i - \alpha) - r)^2 . \quad (4.72)$$

It can be shown that the solution to equation (4.70) in the *weighted* least-squares sense[3] is

$$\alpha = \frac{1}{2}\,\mathrm{atan}\left(\frac{\sum_i w_i \rho_i^2 \sin 2\theta_i - \frac{2}{\sum_i w_i} \sum_i \sum_j w_i w_j \rho_i \rho_j \cos\theta_i \sin\theta_j}{\sum_i w_i \rho_i^2 \cos 2\theta_i - \frac{1}{\sum_i w_i} \sum_i \sum_j w_i w_j \rho_i \rho_j \cos(\theta_i + \theta_j)}\right) \quad (4.73)$$

$$r = \frac{\sum_i w_i \rho_i \cos(\theta_i - \alpha)}{\sum_i w_i} \quad (4.74)$$

In practice, equation (4.73) uses the four-quadrant arc tangent (atan2)[4].

Let us demonstrate equations (4.73) and (4.74) with a concrete example. The seventeen measurements (ρ_i, θ_i) in table 4.2 have been taken with a laser range sensor installed on a mobile robot. We assume that the uncertainties of all measurements are equal, uncorrelated, and that the robot was static during the measurement process.

Direct application of the above solution equations yields the line defined by $\alpha = 37.36$ and $r = 0.4$. This line represents the best fit in a least-squares sense and is shown visually in figure 4.37.

Propagation of uncertainty during line extraction. Returning

to the subject of section 4.2.3, we would like to understand how the uncertainties of specific range sensor measure- ments propagate to govern the uncertainty of the extracted line. In other words, how does uncertainty in α_i and β_i propagate in equations (4.73) and (4.74) to affect the uncertainty of α and r?

---

3. We follow here the notation of [14] and distinguish a weighted least-squares problem if C_X is diagonal (input errors are mutually independent) and a generalized least-squares problem if C_X is nondiagonal.

4. Atan2 computes $\tan(x / y)^{-1}$ but uses the signs of both x and y to determine the quadrant in which the resulting angles lies. For example atan 2(–2, –2) = –135°, whereas atan 2(2, 2) = –45°, a dis-
tinction which would be lost with a single-argument arc tangent function.

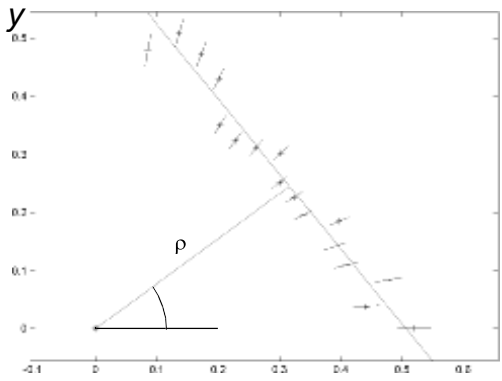


Figure 4.37

Extracted line from laser range measurements (+). The small lines at each measurement point repre- sent the measurement uncertainty that is proportional to the square of the measurement distance.

Table 4.2

Measured values

pointing angle of sensor θ_i [deg]	range ρ_i [m]
0	0.5197
5	0.4404
10	0.4850

15	0.4222
20	0.4132
25	0.4371
30	0.3912
35	0.3949
40	0.3919
45	0.4276
50	0.4075
55	0.3956
60	0.4053
65	0.4752
70	0.5032
75	0.5273
80	0.4879

This requires direct application of equation and representing the (4.60) with A R

random output variables of and r respectively. The goal output is to derive the 2 2 covariance matrix

$$C_{AR} = \begin{bmatrix} \sigma_A^2 & \rho_{AR}\sigma_A\sigma \\ \rho_{AR}\sigma_A\sigma & \sigma^2 \end{bmatrix}, \quad (4.75)$$

given the $2n$ $2n$ input covariance matrix

$$C_X = \begin{bmatrix} C_P & \mathbf{0} \\ \mathbf{0} & C_Q \end{bmatrix} = \begin{bmatrix} diag[\sigma_{P_i}^2] & \mathbf{0} \\ \mathbf{0} & diag[\sigma_{\theta_i}^2] \end{bmatrix} \quad (4.76)$$

and the system relationships [equations (4.73) and (4.74)]. Then by calculating the Jaco- bian,

$$F_{PQ} = \begin{bmatrix} \frac{\partial A}{\partial P_1} & \frac{\partial A}{\partial P_2} & \cdots & \frac{\partial A}{\partial P_n} & \frac{\partial A}{\partial Q_1} & \frac{\partial A}{\partial Q_2} & \cdots & \frac{\partial A}{\partial Q_n} \\ \frac{\partial r}{\partial P_1} & \frac{\partial r}{\partial P_2} & \cdots & \frac{\partial r}{\partial P_n} & \frac{\partial r}{\partial Q_1} & \frac{\partial r}{\partial Q_2} & \cdots & \frac{\partial r}{\partial Q_n} \end{bmatrix} \quad (4.77)$$

we can instantiate the uncertainty propagation equation (4.63) to yield C_{AR} :

$$C_{AR} = F_{PQ} C_{X} F_{PQ} \tag{4.78}$$

Thus we have calculated the probability C_{AR} of the extracted line r based on the probabilities of the measurement points. For more details about this method refer to [6, 37]

4.3.1.2 Segmentation for line extraction

The previous section described how to extract a line feature given a set of range measure- ments. Unfortunately, the feature extraction process is significantly more complex than that. A mobile robot does indeed acquire a set of range measurements, but in general the range measurements are not all part of one line. Rather, only some of the range measure- ments should play a role in line extraction and, further, there may be more than one line

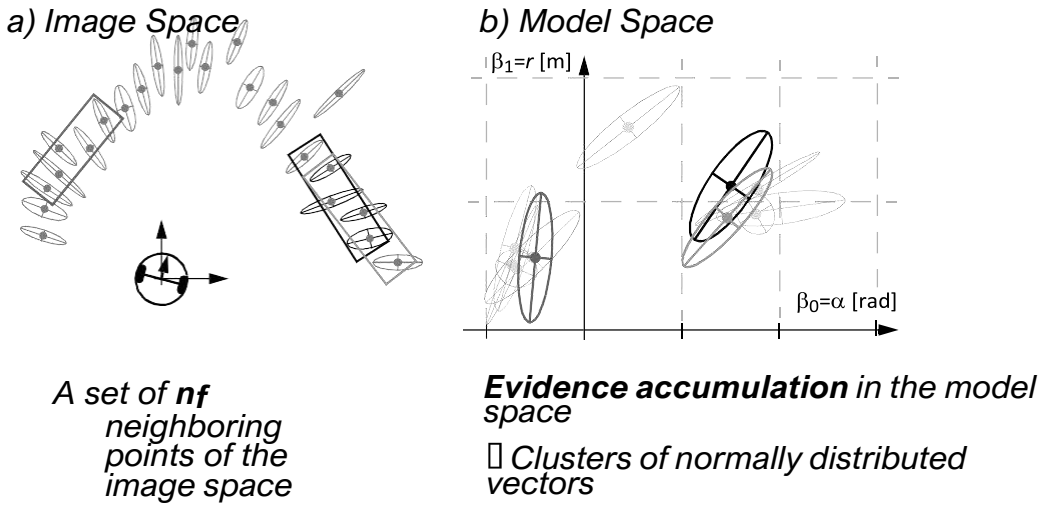


Figure 4.38

Clustering: finding neighboring segments of a common line [37].

feature represented in the measurement set. This more realistic scenario is shown in figure 4.38.

The process of dividing up a set of measurements into subsets that can be interpreted one by one is termed *segmentation* and is an important aspect of both range-based and vision- based perception. A diverse set of techniques exist for segmentation of sensor input in gen- eral. This general problem is beyond the scope of this text and for details concerning seg- mentation algorithms, refer to [91, 131].

For example, one segmentation technique is the merging, or *bottom-up* technique in which smaller features are identified and then merged together based on decision criteria to extract the

goal features. Suppose that the problem of figure 4.38. is solved through merg- ing. First, one may generate a large number of line segments based on adjacent groups of range measurements. The second step would be to identify line segments that have a high probability of belonging to the same extracted light feature. The simplest measure of the closeness of two line segments[5] $x_1 = [\theta_1, r_1]$ and $x_2 = [\theta_2, r_2]$ in the model space is given by Euclidean distance:

$$[x - x]^T[x - x] = [\theta - \theta]^2 + [r - r]^2 \quad (4.79)$$
$$1 \quad 2 \quad 1 \quad 2 \quad 1 \quad 2 \quad 1 \quad 2$$

5. Note: The lines are represented in polar coordinates.

The selection of all line segments x_j that contribute to the same line can now be done in a threshold-based manner according to

$$(x - x)^T (x - x) < d \quad (4.80)$$

where d is a threshold value and x is the representation of the reference line (from a model, average of a group of lines, etc.).

But the approach of equation (4.80) does not take into account the fact that for each mea- surement and therefore for each line segment we have a measure of uncertainty. One can improve upon this equation by selecting line segments that are weighted by their covariance matrix C_j :

$$(x - x)^T (C + C)^{-1} (x - x) < d \quad (4.81)$$

The distance measure of equation (4.81) discriminates the distance of uncertain points in model space considerably more effectively by taking uncertainty into account explicitly.

4.3.1.3 Range histogram features

A histogram is a simple way to combine characteristic elements of an image. An angle his- togram, as presented in figure 4.39, plots the statistics of lines extracted by two adjacent range

measurements. First, a 360-degree scan of the room is taken with the range scanner, and the resulting “hits” are recorded in a map. Then the algorithm measures the relative angle between any two adjacent hits (see figure 4.39b). After compensating for noise in the readings (caused by the inaccuracies in position between adjacent hits), the angle histogram shown in figure 4.39c can be built. The uniform direction of the main walls are clearly vis- ible as peaks in the angle histogram. Detection of peaks yields only two main peaks: one for each pair of parallel walls. This algorithm is very robust with regard to openings in the walls, such as doors and windows, or even cabinets lining the walls.

4.3.1.4 Extracting other geometric features

Line features are of particular value for mobile robots operating in man-made environ- ments, where, for example, building walls and hallway walls are usually straight. In gen- eral, a mobile robot makes use of multiple features simultaneously, comprising a *feature set* that is most appropriate for its operating environment. For indoor mobile robots, the line feature is certainly a member of the optimal feature set.

In addition, other geometric kernels consistently appear throughout the indoor man- made environment. *Corner* features are defined as a point feature with an orientation. *Step discontinuities*, defined as a step change perpendicular to the direction of hallway travel,

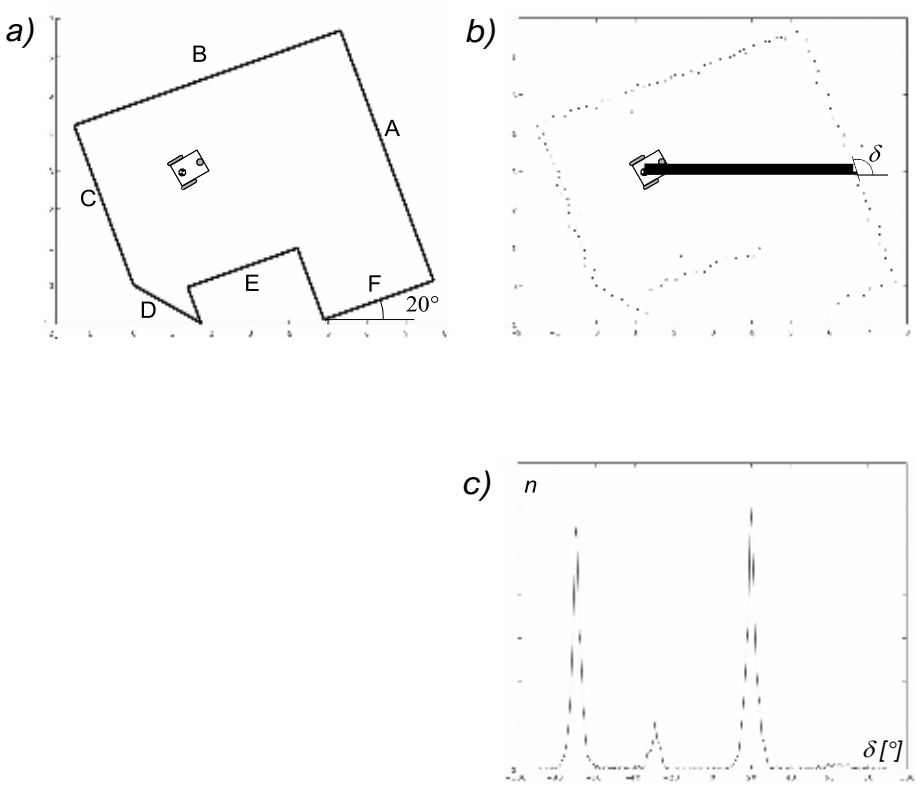


Figure 4.39

Angle histogram [155].

are characterized by their form (convex or concave) and step size. *Doorways*, defined as openings of the appropriate dimensions in walls, are characterized by their width.

Thus, the standard segmentation problem is not so simple as deciding on a mapping from sensor readings to line segments,

but rather it is a process in which features of different types are extracted based on the available sensor measurements. Figure 4.40 shows a model of an indoor hallway environment along with both indentation features (i.e., step disconti- nuities) and doorways.

Note that different feature types can provide quantitatively different information for mobile robot localization. The line feature, for example, provides two degrees of informa- tion, angle and distance. But the step feature provides 2D relative position information as well as angle.

The set of useful geometric features is essentially unbounded, and as sensor perfor- mance improves we can only expect greater success at the feature extraction level. For example, an interesting improvement upon the line feature described above relates to the

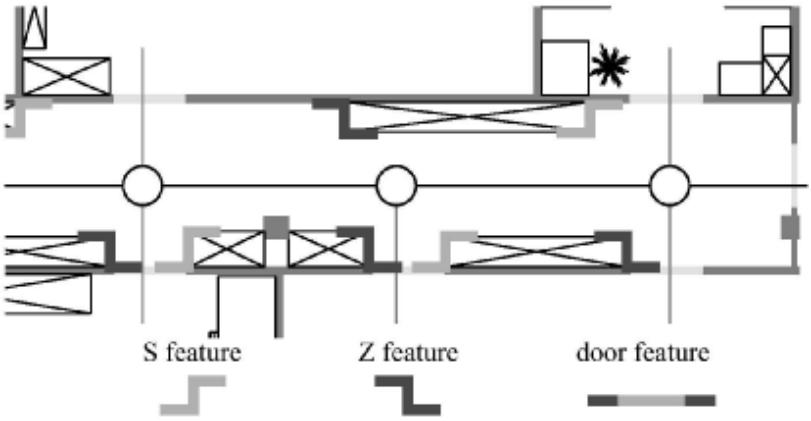


Figure 4.40

Multiple geometric features in a single hallway, including doorways and discontinuities in the width of the hallway.

advent of successful vision-based ranging systems. Because stereo vision provides a full 3D set of range measurements, one can extract plane features in addition to line features from the resulting data set. Plane features are valuable in man-made environments due to the flat walls, floors, and ceilings of our indoor environments. Thus they are promising as another highly informative feature for mobile robots to use for mapping and localization.

4.3.2 Visual appearance based feature extraction

Visual interpretation is, as we have mentioned before, an extremely challenging problem to fully solve. Significant research effort has been dedicated over the past several

decades, to inventing algorithms for understanding a scene based on 2D images and the research efforts have slowly produced fruitful results. Covering the field of computer vision and image processing is, of course, beyond the scope of this book. To explore these disciplines, refer to [18, 29, 159]. An overview on some of the most popular approaches can be seen in figure 4.41.

In section 4.1.8 we have already seen vision-based ranging and color-tracking sensors that are commercially available for mobile robots. These specific vision applications have witnessed commercial solutions primarily because the challenges are in both cases rela- tively well focused and the resulting, problem-specific algorithms are straightforward. But images contain much more than implicit depth information and color blobs. We would like to solve the more general problem of extracting a large number of feature types from images.

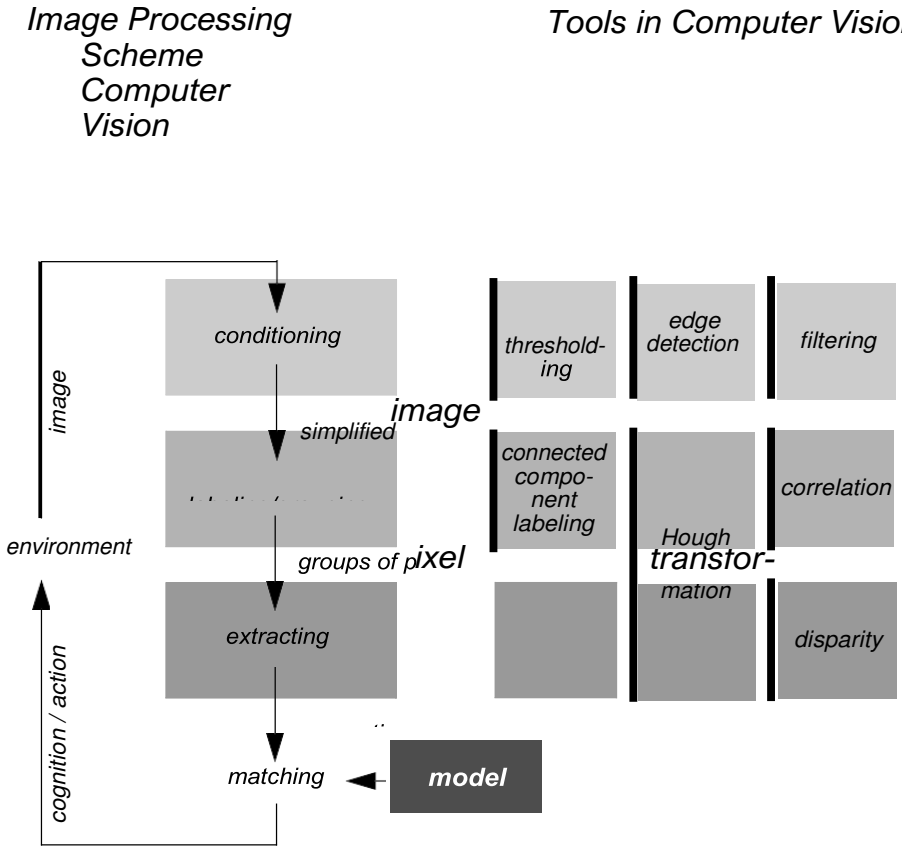


Figure 4.41

Scheme and tools in computer vision. See also [18].

This section presents some appearance-based feature extraction techniques that are rel- evant to mobile robotics along these lines. Two key requirements must be met for a vision-based feature extraction technique to have mobile robotic relevance. First, the method must operate in real time. Mobile robots move through their environment, and so the processing

simply cannot be an off-line operation. Second, the method must be robust to the real-world conditions outside of a laboratory. This means that carefully controlled illumination assumptions and carefully painted objects are unacceptable requirements.

Throughout the following descriptions, keep in mind that vision-based interpretation is primarily about the challenge of *reducing information*. A sonar unit produces perhaps fifty bits of information per second. By contrast, a CCD camera can output 240 *million* bits per second! The sonar produces a tiny amount of information from which we hope to draw broader conclusions. But the CCD chip produces too much information, and this overabun- dance of information mixes together relevant and irrelevant information haphazardly. For example, we may intend to measure the color of a landmark. The CCD camera does not simply report its color, but also measures the general illumination of the environment, the

direction of illumination, the defocusing caused by optics, the side effects imposed by nearby objects with different colors, and so on. Therefore the problem of visual feature extraction is largely one of removing the majority of irrelevant information in an image so that the remaining information unambiguously describes specific features in the environ- ment.

We divide vision-based feature extraction methods into two classes based on their spa- tial extent. *Spatially localized features* are those features found in subregions of one or more images, corresponding to specific locations in the physical world. *Whole-image fea- tures* are those features that are functions of the entire image or set of images, correspond- ing to a large visually connected area in the physical world.

Before continuing it is important to note that all vision-based sensors supply images with such a significant amount of noise that a first step usually consists of "cleaning" the image before launching any feature extraction algorithm. Therefore, we first describe the process of initial image filtering, or preprocessing.

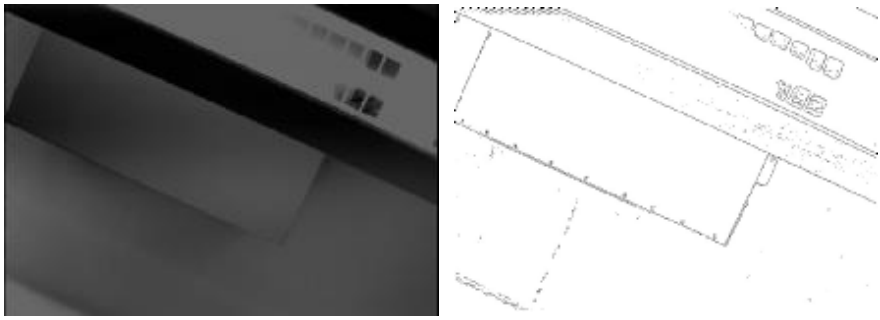
Image preprocessing. Many image-processing algorithms make use of the second deriv- ative of the image intensity. Indeed, the Laplacian of Gaussian method we studied in sec- tion 4.1.8.2 for stereo ranging is such an example. Because of the susceptibility of such high-order derivative algorithms to changes in illumination in the basic signal, it is impor- tant to smooth the signal so that changes in intensity are due to real changes in the luminos- ity of objects in the scene rather than random variations due to imaging noise. A standard approach is convolution with a Gaussian distribution function, as we described earlier in section 4.1.8.2:

$$\hat{I} = G \otimes I \tag{4.82}$$

Of course, when approximated by a discrete kernel, such as a 3 x 3 table, the result is essentially local, weighted averaging:

$$G = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{4.83}$$

Such a low-pass filter effectively removes high-frequency noise, and this in turn causes the first derivative and especially the second derivative of intensity to be far more stable. Because of the importance of gradients and derivatives to image processing, such Gaussian smoothing preprocessing is a popular first step of virtually all computer vision algorithms.



a) b)

Figure 4.42

(a) Photo of a ceiling lamp. (b) Edges computed from (a).

4.3.2.1 Spatially localized features

In the computer vision community many algorithms assume that the object of interest occu- pies only a sub-region of the image, and therefore the features being sought are localized spatially within images of the scene. Local image-processing techniques find features that are local to a subset of pixels, and such local features map to specific locations in the phys- ical world. This makes them particularly applicable to geometric models of the robot's environment.

The single most popular local feature extractor used by the mobile robotics community is the edge detector, and so we begin with a discussion of this classic topic in computer vision. However, mobile robots face the specific mobility challenges of obstacle avoidance and localization. In view of obstacle avoidance, we present vision-based extraction of the floor plane,

enabling a robot to detect all areas that can be safely traversed. Finally, in view of the need for localization we discuss the role of vision-based feature extraction in the detection of robot navigation landmarks.

Edge detection. Figure 4.42 shows an image of a scene containing a part of a ceiling lamp as well as the edges extracted from this image. Edges define regions in the image plane where a *significant* change in the image brightness takes place. As shown in this example, edge detection significantly reduces the amount of information in an image, and is therefore a useful potential feature during image interpretation. The hypothesis is that edge contours in an image correspond to important scene contours. As figure 4.42b shows, this is not entirely true. There is a difference between the output of an edge detector and an ideal line drawing. Typically, there are missing contours, as well as noise contours, that do not cor- respond to anything of significance in the scene.

The basic challenge of edge detection is visualized in figure 4.23. Figure 4.23 (top left) shows the 1D section of an ideal edge. But the signal produced by a camera will look more like figure 4.23 (top right). The location of the edge is still at the same *x* value, but a signif- icant level of high-frequency noise affects the signal quality.

A naive edge detector would simply differentiate, since an edge by definition is located where there are large transitions in intensity. As shown in figure 4.23 (bottom right), dif- ferentiation of the noisy camera signal results in subsidiary peaks that can make edge detec- tion very challenging. A far more stable derivative signal can be generated simply by preprocessing the camera signal using the Gaussian smoothing function described above. Below, we present several popular edge detection algorithms, all of which operate on this same basic principle, that the derivative(s) of intensity, following some form of smoothing, comprises the basic signal from which to extract edge features.

Optimal edge detection Canny. The current reference edge detector throughout the vision community was invented by John Canny in 1983 [30]. This edge detector was born out of a formal approach in which Canny treated edge detection as a signal-processing problem in which there are three explicit goals:

- Maximizing the signal-to-noise ratio;
- Achieving the highest precision possible on the location of edges;
- Minimizing the number of edge responses associated with each edge.

The Canny edge extractor smooths the image *I* via Gaussian convolution and then looks for maxima in the (rectified)

derivative. In practice the smoothing and differentiation are combined into one operation because

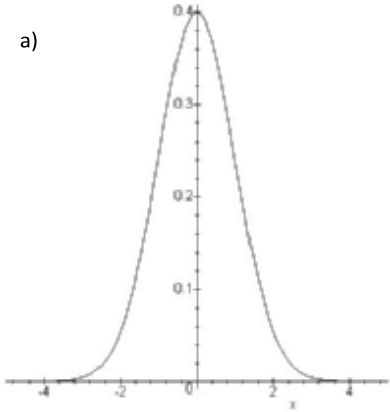
$$\frac{\partial}{\partial x}(G \otimes I) = G' \otimes I \tag{4.84}$$

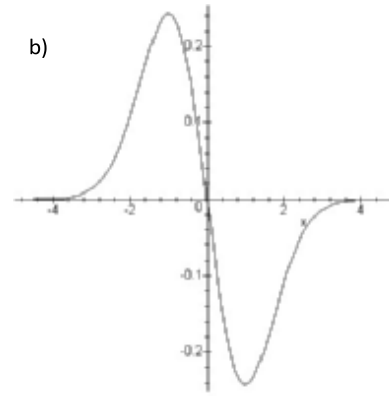
Thus, smoothing the image by convolving with a Gaussian G and then differentiating is equivalent to convolving the image with G', the first derivative of a Gaussian (figure 4.43b).

We wish to detect edges in any direction. Since G' is directional, this requires applica- tion of two perpendicular filters, just as we did for the Laplacian in equation (4.35). We define the two filters as $f_V(x, y) = G'(x)G(y)$ and $f_H(x, y) = G'(y)G(x)$. The result is a basic algorithm for detecting edges at arbitrary orientations:

The algorithm for detecting edge pixels at an arbitrary orientation is as follows:

1. Convolve the image $I(x, y)$ with $f_V(x, y)$ and $f_H(x, y)$ to obtain the gradient compo- nents $R_V(x, y)$ and $R_H(x, y)$, respectively.



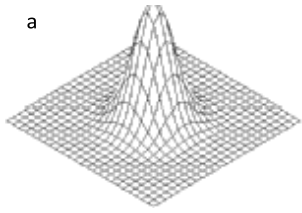


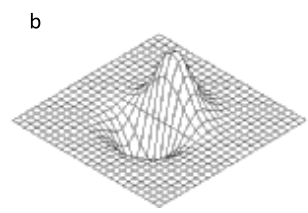
$$G(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{x^2}{2\sigma^2}}$$

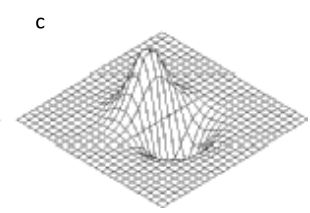
$$G'(x) = \frac{-x}{\sqrt{2\pi\sigma^3}} e^{-\frac{x^2}{2\sigma^2}}$$

Figure 4.43

(a) A Gaussian function. (b) The first derivative of a Gaussian function.







$G(x, y) = G(x)G(y)$

$f_V(x, y) = G'(x)G(y)$

$f_H(x, y) = G'(y)G(x)$

Figure 4.44

(a) Two-dimensional Gaussian function. (b) Vertical filter. (c) Horizontal filter.

2. Define the square of the gradient magnitude $R(x, y) = R_V^2(x, y) + R_H^2(x, y)$.

3. Mark those peaks in $R(x, y)$ that are above some predefined threshold T.

Once edge pixels are extracted, the next step is to construct complete edges. A popular next step in this process is *nonmaxima suppression*. Using edge direction information, the process involves revisiting the gradient value and determining whether or not it is at a local

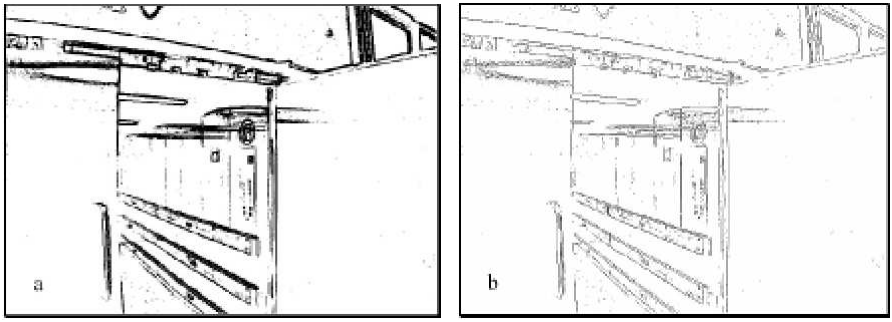


Figure 4.45

(a) Example of an edge image; (b) Nonmaxima suppression of (a).

maximum. If not, then the value is set to zero. This causes only the maxima to be preserved, and thus reduces the thickness of all edges to a single pixel (figure 4.45).

Finally, we are ready to go from edge pixels to complete edges. First, find adjacent (or connected) sets of edges and group them into ordered lists. Second, use thresholding to eliminate the weakest edges.

Gradient edge detectors. On a mobile robot, computation time must be minimized to retain the real-time behavior of the robot. Therefore simpler, discrete kernel operators are commonly used to approximate the behavior of the Canny edge detector. One such early operator was developed by Roberts in 1965 [29]. He used two 2 x 2 masks to calculate the gradient across the edge in two diagonal directions. Let r_1 be the value calculated from the first mask and r_2 from the second mask. Roberts obtained with the gradient magnitude G the equation

$$|G| = \sqrt{r_1^2 + r_2^2}; \quad r_1 = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}; \quad r_2 = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \tag{4.85}$$

Prewitt (1970) [29] used two 3 x 3 masks oriented in the row and column directions. Let p_1 be the value calculated from the first mask and p_2 the value calculated from the second mask. Prewitt obtained the gradient magnitude G and the gradient direction θ taken in a clockwise angle with respect to the column axis shown in the following equation.

$$|G| = \sqrt{p_1^2 + p_2^2};$$

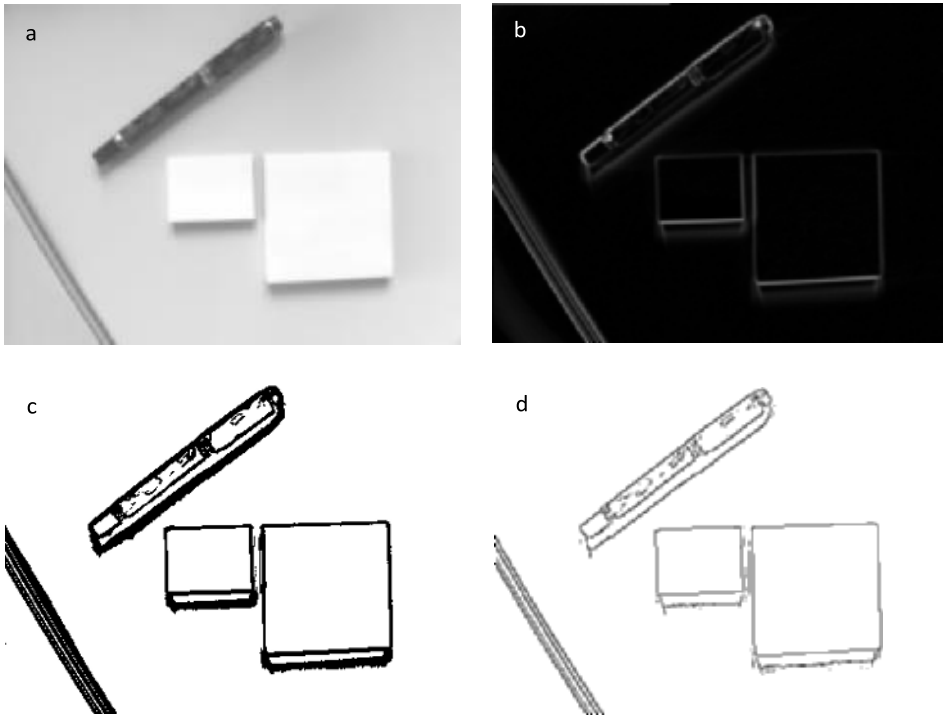


Figure 4.46

Example of vision-based feature extraction with the different processing steps: (a) raw image data;

(b) filtered image using a Sobel filter; (c) thresholding, selection of edge pixels (d) nonmaxima sup- pression.

$$\varphi = \operatorname{atan}\left(\frac{p1}{p2}\right); \quad p1 = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}; \quad p2 = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \tag{4.86}$$

In the same year Sobel [29] used, like Prewitt, two 3 x 3 masks oriented in the row and column direction. Let s_1 be the value calculated from the first mask and s_2 the value cal- culated from the second mask. Sobel obtained the same results as Prewitt for the gradient magnitude G and the gradient direction Φ taken in a clockwise angle with respect to the column axis. Figure 4.46 shows application of the Sobel filter to a visual scene.

$$|G| = \sqrt{s_1^2 + s_2^2};$$

$$\theta = \operatorname{atan}\left(\frac{s_1}{s_2}\right); \quad s_1 = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}; \quad s_2 = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{4.87}$$

Dynamic thresholding. Many image-processing algorithms have generally been tested in laboratory conditions or by using static image databases. Mobile robots, however, operate in dynamic real-world settings where there is no guarantee regarding optimal or even stable illumination. A vision system for mobile robots has to adapt to the changing illumination. Therefore a constant threshold level for edge detection is not suitable. The same scene with different illumination results in edge images with considerable differences. To dynamically adapt the edge detector to the ambient light, a more adaptive threshold is required, and one approach involves calculating that threshold based on a statistical analysis of the image about to be processed.

To do this, a histogram of the gradient magnitudes of the processed image is calculated (figure 4.47). With this simple histogram it is easy to consider only the *n* pixels with the highest gradient magnitude for further calculation steps. The pixels are counted backward starting at the highest magnitude. The gradient magnitude of the point where *n* is reached will be used as the temporary threshold value.

The motivation for this technique is that the *n* pixels with the highest

gradient are expected to be the most relevant ones for the processed image. Furthermore, for each image, the same number of relevant edge pixels is considered, independent of illumination. It is important to pay attention to the fact that the number of pixels in the edge image deliv- ered by the edge detector is not n . Because most detectors use nonmaxima suppression, the number of edge pixels will be further reduced.

Straight edge extraction: Hough transforms. In mobile robotics the straight edge is often extracted as a specific feature. Straight vertical edges, for example, can be used as clues to the location of doorways and hallway intersections. The Hough transform is a simple tool for extracting edges of a particular shape[16, 18]. Here we explain its applica- tion to the problem of extracting straight edges.

Suppose a pixel (x_p, y_p) in the image I is part of an edge. Any straight-line edge including point (x_p, y_p) must satisfy the equation: $y_p = m_1 x_p + b_1$. This equation can only be satisfied with a constrained set of possible values for m_1 and b_1 . In other words, this equa- tion is satisfied only by lines through I that pass through (x_p, y_p) .

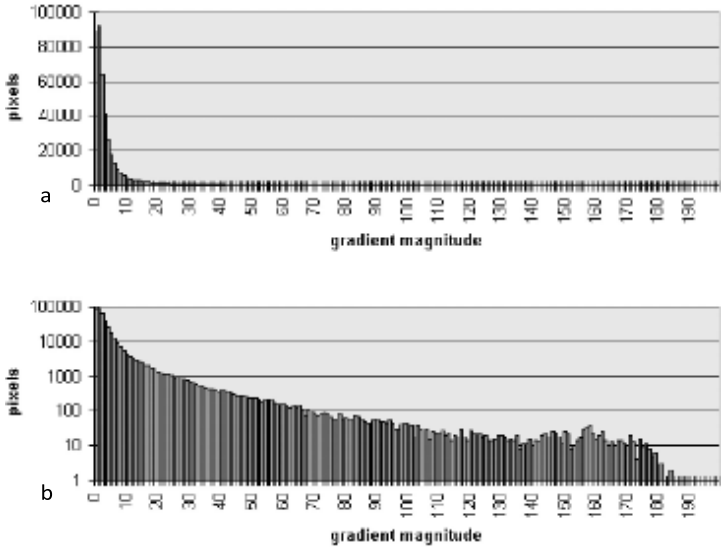


Figure 4.47

(a) Number of pixels with a specific gradient magnitude in the image of figure 4.46(b). (b) Same as (a), but with logarithmic scale

Now consider a second pixel, (x_q, y_q) in I. Any line passing through this second pixel must satisfy the equation: $y_q = m_2 x_q + b_2$. What if $m_1 = m_2$ and $b_1 = b_2$? Then the line defined by both equations is one and the same: it is the line that passes through both (x_p, y_p) and (x_q, y_q).

More generally, for all pixels that are part of a single straight line through I, they must all lie on a line defined by the *same* values for m and b. The general definition of this line is, of course, $y = mx + b$. The Hough transform uses this basic property, creating a mech- anism so that each edge pixel can "vote" for

various values of the (m, b) parameters. The lines with the most votes at the end are straight edge features:

- Create a 2D array A with axes that tessellate the values of m and b.
- Initialize the array to zero: $A(m, b) = 0$ for all values of m, b .
- For each edge pixel (x_p, y_p) in I , loop over all values of m and b : if $y_p = mx_p + b$ then $A(m, b)$+=1 .
- Search the cells in A to identify those with the largest value. Each such cell's indices (m, b) correspond to an extracted straight-line edge in I .

Floor plane extraction. Obstacle avoidance is one of the basic tasks required of most mobile robots. Range-based sensors provide effective means for identifying most types of obstacles facing a mobile robot. In fact, because they directly measure range to objects in the world, range-based sensors such as ultrasonic and laser rangefinders are inherently well suited for the task of obstacle detection. However, each ranging sensor has limitations. Ultrasonics have poor angular resolution and suffer from coherent reflection at shallow angles. Most laser rangefinders are 2D, only detecting obstacles penetrating a specific sensed plane. Stereo vision and depth from focus require the obstacles and floor plane to have texture in order to enable correspondence and blurring respectively.

In addition to each individual shortcoming, range-based obstacle detection systems will have difficulty detecting small or flat objects that are on the ground. For example, a vacuum cleaner may need to avoid large, flat objects, such as paper or money left on the floor. In addition, different types of floor surfaces cannot easily be discriminated by ranging. For example, a sidewalk-following robot will have difficulty discriminating grass from pave- ment using range sensing alone.

Floor plane extraction is a vision-based approach for identifying the traversable portions of the ground. Because it makes use of edges and color in a variety of implementations, such obstacle detection systems can easily detect obstacles in cases that are difficult for tra- ditional ranging devices.

As is the case with all vision-based algorithms, floor plane extraction succeeds only in environments that satisfy several important assumptions:

- Obstacles differ in appearance from the ground.

- The ground is flat and its angle to the camera is known.
- There are no overhanging obstacles.

The first assumption is a requirement in order to discriminate the ground from obstacles using its appearance. A stronger version of this assumption, sometimes invoked, states that the ground is uniform in appearance and different from all obstacles. The second and third assumptions allow floor plane extraction algorithms to estimate the robot's distance to obstacles detected.

Floor plane extraction in artificial environments. In a controlled environment, the floor, walls and obstacles can be designed so that the walls and obstacles appear signifi- cantly different from the floor in a camera image. Shakey, the first autonomous robot devel- oped from 1966 through 1972 at SRI, used vision-based floor plane extraction in a manufactured environment for obstacle detection [115]. Shakey's artificial environment used textureless, homogeneously white floor tiles. Furthermore, the base of each wall was painted with a high-contrast strip of black paint and the edges of all simple polygonal obsta- cles were also painted black.

In Shakey's environment, edges corresponded to nonfloor objects, and so the floor plane extraction algorithm simply consisted of the application of an edge detector to the monochrome camera image. The lowest edges detected in an image corresponded to the closest obstacles, and the direction of straight-line edges extracted from the image provided clues regarding not only the position but also the orientation of walls and polygonal obstacles.

Although this very simple appearance-based obstacle detection system was successful, it should be noted that special care had to be taken at the time to create indirect lighting in the laboratory such that shadows were not cast, as the system would falsely interpret the edges of shadows as obstacles.

Adaptive floor plane extraction. Floor plane extraction has succeeded not only in artifi- cial environments but in real-world mobile robot demonstrations in which a robot avoids both static obstacles such as walls and dynamic obstacles such as passersby, based on seg- mentation of the floor plane at a rate of several hertz. Such floor plane extraction algorithms tend to use edge detection and color detection jointly while making certain assumptions regarding the floor, for example, the floor's maximum texture or approximate color range [78].

Each system based on fixed assumptions regarding the floor's appearance is limited to only those environments satisfying its constraints. A more recent approach is that of adap- tive floor plane extraction, whereby the parameters defining the expected appearance of the floor are allowed to vary over time. In the simplest instance, one can assume that the pixels at the bottom of the image (i.e., closest to the robot) are part of the floor and

contain no obstacles. Then, statistics computed on these "floor sample" pixels can be used to classify the remaining image pixels.

The key challenge in adaptive systems is the choice of what statistics to compute using the "floor sample" pixels. The most popular solution is to construct one or more *histograms* based on the floor sample pixel values. Under "edge detection" above, we found histograms to be useful in determining the best cut point in edge detection thresholding algorithms. Histograms are also useful as discrete representations of distributions. Unlike the Gaussian representation, a histogram can capture multi-modal distributions. Histograms can also be updated very quickly and use very little processor memory. An intensity histogram of the "floor sample" subregion I_f of image I is constructed as follows:

- As preprocessing, smooth I_f , using a Gaussian smoothing operator.
- Initialize a histogram array H with n intensity values: $H(i) = 0$ for $i = 1 \ldots n$.
- For every pixel (x, y) in I_f increment the histogram: $H(I_f(x, y))$ += 1.

The histogram array H serves as a characterization of the appearance of the floor plane. Often, several 1D histograms are constructed, corresponding to intensity, hue, and satura- tion, for example. Classification of each pixel in I as floor plane or obstacle is performed



Figure 4.48

Examples of adaptive floor plane extraction. The trapezoidal polygon identifies the floor sampling region.

by looking at the appropriate histogram counts for the qualities of the target pixel. For example, if the target pixel has a hue that never occurred in the "floor sample," then the corresponding hue histogram will have a count of zero. When a pixel references a histo- gram value below a predefined threshold, that pixel is classified as an obstacle.

Figure 4.48 shows an appearance-based floor plane extraction algorithm operating on both indoor and outdoor images [151]. Note that, unlike the static floor extraction algo- rithm, the adaptive algorithm is able to successfully classify a human shadow due to the adaptive histogram representation. An interesting extension of the work has been to not use the static floor sample assumption, but rather to record visual history and to

use, as the floor sample, only the portion of prior visual images that has successfully rolled under the robot during mobile robot motion.

Appearance-based extraction of the floor plane has been demonstrated on both indoor and outdoor robots for real-time obstacle avoidance with a bandwidth of up to 10 Hz. Applications include robotics lawn mowing, social indoor robots, and automated electric wheelchairs.

4.3.2.2 Whole-image features

A single visual image provides so much information regarding a robot's immediate sur- roundings that an alternative to searching the image for spatially localized features is to make use of the information captured by the entire image to extract a whole-image feature.

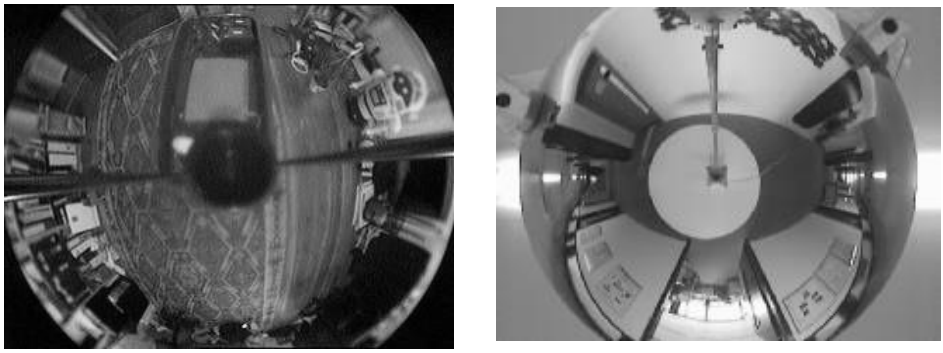


Figure 4.49

Two typical images acquired by the OmniCam catadioptric camera system.

Whole-image features are not designed to identify specific spatial structures such as obsta- cles or the position of specific landmarks. Rather, they serve as compact representations of the entire local region. From the perspective of robot localization, the goal is to extract one or more features from the image that are correlated well with the robot's position. In other words, small changes in robot position should cause only small changes to whole-image features, while large changes in robot position should cause correspondingly large changes to whole-image features.

We present two techniques for whole-image feature extraction below. The first tech- nique is another popular application of the image histogramming approach. The resulting image histogram comprises a set of whole-image features derived directly from the pixel information of an image. The second technique, tiered extraction, covers approaches in which a whole-image feature is

built by first extracting spatially localized features, then composing these features together to form a single metafeature.

Direct extraction: image histograms. Recall that we wish to design whole-image fea- tures that are insensitive to a small amount of robot motion while registering significant changes for large-scale robot motion. A logical first step in designing a vision-based sensor for this purpose is to maximize the field of view of the camera. As the field of view increases, a small-scale structure in the robot's environment occupies a smaller proportion of the image, thereby mitigating the impact of individual scene objects on image character- istics. The catadioptric camera system, now very popular in mobile robotics, offers an extremely wide field of view [114]. This imaging system consists of a high-quality CCD camera mounted, together with customized optics, toward a parabolic mirror. The image provides a 360-degree view of the robot's environment, as shown in figure 4.49.

The catadioptric image is a 360-degree image warped onto a 2D image surface. Because of this, it offers another critical advantage in terms of sensitivity to small-scale robot motion. If the camera is mounted vertically on the robot so that the image represents the environment surrounding the robot (i.e., its horizon), then rotation of the camera and robot simply results in image rotation. In short, the catadioptric camera can be rotationally invari- ant to field of view.

Of course, mobile robot rotation will still change the image; that is, pixel positions will change, although the new image will simply be a rotation of the original image. But we intend to extract image features via histogramming. Because histogramming is a function of the set of pixel values and not the position of each pixel, the process is pixel position invariant. When combined with the catadioptric camera's field of view invariance, we can create a system that is invariant to robot rotation and insensitive to small-scale robot trans- lation.

A color camera's output image generally contains useful information along multiple *bands*: r , g , and b values as well as hue, saturation, and luminance values. The simplest histogram-based extraction strategy is to build separate 1D histograms characterizing each band. Given a color camera image, G , the first step is to create mappings from G to each
of the n available bands. We use G_i to refer to an array storing the values in band i for all pixels in G . Each band-specific histogram H_i is calculated as before:

- As preprocessing, smooth G_i using a Gaussian smoothing operator.
- Initialize H_i with n levels: $H[j] = 0$ for $j = 1 \ldots n$.
- For every pixel (x,y) in G_i , increment the histogram: $H_i[G_i[x, y]]+=1$.

Given the image shown in figure 4.49, the image histogram technique extracts six his- tograms (for each of r , g , b , hue, saturation, and luminance) as shown in figure 4.50. In order to make use of such histograms as whole-image features, we need ways to compare to histograms to quantify the likelihood that the histograms map to nearby robot positions. The problem of defining useful histogram distance metrics is itself an important subfield within the image retrieval field. For an overview refer to [127]. One of the most successful distance metrics encountered in mobile robot localization is the *Jeffrey divergence*. Given two histograms H and K , with h_i and k_i denoting the histogram entries, the Jeffrey diver- gence $d(H, K)$ is defined as

$$d(H, K) = \sum_i \left(h_i \log \frac{2h_i}{h_i + k_i} + k_i \log \frac{2k_i}{h_i + k_i} \right) \quad (4.88)$$

Using measures such as the Jeffrey divergence, mobile robots have used whole-image histogram features to identify their position in real time against a database of previously

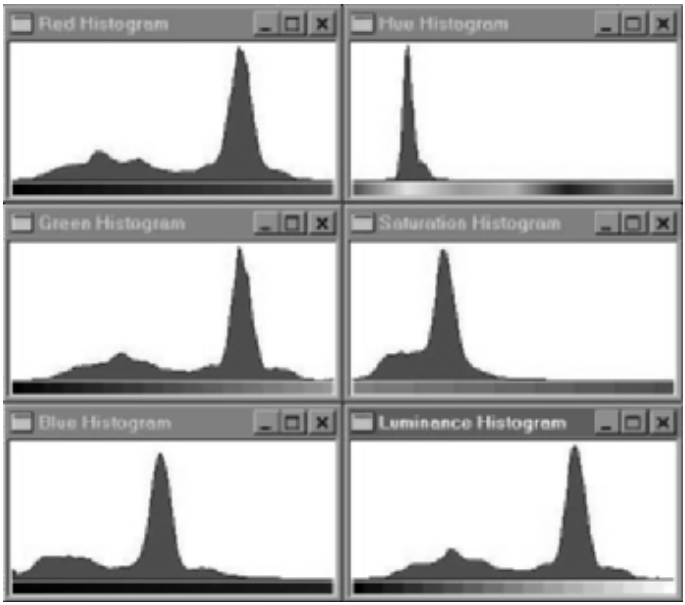


Figure 4.50

Six 1D histograms of the image above. A 5 x 5 smoothing filter was convolved with each band before histogramming.

recorded images of locations in their environment. Using this whole-image extraction approach, a robot can readily recover the particular hallway or particular room in which it is located [152].

Tiered extraction: image fingerprint extraction. An alternative to extracting a whole- image feature directly from pixel values is to use a tiered approach: first identify spatially localized features in the image, then translate from this set of local features to a single metafeature for the whole image. We

describe one particular implementation of this approach, in which the resulting whole-image feature is called the image *fingerprint* [95]. As with other whole-image extraction techniques, because low sensitivity to small robot motions is desired, the system makes use of a 360-degree panoramic image, here con- structed as a mosaic of images captured with a standard CMOS chip camera.

The first extraction tier searches the panoramic image for spatially localized features: vertical edges and sixteen discrete hues of color. The vertical edge detector is a straightfor- ward gradient approach implementing a horizontal difference operator. Vertical edges are “voted upon” by each edge pixel just as in a vertical edge Hough transform. As described

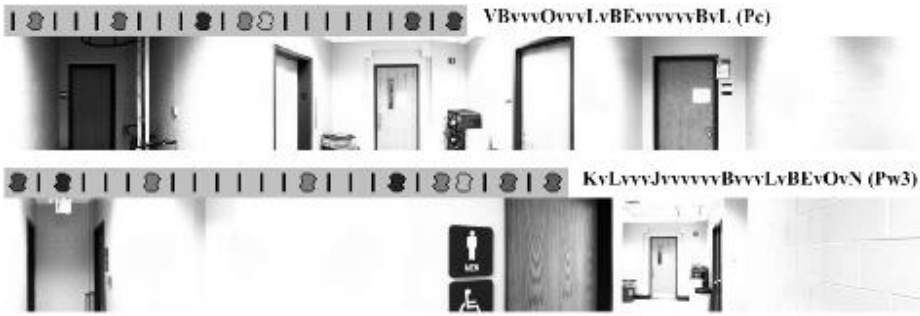


Figure 4.51

Two panoramic images and their associated fingerprint sequences [95].



Figure 4.52

Three actual string sequences. The top two are strings extracted by the robot at the same position [95].

in section 4.3.2.1, an adaptive threshold is used to reduce the number of edges. Suppose the Hough table's tallies for each candidate vertical line have a mean and a standard devia- tion

□ . The chosen threshold is simply □ + □ .

Vertical color bands are identified in largely the same way, identifying statistics over the occurrence of each color, then filtering out all candidate color patches except those with tallies greater than □ + □ . Figure 4.51 shows two sample panoramic images and their asso- ciated fingerprints. Note that each fingerprint is converted to an ASCII string representa- tion.

Just as with histogram distance metrics in the case of image histogramming, we need a quantifiable measure of the distance between two fingerprint strings. String-matching algo- rithms are yet another large field of study, with particularly interesting applications today in the areas of genetics [34]. Note that we may have strings that differ not just in a single element value, but even in their overall length. For example, figure 4.52 depicts three actual

sequences generated using the above algorithm. The top string should match *Place 1*, but note that there are deletions and insertions between the two strings.

The technique used in the fingerprinting approach for string differencing is known as a *minimum energy algorithm*. Taken from the stereo vision community, this optimization- based algorithm will find the minimum energy required to "transform" one sequence into another sequence. The result is a distance metric that is relatively insensitive to the addition or subtraction of individual local features while still able to robustly identify the correct matching string in a variety of circumstances.

It should be clear from the previous two sections that whole-image feature extraction is straightforward with vision-based perception and can be applicable to mobile robot local- ization. But it is spatially localized features that continue to play a dominant role because of their immediate application to the more urgent need for real-time obstacle avoidance.

3. Intelligent Collaborative and Assisted Robots

Cobot

From Wikipedia, the free encyclopedia

Jump to navigation Jump to search



Thanks to sensors, collaborative robots (cobots) are able to interact directly and safely with humans.

A **cobot**[1] or **co-robot**[2] (from **collaborative robot**) is a robot intended to physically interact with humans in a shared workspace.[3] This is in contrast with other robots, designed to operate autonomously or with limited guidance,[4] which is what most industrial robots were up until the decade of the 2010s.

Cobots were invented in 1996 by J. Edward Colgate and Michael Peshkin,[5] professors at Northwestern University. A 1997 US patent filing[6] describes cobots as "an apparatus and method for direct physical interaction between a person and a general purpose manipulator controlled by a computer."

Cobots resulted from a 1994 General Motors initiative led by Prasad Akella of the GM Robotics Center and a 1995 General Motors Foundation research grant intended to find a way to make robots or robot-like equipment safe enough to team with people.[7] The first cobots assured human safety by having no internal source of motive power. Instead, motive power was provided by the human worker.[8] The cobot's function was to allow computer control of motion, by

redirecting or steering a payload, in a cooperative way with the human worker. Later cobots provided limited amounts of motive power as well.[9]

The General Motors team used the term Intelligent Assist Device (IAD) as an alternative to cobot, especially in the context of industrial material handling and automotive assembly operations.[10] A draft safety standard for Intelligent Assist Devices was published in 2002.[11] An updated safety standard was published in 2016.[12]

Cobotics[13] released several cobot models in 2002.[14] [9]

German based industrial robot pioneer KUKA released in 2004 the first cobot LBR 3. This computer controlled lightweight robot was the result of a long collaboration with the German Aerospace Center institute since 1995.[15] KUKA refined the cobotics technology over the KUKA LBR 4 (2008) to the KUKA LBR iiwa (2013).[16] Universal Robots released its first cobot, the UR5, in 2008.[17] In 2012 the UR10 cobot[18] was released, and later a table top cobot, UR3, in 2015. Rethink Robotics released an industrial cobot, Baxter, in 2012[19] and smaller, faster collaborative robot Sawyer in 2015, designed for high precision tasks.

FANUC - the world's largest producer of industrial robots[20] - released its first collaborative robot in 2015 - the FANUC CR-35iA with a heavy 35kg payload.[21] Since that time FANUC has released a smaller line of collaborative robots including the FANUC CR-4iA, CR-7iA and the CR-7/L long arm version.

Cobots can have many roles — from autonomous robots capable of working together with humans in an office environment that can ask you for help,[22] to industrial robots having their protective guards removed. Collaborative industrial robots are highly complex machines which are able to work hand in hand with human beings. The robots support and relieve the human operator in a conjoint work flow.

During revision and restructuring of the standards relevant to industrial robots, the area of collaborative robots has been established. The revised EN ISO 10218 standard Parts 1[23] and 2[24] and the ISO/TS 15066[25] specification define the safety requirements for collaborative robots. In this context, the collaborative

robot also comprises the tools adapted on the robot arm to perform tasks as well as the objects handled by it.

Close cooperation or even direct contact between the operator and the robot inevitably arises the risk of a collision. Consequently, the risk assessment accomplished by the robot manufacturer must also cover the envisioned industrial workplace. EN ISO 10218 Parts 1 and 2 build the basis for the risk assessment, together with the Machinery Directive.

See also[edit]

- Air-Cobot, a collaborative mobile robot to inspect aircraft
- Institute for Occupational Safety and Health of the German Social Accident Insurance (IFA): Safe co-operation between human beings and robots (http://www.dguv.de/ifa%3B/fachinfos/kollaborierende-roboter/index-2.jsp)
- Cobot, manufactured by Universal Robots

Collaborating robots

From OSHWiki
Jump to: navigation, search
Share
2
Vote

▼ Machinery and work equipment
Collaborating robots
Human machine interface
Recommended responsibilities for management of Magnetic Resonance safety
Safety of machinery and work equipment

Michael Huelke, Institute for Occupational Safety and Health of the German Social Accident Insurance (IFA), Germany

Contents

[hide]

- 1 Introduction
- 2 Workplaces with collaborative industrial robots
  - 2.1 Collaborative application design
  - 2.2 Standardization
  - 2.3 Hazard identification and risk assessment
  - 2.4 Requirements for collaborative robot operation
  - 2.5 Safeguarding
  - 2.6 Methods of collaborative working
    - 2.6.1 Safety-rated monitored stop
    - 2.6.2 Hand guiding
    - 2.6.3 Speed and separation monitoring
    - 2.6.4 Power and force limiting
  - 2.7 Organizational requirements, issues of occupational medicine
  - 2.8 Ergonomics requirements, human factors
  - 2.9 Marking and instructions
  - 2.10 Verification and validation
- 3 References
- 4 Links for further reading

Introduction

A growth in the number of manufacturing workplaces at which people work very closely with robots is expected in the coming years [1]. By combining human abilities and dexterity with the precision, strength and speed of the robot, highly efficient production can be achieved while reducing the workload of the persons involved. In the safety strategies applied, occupational safety for the persons involved must be ensured by designing the workplace and particularly the robot

system in conformity with the relevant standards [2] [3] [4]. Typical OSH issues are: Mechanical hazards and Ergonomic hazards.

Workplaces with collaborative industrial robots

This article focuses upon collaborative robots which operate in alongside human operators in the industrial sector. Similar applications exist in the non-industrial sector; these are subject to certain other requirements and will not be addressed here. This article describes the typical hazards associated with collaborative robots and the concepts for accident prevention and health protection. Special robot applications such as those for welding or laser processes require consideration to be given to further hazards; these will not be described here.

Collaborative application design

A collaborative robot shares a common workspace with a human operator in order to carry out, in conjunction with the operator and at the same time, a previously defined task. This collaborative task may have ergonomic benefits for the operator, such as reduced working loads, improved body posture or fewer repetitive movements. The collaboration may be of extended duration, as for example during joint production of assemblies, or be limited to short phases, as for example with the transfer of workpieces to the robot. A range of methods for collaboration exist for different tasks. Common to all is that all necessary protective measures must be active. The robots used must be specially designed for collaborative operation and must comply with the international standards set in the EN ISO 10218 series [2] [3]. Physical contact may arise between persons and collaborative robots and this needs to be managed in order to protect the person.

Standardization

For collaborative industrial robots the European Machinery Directive 2006/42/EC [5] applies. The harmonized two-part EN ISO 10218 series of standards was developed for the specific hazards presented by industrial robots. EN ISO 10218-1:2011 [2] describes how safe robots may be designed and constructed. The integration of a particular robot system into a working environment also influences the safety of the industrial robots during their use. For this purpose, EN ISO 10218-2:2011 [3] provides instructions for protection of the personnel during integration of the robots, their installation, functional testing, programming,

operation, maintenance and repair. In Europe, EN ISO 10218-1/2 superceded European standard EN 775 in 2008. For detailed requirements governing collaborative robots, EN ISO 10218-2 refers to the ISO/TS 15066 specification [4], which is currently under development.

Hazard identification and risk assessment

Robots possess particular characteristics that differentiate them from other machines and which must be addressed by risk assessments:

- They perform movements involving high energy levels and extending over a large operating space.
- It is difficult to determine in advance when a movement will be initiated and what path it will follow. The movements may vary according to operational requirements.
- The robot's operating space may overlap with the working areas of other robots and machinery.
- Persons must work within the operating space of the robot whilst its drive elements are energized.

Where a collaborative robot is used, consideration must be given to the anticipated unobstructed accessibility by persons during collaboration with the robot. This may result in contact between the human operator and the robot. This may be intentional during performance of the work task, or be a result of foreseeable misuse, owing for example to guards having been replaced by other protective measures. It is therefore particularly important that the collaborative task be described and specific hazards identified from the description. Collaboration may be characterized by:

- Frequency and duration of presence of operators or other workers in the collaborative space whilst the robot is energized

- Frequency and duration of contact between operators and the energized robot (e.g. hand guiding, handover of the tool or workpiece)
- Restarting of the robot following violation of the minimum separation distance or following contact
- Switching of the robot between collaborative and autonomous operation when the operator enters or leaves the collaborative space
- Collaboration with several operators simultaneously
- Energise robot arm involuntary movements
- Highest incidence of contention between human and robot is when both are accessing the same object

As with other machines, the hazards must be identified within the boundaries of the robot system, and the risks assessed. This is described in ISO 10218, ISO 12100 and other relevant standards. Mechanical hazards such as clamping, crushing, shear, impact and piercing typically occur (see Annex A of EN ISO 10218-1 [2]).

Suitable measures for risk reduction must then be taken:

1. Design measures for the elimination or reduction of hazards
2. Technical measures for the protection of persons against hazards and unacceptable mechanical effects
3. Provision of supplementary protective measures such as user information, warning instructions, training, personal protective equipment, etc.

The collaborative robot includes the robot itself, the end-effector and the objects moved by it. The end-effector is the tool adapted to the robot arm, with which the robot performs tasks. This gives rise to the possibility of collision with the tool and the workpiece and a risk of crushing against parts of the working area and other

hazards as indicated in Annex A of EN ISO 10218-1 [2]. The risk assessment must therefore also cover the intended industrial workplace and the work task; the robot cannot be adequately assessed in isolation. In addition to the conditions typical of the machine, the risk assessment must consider the following: the characteristics of the robot, the hazards presented by the end-effector and the workpiece, the position of the operator and the path travelled by the robot in relation to the working environment, other machines, parts of building structures, etc.

Requirements for collaborative robot operation

Robots for collaborative operation must satisfy the requirements of EN ISO 10218-1 including functional safety. Functional Safety is achieved when every control based protective measure – the so-called safety function - is carried out and the level of performance required is met. Safety-related parts of the robots control system must satisfy

- EN ISO 13849-1 [6] at Performance Level "d" with Category 3 or
- EN 62061 [7] Safety Integrity Level 2 with a hardware fault tolerance of 1.

This particularly applies to the safety functions described below for the various collaboration methods.

Hazards arising during collaboration can be avoided by reduction of interaction and of the collaborative space, to the extent reasonable in consideration of the task. Any unnecessary approaching of the robot must be avoided. Persons not involved in the task should be prevented by guards (e.g. fixed or interlocking guards, trip devices, fences) from accessing the collaborative space. Entry by a person into the guarded area around the robot beyond the collaborative space must cause immediate stoppage of the robot.

The collaborative space must be clearly defined and demarcated. In order to provide sufficient space for the robot to move, adequate clearance must be ensured and suitable protective equipment provided in order to prevent crushing against parts of building structures or machinery. The alternation between autonomous and collaborative operation is particularly critical for operating

personnel, because the robots movements may change without an intermediate stop.

Safeguarding

During collaborative operation, operating personnel are protected by a safety function or a combination of several safety functions. The operator must be able to stop the robot movement or to end collaborative operation by a single action. Protective measures must be available to all operating personnel within the collaborative space. The following safety functions are typical:

- Stop functions: each collaborative workplace must be equipped with an emergency-stop and a protective-stop function for the robot.
- Enabling device: a robot movement occurs only following an enabling action by the operator (pushing a button for example during hand guiding).
- Limited collaborative speed: a maximum permissible robot speed is determined by the risk assessment. This limit value is stored in the control system, either as a fixed value or as a variable dependent upon the distance from the operator, and is monitored continually.
- Minimum separation distance: a sensor system continually detects the position of the operator and his or her distance to the robot. The required safe distance is typically determined dynamically from the relative speeds of the operator and the robot, the response times of the brake and control system, and the accuracy of measurement of the sensor system or the robot. If this safe distance is violated, a protective stop occurs.
- Limited force/pressure: a sensor system on or in the robot detects a contact between the operator and the robot and stops in order avoid exceeding the allowable force or

pressure. A protective stop is then initiated and possibly reversal of the robot movement to enable the operator to free him or herself.

Methods of collaborative working

For safe collaborative operation, one or more of the following methods and safety functions must be selected as appropriate. Whenever a failure of the safety functions is detected, a safe stop must be initiated. Autonomous operation of the robot following a stop may be resumed only following a deliberate restart executed by the operator from outside the collaborative space.

Safety-rated monitored stop

With this method, simultaneous movement of the operator and the robot is not permissible. The robot functions autonomously until a person enters the collaborative space. The robot must then perform a stop. The stop is monitored by the control system in order to detect control failures. The operator can now work directly at the robot, and for example chuck a workpiece. As soon as the operator leaves the space, the robot can continue its autonomous movement. When the robot is working outside the collaborative space whilst the operator is within it, the robot must not penetrate this space. Movement into this space must again cause the robot to be halted with a stop at the boundary of the collaborative space.

Hand guiding

With this method, the operator and the robot can move simultaneously within the collaborative space, and also work closely together. This may be the method selected when performing some maintenance tasks. The robot however must not move autonomously, and must instead be guided manually by the operator. Hand-guided operation of the robot may be started when the robot is in the safety-rated monitored stop status at a transfer position. In order to position the robot manually, the operator must have a guidance facility (push buttons, joystick) with an emergency-stop feature. The operator must have a clear view of the collaborative space and of the robot's movements. Safe limits are imposed upon the robot's speed and position. The position and posture of the operator and the guidance facility itself must not give rise to additional hazards. As soon as

the guidance facility is released by the operator, the robot must stop again. As soon as the operator leaves the space, the robot can continue its autonomous movement.

Speed and separation monitoring

With this method, the operator and the robot can move simultaneously and autonomously within the collaborative space but cannot work closely together. The risk is reduced by a permanent safe distance between the operator and the robot. This safe distance can be determined dynamically from the instantaneous speed of the robot. At low speeds, the safe distance may be reduced. The maximum robot speed, the minimum distance and other parameters must be determined by the risk assessment. If the safe distance is violated (too low), the robot comes to a halt. As soon as the operator moves away from the robot, the robot can once again move autonomously, with assurance of the minimum separation distance. The safe distance can also be assured by adoption by the robot of an alternative path by which it avoids the operator in the collaborative space. This safety function is based upon the position of all persons within the collaborative space being reliably detected.

Power and force limiting

With this method, the operator and the robot can move simultaneously and autonomously within the collaborative space and work so closely together that they may come into contact with each other. This risk is reduced by limitation of the force and power of the robot system in the event of contact occurring. The limit values for allowable force and pressure are determined and verified by a risk assessment performed for the specific workplace in question.

This method is limited to robots of special design employing safe materials, soft surfaces, cushioning and the absence of sharp edges. The reason is that it mostly needs a contact to trigger the protective-stop function. Therefore, all robot movements should be logical and predictable for the operator in order to reduce the probability of unintentional contact.

Biomechanical requirements

The risk assessment determines the body regions (head, trunk, extremities) described in ISO/TS 15066 [4] which may come into contact with the robot during intended collaboration. Such contact may result only in strain upon the skin and the underlying connective and muscle tissue which does not lead to deeper penetration of the skin and the tissue with bleeding trauma or to fractures or other permanent damage to the skeletal system. In addition, no injuries may occur exceeding injury severity Category 1 of the Abbreviated Injury Scale [8] (AIS1) and the injury severities with the codes for superficial injuries of ICD-10-GM 2006] [9]. Should a risk of injury to the eyes, ears, nose and mouth (sensory organs) exist during use, special protective measures (such as goggles) must be taken to reduce it.

The injury severity is adequately addressed with respect to all individual regions of the body by the following criteria for a range of mechanical hazards (impact, clamping, etc.):

- Clamping/Squeezing force (unit: N)
- Impact force (unit: N)
- Pressure per unit area (unit: N/cm²)

The tolerable range of injury severity for a given individual region of the body is not exceeded when the limit values for the injury criteria are observed. A single, uniform limit value does not exist; instead, ISO/TS 15066 contains a table of limit values for a total of 15 individual body regions, with differing limit values for pressure and force during trapping/crushing for the contact area between the robot and the person.

Within standardization, the use of a two-stage risk concept employing two limit values is also under discussion. The limit for the tolerable injury severity described above relates to undesirable, very infrequent collisions which must not occur during collaboration as intended. In the event of foreseeable misuse of the robot systems, such collisions are quite possible, owing to the absence of guards.

For the contact which may occur more frequently during the intended collaboration, the limit for the acceptable mechanical impact must be set substantially lower: to the so-called pain level. The pain level in this context is defined as the transition from the perception of pressure to the perception of pain during contact. This limit must also be reviewed at each individual workplace by

means of a suitable force and pressure measurement system that resembles the human body in its characteristics.

Organizational requirements, issues of occupational medicine

The suitability, in terms of their health, of persons who work with a collaborative robot and are exposed to a risk of collision should be determined at suitable regular intervals. These persons must be provided with regular instruction on the risks and emergencies and the safety measures which must be taken. This applies in particular to installation, assembly and test work, to set-up mode and during commissioning. The particular underlying conditions for the organization of workplaces involving collaborative robots (such as working hours, breaks, first-aid kits, log books, etc.) must be examined and defined. Following an unintended collision, the fitness for work of the affected individual and the correct setup of the workplace must be examined.

Ergonomics requirements, human factors

The robot system including the human machine interface and its control and display elements must be designed in accordance with ergonomics principles [10], in order to ensure that its operation and maintenance are straightforward. The working space in which a collision between a person and a collaborative robot may occur must be designed such that the physical mobility of the person is not restricted. The individual's perception, attention and thought processes must not be constrained or disturbed by the working environment and the collaborative robot . The beginning of the robot's movement and its course should be visible, predictable and logical. These characteristics assist in preventing contact between the operator and the robot and possible stress upon or loss of control by the operator.

Marking and instructions

Robots for collaborative operation and points of access to collaborative areas must be marked by a suitable warning sign and symbol. The collaborative space in which the operator works directly with the robot must be clearly set out and demarcated (for example by markings on the ground, signs, etc.). The instruction handbooks for machinery must also describe the collaborative aspect:

- Design and use procedures: type of collaborative operation, detailed instructions, maintenance, abnormal situations, system limitations, cautions/warnings, PPE
- Criteria for operator capabilities: skills, training, physical limitations
- Operators must be informed regularly of the risks, procedures in emergencies and required safety measures associated with collaborative operation

Verification and validation

For the "speed and separation monitoring" method, the safe distance, braking distance of the robot and response time of the control system must be verified.

For the "power and force limiting" method, the static and dynamic injury criteria must be validated. For this purpose, a suitable force and pressure measurement system modelling the human body must be employed. Further details are formulated in the annexes of the ISO/TS 15066 specification in draft [3].

References

1. Jump up ↑ Daimler AG (2012). Innovative Cooperation between Workers and Robots at Mercedes-Benz. Retrieved 01 March 2013 from: [1]
2. ↑ Jump up to: 2.0 2.1 2.2 2.3 2.4 ISO 10218-1:2011 Robots and robotic devices - Safety requirements for industrial robots - Part 1: Robots. International Organization for Standardization. Available at: [2]
3. ↑ Jump up to: 3.0 3.1 3.2 ISO 10218-2:2011 Robots and robotic devices - Safety requirements for industrial robots - Part 2: Robot systems and

integration. International Organization for Standardization. Available at: [3]
4. ↑ Jump up to: 4.0 4.1 4.2 Robots and robotic devices - industrial Safety requirements Collaborative industrial robots (ISO/TS 15066 in draft). International Organization for Standardization. Not yet publically available
5. Jump up ↑ Directive 2006/42/EC of the European Parliament and The Council of 17 May 2006 on machinery, and amending Directive 95/16/EC (recast). OJ L 157/21, 9.6.2006. Available at: [4]
6. Jump up ↑ ISO 13849-1:2006 Safety of machinery - Safety-related parts of control systems - Part 1: General principles for design. International Organization for Standardization. Available at: [5]
7. Jump up ↑ IEC 62061:2012 Safety of machinery - Functional safety of safety-related electrical, electronic and programmable electronic control systems. International Electrotechnical Commission. Available at: [6]
8. Jump up ↑ Wikimedia Foundation - (2013). Abbreviated Injury Scale - Wikipedia. Retrieved 01 March 2013, from: [7]
9. Jump up ↑ Wikimedia Foundation - (2013). International Classification of Diseases and Related Health Problems - Wikipedia. Retrieved 01 March 2013, from: [8]
10. Jump up ↑ ISO 26800:2011 Ergonomics - General approach, principles and concepts. International Organization for Standardization. Available at: [9]

Links for further reading

HSE - Health and Safety Executive, Human Science Group, HSE report RR906 - Collision and injury criteria when working with collaborative robots, Derbyshire, 2012. Available at: [10]

IFA - Institut für Arbeitsschutz der DGUV (2013). Collaborative Robots. Retrieved 01 September 2014, from: [11]

DGUV - Deutsche Gesetzliche Unfallversicherung, BG/BGIA risk assessment recommendations according to machinery directive - Design of workplaces with collaborative robots (edition 2011), DGUV, Sankt Augustin, 2011. Available at: [12]

4. Research Overview

5 Mobile Robot Localization

5.1 Introduction

Navigation is one of the most challenging competences required of a mobile robot. Success in navigation requires success at the four building blocks of navigation: *perception,* the robot must interpret its sensors to extract meaningful data; *localization,* the robot must determine its position in the environment (figure 5.1); *cognition,* the robot must decide how to act to achieve its goals; and *motion control,* the robot must modulate its motor outputs to achieve the desired trajectory.

Of these four components (figure 5.2), localization has received the greatest research attention in the past decade and, as a result, significant advances have been made on this front. In this chapter, we explore the successful localization methodologies of recent years. First, section 5.2 describes how sensor and effector uncertainty is responsible for the diffi- culties of localization. Then, section 5.3 describes two extreme approaches to dealing with the challenge of robot localization: avoiding localization altogether, and performing explicit map-based localization. The remainder of the chapter discusses the question of rep-

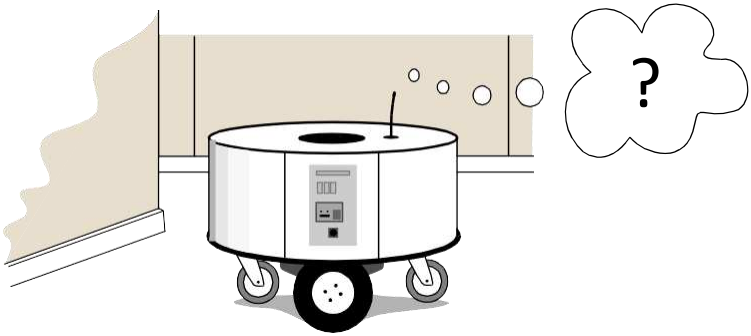


Figure 5.1

Where am I?

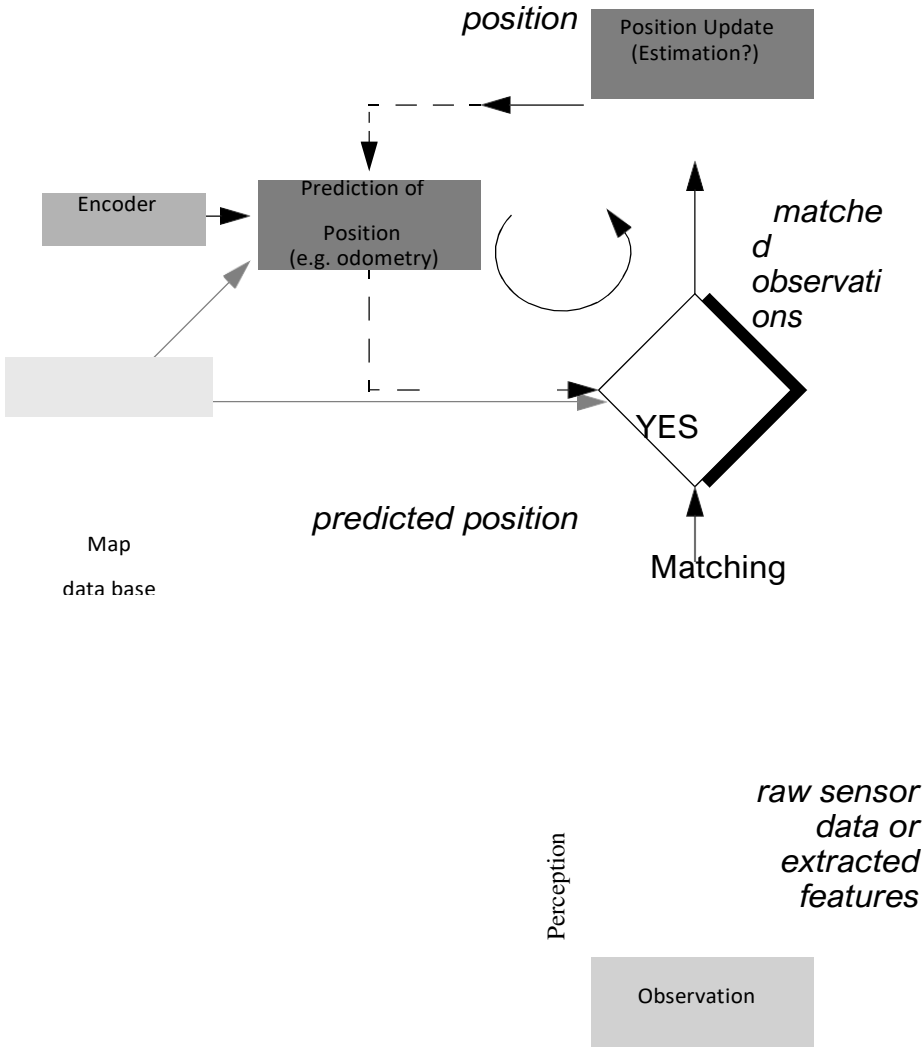


Figure 5.2

General schematic for mobile robot localization.

resentation, then presents case studies of successful localization systems using a variety of representations and techniques to achieve mobile robot localization competence.

5.2 The Challenge of Localization: Noise and Aliasing

If one could attach an accurate GPS (global positioning system) sensor to a mobile robot, much of the localization problem would be obviated. The GPS would inform the robot of its exact position, indoors and outdoors, so that the answer to the question, "Where am I?", would always be immediately available. Unfortunately, such a sensor is not currently prac- tical. The existing GPS network provides accuracy to within several meters, which is unac- ceptable for localizing human-scale mobile robots as well as miniature mobile robots such as desk robots and the body-navigating nanorobots of the future. Furthermore, GPS technologies cannot function indoors or in obstructed areas and are thus limited in their work- space.

But, looking beyond the limitations of GPS, localization implies more than knowing one's absolute position in the Earth's reference frame. Consider a robot that is interacting with humans. This robot may need to identify its absolute position, but its relative position

with respect to target humans is equally important. Its localization task can include identi- fying humans using its sensor array, then computing its relative position to the humans. Furthermore, during the *cognition* step a robot will select a strategy for achieving its goals. If it intends to reach a particular location, then localization may not be enough. The robot may need to acquire or build an environmental model, a *map*, that aids it in planning a path to the goal. Once again, localization means more than simply determining an absolute pose in space; it means building a map, then identifying the robot's position relative to that map. Clearly, the robot's sensors and effectors play an integral role in all the above forms of localization. It is because of the inaccuracy and incompleteness of these sensors and effec- tors that localization poses difficult challenges. This section identifies important aspects of this sensor and effector suboptimality.

5.2.1 Sensor noise

Sensors are the fundamental robot input for the process of *perception*, and therefore the degree to which sensors can discriminate the world state is critical. *Sensor noise* induces a limitation on the consistency of sensor readings in the same environmental state and, there- fore, on the number of useful bits available from each sensor reading. Often, the source of sensor noise problems is that some environmental features are not captured by the robot's representation and are thus overlooked.

For example, a vision system used for indoor navigation in an office building may use the color values detected by its color CCD camera. When the sun is hidden by clouds, the illumination of the building's interior changes because of the windows throughout the building. As a result, hue values are not constant.

The color CCD appears noisy from the robot's perspective as if subject to random error, and the hue values obtained from the CCD camera will be unusable, unless the robot is able to note the position of the sun and clouds in its representation.

Illumination dependence is only one example of the apparent noise in a vision-based sensor system. Picture jitter, signal gain, blooming, and blurring are all additional sources of noise, potentially reducing the useful content of a color video image.

Consider the noise level (i.e., apparent random error) of ultrasonic range-measuring sen- sors (e.g., sonars) as discussed in section 4.1.2.3. When a sonar transducer emits sound toward a relatively smooth and angled surface, much of the signal will coherently reflect away, failing to generate a return echo. Depending on the material characteristics, a small amount of energy may return nonetheless. When this level is close to the gain threshold of the sonar sensor, then the sonar will, at times, succeed and, at other times, fail to detect the object. From the robot's perspective, a virtually unchanged environmental state will result in two different possible sonar readings: one short and one long.

The poor signal-to-noise ratio of a sonar sensor is further confounded by interference between multiple sonar emitters. Often, research robots have between twelve and forty-

eight sonars on a single platform. In acoustically reflective environments, multipath inter- ference is possible between the sonar emissions of one transducer and the echo detection circuitry of another transducer. The result can be dramatically large errors (i.e., underesti- mation) in ranging values due to a set of coincidental angles. Such errors occur rarely, less than 1% of the time, and are virtually random from the robot's perspective.

In conclusion, sensor noise reduces the useful information content of sensor readings. Clearly, the solution is to take multiple readings into account, employing temporal fusion or multisensor fusion to increase the overall information content of the robot's inputs.

5.2.2 Sensor aliasing

A second shortcoming of mobile robot sensors causes them to yield little information con- tent, further exacerbating the problem of perception and, thus, localization. The problem, known as *sensor aliasing*, is a phenomenon that humans rarely encounter. The human sen- sory system, particularly the visual system, tends to receive unique inputs in each unique local state. In other words, every different place looks different. The power of this unique mapping is only apparent when one considers situations where this fails to hold. Consider moving through an unfamiliar building that is completely dark. When the visual system sees only black, one's localization system quickly degrades. Another useful example is that of a human-sized maze made from tall hedges. Such mazes have been created for centuries, and humans find them extremely difficult to solve without landmarks or clues because, without visual uniqueness, human localization competence degrades rapidly.

In robots, the nonuniqueness of sensor readings, or *sensor aliasing*, is the norm and not the exception. Consider a narrow-beam rangefinder such as an ultrasonic or infrared rangefinder. This sensor provides range information in a single direction without any addi- tional data regarding material composition such as color, texture, and hardness. Even for a robot with several such sensors in an array, there are a variety of environmental states that would trigger the same sensor values across the array. Formally, there is a many-to-one mapping from environmental states to the robot's perceptual inputs. Thus, the robot's percepts cannot distinguish from among these many states. A classic problem with sonar- based robots involves distinguishing between humans and inanimate objects in an indoor setting. When facing an apparent obstacle in front of itself, should the robot say "Excuse me" because the obstacle may be a moving human, or should the robot plan a path around the object because it may be a cardboard box? With sonar alone, these states are aliased and differentiation is impossible.

The problem posed to navigation because of sensor aliasing is that, even with noise-free sensors, the amount of information is generally insufficient to identify the robot's position from a single-percept reading. Thus techniques must be employed by the robot programmer that base the robot's localization on a series of readings and, thus, sufficient information to recover the robot's position over time.

5.2.3 Effector noise

The challenges of localization do not lie with sensor technologies alone. Just as robot sen- sors are noisy, limiting the information content of the signal, so robot effectors are also noisy. In particular, a single action taken by a mobile robot may have several different pos- sible results, even though from the robot's point of view the initial state before the action was taken is well known.

In short, mobile robot effectors introduce uncertainty about future state. Therefore the simple act of moving tends to increase the uncertainty of a mobile robot. There are, of course, exceptions. Using *cognition,* the motion can be carefully planned so as to minimize this effect, and indeed sometimes to actually result in more certainty. Furthermore, when the robot's actions are taken in concert with careful interpretation of sensory feedback, it can compensate for the uncertainty introduced by noisy actions using the information pro- vided by the sensors.

First, however, it is important to understand the precise nature of the effector noise that impacts mobile robots. It is important to note that, from the robot's point of view, this error in motion is viewed as an error in odometry, or the robot's inability to estimate its own posi- tion over time using knowledge of its kinematics and dynamics. The true source of error generally lies in an incomplete model of the environment. For instance, the robot does not model the fact that the floor may be sloped, the wheels may slip, and a human may push the robot. All of these unmodeled sources of error result in inaccuracy between the physical motion of the robot, the intended motion of the robot, and the proprioceptive sensor esti- mates of motion.

In odometry (wheel sensors only) and dead reckoning (also

heading sensors) the posi- tion update is based on *proprioceptive* sensors. The movement of the robot, sensed with wheel encoders or heading sensors or both, is integrated to compute position. Because the sensor measurement errors are integrated, the position error accumulates over time. Thus the position has to be updated from time to time by other localization mechanisms. Other- wise the robot is not able to maintain a meaningful position estimate in the long run.

In the following we concentrate on odometry based on the wheel sensor readings of a differential-drive robot only (see also [4, 57, 58]). Using additional heading sensors (e.g., gyroscope) can help to reduce the cumulative errors, but the main problems remain the same.

There are many sources of odometric error, from environmental factors to resolution:

- Limited resolution during integration (time increments, measurement resolution, etc.);
- Misalignment of the wheels (deterministic);
- Uncertainty in the wheel diameter and in particular unequal wheel diameter (determin- istic);
- Variation in the contact point of the wheel;

- Unequal floor contact (slipping, nonplanar surface, etc.).

Some of the errors might be *deterministic (systematic)*, thus they can be eliminated by proper calibration of the system. However, there are still a number of *nondeterministic (random)* errors which remain, leading to uncertainties in position estimation over time. From a geometric point of view one can classify the errors into three types:

1. Range error: integrated path length (distance) of the robot's movement
   □ sum of the wheel movements
2. Turn error: similar to range error, but for turns
   □ difference of the wheel motions
3. Drift error: difference in the error of the wheels leads to an error in the robot's angular orientation

Over long periods of time, turn and drift errors far outweigh range errors, since their con- tribution to the overall position error is nonlinear. Consider a robot whose position is ini- tially perfectly well-known, moving forward in a straight line along the x -axis. The error in the y -position introduced by a move of d meters will have a component of $d \sin \Box\Box$,

which can be quite large as the angular error $\Box\Box$ grows. Over time, as a mobile robot moves about the environment, the rotational error between its internal reference frame and its orig- inal reference frame grows quickly. As the robot moves away from the origin of these ref- erence frames, the resulting linear error in position grows quite large. It is instructive to establish an error model for odometric accuracy and see how the errors propagate over time.

5.2.4 An error model for odometric position estimation

Generally the pose (position) of a robot is represented by the vector

$$p = \begin{bmatrix} x \\ y \end{bmatrix} \tag{5.1}$$

For a differential-drive robot the position can be estimated starting from a known posi- tion by integrating the movement (summing the incremental travel distances). For a discrete system with a fixed sampling interval Δt the incremental travel distances $\Delta x; \Delta y; \Delta\Delta$ are

$$\Delta x = \Delta s \cos\Delta\Delta + \Delta\Delta \Delta 2\Delta \tag{5.2}$$

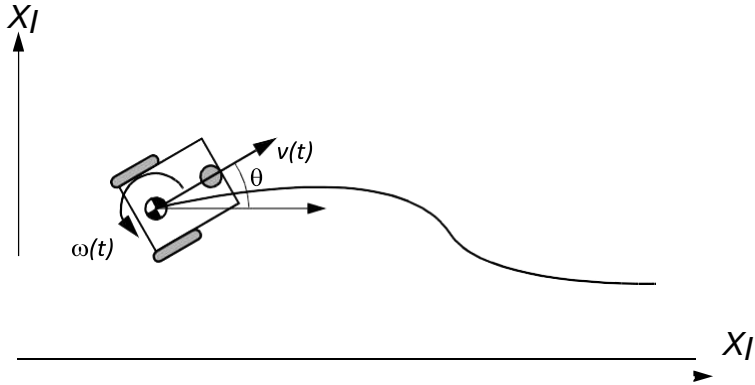


Figure 5.3

Movement of a differential-drive robot.

$$\Delta y = \Delta s \sin\Delta\theta + \frac{\Delta\theta}{2} \tag{5.3}$$

$$\Delta\theta = \frac{\Delta s_r - \Delta s_l}{b} \tag{5.4}$$

$$\Delta s = \frac{\Delta s_r + \Delta s_l}{2} \tag{5.5}$$

where

$\Delta x;\Delta y;\Delta\theta$ = path traveled in the last sampling interval;

$\Delta s_r;\Delta s_l$ = traveled distances for the right and left wheel respectively;

b = distance between the two wheels of

differential-drive robot. Thus we get the

updated position p' :

$$p' = \begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = p + \begin{bmatrix} \Delta s \cos\left(\theta + \frac{\Delta\theta}{2}\right) \\ \Delta s \sin\left(\theta + \frac{\Delta\theta}{2}\right) \\ \Delta\theta \end{bmatrix} \quad (5.6)$$

By using the relation for $\Delta s, \Delta\theta$ of equations (5.4) and (5.5) we further obtain the

basic equation for odometric position update (for differential drive robots):

$$p' = f(x, y, \theta, \Delta s_r, \Delta s_l) = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} \frac{\Delta s_r + \Delta s_l}{2} \cos\left(\theta + \frac{\Delta s_r - \Delta s_l}{2b}\right) \\ \frac{\Delta s_r + \Delta s_l}{2} \sin\left(\theta + \frac{\Delta s_r - \Delta s_l}{2b}\right) \\ \frac{\Delta s_r - \Delta s_l}{b} \end{bmatrix} \tag{5.7}$$

As we discussed earlier, odometric position updates can give only a very rough estimate of the actual position. Owing to integration errors of the uncertainties of p and the motion errors during the incremental motion $(\Delta s_r, \Delta s_l)$ the position error based on odometry inte- gration grows with time.

In the next step we will establish an error model for the integrated position p' to obtain the covariance matrix $\Sigma_{p'}$ of the odometric position estimate. To do so, we assume that at the starting point the initial covariance matrix Σ_p is known. For the motion increment $(\Delta s_r, \Delta s_l)$ we assume the following covariance matrix Σ_Δ:

$$\Sigma_\Delta \equiv covar(\Delta s, \Delta s) = \begin{bmatrix} k_r |\Delta s_r| & 0 \\ 0 & k_l |\Delta s_l| \end{bmatrix} \tag{5.8}$$

where Δs_r and Δs_l are the distances traveled by each wheel,

and k_r , k_l are error con- stants representing the nondeterministic parameters of the motor drive and the wheel-floor interaction. As you can see, in equation (5.8) we made the following assumptions:

- The two errors of the individually driven wheels are independent[5];
- The variance of the errors (left and right wheels) are proportional to the absolute value of the traveled distances $\square\square s_r$; $\square s_l\square$.

These assumptions, while not perfect, are suitable and will thus be used for the further development of the error model. The *motion errors* are due to imprecise movement because of deformation of wheel, slippage, unequal floor, errors in encoders, and so on. The values for the error constants k_r and k_l depend on the robot and the environment and should be experimentally established by performing and analyzing representative movements.

If we assume that p and $\square_{rl} = \square\square s_r; \square s_l\square$ are uncorrelated and the derivation of f [equa- tion (5.7)] is reasonably approximated by the first-order Taylor expansion (linearization), we conclude, using the error propagation law (see section 4.2.2),

---

5. If there is more knowledge regarding the actual robot kinematics, the correlation terms of the covariance matrix could also be used.

$$\Sigma_{p'} = F_p \Sigma_p F_p^T + F_{\Delta rl} \Sigma_{\Delta rl} F_{\Delta rl}^T \qquad (5.9)$$

The covariance matrix Σ_p is, of course, always given by the $\Sigma_{p'}$ of the previous step, and can thus be calculated after specifying an initial value (e.g., 0).

Using equation (5.7) we can develop the two *Jacobians*, $F_p = \frac{\partial f}{\partial p}$ and $F_{\Delta rl} = \frac{\partial f}{\partial \Delta rl}$:

$$F_p = \frac{\partial f}{\partial p} = \begin{bmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} & \frac{\partial f}{\partial \theta} \end{bmatrix} = \begin{bmatrix} 1 & 0 & -s \sin\left(\theta + \frac{\Delta\theta}{2}\right) \\ 0 & 1 & s \cos\left(\theta + \frac{\Delta\theta}{2}\right) \\ 0 & 0 & 1 \end{bmatrix} \qquad (5.10)$$

$$F_{rl} = \begin{bmatrix} \frac{1}{2} \cos\left(\theta + \frac{\Delta\theta}{2}\right) + \frac{s}{2b} \sin\left(\theta + \frac{\Delta\theta}{2}\right) & \frac{1}{2} \cos\left(\theta + \frac{\Delta\theta}{2}\right) - \frac{s}{2b} \sin\left(\theta + \frac{\Delta\theta}{2}\right) \\ \frac{1}{2} \sin\left(\theta + \frac{\Delta\theta}{2}\right) - \frac{s}{2b} \cos\left(\theta + \frac{\Delta\theta}{2}\right) & \frac{1}{2} \sin\left(\theta + \frac{\Delta\theta}{2}\right) + \frac{s}{2b} \cos\left(\theta + \frac{\Delta\theta}{2}\right) \\ \frac{1}{b} & -\frac{1}{b} \end{bmatrix} \qquad (5.11)$$

The details for arriving at equation (5.11) are

$$F_{rl} = \Delta f \begin{bmatrix} \frac{\partial f}{\partial s_r} & \frac{\partial f}{\partial s_l} \end{bmatrix} = \tag{5.12}$$

$$\begin{bmatrix} \frac{\partial s}{\partial s_r}\cos\theta + \frac{\partial}{\partial s_r}\left(-\theta + \frac{s}{2}\right)\sin\theta + \frac{\partial}{\partial s_r}\left(\frac{s}{2}\right)\left(\frac{\sin\theta}{\theta}\right) & \frac{\partial s}{\partial s_l}\cos\theta + \frac{\partial}{\partial s_l}\left(-\theta + \frac{s}{2}\right)\sin\theta + \frac{\partial}{\partial s_l}\left(\frac{s}{2}\right)\left(\frac{\sin\theta}{\theta}\right) \\ \frac{\partial s}{\partial s_r}\sin\theta + \frac{\partial}{\partial s_r}\left(-\theta + \frac{s}{2}\right)\cos\theta + \frac{\partial}{\partial s_r}\left(\frac{s}{2}\right)\left(\frac{1-\cos\theta}{\theta}\right) & \frac{\partial s}{\partial s_l}\sin\theta + \frac{\partial}{\partial s_l}\left(-\theta + \frac{s}{2}\right)\cos\theta + \frac{\partial}{\partial s_l}\left(\frac{s}{2}\right)\left(\frac{1-\cos\theta}{\theta}\right) \end{bmatrix}$$

$$\begin{bmatrix} \frac{\partial s}{\partial s_r} - \frac{1}{2}\frac{\partial \theta}{\partial s_r}\left(\frac{\sin\theta}{\theta}\right) & \frac{\partial s}{\partial s_l} - \frac{1}{2}\frac{\partial \theta}{\partial s_l}\left(\frac{\sin\theta}{\theta}\right) \\ \frac{1}{2}\frac{\partial \theta}{\partial s_r}\left(\frac{1-\cos\theta}{\theta}\right) & \frac{1}{2}\frac{\partial \theta}{\partial s_l}\left(\frac{1-\cos\theta}{\theta}\right) \end{bmatrix} \tag{5.13}$$

and with

$$\Delta s = \frac{\Delta s_r + \Delta s_l}{2}; \quad \Delta\theta = \frac{\Delta s_r - \Delta s_l}{b} \tag{5.14}$$

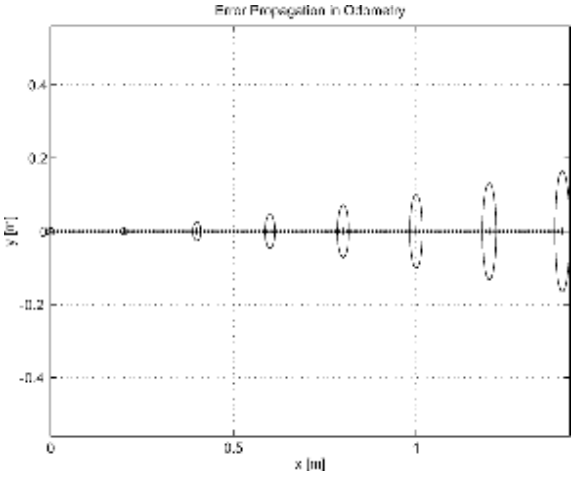


Figure 5.4

Growth of the pose uncertainty for straight-line movement: Note that the uncertainty in y grows much faster than in the direction of movement. This results from the integration of the uncertainty about the robot's orientation. The ellipses drawn around the robot positions represent the uncertainties in the *x,y* direction (e.g. 3). The uncertainty of the orientation is not represented in the picture although its effect can be indirectly observed.

$$\frac{\partial s}{\partial s_r} = \frac{1}{2}\cdot\cos\theta - \frac{1}{2}\cdot\sin\theta - \frac{s}{b}\cdot\sin\theta \qquad \frac{\partial s}{\partial s_l} = \frac{1}{2}\cdot\cos\theta + \frac{1}{2}\cdot\sin\theta + \frac{s}{b}\cdot\cos\theta \qquad \frac{\partial \theta}{\partial s_r} = \frac{1}{b} \qquad \frac{\partial \theta}{\partial s_l} = -\frac{1}{b} \tag{5.15}$$

we obtain equation (5.11).

Figures 5.4 and 5.5 show typical examples of how the position errors grow with time.

The results have been computed using the error model presented above.

Once the error model has been established, the error parameters must be specified. One can compensate for deterministic errors properly calibrating the robot. However the error parameters specifying the nondeterministic errors can only be quantified by statistical (repetitive) measurements. A detailed discussion of odometric errors and a method for cal- ibration and quantification of deterministic and nondeterministic errors can be found in [5]. A method for on-the-fly odometry error estimation is presented in [105].

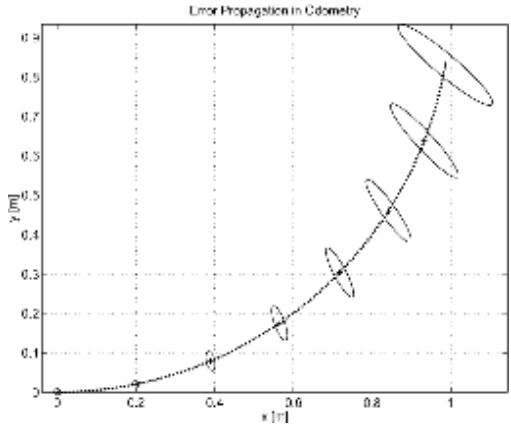


Figure 5.5

Growth of the pose uncertainty for circular movement (r = const): Again, the uncertainty perpendic- ular to the movement grows much faster than that in the direction of movement. Note that the main axis of the uncertainty ellipse does not remain perpendicular to the direction of movement.

5.3 To Localize or Not to Localize: Localization-Based Navigation versus Programmed Solutions

Figure 5.6 depicts a standard indoor environment that a mobile robot navigates. Suppose that the mobile robot in question must deliver messages between two specific rooms in this environment: rooms *A* and *B*. In creating a navigation system, it is clear that the mobile robot will need sensors and a motion control system. Sensors are absolutely required to avoid hitting moving obstacles such as humans, and some motion control

system is required so that the robot can deliberately move.

It is less evident, however, whether or not this mobile robot will require a *localization system*. Localization may seem mandatory in order to successfully navigate between the two rooms. It is through localizing on a map, after all, that the robot can hope to recover its position and detect when it has arrived at the goal location. It is true that, at the least, the robot must have a way of detecting the goal location. However, explicit localization with reference to a map is not the only strategy that qualifies as a goal detector.

An alternative, espoused by the behavior-based community, suggests that, since sensors and effectors are noisy and information-limited, one should avoid creating a geometric map for localization. Instead, this community suggests designing sets of behaviors that together result in the desired robot motion. Fundamentally, this approach avoids explicit reasoning about localization and position, and thus generally avoids explicit path planning as well.

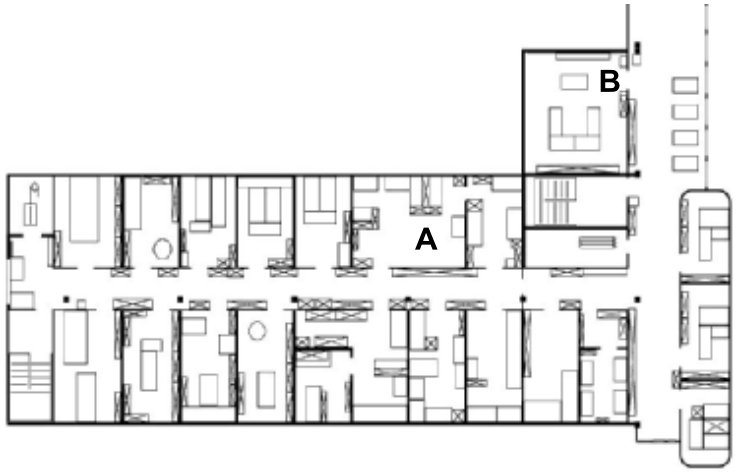


Figure 5.6

A sample environment.

This technique is based on a belief that there exists a procedural solution to the particular navigation problem at hand. For example, in figure 5.6, the behavioralist approach to nav- igating from room *A* to room *B* might be to design a left-wall following behavior and a detector for room *B* that is triggered by some unique queue in room *B*, such as the color of the carpet. Then the robot can reach room *B* by engaging the left-wall follower with the room *B* detector as the termination condition for the program.

The architecture of this solution to a specific navigation problem is shown in figure 5.7. The key advantage of this method is that, when possible, it may be implemented very quickly for a single environment with a small number of goal positions. It suffers from some disadvantages, however. First,

the method does not directly scale to other environ- ments or to larger environments. Often, the navigation code is location-specific, and the same degree of coding and debugging is required to move the robot to a new environment.

Second, the underlying procedures, such as *left-wall-follow,* must be carefully designed to produce the desired behavior. This task may be time-consuming and is heavily dependent on the specific robot hardware and environmental characteristics.

Third, a behavior-based system may have multiple active behaviors at any one time. Even when individual behaviors are tuned to optimize performance, this fusion and rapid switching between multiple behaviors can negate that fine-tuning. Often, the addition of each new incremental behavior forces the robot designer to retune all of the existing behav- iors again to ensure that the new interactions with the freshly introduced behavior are all stable.

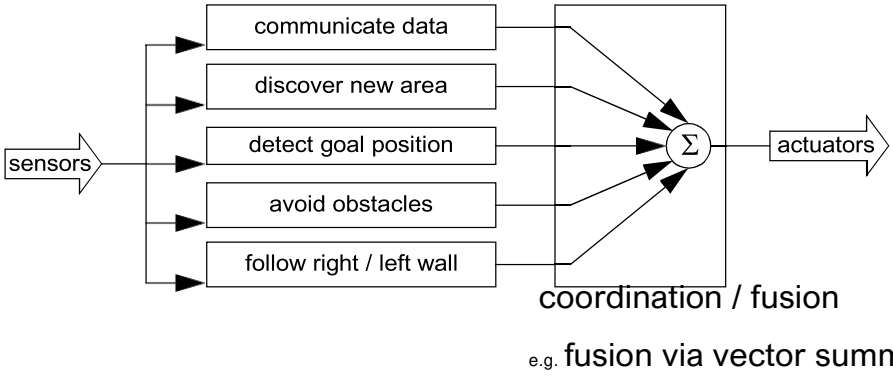


Figure 5.7

An architecture for behavior-based navigation.

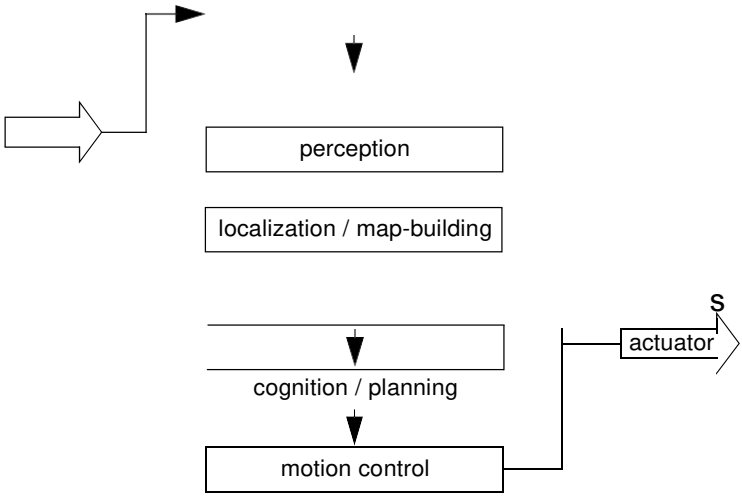


Figure 5.8

An architecture for map-based (or model-based) navigation.

In contrast to the behavior-based approach, the map-based approach includes both *local- ization* and *cognition* modules (see

figure 5.8). In map-based navigation, the robot explic- itly attempts to localize by collecting sensor data, then updating some belief about its position with respect to a map of the environment. The key advantages of the map-based approach for navigation are as follows:

- The explicit, map-based concept of position makes the system's belief about position transparently available to the human operators.
- The existence of the map itself represents a medium for communication between human and robot: the human can simply give the robot a new map if the robot goes to a new environment.

- The map, if created by the robot, can be used by humans as well, achieving two uses. The map-based approach will require more up-front development effort to create a navigating mobile robot. The hope is that the development effort results in an architecture that can successfully map and navigate a variety of environments, thereby amortizing the up- front design cost over time.

Of course the key risk of the map-based approach is that an internal representation, rather than the real world itself, is being constructed and *trusted* by the robot. If that model diverges from reality (i.e., if the map is wrong), then the robot's behavior may be undesir- able, even if the raw sensor values of the robot are only transiently incorrect.

In the remainder of this chapter, we focus on a discussion of map-based approaches and, specifically, the localization component of these techniques. These approaches are particularly appropriate for study given their significant recent successes in enabling mobile robots to navigate a variety of environments, from academic research buildings, to factory floors, and to museums around the world.

5.4 Belief Representation

The fundamental issue that differentiates various map-based localization systems is the issue of *representation*. There are two specific concepts that the robot must represent, and each has its own unique possible solutions. The robot must have a representation (a model) of the environment, or a map. What aspects of the environment are contained in this map? At what level of fidelity does the map represent the environment? These are the design questions for *map representation*.

The robot must also have a representation of its belief regarding its position on the map. Does the robot identify a single unique position as its current position, or does it describe its position in terms of a set of possible positions? If multiple possible positions are expressed in a single belief, how are those multiple positions ranked, if at all? These are the design questions for *belief representation*.

Decisions along these two design axes can result in varying levels of architectural com- plexity, computational complexity, and overall localization accuracy. We begin by discuss- ing belief representation. The first major branch in a taxonomy of belief representation systems differentiates between single-hypothesis and multiple-hypothesis belief systems. The former covers solutions in which the robot postulates its unique position, whereas the latter enables a mobile robot to describe the degree to which it is uncertain about its posi- tion. A sampling of different belief and map representations is shown in figure 5.9.

5.4.1 Single-hypothesis belief

The single-hypothesis belief representation is the most direct possible postulation of mobile robot position. Given some environmental map, the robot's belief about position is

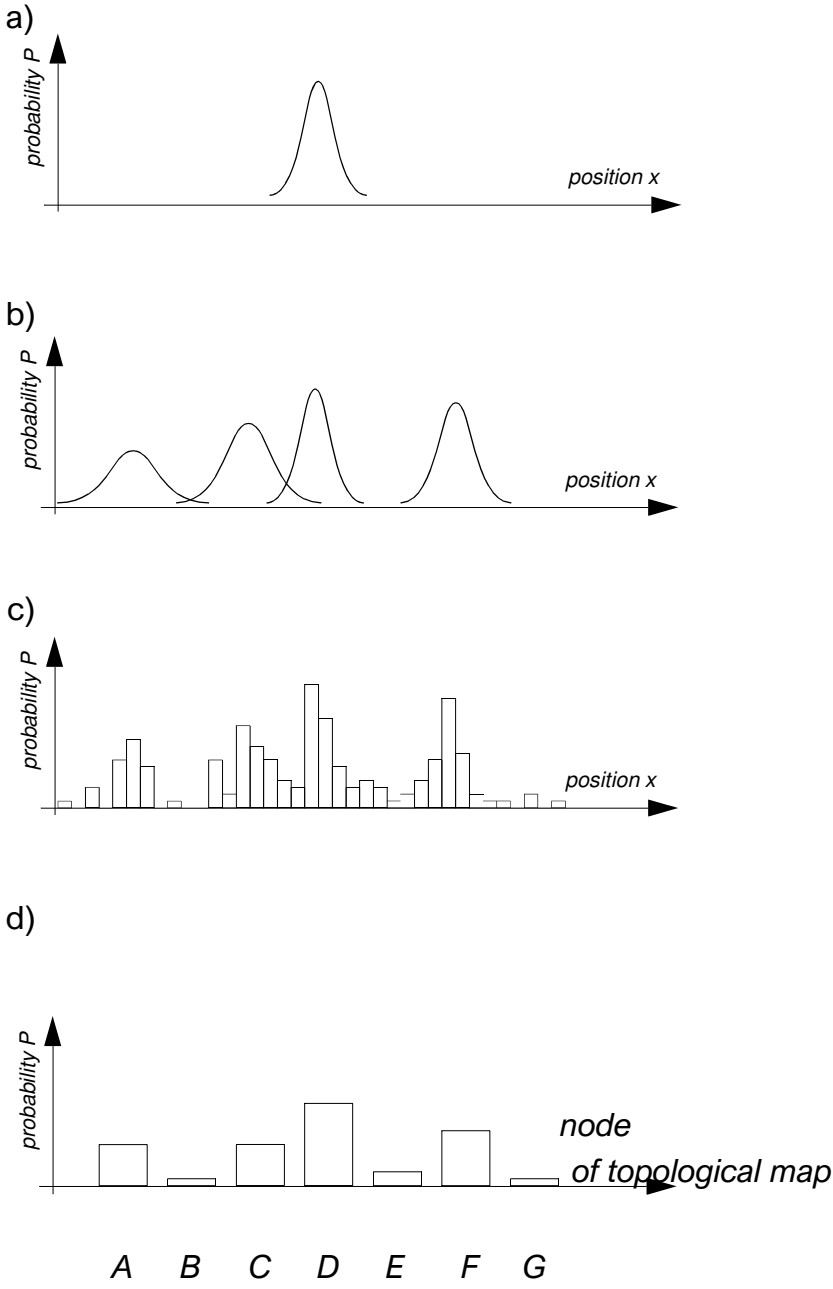

a)
probability P
position x
b)
probability P
position x
c)
probability P
position x
d)
probability P
node
of topological map
A
B
C
D
E
F
G

Figure 5.9

Belief representation regarding the robot position (1D) in continuous and discretized (tessellated) maps. (a) Continuous map with single-hypothesis belief, e.g., single Gaussian centered at a single continuous value. (b) Continuous map with multiple-hypothesis belief, e.g;. multiple Gaussians cen- tered at multiple continuous values. (c) Discretized (decomposed) grid map with probability values for all possible robot positions, e.g.; Markov approach. (d) Discretized topological map with proba- bility value for all possible nodes (topological robot positions), e.g.; Markov approach.

expressed as a single unique point on the map. In figure 5.10, three examples of a single- hypothesis belief are shown using three different map representations of the same actual environment (figure 5.10a). In figure 5.10b, a single point is geometrically annotated as the robot's position in a continuous 2D geometric map. In figure 5.10c, the map is a discrete, tessellated map, and the position is noted at the same level of fidelity as the map cell size. In figure 5.10d, the map is not geometric at all but abstract and topological. In this case, the single hypothesis of position involves identifying a single node *i* in the topological graph as the robot's position.

The principal advantage of the single-hypothesis representation of position stems from the fact that, given a unique belief, there is no position ambiguity. The unambiguous nature of this representation facilitates decision-making at the robot's cognitive level (e.g., path planning). The robot can simply assume that its belief is correct, and can then select its future actions based on its unique position.

Just as decision-making is facilitated by a single-position hypothesis, so updating the robot's belief regarding position is also facilitated, since the single position must be updated by definition to a new, single position. The challenge with this position update approach, which ultimately is the principal disadvantage of single-hypothesis representa- tion, is that robot motion often induces uncertainty due to effector and sensor noise. There- fore, forcing the position update process to always generate a *single* hypothesis of position is challenging and, often, impossible.

5.4.2 Multiple-hypothesis belief

In the case of multiple-hypothesis beliefs regarding position, the robot tracks not just a single possible position but a possibly infinite set of positions.

In one simple example originating in the work of Jean-Claude Latombe [21, 99], the robot's position is described in terms of a convex polygon positioned in a 2D map of the environment. This multiple-hypothesis representation communicates the set of possible robot positions geometrically, with no preference ordering over the positions. Each point in the map is simply either contained by the polygon and, therefore, in the robot's belief set, or outside the polygon and thereby excluded. Mathematically, the position polygon serves to partition the space of possible robot positions. Such a polygonal representation of the multiple-hypothesis belief can apply to a continuous, geometric map of the environment
[35] or, alternatively, to a tessellated, discrete approximation to the continuous environ- ment.

It may be useful, however, to incorporate some ordering on the possible robot positions, capturing the fact that some robot positions are likelier than others. A strategy for repre- senting a continuous multiple-hypothesis belief state along with a preference ordering over possible positions is to model the belief as a mathematical distribution. For example, [50, 142] notate the robot's position belief using an (X, Y) point in the 2D environment as the

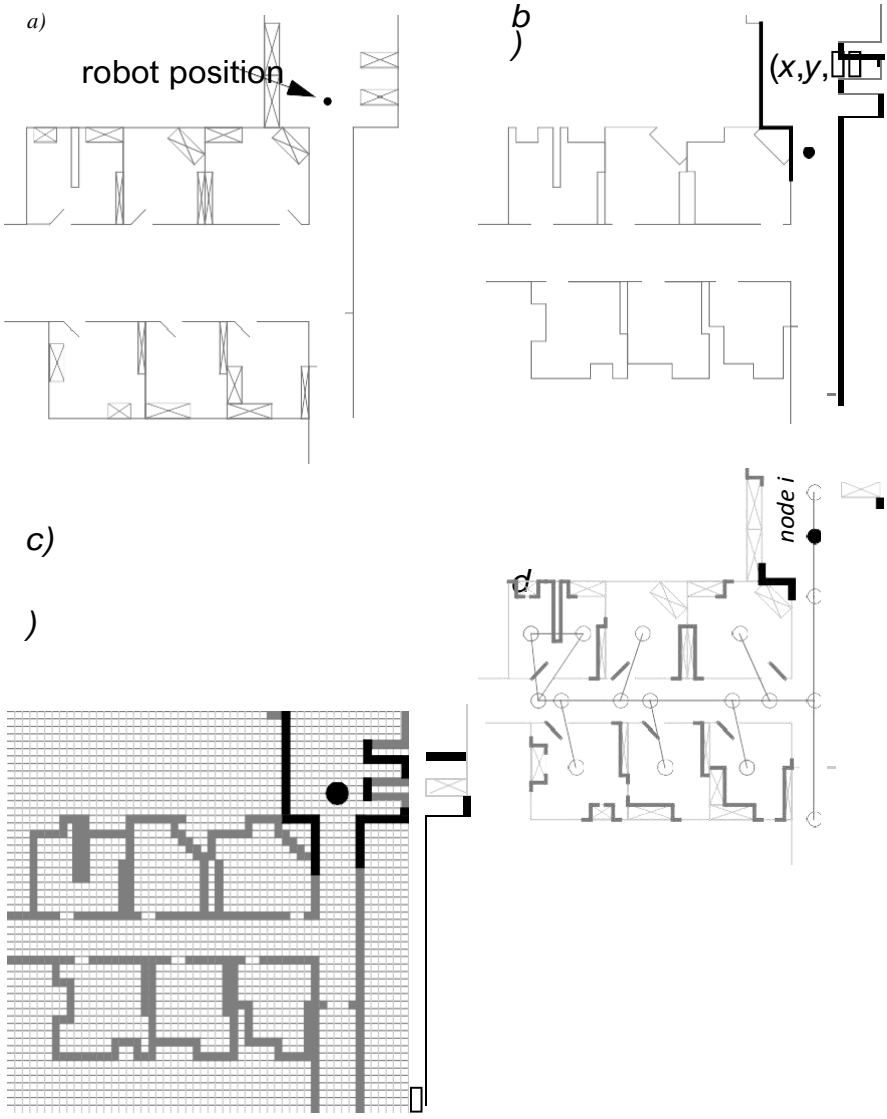
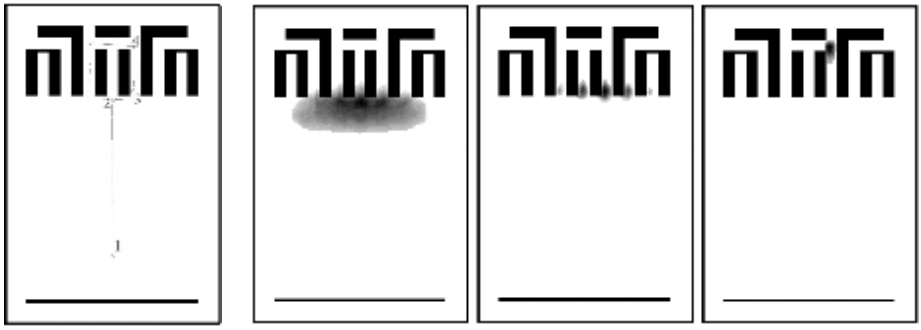


Figure 5.10

Three examples of single hypotheses of position using different map representations: (a) real map with walls, doors and furniture; (b) line-based map around 100 lines with two parameters; (c) occupancy grid-based map around 3000 grid

cells size 50 x 50 cm; (d) topological map using line features (Z/S lines) and doors around 50 features and 18 nodes.



Path of the robot *Belief states at positions 2, 3, and 4*

Figure 5.11

Example of multiple-hypothesis tracking (courtesy of W. Burgard [49]). The belief state that is largely distributed becomes very certain after moving to position 4. Note that darker coloring repre- sents higher probability.

mean plus a standard deviation parameter , thereby defining a Gaussian distribution. The intended interpretation is that the distribution at each position represents the probabil- ity assigned to the robot being at that location. This representation is particularly amenable to mathematically defined tracking functions, such as the Kalman filter, that are designed to operate efficiently on Gaussian distributions.

An alternative is to represent the set of possible robot positions, not using a single Gaus- sian probability density function, but using discrete markers for each possible position. In this case, each possible robot position is individually noted along with a confidence or probability parameter (see figure 5.11). In the case of a highly tessellated map this can result in thousands or

even tens of thousands of possible robot positions in a single-belief state.

The key advantage of the multiple-hypothesis representation is that the robot can explic- itly maintain uncertainty regarding its position. If the robot only acquires partial informa- tion regarding position from its sensors and effectors, that information can conceptually be incorporated in an updated belief.

A more subtle advantage of this approach revolves around the robot's ability to explic- itly measure its own degree of uncertainty regarding position. This advantage is the key to a class of localization and navigation solutions in which the robot not only reasons about reaching a particular goal but reasons about the future trajectory of its own belief state. For instance, a robot may choose paths that minimize its future position uncertainty. An exam- ple of this approach is [141], in which the robot plans a path from point *A* to point *B* that takes it near a series of landmarks in order to mitigate localization difficulties. This type of

explicit reasoning about the effect that trajectories will have on the quality of localization requires a multiple-hypothesis representation.

One of the fundamental disadvantages of multiple-hypothesis approaches involves deci- sion-making. If the robot represents its position as a region or set of possible positions, then how shall it decide what to do next? Figure 5.11 provides an example. At position 3, the robot's belief state is distributed among five hallways separately. If the goal of the robot is to travel down one particular hallway, then given this belief state, what action should the robot choose?

The challenge occurs because some of the robot's possible positions imply a motion tra- jectory that is inconsistent with some of its other possible positions. One approach that we will see in the case studies below is to assume, for decision-making purposes, that the robot is physically at the most probable location in its belief state, then to choose a path based on that current position. But this approach demands that each possible position have an asso- ciated probability.

In general, the right approach to such decision-making problems would be to decide on trajectories that eliminate the ambiguity explicitly. But this leads us to the second major disadvantage of multiple-hypothesis approaches. In the most general case, they can be computationally very expensive. When one reasons in a 3D space of discrete possible posi- tions, the number of possible belief states in the single-hypothesis case is limited to the number of possible positions in the 3D world. Consider this number to be N . When one moves to an arbitrary multiple-hypothesis representation, then the number of possible belief states is the power set of N , which is far larger: 2^N . Thus

explicit reasoning about the possible trajectory of the belief state over time quickly becomes computationally unten- able as the size of the environment grows.

There are, however, specific forms of multiple-hypothesis representations that are some- what more constrained, thereby avoiding the computational explosion while allowing a limited type of multiple-hypothesis belief. For example, if one assumes a Gaussian distri- bution of probability centered at a single position, then the problem of representation and tracking of belief becomes equivalent to Kalman filtering, a straightforward mathematical process described below. Alternatively, a highly tessellated map representation combined with a limit of ten possible positions in the belief state, results in a discrete update cycle that is, at worst, only ten times more computationally expensive than a single-hypothesis belief update. And other ways to cope with the complexity problem, still being precise and computationally cheap, are hybrid metric-topological approaches [145, 147] or multi-Gaussian position estimation [35, 60, 81].

In conclusion, the most critical benefit of the multiple-hypothesis belief state is the abil- ity to maintain a sense of position while explicitly annotating the robot's uncertainty about its own position. This powerful representation has enabled robots with limited sensory

information to navigate robustly in an array of environments, as we shall see in the case studies below.

5.5 Map Representation

The problem of representing the environment in which the robot moves is a dual of the problem of representing the robot's possible position or positions. Decisions made regard- ing the environmental representation can have impact on the choices available for robot position representation. Often the fidelity of the position representation is bounded by the fidelity of the map.

Three fundamental relationships must be understood when choosing a particular map representation:

1. The precision of the map must appropriately match the precision with which the robot needs to achieve its goals.
2. The precision of the map and the type of features represented must match the precision and data types returned by the robot's sensors.
3. The complexity of the map representation has direct impact on the computational com- plexity of reasoning about mapping, localization, and navigation.

In the following sections, we identify and discuss critical design choices in creating a map representation. Each such choice has great impact on the relationships listed above and on the resulting robot localization architecture. As we shall see, the choice of possible map representations is broad. Selecting an appropriate representation requires understanding all of the trade-offs inherent in that choice as well as understanding the specific context in which a particular mobile robot

implementation must perform localization. In general, the environmental representation and model can be roughly classified as presented in chapter 4, section 4.3.

5.5.1 Continuous representations

A continuous-valued map is one method for *exact* decomposition of the environment. The position of environmental features can be annotated precisely in continuous space. Mobile robot implementations to date use continuous maps only in 2D representations, as further dimensionality can result in computational explosion.

A common approach is to combine the exactness of a continuous representation with the compactness of the *closed-world assumption*. This means that one assumes that the representation will specify all environmental objects in the map, and that any area in the map that is devoid of objects has no objects in the corresponding portion of the environment. Thus, the total storage needed in the map is proportional to the density of objects in the environment, and a sparse environment can be represented by a low-memory map.

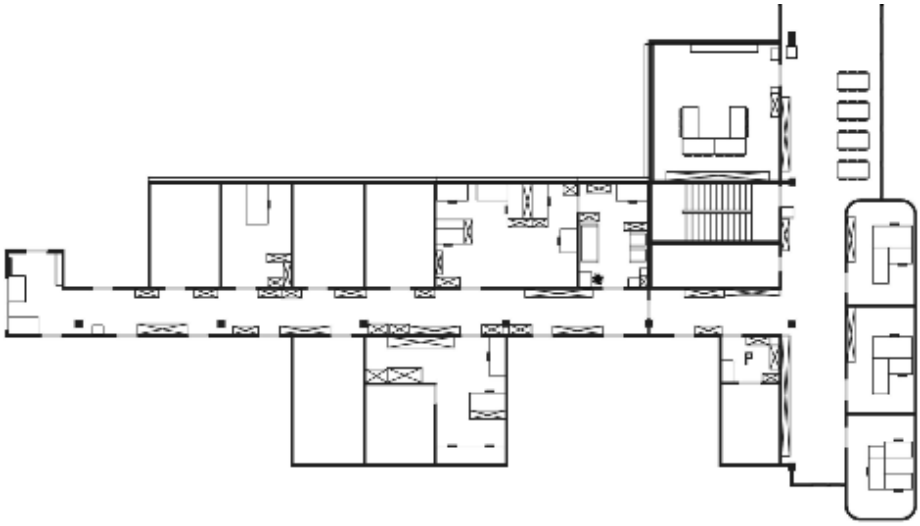


Figure 5.12

A continuous representation using polygons as environmental obstacles.

One example of such a representation, shown in figure 5.12, is a 2D representation in which polygons represent all obstacles in a continuous-valued coordinate space. This is similar to the method used by Latombe [21, 98] and others to represent environments for mobile robot path-planning techniques.

In the case of [21, 98], most of the experiments are in fact simulations run exclusively within the computer's memory. Therefore, no real effort would have been expended to attempt to use sets of polygons to describe a real-world environment, such as a park or office building.

In other work in which real environments must be captured by the maps, one sees a trend toward selectivity and abstraction. The human map maker tends to capture on the map, for

localization purposes, only objects that can be detected by the robot's sensors and, further- more, only a subset of the features of real-world objects.

It should be immediately apparent that geometric maps can capably represent the phys- ical locations of objects without referring to their texture, color, elasticity, or any other such secondary features that do not relate directly to position and space. In addition to this level of simplification, a mobile robot map can further reduce memory usage by capturing only aspects of object geometry that are immediately relevant to localization. For example, all objects may be approximated using very simple convex polygons, sacrificing map felicity for the sake of computational speed.

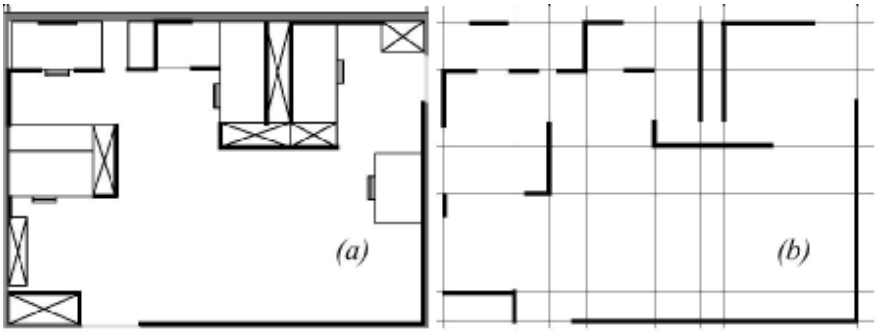


Figure 5.13

Example of a continuous-valued line representation of EPFL. (a) Real map. (b) Representation with a set of infinite lines.

One excellent example involves line extraction. Many indoor mobile robots rely upon laser rangefinding devices to recover distance readings to nearby objects. Such robots can automatically extract best-fit lines from the dense range data provided by thousands of points of laser strikes. Given such a line extraction sensor, an appropriate continuous map- ping approach is to populate the map with a set of infinite lines. The continuous nature of the map guarantees that lines can be positioned at arbitrary positions in the plane and at arbitrary angles. The abstraction of real environmental objects such as walls and intersec- tions captures only the information in the map representation that matches the type of infor- mation recovered by the mobile robot's rangefinding sensor.

Figure 5.13 shows a map of an indoor environment at EPFL using a continuous line rep- resentation. Note that the only environmental features captured by the map are straight lines,

such as those found at corners and along walls. This represents not only a sampling of the real world of richer features but also a simplification, for an actual wall may have texture and relief that is not captured by the mapped line.

The impact of continuous map representations on position representation is primarily positive. In the case of single-hypothesis position representation, that position may be specified as any continuous-valued point in the coordinate space, and therefore extremely high accuracy is possible. In the case of multiple-hypothesis position representation, the contin- uous map enables two types of multiple position representation.

In one case, the possible robot position may be depicted as a geometric shape in the hyperplane, such that the robot is known to be within the bounds of that shape. This is shown in figure 5.29, in which the position of the robot is depicted by an oval bounding area.

Yet, the continuous representation does not disallow representation of position in the form of a discrete set of possible positions. For instance, in [62] the robot position belief state is captured by sampling nine continuous-valued positions from within a region near the robot's best-known position. This algorithm captures, within a continuous space, a dis- crete sampling of possible robot positions.

In summary, the key advantage of a continuous map representation is the potential for high accuracy and expressiveness with respect to the environmental configuration as well as the robot position within that environment. The danger of a continuous representation is that the map may be computationally costly. But this danger can be tempered by employing abstraction and capturing only the most relevant environmental features. Together with the use of the *closed-world assumption*, these techniques can enable a continuous-valued map to be no more costly, and sometimes even less costly, than a standard discrete representa- tion.

5.5.2 Decomposition strategies

In the section above, we discussed one method of simplification, in which the continuous map representation contains a set of infinite lines that approximate real-world environmen- tal lines based on a 2D slice of the world. Basically this transformation from the real world to the map representation is a filter that removes all nonstraight data and furthermore extends line segment data into infinite lines that require fewer parameters.

A more dramatic form of simplification is *abstraction*: a general decomposition and selection of environmental features. In this section, we explore decomposition as applied in its more

extreme forms to the question of map representation.

Why would one radically decompose the real environment during the design of a map representation? The immediate disadvantage of decomposition and abstraction is the loss of fidelity between the map and the real world. Both qualitatively, in terms of overall struc- ture, and quantitatively, in terms of geometric precision, a highly abstract map does not compare favorably to a high-fidelity map.

Despite this disadvantage, decomposition and abstraction may be useful if the abstrac- tion can be planned carefully so as to capture the relevant, *useful* features of the world while discarding all other features. The advantage of this approach is that the map representation can potentially be minimized. Furthermore, if the decomposition is hierarchical, such as in a pyramid of recursive abstraction, then reasoning and planning with respect to the map representation may be computationally far superior to planning in a fully detailed world model.

A standard, lossless form of *opportunistic decomposition* is termed *exact cell decompo- sition*. This method, introduced by Latombe [21], achieves decomposition by selecting boundaries between discrete cells based on geometric criticality.

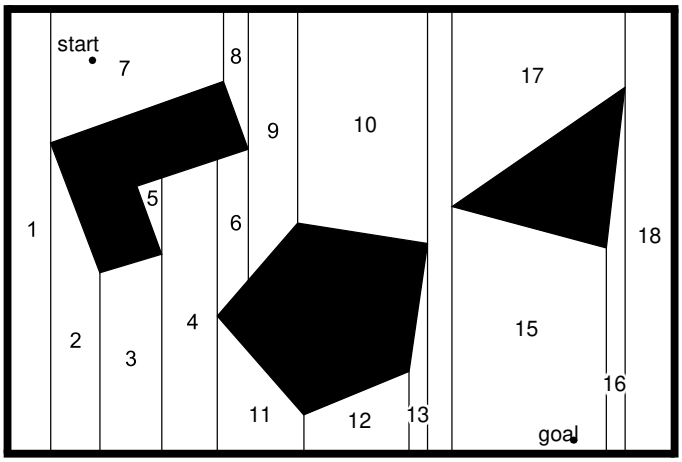


Figure 5.14

Example of exact cell decomposition.

Figure 5.14 depicts an exact decomposition of a planar workspace populated by polyg- onal obstacles. The map representation tessellates the space into areas of free space. The representation can be extremely compact because each such area is actually stored as a single node, resulting in a total of only eighteen nodes in this example.

The underlying assumption behind this decomposition is that the particular position of a robot within each area of free space does not matter. What matters is the robot's ability to traverse from each area of free space to the adjacent areas. Therefore, as with other rep- resentations we will see, the resulting graph captures the adjacency of map locales. If indeed the assumptions are valid and the robot does not care about its precise position within a single area, then this can be an effective

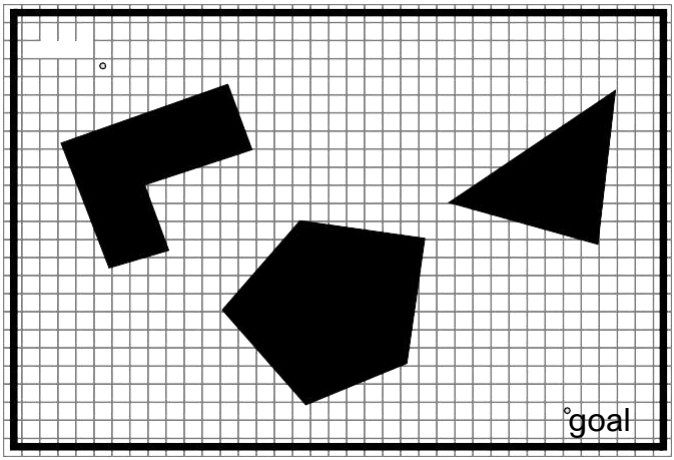
representation that nonetheless captures the connectivity of the environment.

Such an exact decomposition is not always appropriate. Exact decomposition is a func- tion of the particular environment obstacles and free space. If this information is expensive to collect or even unknown, then such an approach is not feasible.

An alternative is *fixed decomposition*, in which the world is tessellated, transforming the continuous real environment into a discrete approximation for the map. Such a transforma- tion is demonstrated in figure 5.15, which depicts what happens to obstacle-filled and free areas during this transformation. The key disadvantage of this approach stems from its *inex- act* nature. It is possible for narrow passageways to be lost during such a transformation, as shown in figure 5.15. Formally, this means that fixed decomposition is sound but not com- plete. Yet another approach is adaptive cell decomposition, as presented in figure 5.16.

ar

st t



ar

st t

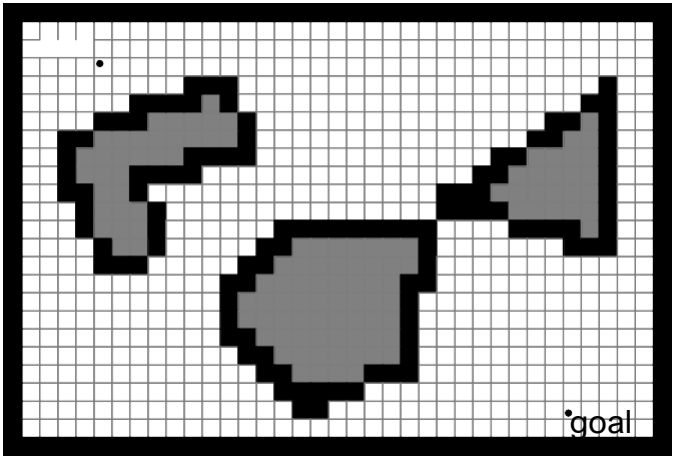


Figure 5.15

Fixed decomposition of the same space (narrow passage disappears).

The concept of fixed decomposition is extremely popular in mobile robotics; it is per- haps the single most common map representation technique currently utilized. One very popular version of fixed decomposition is known as the *occupancy grid* representation [112]. In an occupancy grid, the environment is represented by a discrete grid, where each cell is either filled (part of an obstacle) or empty (part of free space). This method is of par- ticular value when a robot is equipped with range-based sensors because the range values

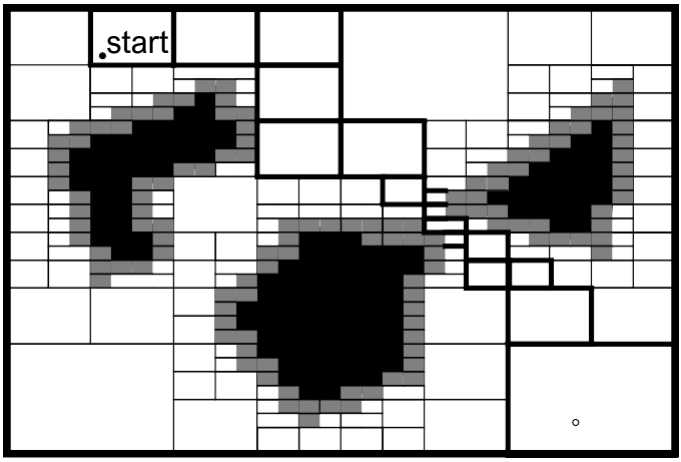


Figure 5.16

Example of adaptive (approximate variable-cell) decomposition of an environment [21]. The rectan- gle, bounding the free space, is decomposed into four identical rectangles. If the interior of a rectangle lies completely in free space or in the configuration space obstacle, it is not decomposed further. Oth- erwise, it is recursively decomposed into four rectangles until some predefined resolution is attained. The white cells lie outside the obstacles, the black inside, and the gray are part of both regions.

of each sensor, combined with the absolute position of the robot, can be used directly to update the filled or empty value of each cell.

In the occupancy grid, each cell may have a counter, whereby

the value 0 indicates that the cell has not been “hit” by any ranging measurements and, therefore, it is likely free space. As the number of ranging strikes increases, the cell’s value is incremented and, above a certain threshold, the cell is deemed to be an obstacle. The values of cells are com- monly discounted when a ranging strike travels *through* the cell, striking a further cell. By also discounting the values of cells over time, both hysteresis and the possibility of transient obstacles can be represented using this occupancy grid approach. Figure 5.17 depicts an occupancy grid representation in which the darkness of each cell is proportional to the value of its counter. One commercial robot that uses a standard occupancy grid for mapping and navigation is the Cye robot [163].

There remain two main disadvantages of the occupancy grid approach. First, the size of the map in robot memory grows with the size of the environment and if a small cell size is used, this size can quickly become untenable. This occupancy grid approach is not compat- ible with the *closed-world assumption*, which enabled continuous representations to have potentially very small memory requirements in large, sparse environments. In contrast, the

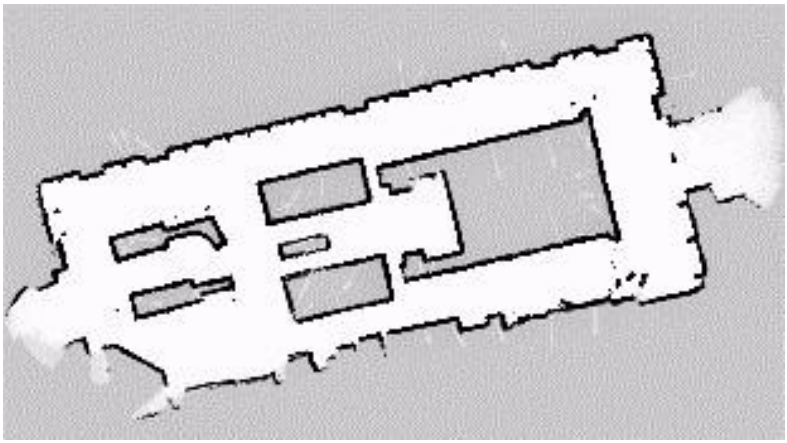


Figure 5.17

Example of an occupancy grid map representation (courtesy of S. Thrun [145]).

occupancy grid must have memory set aside for every cell in the matrix. Furthermore, any fixed decomposition method such as this imposes a geometric grid on the world *a priori*, regardless of the environmental details. This can be inappropriate in cases where geometry is not the most salient feature of the environment.

For these reasons, an alternative, called *topological* decomposition, has been the subject of some exploration in mobile robotics. Topological approaches avoid direct measurement of geometric environmental qualities, instead concentrating on characteristics of the envi- ronment that are most relevant to the robot for localization.

Formally, a topological representation is a graph that specifies two things: *nodes* and the *connectivity* between those nodes.

Insofar as a topological representation is intended for the use of a mobile robot, nodes are used to denote areas in the world and arcs are used to denote adjacency of pairs of nodes. When an arc connects two nodes, then the robot can traverse from one node to the other without requiring traversal of any other intermediary node.

Adjacency is clearly at the heart of the topological approach, just as adjacency in a cell decomposition representation maps to geometric adjacency in the real world. However, the topological approach diverges in that the nodes are not of fixed size or even specifications of free space. Instead, nodes document an area based on any sensor discriminant such that the robot can recognize entry and exit of the node.

Figure 5.18 depicts a topological representation of a set of hallways and offices in an indoor environment. In this case, the robot is assumed to have an intersection detector, per- haps using sonar and vision to find intersections between halls and between halls and

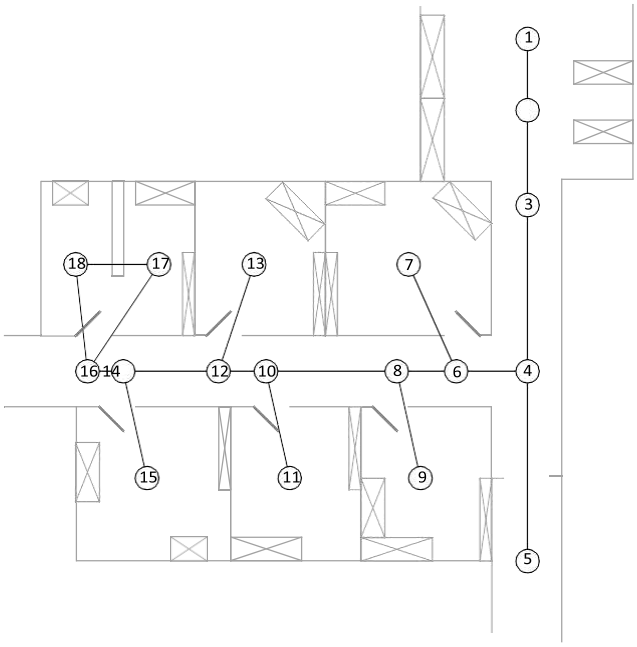


Figure 5.18

A topological representation of an indoor office area.

rooms. Note that nodes capture geometric space, and arcs in this representation simply rep- resent connectivity.

Another example of topological representation is the work of Simhon and Dudek [134], in which the goal is to create a mobile robot that can capture the most interesting aspects of an area for human consumption. The nodes in their representation are visually striking locales rather than route intersections.

In order to navigate using a topological map robustly, a robot must satisfy two con- straints. First, it must have a means for detecting its current position in terms of the nodes of the topological graph. Second, it must have a means for traveling

between nodes using robot motion. The node sizes and particular dimensions must be optimized to match the sensory discrimination of the mobile robot hardware. This ability to “tune” the representa- tion to the robot’s particular sensors can be an important advantage of the topological approach. However, as the map representation drifts further away from true geometry, the expressiveness of the representation for accurately and precisely describing a robot position is lost. Therein lies the compromise between the discrete cell-based map representations and the topological representations. Interestingly, the continuous map representation has



Figure 5.19

An artificial landmark used by Chips during autonomous docking.

the potential to be both compact like a topological representation and precise as with all direct geometric representations.

Yet, a chief motivation of the topological approach is that the environment may contain important nongeometric features – features that have no ranging relevance but are useful for localization. In chapter 4 we described such whole-image vision-based features.

In contrast to these whole-image feature extractors, often spatially localized landmarks are artificially placed in an environment to impose a particular visual-topological connectivity upon the environment. In effect, the artificial landmark can

impose artificial struc- ture. Examples of working systems operating with this landmark-based strategy have also demonstrated success. Latombe's landmark-based navigation research [99] has been implemented on real-world indoor mobile robots that employ paper landmarks attached to the ceiling as the locally observable features. Chips, the museum robot, is another robot that uses man-made landmarks to obviate the localization problem. In this case, a bright pink square serves as a landmark with dimensions and color signature that would be hard to accidentally reproduce in a museum environment [118]. One such museum landmark is shown in figure 5.19.

In summary, range is clearly not the only measurable and useful environmental value for a mobile robot. This is particularly true with the advent of color vision, as well as laser

rangefinding, which provides reflectance information in addition to range information. Choosing a map representation for a particular mobile robot requires, first, understanding the sensors available on the mobile robot, and, second, understanding the mobile robot's functional requirements (e.g., required goal precision and accuracy).

5.5.3 State of the art: current challenges in map representation

The sections above describe major design decisions in regard to map representation choices. There are, however, fundamental real-world features that mobile robot map repre- sentations do not yet represent well. These continue to be the subject of open research, and several such challenges are described below.

The real world is dynamic. As mobile robots come to inhabit the same spaces as humans, they will encounter moving people, cars, strollers, and the transient obstacles placed and moved by humans as they go about their activities. This is particularly true when one con- siders the home environment with which domestic robots will someday need to contend.

The map representations described above do not, in general, have explicit facilities for identifying and distinguishing between permanent obstacles (e.g., walls, doorways, etc.) and transient obstacles (e.g., humans, shipping packages, etc.). The current state of the art in terms of mobile robot sensors is partly to blame for this shortcoming. Although vision research is rapidly advancing, robust sensors that discriminate between moving animals and static structures *from a moving reference frame* are not yet available. Furthermore, esti- mating the motion vector of transient objects remains a research problem.

Usually, the assumption behind the above map representations

is that all objects on the map are effectively static. Partial success can be achieved by discounting mapped objects over time. For example, occupancy grid techniques can be more robust to dynamic settings by introducing temporal discounting, effectively treating transient obstacles as noise. The more challenging process of map creation is particularly fragile to environmental dynam- ics; most mapping techniques generally require that the environment be free of moving objects during the mapping process. One exception to this limitation involves topological representations. Because precise geometry is not important, transient objects have little effect on the mapping or localization process, subject to the critical constraint that the tran- sient objects must not change the topological connectivity of the environment. Still, neither the occupancy grid representation nor a topological approach is actively recognizing and representing transient objects as distinct from both sensor error and permanent map fea- tures.

As vision sensing provides more robust and more informative content regarding the transience and motion details of objects in the world, mobile roboticists will in time pro- pose representations that make use of that information. A classic example involves occlu- sion by human crowds. Museum tour guide robots generally suffer from an extreme amount of occlusion. If the robot's sensing suite is located along the robot's body, then the robot is

effectively blind when a group of human visitors completely surround the robot. This is because its map contains only environmental features that are, at that point, fully hidden from the robot's sensors by the wall of people. In the best case, the robot should recognize its occlusion and make no effort to localize using these invalid sensor readings. In the worst case, the robot will localize with the fully occluded data, and will update its location incor- rectly. A vision sensor that can discriminate the local conditions of the robot (e.g,. we are surrounded by people) can help eliminate this error mode.

A second open challenge in mobile robot localization involves the traversal of open spaces. Existing localization techniques generally depend on local measures such as range, thereby demanding environments that are somewhat densely filled with objects that the sensors can detect and measure. Wide-open spaces such as parking lots, fields of grass, and indoor atriums such as those found in convention centers pose a difficulty for such systems because of their relative sparseness. Indeed, when populated with humans, the challenge is exacerbated because any mapped objects are almost certain to be occluded from view by the people.

Once again, more recent technologies provide some hope of overcoming these limita- tions. Both vision and state-of-the-art laser rangefinding devices offer outdoor performance with ranges of up to a hundred meters and more. Of course, GPS performs even better. Such long-range sensing may be required for robots to localize using distant features.

This trend teases out a hidden assumption underlying most topological map representa- tions. Usually, topological representations make assumptions regarding spatial locality: a

node contains objects and features that are themselves within that node. The process of map creation thus involves making nodes that are, in their own self-contained way, recognizable by virtue of the objects contained within the node. Therefore, in an indoor environment, each room can be a separate node, and this is reasonable because each room will have a layout and a set of belongings that are unique to that room.

However, consider the outdoor world of a wide-open park. Where should a single node end and the next node begin? The answer is unclear because objects that are far away from the current node, or position, can yield information for the localization process. For exam- ple, the hump of a hill at the horizon, the position of a river in the valley, and the trajectory of the sun all are nonlocal features that have great bearing on one's ability to infer current position. The spatial locality assumption is violated and, instead, replaced by a visibility criterion: the node or cell may need a mechanism for representing objects that are measur- able and visible from that cell. Once again, as sensors improve and, in this case, as outdoor locomotion mechanisms improve, there will be greater urgency to solve problems associ- ated with localization in wide-open settings, with and without GPS-type global localization sensors.

We end this section with one final open challenge that represents one of the fundamental academic research questions of robotics: sensor fusion. A variety of measurement types are

possible using off-the-shelf robot sensors, including heat, range, acoustic and light-based reflectivity, color, texture, friction, and so on. Sensor fusion is a research topic closely related to map representation. Just as a map must embody an environment in sufficient detail for a robot to perform localization and reasoning, sensor fusion demands a represen- tation of the world that is sufficiently general and expressive that a variety of sensor types can have their data correlated appropriately, strengthening the resulting percepts well beyond that of any individual sensor's readings.

Perhaps the only general implementation of sensor fusion to date is that of neural net- work classifier. Using this technique, any number and any type of sensor values may be jointly combined in a network that will use whatever means necessary to optimize its clas- sification accuracy. For the mobile robot that must use a human-readable internal map rep- resentation, no equally general sensor fusion scheme has yet been born. It is reasonable to expect that, when the sensor fusion problem is solved, integration of a large number of dis- parate sensor types may easily result in sufficient discriminatory power for robots to achieve real-world navigation, even in wide-open and dynamic circumstances such as a public square filled with people.

5.6 Probabilistic Map-Based Localization

5.6.1 Introduction

As stated earlier, multiple-hypothesis position representation is advantageous because the robot can explicitly track its own beliefs regarding its possible positions in the environment. Ideally, the robot's *belief state* will change, over time, as is consistent with its motor outputs and perceptual inputs. One geometric approach

to multiple-hypothesis representation, men- tioned earlier, involves identifying the possible positions of the robot by specifying a poly- gon in the environmental representation [98]. This method does not provide any indication of the relative chances between various possible robot positions.

Probabilistic techniques differ from this because they explicitly identify probabilities with the possible robot positions, and for this reason these methods have been the focus of recent research. In the following sections we present two classes of probabilistic localiza- tion. The first class, *Markov localization*, uses an explicitly specified probability distribu- tion across all possible robot positions. The second method, *Kalman filter localization*, uses a Gaussian probability density representation of robot position and scan matching for local- ization. Unlike Markov localization, Kalman filter localization does not independently consider each possible pose in the robot's configuration space. Interestingly, the Kalman filter localization process results from the Markov localization axioms if the robot's position uncertainty is assumed to have a Gaussian form [3, pp. 43-44].

Before discussing each method in detail, we present the general robot localization prob- lem and solution strategy. Consider a mobile robot moving in a known environment. As it

starts to move, say from a precisely known location, it can keep track of its motion using odometry. Due to odometry uncertainty, after some movement the robot will become very uncertain about its position (see section 5.2.4). To keep position uncertainty from growing unbounded, the robot must localize itself in relation to its environment map. To localize, the robot might use its on-board sensors (ultrasonic, range sensor, vision) to make observa- tions of its environment. The information provided by the robot's odometry, plus the infor- mation provided by such exteroceptive observations, can be combined to enable the robot to localize as well as possible with respect to its map. The processes of updating based on proprioceptive sensor values and exteroceptive sensor values are often separated logically, leading to a general two-step process for robot position update.

Action update represents the application of some action model *Act* to the mobile robot's proprioceptive encoder measurements o_t and prior belief state s_{t-1} to yield a new belief state representing the robot's belief about its current position. Note that throughout this
chapter we assume that the robot's proprioceptive encoder measurements are used as the best possible measure of its actions over time. If, for instance, a differential-drive robot had motors without encoders connected to its wheels and employed open-loop control, then instead of encoder measurements the robot's highly uncertain estimates of wheel spin would need to be incorporated. We ignore such cases and therefore have a simple formula:

$$s'_t = Act(o_t, s_{t-1}). \tag{5.16}$$

Perception update represents the application of some perception model *See* to the *e*

mobile robot's exteroceptive sensor inputs i_t and updated belief state s'_t to yield a refined belief state representing the robot's current position:

$$s_t = See(i_t, s'_t) \quad (5.17)$$

The perception model *See* and sometimes the action model *Act* are abstract functions of both the map and the robot's physical configuration (e.g., sensors and their positions, kinematics, etc.).

In general, the action update process contributes uncertainty to the robot's belief about position: encoders have error and therefore motion is somewhat nondeterministic. By con- trast, perception update generally refines the belief state. Sensor measurements, when com- pared to the robot's environmental model, tend to provide clues regarding the robot's possible position.

In the case of Markov localization, the robot's belief state is usually represented as sep- arate probability assignments for every possible robot pose in its map. The action update and perception update processes must update the probability of every cell in this case. Kalman filter localization represents the robot's belief state using a single, well-defined

Gaussian probability density function, and thus retains just a and parameterization of the robot's belief about position with respect to the map. Updating the parameters of the Gaussian distribution is all that is required. This fundamental difference in the representa- tion of belief state leads to the following advantages and disadvantages of the two methods, as presented in [73]:

- Markov localization allows for localization starting from any unknown position and can thus recover from ambiguous situations because the robot can track multiple, completely disparate possible positions. However, to update the probability of all positions within the whole state space at any time requires a discrete representation of the space, such as a geometric grid or a topological graph (see section 5.5.2). The required memory and computational power can thus limit precision and map size.
- Kalman filter localization tracks the robot from an initially known position and is inher- ently both precise and efficient. In particular, Kalman filter localization can be used in continuous world representations. However, if the uncertainty of the robot becomes too large (e.g., due to a robot collision with an object) and thus not truly unimodal, the Kalman filter can fail to capture the multitude of possible robot positions and can become irrevocably lost.

In recent research projects improvements are achieved or proposed by either only updat- ing the state space of interest within the Markov approach [49] or by tracking multiple hypotheses with Kalman filters [35], or by combining both methods to create a hybrid localization system [73, 147]. In the

next two sections we will each approach in detail.

5.6.2 Markov localization

Markov localization tracks the robot's belief state using an arbitrary probability density function to represent the robot's position (see also [50, 88, 116, 119]). In practice, all known Markov localization systems implement this generic belief representation by first tessellating the robot configuration space into a finite, discrete number of possible robot poses in the map. In actual applications, the number of possible poses can range from sev- eral hundred to millions of positions.

Given such a generic conception of robot position, a powerful update mechanism is required that can compute the belief state that results when new information (e.g., encoder values and sensor values) is incorporated into a prior belief state with arbitrary probability density. The solution is born out of probability theory, and so the next section describes the foundations of probability theory that apply to this problem, notably the Bayes formula. Then, two subsequent sections provide case studies, one robot implementing a simple fea- ture-driven topological representation of the environment [88, 116, 119] and the other using a geometric grid-based map [49, 50].

5.6.2.1 Introduction: applying probability theory to robot localization

Given a discrete representation of robot positions, in order to express a belief state we wish to assign to each possible robot position a probability that the robot is indeed at that posi- tion. From probability theory we use the term $p(A)$ to denote the probability that A is true. This is also called the *prior probability* of A because it measures the probability that A is true independent of any additional knowledge we may have. For example we can use $p(r_t = l)$ to denote the prior probability that the robot r is at position l at time t.

In practice, we wish to compute the probability of each individual robot position given

the encoder and sensor evidence the robot has collected. In probability theory, we use the term $p(A \mid B)$ to denote the *conditional* probability of A given that we know B. For exam- ple, we use $p(r_t = l \mid i_t)$ to denote the probability that the robot is at position l given that the robot's sensor inputs i.

The question is, how can a term such as $p(r_t = l \mid i_t)$ be simplified to its constituent parts so that it can be computed? The answer lies in the product rule, which states

$$p(A \wedge B) = p(A \mid B)\,p(B) \tag{5.18}$$

Equation (5.18) is intuitively straightforward, as the probability of both A and B being true is being related to B being true *and* the other being conditionally true. But you should be able to convince yourself that the alternate equation is equally correct:

$$p(A \cap B) = p(B \mid A) \, p(A) \tag{5.19}$$

Using equations (5.18) and (5.19) together, we can derive the Bayes formula for com- puting $p(A \mid B)$:

$$p(A \mid B) = \frac{p(B \mid A) \, p(A)}{p(B)} \tag{5.20}$$

We use the Bayes rule to compute the robot's new belief state as a function of its sensory inputs and its former belief state. But to do this properly, we must recall the basic goal of the Markov localization approach: a discrete set of possible robot positions L are repre- sented. The belief state of the robot must assign a probability $p(r_t = l)$ for each location l in L .

The *See* function described in equation (5.17) expresses a mapping from a belief state and sensor input to a refined belief state. To do this, we must update the probability asso- ciated with each position l in L , and we can do this by directly applying the Bayes formula to every such l . In denoting this, we will stop representing the temporal index t for sim- plicity and will further use $p(l)$ to mean $p(r = l)$:

$$p(l \mid i) = \frac{p(i \mid l)\, p(l)}{p(i)} \tag{5.21}$$

The value of $p(i \mid l)$ is key to equation (5.21), and this probability of a sensor input at each robot position must be computed using some model. An obvious strategy would be to consult the robot's map, identifying the probability of particular sensor readings with each possible map position, given knowledge about the robot's sensor geometry and the mapped environment. The value of $p(l)$ is easy to recover in this case. It is simply the probability $p(r = l)$ associated with the belief state before the perceptual update process. Finally, note that the denominator $p(i)$ does not depend upon l; that is, as we apply equation (5.21) to all positions l in L, the denominator never varies. Because it is effectively constant, in practice this denominator is usually dropped and, at the end of the perception update step, all probabilities in the belief state are re-normalized to sum at 1.0.

Now consider the *Act* function of equation (5.16). *Act* maps a former belief state and encoder measurement (i.e., robot action) to a new belief state. In order to compute the prob- ability of position l in the new belief state, one must integrate over all the possible ways in which the robot may have reached l according to the potential positions expressed in the former belief state. This is subtle but fundamentally important. The same location l can be reached from multiple source locations with the same encoder measurement o because the encoder measurement is uncertain. Temporal indices are required in this update equation:

$$p(l_t \mid o_t) = \int p(l_t \mid l'_{t-1}, o_t)\, p(l'_{t-1} \mid o_t)\, dl'_{t-1} \quad (5.22)$$

Thus, the total probability for a specific position l is built up from the individual con- tributions from every location l' in the former belief state given encoder measurement o. Equations (5.21) and (5.22) form the basis of Markov localization, and they incorporate the *Markov assumption*. Formally, this means that their output is a function only of the robot's previous state and its most recent actions (odometry) and perception. In a general, non-Markovian situation, the state of a system depends upon all of its history. After all, the values of a robot's sensors at time t do not really depend only on its position at time t. They depend to some degree on the trajectory of the robot over time; indeed, on the entire history of the robot. For example, the robot could have experienced a serious collision recently that has biased the sensor's behavior. By the same token, the position of the robot at time t does not really depend only on its position at time $t - 1$ and its odometric measurements. Due to its history of motion, one wheel may have worn more than the other, causing a left-turning bias over time that affects its current position.

So the Markov assumption is, of course, not a valid assumption. However the Markov assumption greatly simplifies tracking, reasoning, and planning and so it is an approxima- tion that continues to be extremely popular in mobile robotics.





Figure 5.20

Dervish exploring its environment.

5.6.2.2 Case study 1: Markov localization using a topological map

A straightforward application of Markov localization is possible when the robot's environ- ment representation already provides an appropriate decomposition. This is the case when the environmental representation is purely topological.

Consider a contest in which each robot is to receive a topological description of the envi- ronment. The description would include only the connectivity of hallways and rooms, with no mention of geometric distance. In addition, this supplied *map* would be imperfect, con- taining several false arcs (e.g., a closed door). Such was the case for the 1994 American Association for

Artificial Intelligence (AAAI) National Robot Contest, at which each robot's mission was to use the supplied map and its own sensors to navigate from a chosen starting position to a target room.

Dervish, the winner of this contest, employed probabilistic Markov localization and used a multiple-hypothesis belief state over a topological environmental representation. We now describe Dervish as an example of a robot with a discrete, topological representation and a probabilistic localization algorithm.

Dervish, shown in figure 5.20, includes a sonar arrangement custom-designed for the 1994 AAAI National Robot Contest. The environment in this contest consisted of a recti-

linear indoor office space filled with real office furniture as obstacles. Traditional sonars were arranged radially around the robot in a ring. Robots with such sensor configurations are subject to both tripping over short objects below the ring and to decapitation by tall objects (such as ledges, shelves, and tables) that are above the ring.

Dervish's answer to this challenge was to arrange one pair of sonars diagonally upward to detect ledges and other overhangs. In addition, the diagonal sonar pair also proved to ably detect tables, enabling the robot to avoid wandering underneath tall tables. The remaining sonars were clustered in sets of sonars, such that each individual transducer in the set would be at a slightly varied angle to minimize specularity. Finally, two sonars near the robot's base were positioned to detect low obstacles, such as paper cups, on the floor.

We have already noted that the representation provided by the contest organizers was purely topological, noting the connectivity of hallways and rooms in the office environ- ment. Thus, it would be appropriate to design Dervish's perceptual system to detect match- ing perceptual events: the detection and passage of connections between hallways and offices.

This *abstract* perceptual system was implemented by viewing the trajectory of sonar strikes to the left and right sides of Dervish over time. Interestingly, this perceptual system would use time alone and no concept of encoder value to trigger perceptual events. Thus, for instance, when the robot detects a 7 to 17 cm indentation in the width of the hallway for more than 1 second continuously, a *closed door* sensory event is triggered. If the sonar strikes jump well beyond 17 cm for more than 1 second, an *open door* sensory event trig- gers.

To reduce coherent reflection sensor noise (see section 4.1.6) associated with Dervish's sonars, the robot would track its angle relative to the hallway centerline and completely suppress sensor events when its angle to the hallway exceeded 9 degrees. Interestingly, this would result in a conservative perceptual system that frequently misses features, particu- larly when the hallway is crowded with obstacles that Dervish must negotiate. Once again, the conservative nature of the perceptual system, and in particular its tendency to issue false negatives, would point to a probabilistic solution to the localization problem so that a com- plete trajectory of perceptual inputs could be considered.

Dervish's environmental representation was a discrete topological map, identical in abstraction and information to the map provided by the contest organizers. Figure 5.21 depicts a geometric representation of a typical office environment overlaid with the topo- logical map for the same office environment. Recall that for a topological representation the key decision involves assignment of nodes and connectivity between nodes (see section 5.5.2). As shown on the left in figure 5.21 Dervish uses a topology in which node bound- aries are marked primarily by doorways (and hallways and foyers). The topological graph shown on the right depicts the information captured in the example shown.

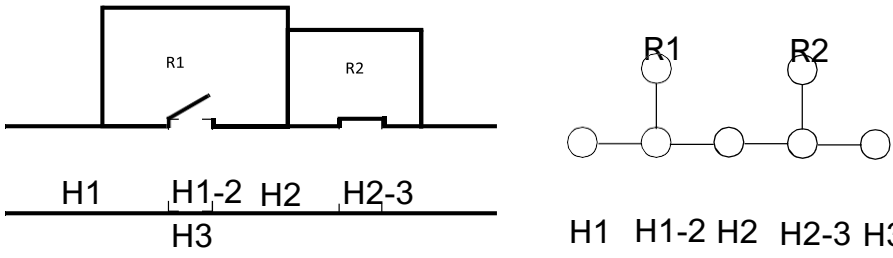


Figure 5.21

A geometric office environment (left) and its topological analog (right).

Note that in this particular topological model arcs are zero-length while nodes have spa- tial expansiveness and together cover the entire space. This particular topological representation is particularly apt for Dervish given its task of navigating through hallways into a specific room and its perceptual capability of recognizing discontinuities in hallway walls. In order to represent a specific belief state, Dervish associated with each topological node *n* a probability that the robot is at a physical position within the boundaries of *n* : $p(r_t = n)$. As will become clear below, the probabilistic update used by Dervish was approximate, therefore technically one should refer to the resulting values as *likelihoods* rather than probabilities.

Table 5.1

Dervish's certainty matrix.

	Wall	Closed door	Open door	Open hallway	Foyer
Nothing detected	0.70	0.40	0.05	0.001	0.30
Closed door detected	0.30	0.60	0	0	0.05
Open door detected	0	0	0.90	0.10	0.15
Open hallway detected	0	0	0.001	0.90	0.50

The perception update process for Dervish functions precisely as in equation (5.21). Per- ceptual events are generated asynchronously, each time the feature extractor is able to recognize a large scale feature (e.g., doorway, intersection) based on recent ultrasonic values. Each perceptual event consists of a percept-pair (a feature on one side of the robot or two features on both sides).

Given a specific percept-pair i , equation (5.21) enables the likelihood of each possible position n to be updated using the formula:

$$p(n \mid i) = \frac{p(i \mid n)\, p(n)}{p(i)} \tag{5.23}$$

The value of $p(n)$ is already available from the current belief state of Dervish, and so the challenge lies in computing $p(i \mid n)$. The key simplification for Dervish is based upon the realization that, because the feature extraction system only extracts four total features and because a node contains (on a single side) one of five total features, every possible com- bination of node type and extracted feature can be represented in a 4 x 5 table.

Dervish's *certainty matrix* (show in table 5.1) is just this lookup table. Dervish makes the simplifying assumption that the performance of the feature detector (i.e., the probability that it is correct) is only a function of the feature extracted and the actual feature in the node. With this assumption in hand, we can populate the *certainty matrix* with confidence esti- mates for each possible pairing of perception and node type. For each of the five world fea- tures that the robot can encounter (wall, closed door, open door, open hallway-and foyer) this matrix assigns a likelihood for each of the three one-sided percepts that the sensory system can issue. In addition, this matrix assigns a likelihood that the sensory system will fail to issue a perceptual event altogether (*nothing detected*).

For example, using the specific values in table 5.1, if Dervish is next to an open hallway, the likelihood of mistakenly recognizing it as an open door is 0.10. This means that for any node n that is of type *open hallway* and for the sensor value i =*open door,* $p(i \mid n) = 0.10$. Together with a specific topological map, the certainty matrix enables straightforward computation of $p(i \mid n)$

during the perception update process.

For Dervish's particular sensory suite and for any specific environment it intends to nav- igate, humans generate a specific certainty matrix that loosely represents its perceptual con- fidence, along with a global measure for the probability that any given door will be closed versus opened in the real world.

Recall that Dervish has no encoders and that perceptual events are triggered asynchro- nously by the feature extraction processes. Therefore, Dervish has no action update step as depicted by equation (5.22). When the robot does detect a perceptual event, multiple per- ception update steps will need to be performed to update the likelihood of every possible robot position given Dervish's former belief state. This is because there is a chance that the robot has traveled *multiple* topological nodes since its previous perceptual event (i.e., false- negative errors). Formally, the perception update formula for Dervish is in reality a combi- nation of the general form of action update and perception update. The likelihood of posi- tion n given perceptual event i is calculated as in equation (5.22):

$$p(n_t \mid i_t) = \frac{p(n_t \mid n'_{t-i}, i_t)}{\int p(n_t \mid n'_{t-i}) \, dn'_{t-i}} \tag{5.24}$$

The value of $p(n'_{t-i})$ denotes the likelihood of Dervish being at position n' as repre- sented by Dervish's former belief state. The temporal subscript $t - i$ is used in lieu of $t - 1$

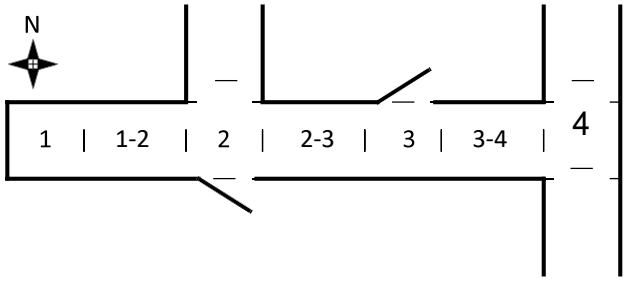


Figure 5.22

A realistic indoor topological environment.

because for each possible position n' the discrete topological distance from n' to n can vary depending on the specific topological map. The calculation of $p(n_t \mid n'_{t-i}, i_t)$ is performed by multiplying the probability of generating perceptual event i at position n by the probability of having failed to generate perceptual events at all nodes between n' and n:

$$p(n_t \mid n'_{t-i}, i_t) = p(i_t \mid n_t) \cdot p(n_t \mid n_{t-1}) \cdot p(n_t \mid n_{t-2}) \cdots p(n_t \mid n_{t-i+1}) \quad (5.25)$$

For example (figure 5.22), suppose that the robot has only two nonzero nodes in its belief state, {1-2, 2-3}, with likelihoods associated with each possible position: $p(1-2) = 1.0$ and

$p(2-3) = 0.2$. For simplicity assume the robot is facing east with certainty. Note that the likelihoods for nodes 1-2 and 2-3 do not sum to 1.0. These values are not formal probabilities, and so computational effort is minimized in Dervish by avoid- ing normalization altogether. Now suppose that a perceptual event is generated: the robot detects an open hallway on its left and an open door on its right simultaneously.

State 2-3 will progress potentially to states 3, 3-4, and 4. But states 3 and 3-4 can be eliminated because the likelihood of detecting an open door when there is only wall is zero. The likelihood of reaching state 4 is the product of the initial likelihood for state 2-3, 0.2, the likelihood of not detecting anything at node 3, (a), and the likelihood of detecting a hall- way on the left and a door on the right at node 4, (b). Note that we assume the likelihood of detecting nothing at node 3-4 is 1.0 (a simplifying approximation).

(a) occurs only if Dervish fails to detect the door on its left at node 3 (either closed or open), $(0.6 \times 0.4 + (1 - 0.6) \times 0.05)$, and correctly detects nothing on its right, 0.7.

(b) occurs if Dervish correctly identifies the open hallway on its left at node 4, 0.90, and mistakes the right hallway for an open door, 0.10.

The final formula, $0.2 \times (0.6 \times 0.4 + 0.4 \times 0.05) \times 0.7 \times (0.9 \times 0.1)$, yields a likelihood of 0.003 for state 4. This is a partial result for $p(4)$ following from the prior belief state node 2-3.

Turning to the other node in Dervish's prior belief state, 1-2 will potentially progress to states 2, 2-3, 3, 3-4, and 4. Again, states 2-3, 3, and 3-4 can all be eliminated since the like- lihood of detecting an open door when a wall is present is zero. The likelihood of state 2 is the product of the prior likelihood for state 1-2, (1.0), the likelihood of detecting the door on the right as an open door, $(0.6 \times 0 + 0.4 \times 0.9)$, and the likelihood of correctly detecting an open hallway to the left, 0.9. The likelihood for being at state 2 is then $1.0 \times 0.4 \times 0.9 \times 0.9 = 0.3$. In addition, 1-2 progresses to state 4 with a certainty factor of 4.3×10^{-6}, which is added to the certainty factor above to bring the total for state 4 to 0.00328. Dervish would therefore track the new belief state to be {2, 4}, assigning a very high likelihood to position 2 and a low likelihood to position 4.

Empirically, Dervish's map representation and localization system have proved to be sufficient for navigation of four indoor office environments: the artificial office environ- ment created explicitly for the 1994 National Conference on Artificial Intelligence; and the psychology, history, and computer science departments at Stanford University. All of these experiments were run while providing Dervish with no notion of the distance between adja- cent nodes in its topological map. It is a demonstration of the power of probabilistic local- ization that, in spite of the tremendous lack of action and encoder information, the robot is able to navigate several real-world office buildings successfully.

One open question remains with respect to Dervish's localization system. Dervish was not just a localizer but also a navigator. As with all multiple hypothesis systems, one must ask the question, how does the robot decide how to move, given that

it has multiple possible robot positions in its representation? The technique employed by Dervish is a common technique in the mobile robotics field: plan the robot's actions by assuming that the robot's actual position is its most likely node in the belief state. Generally, the most likely position is a good measure of the robot's actual world position. However, this technique has shortcomings when the highest and second highest most likely positions have similar values. In the case of Dervish, it nonetheless goes with the highest-likelihood position at all times, save at one critical juncture. The robot's goal is to enter a target room and remain there. Therefore, from the point of view of its goal, it is critical that Dervish finish navigating only when the robot has strong confidence in being at the correct final location. In this particular case, Dervish's execution module refuses to enter a room if the gap between the most likely position and the second likeliest position is below a preset threshold. In such a case, Dervish will actively plan a path that causes it to move further down the hallway in an attempt to collect more sensor data and thereby increase the relative likelihood of one position in the multiple-hypothesis belief state.

Although computationally unattractive, one can go further, imagining a planning system for robots such as Dervish for which one specifies a *goal belief state* rather than a goal posi- tion. The robot can then reason and plan in order to achieve a goal confidence level, thus explicitly taking into account not only robot position but also the measured likelihood of

each position. An example of just such a procedure is the sensory uncertainty field of Latombe [141], in which the robot must find a trajectory that reaches its goal while maxi- mizing its localization confidence on-line.

5.6.2.3 Case study 2: Markov localization using a grid map

The major weakness of a purely topological decomposition of the environment is the reso- lution limitation imposed by such a granular representation. The position of the robot is usually limited to the resolution of a single node in such cases, and this may be undesirable for certain applications.

In this case study, we examine the work of Burgard and colleagues [49, 50] in which far more precise navigation is made possible using a grid-based representation while still employing the Markov localization technique.

The robot used by this research, Rhino, is an RWI B24 robot with twenty-four sonars and two Sick laser rangefinders. Clearly, at the sensory level this robot accumulates greater and more accurate range data than is possible with the handful of sonar sensors mounted on Dervish. In order to make maximal use of these fine-grained sensory data, Rhino uses a 2D geometric environmental representation of free and occupied space. This metric map is tes- sellated regularly into a *fixed decomposition* grid with each cell occupying 4 to 64 cm in various instantiations.

Like Dervish, Rhino uses multiple-hypothesis belief representation. In line with the far improved resolution of the environmental representation, the belief state representation of Rhino consists of a $15 \times 15 \times 15$ 3D array representing the probability of 15^3 possible robot positions (see figure 5.23). The resolution of the array is 15 cm × 15 cm × 1°. Note that unlike Dervish, which assumes its

orientation is approximate and known, Rhino explicitly represents fine-grained alternative orientations, and so its belief state formally represents three degrees of freedom. As we have stated before, the resolution of the belief state representation must match the environmental representation in order for the overall system to function well.

Whereas Dervish made use of only perceptual events, ignoring encoder inputs and there- fore metric distance altogether, Rhino uses the complete Markov probabilistic localization approach summarized in section 5.6.2.1, including both an explicit action update phase and a perception update phase at every cycle.

The discrete Markov chain version of action update is performed because of the tessel- lated representation of position. Given encoder measurements o at time t , each updated position probability in the belief state is expressed as a sum over previous possible positions and the motion model:

$$P(l_t \mid o_t) = \sum_{l'} P(l_t \mid l'_{t-1}) \, p(l'_{t-1} \mid o_t) \tag{5.26}$$

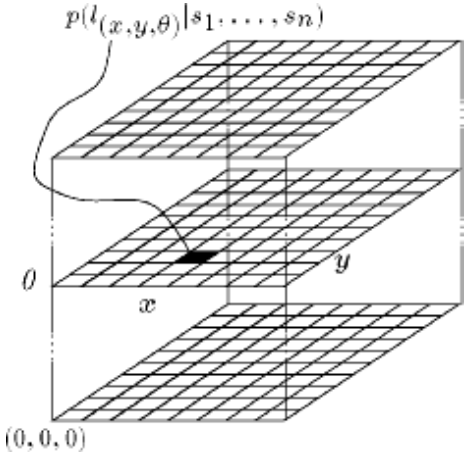


Figure 5.23

The belief state representation 3D array used by Rhino (courtesy of W. Burgard and S. Thrun).

Note that equation (5.26) is simply a discrete version of equation (5.22). The specific motion model used by Rhino represents the result of motion as a Gaussian that is bounded (i.e., the tails of the distribution are finite). Rhino's kinematic configuration is a three- wheel synchro-drive rather than a differential-drive robot. Nevertheless, the error ellipses depicted in figures 5.4 and 5.5 are similar to the Gaussian bounds that result from Rhino's motion model.

The perception model follows the Bayes formula precisely, as in equation (5.21). Given a range perception *i* the probability of the robot being at each location *l* is updated as fol- lows:

$$p(l \mid i) =$$

$$\frac{p(i \mid l)\, p(l)}{p(i)} \qquad (5.27)$$

Note that a denominator is used by Rhino, although the denominator is constant for vary- ing values of l . This denominator acts as a normalizer to ensure that the probability mea- sures in the belief state continue to sum to 1.

The critical challenge is, of course, the calculation of $p(i \mid l)$. In the case of Dervish, the number of possible values for i and l were so small that a simple table could suffice. How- ever, with the fine-grained metric representation of Rhino, the number of possible sensor readings and environmental geometric contexts is extremely large. Thus, Rhino computes

$p(i \mid l)$ directly using a model of the robot's sensor behavior, its position l, and the local environmental metric map around l.

The sensor model must calculate the probability of a specific perceptual measurement given that its likelihood is justified by known errors of the sonar or laser rangefinder sen- sors. Three key assumptions are used to construct this sensor model:

1. If an object in the metric map is detected by a range sensor, the measurement error can be described with a distribution that has a mean at the correct reading.
2. There should always be a nonzero chance that a range sensor will read any measurement value, even if this measurement disagrees sharply with the environmental geometry.
3. In contrast to the generic error described in (2), there is a specific failure mode in ranging sensors whereby the signal is absorbed or coherently reflected, causing the sensor's range measurement to be maximal. Therefore, there is a local peak in the probability density distribution at the maximal reading of a range sensor.

By validating these assumptions using empirical sonar trials in multiple environments, the research group has delivered to Rhino a conservative and powerful sensor model for its particular sensors.

Figure 5.24 provides a simple 1D example of the grid-based Markov localization algo- rithm. The robot begins with a flat probability density function for its possible location. In other words, it initially has no bias regarding position. As the robot encounters first one door and then a second door, the probability density function over possible positions becomes first multimodal and finally unimodal and sharply defined. The ability

of a Markov localization system to *localize* the robot from an initially lost belief state is its key distinguishing feature.

The resulting robot localization system has been part of a navigation system that has demonstrated great success both at the University of Bonn (Germany) and at a public museum in Bonn. This is a challenging application because of the dynamic nature of the environment, as the robot's sensors are frequently subject to occlusion due to humans gath- ering around the robot. Rhino's ability to function well in this setting is a demonstration of the power of the Markov localization approach.

Reducing computational complexity: randomized sampling. A great many steps are taken in real-world implementations such as Rhino in order to effect computational gains. These are valuable because, with an exact cell decomposition representation and use of raw sensor values rather than abstraction to features, such a robot has a massive computational effort associated with each perceptual update.

One class of techniques deserves mention because it can significantly reduce the com- putational overhead of techniques that employ fixed-cell decomposition representations.

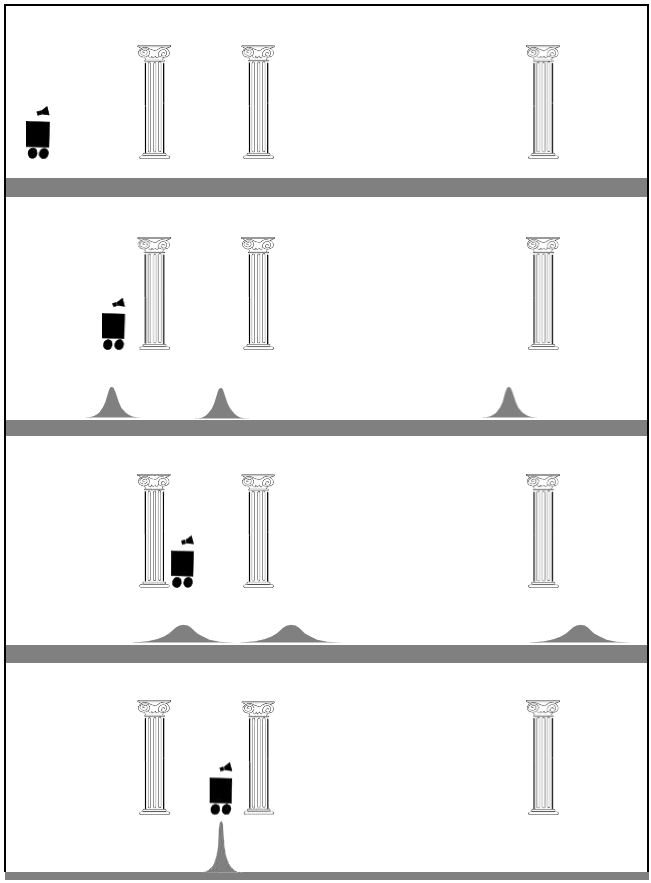


Figure 5.24

Improving belief state by moving.

The basic idea, which we call *randomized sampling,* is known alternatively as particle filter algorithms, condensation algorithms, and Monte Carlo algorithms [68, 144].

Irrespective of the specific technique, the basic algorithm is the

same in all these cases. Instead of representing *every* possible robot position by representing the complete and cor- rect belief state, an approximate belief state is constructed by representing only a *subset* of the complete set of possible locations that should be considered.

For example, consider a robot with a complete belief state of 10,000 possible locations at time *t*. Instead of tracking and updating all 10,000 possible locations based on a new sensor measurement, the robot can select only 10% of the stored locations and update only those locations. By weighting this sampling process with the probability values of the loca- tions, one can bias the system to generate more samples at local peaks in the probability

density function. So the resulting 1000 locations will be concentrated primarily at the high- est probability locations. This biasing is desirable, but only to a point.

We also wish to ensure that *some* less likely locations are tracked, as otherwise, if the robot does indeed receive unlikely sensor measurements, it will fail to localize. This *ran- domization* of the sampling process can be performed by adding additional samples from a flat distribution, for example. Further enhancements of these randomized methods enable the number of statistical samples to be varied on the fly, based, for instance, on the ongoing localization confidence of the system. This further reduces the number of samples required on average while guaranteeing that a large number of samples will be used when necessary [68].

These sampling techniques have resulted in robots that function indistinguishably as compared to their full belief state set ancestors, yet use computationally a fraction of the resources. Of course, such sampling has a penalty: completeness. The probabilistically complete nature of Markov localization is violated by these sampling approaches because the robot is failing to update *all* the nonzero probability locations, and thus there is a danger that the robot, due to an unlikely but correct sensor reading, could become truly lost. Of course, recovery from a lost state is feasible just as with all Markov localization techniques.

5.6.3 Kalman filter localization

The Markov localization model can represent any probability density function over robot position. This approach is very general but, due to its generality, inefficient. Consider instead

the key demands on a robot localization system. One can argue that it is not the exact replication of a probability density curve but the *sensor fusion* problem that is key to robust localization. Robots usually include a large number of heterogeneous sensors, each providing clues as to robot position and, critically, each suffering from its own failure modes. Optimal localization should take into account the information provided by all of these sensors. In this section we describe a powerful technique for achieving this sensor fusion, called the Kalman filter. This mechanism is in fact more efficient than Markov localization because of key simplifications when representing the probability density func- tion of the robot's belief state and even its individual sensor readings, as described below. But the benefit of this simplification is a resulting *optimal recursive data-processing algo- rithm*. It incorporates all information, regardless of precision, to estimate the current value of the variable of interest (i.e., the robot's position). A general introduction to Kalman filters can be found in [106] and a more detailed treatment is presented in [3].

Figure 5.25 depicts the general scheme of Kalman filter estimation, where a system has a control signal and system error sources as inputs. A measuring device enables measuring some system states with errors. The Kalman filter is a mathematical mechanism for produc- ing an optimal estimate of the system state based on the knowledge of the *system* and the *measuring device*, the description of the system noise and measurement errors and the

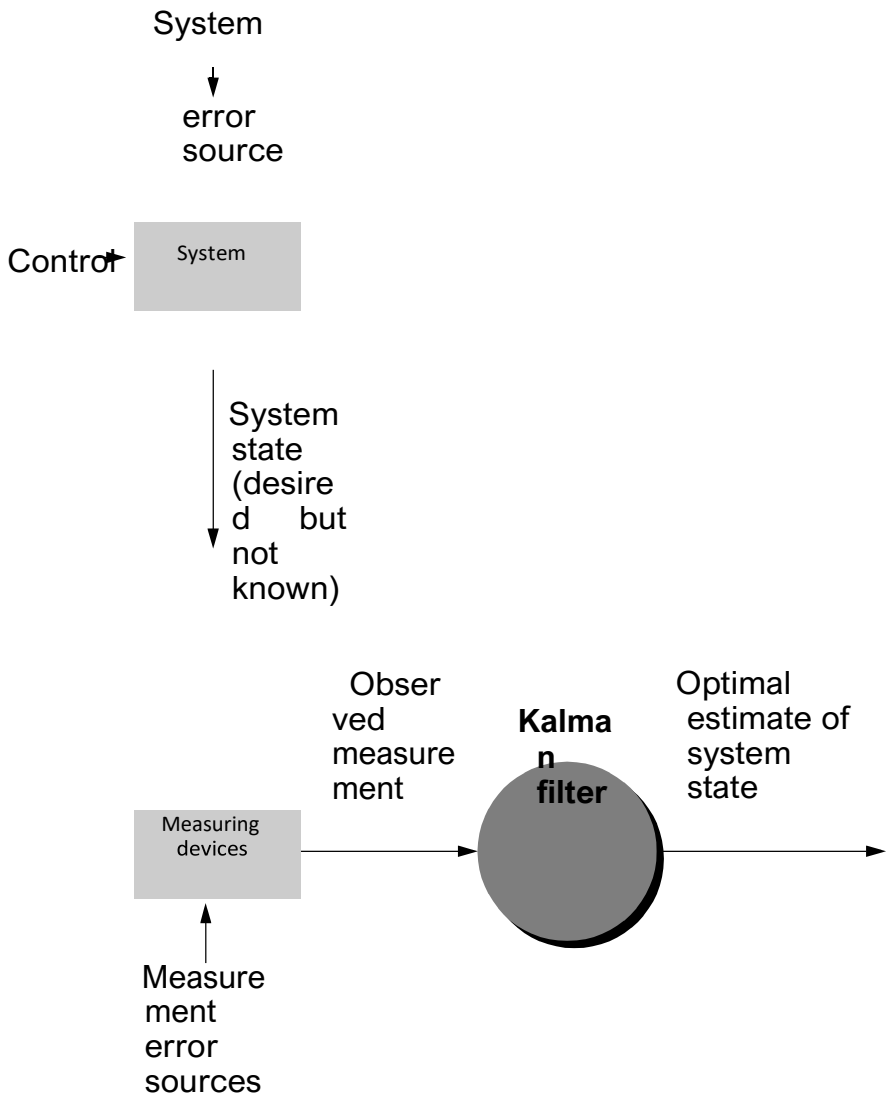


Figure 5.25

Typical Kalman filter application [106].

uncertainty in the dynamics models. Thus the Kalman filter *fuses* sensor signals and system knowledge in an optimal way. Optimality depends on the criteria chosen to evaluate the performance and on the assumptions. Within the Kalman filter theory the system is assumed to be *linear* and *white* with *Gaussian* noise. As we have discussed earlier, the assumption of Gaussian error is invalid for our mobile robot applications but, nevertheless, the results are extremely useful. In other engineering disciplines, the Gaussian error assumption has in some cases been shown to be quite accurate [106].

We begin with a section that introduces Kalman filter theory, then we present an appli- cation of that theory to the problem of mobile robot localization (5.6.3.2). Finally, section 5.6.3.2 presents a case study of a mobile robot that navigates indoor spaces by virtue of Kalman filter localization.

5.6.3.1 Introduction to Kalman filter theory

The basic Kalman filter method allows multiple measurements to be incorporated opti- mally into a single estimate of state. In demonstrating this, first we make the simplifying assumption that the state does not change (e.g., the robot does not move) between the acqui- sition of the first and second measurement. After presenting this static case, we can intro- duce *dynamic prediction* readily.

Static estimation. Suppose that our robot has two sensors, an ultrasonic range sensor and a laser rangefinding sensor. The laser rangefinder provides far richer and more accurate data for localization, but it will suffer from failure modes that differ from that of the sonar ranger. For instance, a glass wall will be transparent to the laser but, when measured head- on, the sonar will provide an accurate reading. Thus we wish to combine the information provided by the two sensors, recognizing that such sensor fusion, when done in a principled way, can only result in information gain.

The Kalman filter enables such fusion extremely efficiently, as long as we are willing to approximate the error characteristics of these sensors with unimodal, zero-mean, Gaussian noise. Specifically, assume we have taken two measurements, one with the sonar sensor at time k and one with the laser rangefinder at time $k + 1$. Based on each measurement indi- vidually we can estimate the robot's position. Such an estimate derived from the sonar is

q_1 and the estimate of position based on the laser is q_2 . As a simplified way of character- izing the error associated with each of these estimates, we presume a (unimodal) Gaussian probability density curve and thereby associate one variance with each measurement: σ_1^2 and σ_2^2 . The two dashed probability densities in figure 5.26 depict two such measurements.

In summary, this yields two robot position estimates:

$$\hat{q}_1 = q_1 \text{ with variance } \sigma_1^2 \tag{5.28}$$

$$\hat{q}_2 = q_2$$

with variance σ_2^2. (5.29)

The question is, how do we *fuse* (combine) these data to get the best estimate $\hat{q}$ for the robot position? We are assuming that there was no robot motion between time k and time $k + 1$, and therefore we can directly apply the same weighted least-squares technique of equation (5.26) in section 4.3.1.1. Thus we write

$$S = \sum_{i=1}^{n} w_i (\hat{q} - q_i)^2 \tag{5.30}$$

with w_i being the weight of measurement i. To find the minimum error we set the deriv- ative of S equal to zero.

$$\frac{\partial S}{\partial \hat{q}} = \sum_{i=1}^{n} w_i (q - \hat{q}) = 2 \sum_{i=1}^{n} w_i (q - q_i) = 0 \tag{5.31}$$

$$\sum_{i=1}^{n} w_i \hat{q} - \sum_{i=1}^{n} w_i q_i = 0 \tag{5.32}$$

$$\hat{q} = \frac{\sum_{i=1}^{n} w_i q_i}{\sum_{i=1}^{n} w_i} \tag{5.33}$$

If we take as the weight w_i

$$w_i = \frac{1}{\sigma_i^2} \tag{5.34}$$

then the value of $\hat{q}$ in terms of two measurements can be defined as follows:

$$\frac{1}{\sigma_1^2} q_1 + \frac{1}{\sigma_2^2} q_2$$

$$\hat{q} = \frac{\frac{1}{\sigma_1^2}}{\frac{1}{\sigma_1^2} + \frac{1}{\sigma_2^2}} q_1 + \frac{\frac{1}{\sigma_2^2}}{\frac{1}{\sigma_1^2} + \frac{1}{\sigma_2^2}} q_2 \tag{5.35}$$

$$\frac{1}{\sigma^2} = \frac{1}{\sigma_1^2} + \frac{1}{\sigma_2^2} = \frac{\sigma_1^2 + \sigma_2^2}{\sigma_1^2 \sigma_2^2}; \quad \sigma^2 = \frac{\sigma_1^2 \sigma_2^2}{\sigma_1^2 + \sigma_2^2} \tag{5.36}$$

Note that from equation (5.36) we can see that the resulting variance σ^2 is less than all the variances σ^2 of the individual measurements. Thus the uncertainty of the position esti- mate has been decreased by combining the two measurements. The solid probability den- sity curve represents the result of the Kalman filter in figure 5.26, depicting this result. Even poor measurements, such as are provided by the sonar, will only increase the precision of an estimate. This is a result that we expect based on information theory.

Equation (5.35) can be rewritten as

$$\hat{q} = q_1 + \frac{\sigma_1^2}{\sigma_1^2 + \sigma_2^2}(q_2 - q_1) \tag{5.37}$$

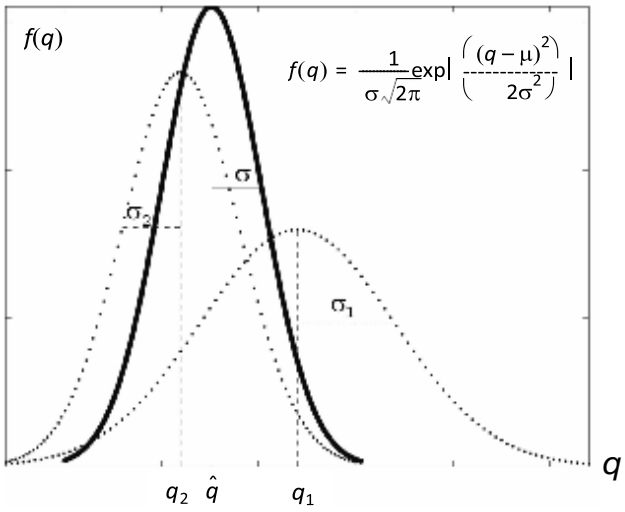


Figure 5.26

Fusing probability density of two estimates [106].

or, in the final form that is used in Kalman filter implementation,

$$\hat{x}_{k+1} = \hat{x}_k + K_{k+1}[z_{k+1} - \hat{x}_k] \tag{5.38}$$

where

$$K = \frac{\sigma^2}{\quad} \qquad ; \sigma^2 = \sigma^2$$

; $\sigma_z^2 = \sigma^2$

$$k+1 \qquad \sigma_k^2 + \sigma_z^2 \qquad k \qquad 1 \qquad z \qquad 2 \tag{5.39}$$

Equation (5.38) tells us, that the best estimate $\hat{x}_{k+1}$ of the state x_{k+1} at time $k+1$ is

equal to the best prediction of the value $\hat{x}_k$ before the new measurement z_{k+1} is taken, plus a correction term of an optimal weighting value times the difference between z_{k+1} and the best prediction $\hat{x}_k$ at time $k+1$. The updated variance of the state $\hat{x}_{k+1}$ is given using equation (5.36)

$$\sigma_{k+1}^2 = \sigma^2 - K_k \sigma_k^2 + 1 \tag{5.40}$$

The new, fused estimate of robot position provided by the Kalman filter is again subject to a Gaussian probability density curve. Its mean and variance are simply functions of the inputs' means and variances. Thus the Kalman filter provides both a compact, simplified representation of uncertainty and an extremely efficient technique for combining heteroge- neous estimates to yield a new estimate for our robot's position.

Dynamic estimation. Next, consider a robot that moves between successive sensor mea- surements. Suppose that the motion of the robot between times k and $k + 1$ is described by the velocity u and the noise w which represents the uncertainty of the actual velocity:

$$\frac{dx}{dt} = u + w \tag{5.41}$$

If we now start at time k , knowing the variance σ^2 of the robot position at this time and knowing the variance σ_k^2 of the motion, we obtain for the time just when the measurement is taken,

$$\hat{x}_{k+1} = \hat{x}_k + u(t_{k+1} - t_k) \tag{5.42}$$

$$\sigma_t^2 = \sigma^2 + \sigma_w^2 t \tag{5.43}$$

$$k\square \quad k \quad w \quad k+1\ k$$

where

$$t_k' = t_{k+1};$$

t_{k+1} and t_k are the time in seconds at $k+1$ and k respectively.

Thus $\hat{x}_{k\square}$ is the optimal prediction of the robot's position just as the measurement is

taken at time $k+1$. It describes the growth of position error until a new measurement is taken (figure 5.27).

We can now rewrite equations (5.38) and (5.39) using equations (5.42) and (5.43).

$$\hat{x}_{k+1} = \hat{x}_{k\square} + K_{k+1}\square z_{k+1} - \hat{x}_{k\square}\square$$

$$= \square\hat{x}_k + u\square t_{k+1} - t_k\square\square + K_{k+1}\square z_k + 1 - \hat{x}_k - u\square t_{k+1} - t_k\square\square \quad (5.44)$$

$$K_{k+1} = \frac{\square^2_{k\square}}{\sigma^2 + \square^2_{k\square}} = \frac{\square^2 + \square^2\square t - t\ \square}{\square^2_k + \square^2_w\ \square t_{k+1} - t_k\ \square + \square^2_z} \quad (5.45)$$

$$2$$

$$k\square$$

$$z$$

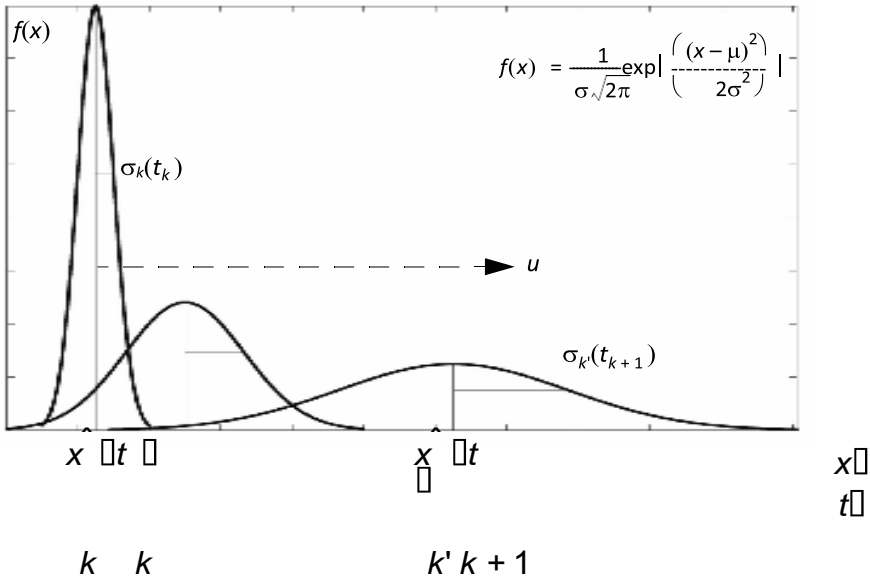


Figure 5.27

Propagation of probability density of a moving robot [106].

The optimal estimate at time $k + 1$ is given by the last estimate at k and the estimate of the robot motion including the estimated movement errors.

By extending the above equations to the vector case and allowing time-varying param- eters in the system and a description of noise, we can derive the Kalman filter localization algorithm.

5.6.3.2 Application to mobile robots: Kalman filter localization

The Kalman filter is an optimal and efficient sensor fusion

technique. Application of the Kalman filter to localization requires posing the robot localization problem as a sensor fusion problem. Recall that the basic probabilistic update of robot belief state can be seg- mented into two phases, *perception update* and *action update,* as specified by equations (5.21) and (5.22).

The key difference between the Kalman filter approach and our earlier Markov localiza- tion approach lies in the perception update process. In Markov localization, the entire per- ception, that is, the robot's set of instantaneous sensor measurements, is used to update each possible robot position in the belief state individually using the Bayes formula. In some cases, the perception is abstract, having been produced by a feature extraction mechanism, as in Dervish. In other cases, as with Rhino, the perception consists of raw sensor readings. By contrast, perception update using a Kalman filter is a multistep process. The robot's total sensory input is treated not as a monolithic whole but as a set of extracted features that

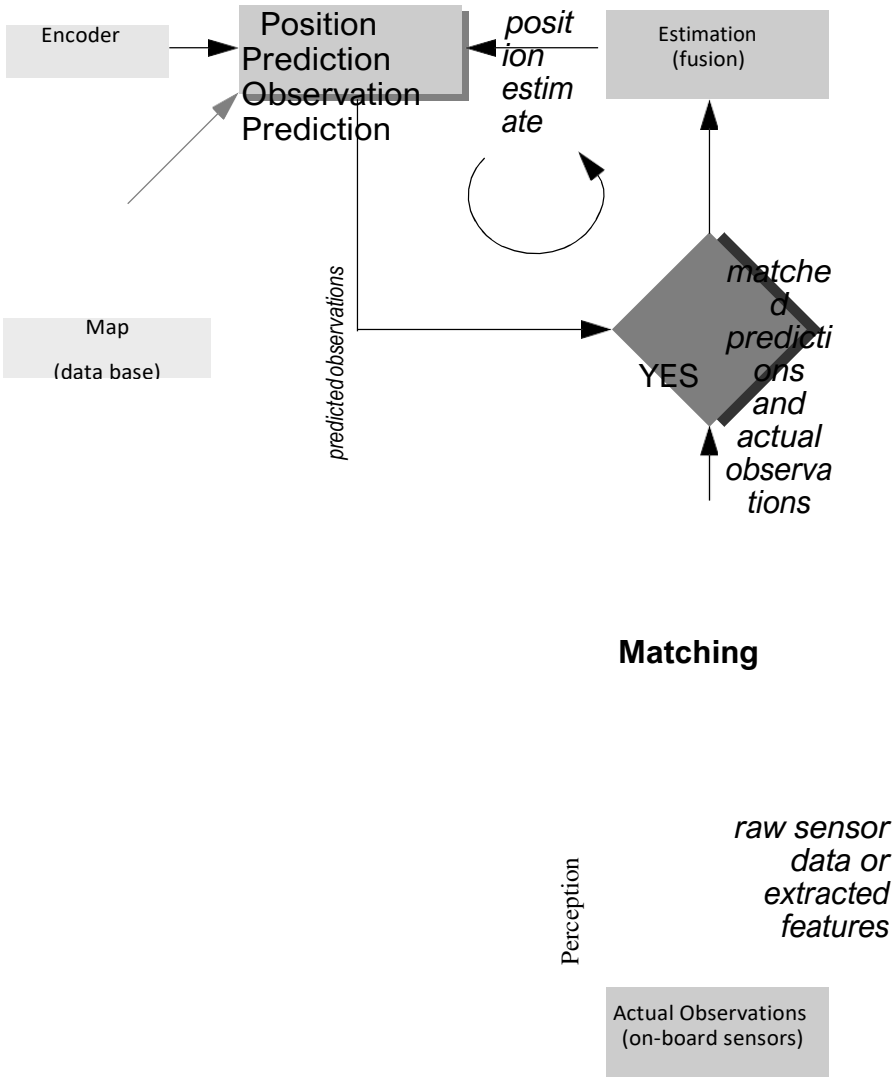


Figure 5.28

Schematic for Kalman filter mobile robot localization (see [23]).

each relate to objects in the environment. Given a set of possible features, the Kalman filter is used to fuse the distance estimate from each feature to a matching object in the map. Instead of carrying out this matching process for many possible robot locations individually as in the Markov approach, the Kalman filter accomplishes the same probabilistic update by treating the whole, unimodal, and Gaussian belief state at once. Figure 5.28 depicts the particular schematic for Kalman filter localization.

The first step is action update or *position prediction*, the straightforward application of a Gaussian error motion model to the robot's measured encoder travel. The robot then col- lects actual sensor data and extracts appropriate features (e.g., lines, doors, or even the value of a specific sensor) in the *observation* step. At the same time, based on its predicted position in the map, the robot generates a *measurement prediction* which identifies the fea- tures that the robot expects to find and the positions of those features. In *matching* the robot identifies the best pairings between the features actually extracted during observation and the expected features due to measurement prediction. Finally, the Kalman filter can fuse the information provided by all of these matches to update the robot belief state in *estimation*.

In the following sections these five steps are described in greater detail. The presentation is based on the work of Leonard and Durrant-Whyte [23, pp. 61–65].

1. **Robot position prediction.** The robot's position at timestep $k + 1$ is predicted based on its old location (at timestep k) and its movement due to the control input $u(k)$:

$$\hat{p}(k+1 \mid k) = f(\hat{p}(k \mid k), u(k)) \tag{5.46}$$

For the differential-drive robot, $\hat{p}(k+1 \mid k) = p'$ is derived in equations (5.6) and (5.7) respectively.

Knowing the plant and error model, we can also compute the variance $\Sigma_p(k+1 \mid k)$ asso- ciated with this prediction [see equation. (5.9), section 5.2.4]:

$$\Sigma_p(k+1 \mid k) = \nabla f_p \cdot \Sigma_p(k \mid k) \cdot \nabla f_p^T + \nabla f_u \cdot \Sigma_u \cdot \nabla f_u^T \tag{5.47}$$

This allows us to predict the robot's position and its uncertainty after a movement spec- ified by the control input $u(k)$. Note that the belief state is assumed to be Gaussian, and so we can characterize the belief state with just the two parameters $\hat{p}(k+1 \mid k)$ and $\Sigma_p(k+1 \mid k)$.

2. **Observation.** The second step it to obtain $Z(k+1)$ from the sensor measurements

robot at time $k + 1$. In this presentation, we assume that the observation is the result of a feature extraction process executed on the raw sensor data. Therefore, the observation con- sists of a set n_0 of single observations $z_j(k + 1)$ extracted from various sensors. Formally, each single observation can represent an extracted feature such as a line or door, or even a single, raw sensor value.

The parameters of the features are usually specified in the sensor frame and therefore in a local reference frame of the robot. However, for matching we need to represent the obser- vations and measurement predictions in the same frame (S) . In our presentation we will transform the measurement predictions from the global coordinate frame to the sensor frame (S) . This transformation is specified in the function h_i discussed in the next para- graph.

3. **Measurement prediction.** We use the predicted robot position $\hat{p}(k + 1 \mid k)$ and the map $M(k)$ to generate multiple predicted feature observations z_t . Each predicted feature has its position transformed into the sensor frame:

$$\hat{z}_i(k + 1) = h_i(z_t \mid \hat{p}(k + 1 \mid k)) \tag{5.48}$$

We can define the measurement prediction as the set containing all n_t predicted feature observations:

$$\hat{Z}(k+1) = \{\hat{z}_i(k+1) \mid 1 \le i \le n\} \tag{5.49}$$

The predicted state estimate $\hat{p}(k+1 \mid k)$ is used to compute the measurement Jacobian H_i for each prediction. As you will see in the example below, the function h_i is mainly a coordinate transformation between the world frame and the sensor frame.

4. **Matching.** At this point we have a set of actual, single observations, which are features in sensor space, and we also have a set of predicted features, also positioned in sensor space. The matching step has the purpose of identifying all of the single observations that match specific predicted features well enough to be used during the estimation process. In other words, we will, for a subset of the observations and a subset of the predicted features, find pairings that intuitively say "this observation is the robot's measurement of this predicted feature based on the map."

Formally, the goal of the matching procedure is to produce an assignment from obser- vations $z_j(k+1)$ to the targets z_t (stored in the map). For each measurement prediction for which a corresponding observation is found we calculate the innovation $v_{ij}(k+1)$. *Inno- vation* is a measure of the difference between the predicted and observed measurements:

$$\begin{aligned} v_{ij}(k+1) &= z_j(k+1) - \hat{z}_i(k+1) \\ &= z_j(k+1) - h_i(z_t, \hat{p}(k+1 \mid k)) \end{aligned} \tag{5.50}$$

The innovation covariance $\Omega_{IN,ij}(k+1)$ can be found by applying the error propagation law [section 4.2.2, equation (4.60)]:

$$\Omega_{IN,ij}(k+1) = h_i \, \Omega_p(k+1|k) \, h_i^T + \Omega_{R,i}(k+1) \tag{5.51}$$

where $\Omega_{R,i}(k+1)$ represents the covariance (noise) of the measurement $z_i(k+1)$.

To determine the validity of the correspondence between measurement prediction and observation, a *validation gate* has to be specified. A possible definition of the validation gate is the Mahalanobis distance:

$$v_{ij}^T(k+1) \, \Omega_{IN,ij}^{-1}(k+1) \, v_{ij}(k+1) \le g^2 \tag{5.52}$$

However, dependent on the application, the sensors, and the environment models, more sophisticated validation gates might be employed.

The validation equation is used to test observation $z_j(k+1)$ for membership in the validation gate for each predicted measurement. When a single observation falls in the valida- tion gate, we get a successful match. If one observation falls in multiple validation gates,

the best matching candidate is selected or multiple hypotheses are tracked. Observations that do not fall in the validation gate are simply ignored for localization. Such observations could have resulted from objects not in the map, such as new objects (e.g., someone places a large box in the hallway) or transient objects (e.g., humans standing next to the robot may form a line feature). One approach is to take advantage of such unmatched observations to populate the robot's map.

5. **Estimation: applying the Kalman filter.** Next we compute the best estimate $\hat{p}(k+1 \mid k+1)$ of the robot's position based on the position prediction and all the observa- tions at time $k+1$. To do this position update, we first stack the validated observations $z_j(k+1)$ into a single vector to form $z(k+1)$ and designate the composite innovation $v(k+1)$. Then we stack the measurement Jacobians $(h_i$ for each validated measurement together to form the composite Jacobian $(h$ and the measurement error (noise) vector $(R(k+1) = diag((R_i(k+1))$. We can then compute the composite innovation covariance $(_{IN}(k+1)$ according to equation (5.51) and by utilizing the well-known result [3] that the Kalman gain can be written as

$$K(k+1) = (p(k+1 \mid k)(h^T((^{-1}_{IN}(k+1) \tag{5.53}$$

we can update the robot's position estimate

$$\hat{p}(k+1 \mid k+1) = \hat{p}(k+1 \mid k) + K(k+1)(v(k+1) \tag{5.54}$$

with the associated variance

$$\Box\Box k+1\ k+1\Box = \Box\Box k+1\ k\Box - K\Box k+1\Box\ \Box\ \Box\Box k+1\Box\ \Box\ K^{T}\Box k+1\Box \quad (5.55)$$

For the 1D case and with $h_i\Box z_t\Box\ \hat{p}\ \Box k+1\ \ k\Box\Box = z_t$ we can show that this formula corre- sponds to the 1D case derived earlier

Equation (5.53) is simplified to

$$K\Box k+1\Box = \frac{\Box^2\Box k+1\ k\Box}{\Box^2\Box k+1\Box} \qquad \frac{\Box^2\Box k+1\ k\Box}{\Box^2\Box k+1\ k\Box + \Box^2\Box k+1\Box} \quad (5.56)$$

IN p R

corresponding to equation (5.45), and equation (5.54) simplifies to

$$\hat{p}(k+1 \mid k+1) = \hat{p}(k+1 \mid k) + K(k+1)\,v(k+1)$$

$$= \hat{p}(k+1 \mid k) + K(k+1)\,(z_j(k+1) - h_i(z_t, \hat{p}(k+1 \mid k))) \quad (5.57)$$

$$= \hat{p}(k+1 \mid k) + K(k+1)\,(z_j(k+1) - z_t)$$

corresponding to equation (5.44).

5.6.3.3 Case study: Kalman filter localization with line feature extraction

The Pygmalion robot at EPFL is a differential-drive robot that uses a laser rangefinder as its primary sensor [37, 38]. In contrast to both Dervish and Rhino, the environmental rep- resentation of Pygmalion is continuous and abstract: the map consists of a set of infinite lines describing the environment. Pygmalion's belief state is, of course, represented as a Gaussian distribution since this robot uses the Kalman filter localization algorithm. The value of its mean position $\hat{p}$ is represented to a high level of precision, enabling Pygmalion to localize with very high precision when desired. Below, we present details for Pygma- lion's implementation of the five Kalman filter localization steps. For simplicity we assume that the sensor frame (S) is equal to the robot frame (R). If not specified all the vectors are represented in the world coordinate system (W).

1. **Robot position prediction.** At the time increment k the robot is at position

$p(k) = \begin{bmatrix} x(k) \\ y(k) \\ \theta(k) \end{bmatrix}$ and its best position estimate is $\hat{p}(k \mid k)$. The control input

$u(k)$ drives the robot to the position $p(k+1)$ (figure 5.29).

The robot position prediction $\hat{p}(k+1 \mid k)$ at the time increment $k+1$ can be computed from the previous estimate $\hat{p}(k \mid k)$ and the odometric integration of the movement. For the differential drive that Pygmalion has we can use the model (odometry) developed in section 5.2.4:

$$\hat{p}(k+1 \mid k) = \hat{p}(k \mid k) + u(k) = \hat{p}(k \mid k) + \begin{bmatrix} \frac{\Delta s_r + \Delta s_l}{2} \cos\left(\theta + \frac{\Delta s_r - \Delta s_l}{2b}\right) \\ \frac{\Delta s_r + \Delta s_l}{2} \sin\left(\theta + \frac{\Delta s_r - \Delta s_l}{2b}\right) \\ \frac{\Delta s_r - \Delta s_l}{b} \end{bmatrix} \quad (5.58)$$

with the updated covariance matrix

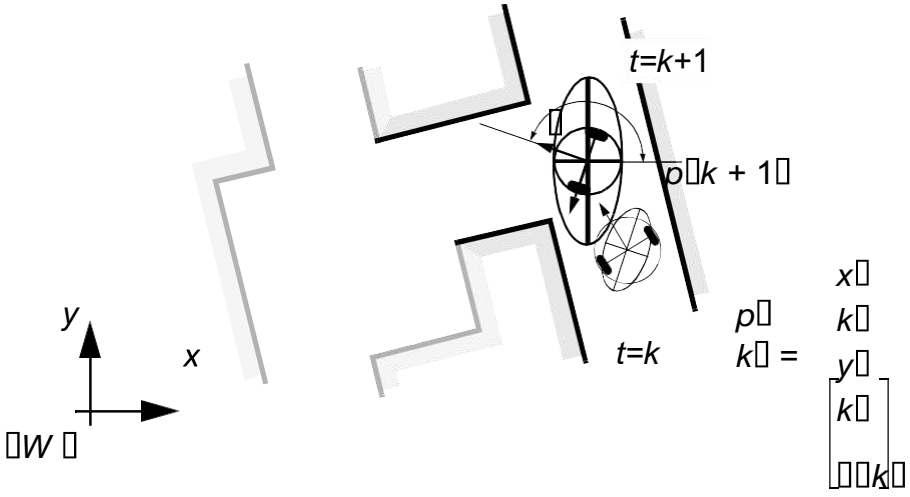


Figure 5.29

Prediction of the robot's position (thick) based on its former position (thin) and the executed move- ment. The ellipses drawn around the robot positions represent the uncertainties in the *x,y* direction (e.g.; 3). The uncertainty of the orientation is not represented in the picture.

$$\Sigma_{p}(k+1|k) = \nabla f_p \cdot \Sigma_p(k|k) \cdot \nabla f_p^T + \nabla f_u \cdot \Sigma_u \cdot \nabla f_u^T \qquad (5.59)$$

where

$$\Sigma_u = cov(s_r, s_l) = \begin{bmatrix} k_r |s_r| & 0 \\ 0 & k_l |s_l| \end{bmatrix}$$

(5.60)

2. **Observation.** For line-based localization, each single observation (i.e., a line feature) is extracted from the raw laser rangefinder data and consists of the two line parameters $\alpha_{0,j}$, $\alpha_{1,j}$ or α_j, r_j (figure 4.36) respectively. For a rotating laser rangefinder, a representation
in the polar coordinate frame is more appropriate and so we use this coordinate frame here:

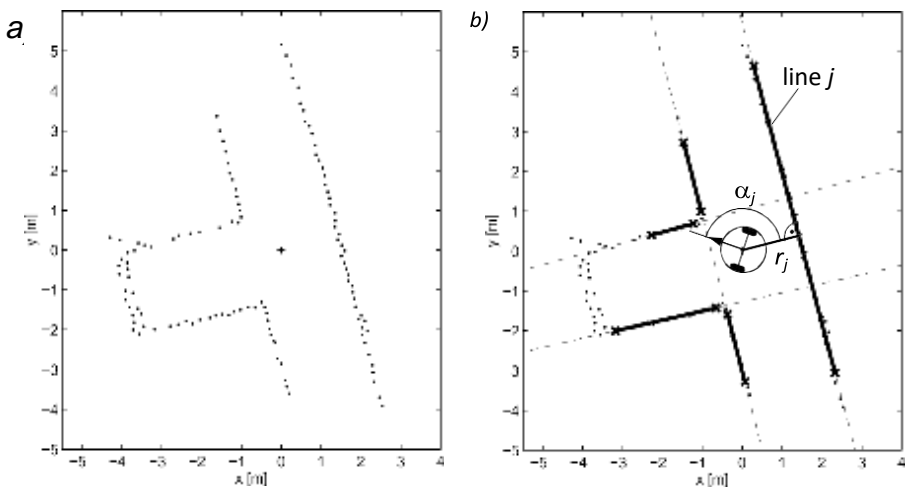
$$z_j(k+1) = \begin{bmatrix} \alpha_j \\ r_j \end{bmatrix}^R \tag{5.61}$$

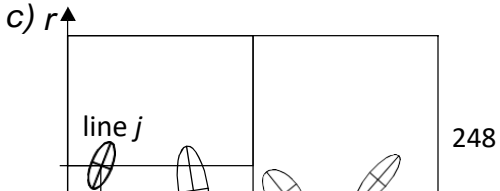
After acquiring the raw data at time k+1, lines and their uncertainties are extracted (fig- ure 5.30a, b). This leads to n_0 observed lines with $2n_0$ line parameters (figure 5.30c) and a covariance matrix for each line that can be calculated from the uncertainties of all the

measurement points contributing to each line as developed for line extraction in section 4.3.1.1:

$$\begin{bmatrix} \\ \\ \end{bmatrix}_r = \begin{bmatrix} \\ \\ \end{bmatrix} \qquad (5.62)$$

$$R_j \quad r \quad rr_j$$





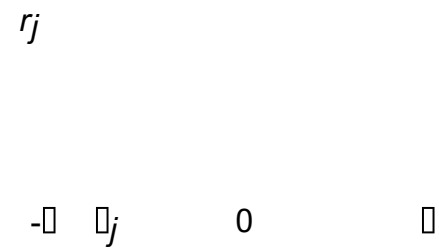


Figure 5.30

Observation: From the raw data (a) acquired by the laser scanner at time $k + 1$, lines are extracted (b). The line parameters α_j and r_j and its uncertainties can be represented in the model space (c).

3. **Measurement prediction.** Based on the stored map and the predicted robot position $\hat{p}(k \mid k)$, the measurement predictions of expected features $z_t(i)$ are generated (figure 5.31). To reduce the required calculation power, there is often an additional step that first selects the possible features, in this case lines, from the whole set of features in the map. These lines are stored in the map and specified in the world coordinate system (W). Therefore they need to be transformed to the robot frame (R):

$$W_z = \begin{bmatrix} W \\ \alpha_t(i) \\ R_z \end{bmatrix} \quad \overline{i} = \begin{bmatrix} R \\ \alpha_t(i) \\ r_t(i) \end{bmatrix} \tag{5.63}$$

$t(i) \quad r_t(i) \quad t(i) \quad r_t(i)$

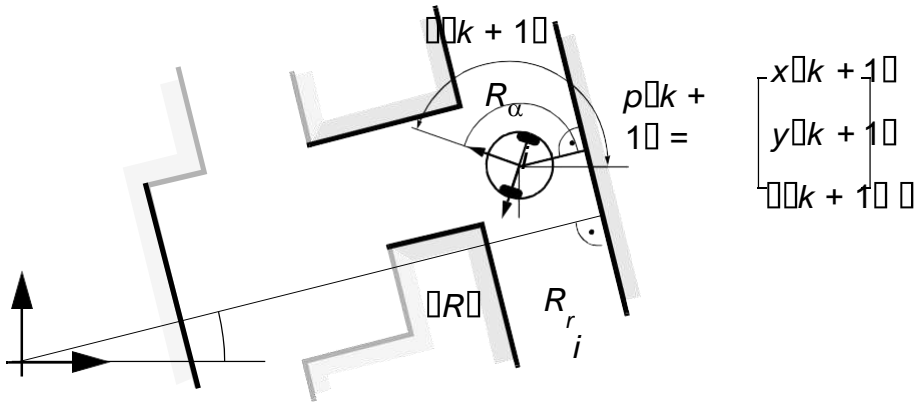
According to figure (5.31), the transformation is given by

$$\hat{z}_i(k+1) = \begin{bmatrix} R\ \alpha_t(i) \\ r_t(i) \end{bmatrix} = h_i(z_t(i), \hat{p}(k+1 \mid k))$$

$$= \begin{bmatrix} W \\ \alpha_t(i) - W\hat{\alpha}(k+1 \mid k) \\ W \\ r_t(i) - {}^w\hat{x}(k+1 \mid k)\cos({}^w\alpha + {}^w\hat{y}(k+1 \mid k)\sin({}^w\alpha \end{bmatrix} \tag{5.64}$$

$t(i)$ and its Jacobian $(h_i$ by

t_i $t\_$
i



y W_r
i

W_α

i

$\_W\_$ x

Figure 5.31

Representation of the target position in the world coordinate frame $\_W\_$ and robot coordinate frame

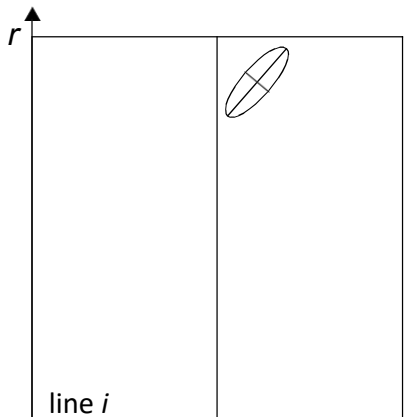
$\_R\_$.

$$h_i = \begin{bmatrix} \hat{x} & \hat{y} & \hat{\alpha} \\ = & & \end{bmatrix} \begin{bmatrix} 0 & 0 & - \\ 1 & & \end{bmatrix} \begin{bmatrix} r & r & \\ r & & \end{bmatrix} \begin{bmatrix} \cos & -\sin & 0 \end{bmatrix} \quad (5.65)$$

$$\hat{x} \quad \hat{y} \quad \hat{\alpha}$$

The measurement prediction results in predicted lines represented in the robot coordi- nate frame (figure 5.32). They are uncertain, because the prediction of robot position is uncertain.

4. **Matching.** For matching, we must find correspondence (or a pairing) between predicted and observed features (figure 5.33). In our case we take the Mahalanobis distance



r_j

$-\pi$ α_j 0 π

Figure 5.32

Measurement predictions: Based on the map and the estimated robot position the targets (visible lines) are predicted. They are represented in the model space similar to the observations.

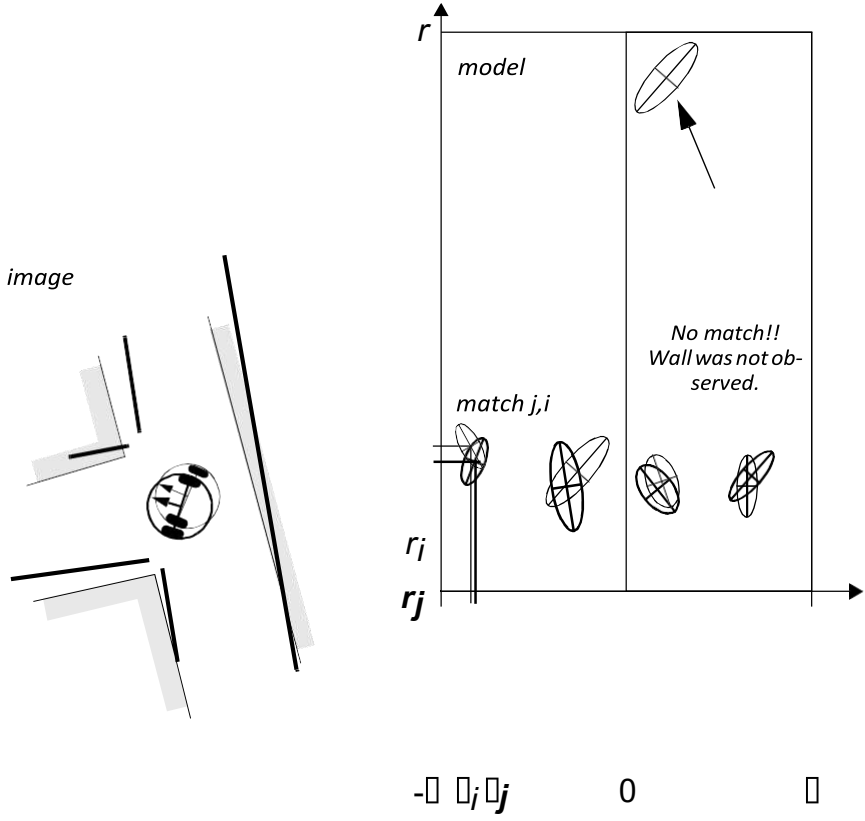


Figure 5.33

Matching: The observations (thick) and measurement prediction (thin) are matched and the innova- tion and its uncertainties are calculated.

$$v_{ij}^T [IN_{ij}]^{-1} v_{ij} + 1 \quad g_2 + 1 \quad (5.66)$$

with

$$v_{ij}(k+1) = (z_j(k+1) - h_i(z_t \hat{p}(k+1\ k)))$$

$$= \begin{bmatrix} \\ j \end{bmatrix} - \begin{bmatrix} W_{t,i} - W\hat{[}\,|k+1\ k) \end{bmatrix}$$

$$r_j \quad W_r - (^w\hat{x}(k+1\ k)\cos(^w) + {}^w\hat{y}(k\ t)\,i))$$

$$+ 1\ k)\sin(^w)$$

$$_{t,i} \qquad | \qquad _{t,i} \qquad |$$

(5.67)

$$+ (k+1) = (h \cdot (k+1\ k) \cdot (h^T \qquad (5.68)$$

$$|$$

$$IN(ij \qquad i \quad p \qquad i \quad R(i$$

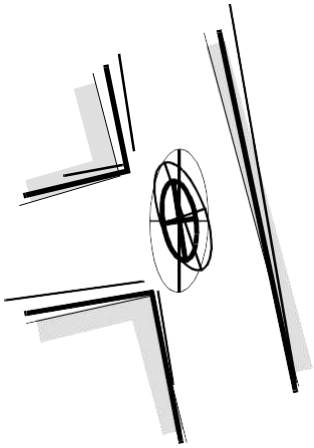


Figure 5.34

Kalman filter estimation of the new robot position: By fusing the prediction of robot position (thin) with the innovation gained by the measurements (thick) we get the updated estimate $\hat{p}(k \mid k)$ of the robot position (very thick).

to enable finding the best matches while eliminating all other remaining observed and pre- dicted unmatched features.

5. **Estimation.** Applying the Kalman filter results in a final pose estimate corresponding to the weighted sum of (figure 5.34)

- the pose estimates of each matched pairing of observed and predicted features;
- the robot position estimation based on odometry and observation positions.

5.7 Other Examples of Localization Systems

Markov localization and Kalman filter localization have been two extremely popular strat- egies for research mobile robot systems navigating indoor environments. They have strong formal bases and therefore well-defined behavior. But there are a large number of other localization techniques that have been used with varying degrees of success on commercial and research mobile robot platforms. We will not explore the space of all localization sys- tems in detail. Refer to surveys such as [5] for such information.

There are, however, several categories of localization techniques that deserve mention. Not surprisingly, many implementations of these techniques in commercial robotics

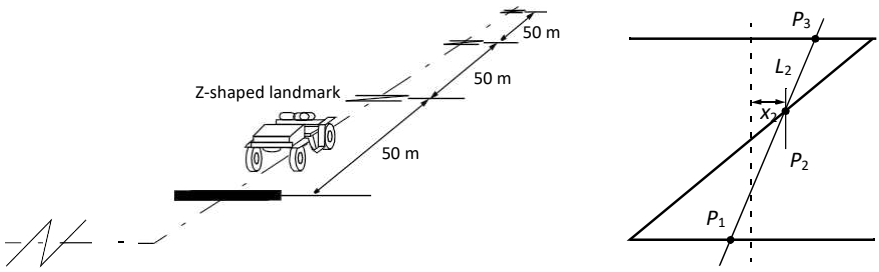


Figure 5.35

Z-shaped landmarks on the ground. Komatsu Ltd., Japan [5 pp. 179-180]

employ modifications of the robot's environment, something that the Markov localization and Kalman filter localization communities eschew. In the following sections, we briefly identify the general strategy incorporated by each category and reference example systems, including, as appropriate, those that modify the environment and those that function with- out environmental modification.

5.7.1 Landmark-based navigation

Landmarks are generally defined as passive objects in the environment that provide a high degree of localization accuracy when they are within the robot's field of view. Mobile robots that make use of landmarks for localization generally use artificial markers that have been placed by the robot's designers to make localization easy.

The control system for a landmark-based navigator consists of two discrete phases. When a landmark is in view, the robot

localizes frequently and accurately, using action update and perception update to track its position without cumulative error. But when the robot is in no landmark “zone,” then only action update occurs, and the robot accumulates position uncertainty until the next landmark enters the robot’s field of view.

The robot is thus effectively *dead-reckoning* from landmark zone to landmark zone. This in turn means the robot must consult its map carefully, ensuring that each motion between landmarks is sufficiently short, given its motion model, that it will be able to local- ize successfully upon reaching the next landmark.

Figure 5.35 shows one instantiation of landmark-based localization. The particular shape of the landmarks enables reliable and accurate pose estimation by the robot, which must travel using *dead reckoning* between the landmarks.

One key advantage of the landmark-based navigation approach is that a strong formal theory has been developed for this general system architecture [98]. In this work, the authors have shown precise assumptions and conditions which, when satisfied, guarantee

that the robot will always be able to localize successfully. This work also led to a real-world demonstration of landmark-based localization. Standard sheets of paper were placed on the ceiling of the Robotics Laboratory at Stanford University, each with a unique checkerboard pattern. A Nomadics 200 mobile robot was fitted with a monochrome CCD camera aimed vertically up at the ceiling. By recognizing the paper landmarks, which were placed approx- imately 2 m apart, the robot was able to localize to within several centimeters, then move, using dead reckoning, to another landmark zone.

The primary disadvantage of landmark-based navigation is that in general it requires sig- nificant environmental modification. Landmarks are local, and therefore a large number are usually required to cover a large factory area or research laboratory. For example, the Robotics Laboratory at Stanford made use of approximately thirty discrete landmarks, all affixed individually to the ceiling.

5.7.2 Globally unique localization

The landmark-based navigation approach makes a strong general assumption: when the landmark is in the robot's field of view, localization is essentially perfect. One way to reach the Holy Grail of mobile robotic localization is to effectively enable such an assumption to be valid no matter *where* the robot is located. It would be revolutionary if a look at the robot's sensors immediately identified its particular location, uniquely and repeatedly.

Such a strategy for localization is surely aggressive, but the question of whether it can be done is primarily a question of sensor technology and sensing software. Clearly, such a

localization system would need to use a sensor that collects a very large amount of infor- mation. Since vision does indeed collect far more information than previous sensors, it has been used as the sensor of choice in research toward globally unique localization.

Figure 4.49 depicts the image taken by a catadioptric camera system. If humans were able to look at an individual such picture and identify the robot's location in a well-known environment, then one could argue that the information for globally unique localization does exist within the picture; it must simply be teased out.

One such approach has been attempted by several researchers and involves constructing one or more image histograms to represent the information content of an image stably (see e.g., figure 4.50 and section 4.3.2.2). A robot using such an image-histogramming system has been shown to uniquely identify individual rooms in an office building as well as indi- vidual sidewalks in an outdoor environment. However, such a system is highly sensitive to external illumination and provides only a level of localization resolution equal to the visual footprint of the camera optics.

The angular histogram depicted in figure 4.39 of the previous chapter is another example in which the robot's sensor values are transformed into an identifier of location. However, due to the limited information content of sonar ranging strikes, it is likely that two *places*

in the robot's environment may have angular histograms that are too similar to be differen- tiated successfully.

One way of attempting to gather sufficient sonar information for global localization is to allow the robot time to gather a large amount of sonar data into a local evidence grid (i.e., occupancy grid) first, then match the local evidence grid with a global metric map of the environment. In [129] the researchers demonstrate such a system as able to localize on the fly even as significant changes are made to the environment, degrading the fidelity of the map. Most interesting is that the local evidence grid represents information well enough that it can be used to correct and update the map over time, thereby leading to a localization system that provides corrective feedback to the environmental representation directly. This is similar in spirit to the idea of taking rejected observed features in the Kalman filter local- ization algorithm and using them to create new features in the map.

A most promising, new method for globally unique localization is called *mosaic-based localization* [83]. This fascinating approach takes advantage of an environmental feature that is rarely used by mobile robots: fine-grained floor texture. This method succeeds pri- marily because of the recent ubiquity of very fast processors, very fast cameras, and very large storage media.

The robot is fitted with a high-quality high-speed CCD camera pointed toward the floor, ideally situated between the robot's wheels, and illuminated by a specialized light pattern off the camera axis to enhance floor texture. The robot begins by collecting images of the entire floor in the robot's workspace using this camera. Of course, the memory require- ments are significant, requiring a 10 GB drive in order to store the complete

image library of a 300 x 300 area.

Once the complete image mosaic is stored, the robot can travel any trajectory on the floor while tracking its own position without difficulty. Localization is performed by simply recording one image, performing action update, then performing perception update by matching the image to the mosaic database using simple techniques based on image database matching. The resulting performance has been impressive: such a robot has been shown to localize repeatedly with 1 mm precision while moving at 25 km/hr.

The key advantage of globally unique localization is that, when these systems function correctly, they greatly simplify robot navigation. The robot can move to any point and will always be assured of localizing by collecting a sensor scan.

But the main disadvantage of globally unique localization is that it is likely that this method will *never* offer a complete solution to the localization problem. There will always be cases where local sensory information is truly ambiguous and, therefore, globally unique localization using only current sensor information is unlikely to succeed. Humans often have excellent *local positioning systems*, particularly in nonrepeating and well-known environments such as their homes. However, there are a number of environments in which such immediate localization is challenging even for humans: consider hedge mazes and

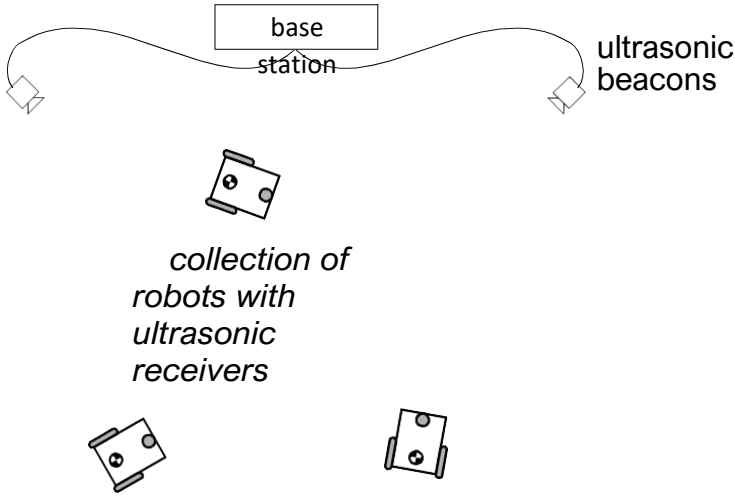


Figure 5.36

Active ultrasonic beacons.

large new office buildings with repeating halls that are identical. Indeed, the mosaic-based localization prototype described above encountered such a problem in its first implementa- tion. The floor of the factory floor had been freshly painted and was thus devoid of suffi- cient micro fractures to generate texture for correlation. Their solution was to modify the environment after all, painting random texture onto the factory floor.

5.7.3 Positioning beacon systems

One of the most reliable solutions to the localization problem is to design and deploy an active beacon system specifically for the target environment. This is the preferred tech- nique used by

both industry and military applications as a way of ensuring the highest pos- sible reliability of localization. The GPS system can be considered as just such a system (see section 4.1.5.1).

Figure 5.36 depicts one such beacon arrangement for a collection of robots. Just as with GPS, by designing a system whereby the robots localize passively while the beacons are active, any number of robots can simultaneously take advantage of a single beacon system. As with most beacon systems, the design depicted depends foremost upon geometric prin- ciples to effect localization. In this case the robots must know the positions of the two active ultrasonic beacons in the global coordinate frame in order to localize themselves to the global coordinate frame.

A popular type of beacon system in industrial robotic applications is depicted in figure 5.37. In this case beacons are retroreflective markers that can be easily detected by a mobile robot based on their reflection of energy back to the robot. Given known positions for the optical retroreflectors, a mobile robot can identify its position whenever it has three such

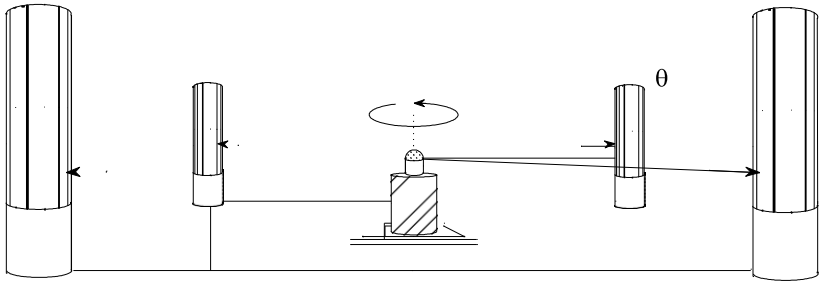


Figure 5.37

Passive optical beacons.

beacons in sight simultaneously. Of course, a robot with encoders can localize over time as well, and does not need to measure its angle to all three beacons at the same instant.

The advantage of such beacon-based systems is usually extremely high engineered reli- ability. By the same token, significant engineering usually surrounds the installation of such a system in a specific commercial setting. Therefore, moving the robot to a different factory floor will be both, time consuming and expensive. Usually, even changing the routes used by the robot will require serious re-engineering.

5.7.4 Route-based localization

Even more reliable than beacon-based systems are route-based localization strategies. In this case, the route of the robot is explicitly marked so that it can determine its position, not relative to some global coordinate frame but relative to the specific path it is allowed to travel. There are many techniques for marking such a route and the subsequent intersec- tions. In all cases, one

is effectively creating a railway system, except that the railway system is somewhat more flexible and certainly more human-friendly than a physical rail. For example, high ultraviolet-reflective, optically transparent paint can mark the route such that only the robot, using a specialized sensor, easily detects it. Alternatively, a guidewire buried underneath the hall can be detected using inductive coils located on the robot chas- sis.

In all such cases, the robot localization problem is effectively trivialized by forcing the robot to always follow a prescribed path. To be fair, there are new industrial *unmanned guided vehicles* that do deviate briefly from their route in order to avoid obstacles. Never- theless, the cost of this extreme reliability is obvious: the robot is much more inflexible given such localization means, and therefore any change to the robot's behavior requires significant engineering and time.

5.8 Autonomous Map Building

All of the localization strategies we have discussed require human effort to install the robot into a space. Artificial environmental modifications may be necessary. Even if this not be case, a map of the environment must be created for the robot. But a robot that localizes suc- cessfully has the right sensors for detecting the environment, and so the robot ought to build its own map. This ambition goes to the heart of autonomous mobile robotics. In prose, we can express our eventual goal as follows:

Starting from an arbitrary initial point, a mobile robot should be able to autonomously explore the environment with its on-board sensors, gain knowledge about it, interpret the scene, build an appropriate map, and localize itself relative to this map.

Accomplishing this goal robustly is probably years away, but an important subgoal is the invention of techniques for autonomous creation and modification of an environmental map. Of course a mobile robot's sensors have only a limited range, and so it must physically explore its environment to build such a map. So, the robot must not only create a map but it must do so while moving and localizing to explore the environment. In the robotics com- munity, this is often called the simultaneous localization and mapping (SLAM) problem, arguably the most difficult problem specific to mobile robot systems.

The reason that SLAM is difficult is born precisely from the interaction between the robot's position updates as it localizes and its mapping actions. If a mobile robot updates its position based on an observation of an imprecisely known feature, the resulting position estimate becomes correlated with the feature location estimate. Similarly, the map becomes correlated with the position estimate if an observation taken from an imprecisely

known position is used to update or add a feature to the map. The general problem of map-building is thus an example of the chicken-and-egg problem. For localization the robot needs to know where the features are, whereas for map-building the robot needs to know where it is on the map.

The only path to a complete and optimal solution to this joint problem is to consider all the correlations between position estimation and feature location estimation. Such cross-correlated maps are called *stochastic* maps, and we begin with a discussion of the theory behind this approach in the following section [55].

Unfortunately, implementing such an optimal solution is computationally prohibitive. In response a number of researchers have offered other solutions that have functioned well in limited circumstances. Section 5.8.2 characterizes these alternative partial solutions.

5.8.1 The stochastic map technique

Figure 5.38 shows a general schematic incorporating map building and maintenance into the standard localization loop depicted by figure 5.28 during the discussion of Kalman filter

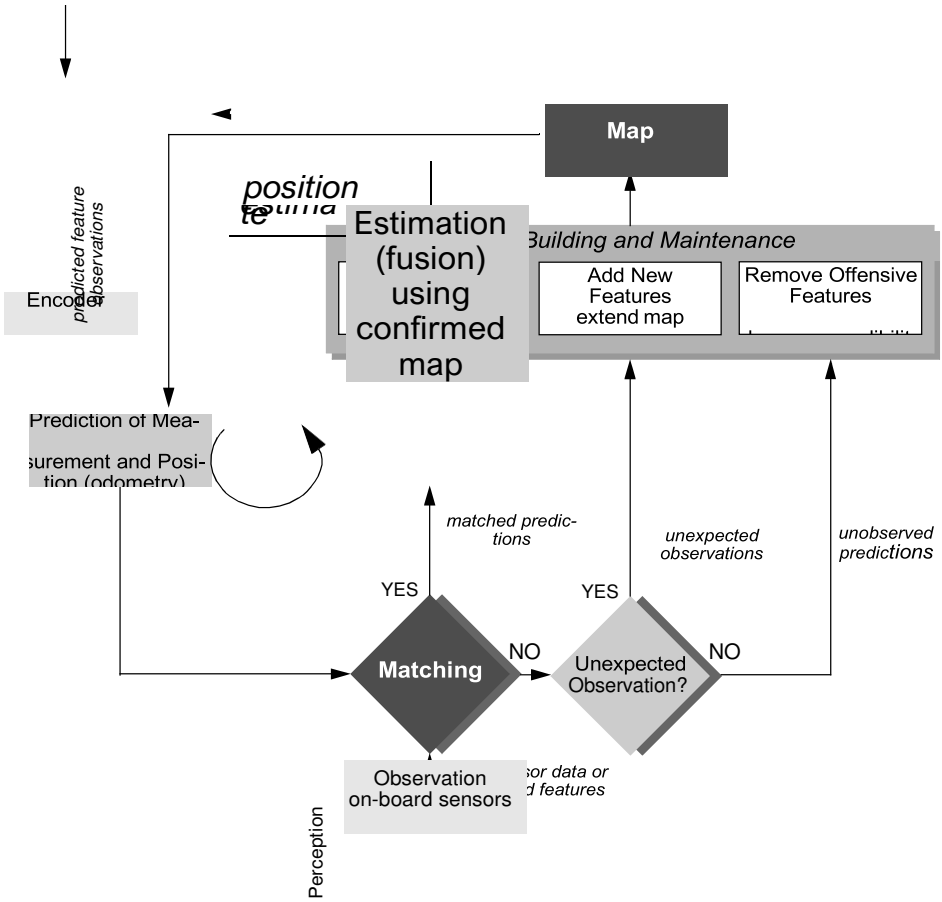


Figure 5.38

General schematic for concurrent localization and map building (see [23]).

localization [23]. The added arcs represent the additional flow of information that occurs when there is an imperfect match between observations and measurement predictions.

Unexpected observations will effect the creation of new features in the map, whereas unobserved measurement predictions will effect the removal of features from the map. As discussed earlier, each specific prediction or observation has an unknown exact value and so it is represented by a distribution. The uncertainties of all of these quantities must be con- sidered throughout this process.

The new type of map we are creating not only has features in it, as did previous maps, but it also has varying degrees of probability that each feature is indeed part of the environ- ment. We represent this new map M with a set n of probabilistic feature locations $\hat{z}_t$, each

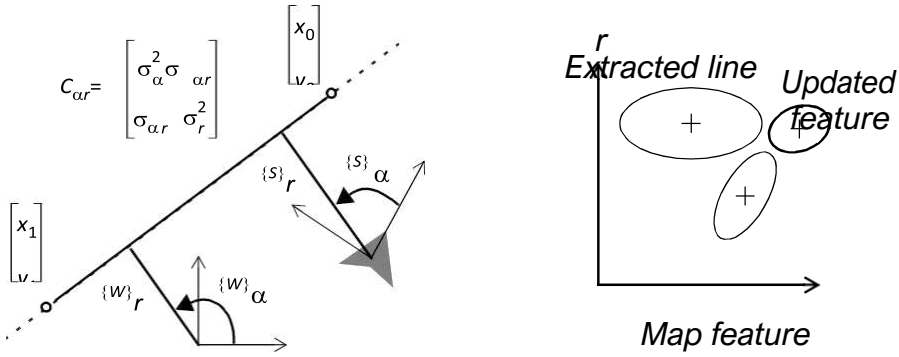


Figure 5.39

Uncertainties in the map.

with the covariance matrix Σ_t and an associated *credibility factor* c_t between 0 and 1 quan- tifying the belief in the existence of the feature in the environment (see figure 5.39):

$$M = \{(\hat{z}_t, \Sigma_t, c_t) \mid 1 \le t \le n\} \tag{5.69}$$

In contrast to the map used for Kalman filter localization previously, the map M is not assumed to be precisely known because it will be created by an uncertain robot over time. This is why the features $\hat{z}_t$ are described with associated covariance matrices Σ_t.

Just as with Kalman filter localization, the matching step yields has

three outcomes in
regard to measurement predictions and observations: *matched prediction and observations*, *unexpected observations,* and *unobserved predictions*. Localization, or the position update of the robot, proceeds as before. However, the map is also updated now, using all three out- comes and complete propagation of all the correlated uncertainties (see [23] for more details).

An interesting variable is the credibility factor c_t , which governs the likelihood that the mapped feature is indeed in the environment. How should the robot's failure to match observed features to a particular map feature reduce that map feature's credibility? And
also, how should the robot's success at matching a mapped feature increase the chance that the mapped feature is "correct?" In [23] the following function is proposed for calculating credibility:

$$c_t(k) = 1 - e^{-\left(\frac{n_s}{a} - \frac{n_u}{b}\right)} \tag{5.70}$$

where a and b define the learning and forgetting rate and n_S and n_U are the number of matched and unobserved predictions up to time k, respectively. The update of the covari- ance matrix Σ_t can be done similarly to the position update seen in the previous section. In map-building the feature positions and the robot's position are strongly correlated. This
forces us to use a *stochastic map*, in which all cross-correlations must be updated in each cycle [55, 113, 136].

The stochastic map consists of a stacked system state vector:

$$X = \begin{bmatrix} x_r(k) & x_1(k) & x_2(k) & \cdots \\ x_n(k) \end{bmatrix}^T \tag{5.71}$$

and a system state covariance matrix:

$$\Sigma = \begin{bmatrix} C_{rr} & C_{r1} & C_{r2} & \cdots & C_{rn} \\ C_{1r} & C_{11} & \cdots & & C_{1n} \\ C_{2r} & \cdots & & & C_{2n} \\ \cdots & & & & \cdots \\ C_{nr} & C_{n1} & C_{n2} & \cdots & C_{nn} \end{bmatrix} \tag{5.72}$$

where the index r stands for the robot and the index $i = 1$ to n for the features in the map. In contrast to localization based on

an a priori accurate map, in the case of a stochastic map the cross-correlations must be maintained and updated as the robot is performing auto- matic map-building. During each localization cycle, the cross-correlations robot-to-feature and feature-to-robot are also updated. In short, this optimal approach requires every value in the map to depend on every other value, and therein lies the reason that such a complete solution to the automatic mapping problem is beyond the reach of even today's computational resources.

5.8.2 Other mapping techniques

The mobile robotics research community has spent significant research effort on the prob- lem of automatic mapping, and has demonstrated working systems in many environments without having solved the complete stochastic map problem described earlier. This field of mobile robotics research is extremely large, and this text will not present a comprehensive survey of the field. Instead, we present below two key considerations associated with auto- matic mapping, together with brief discussions of the approaches taken by several auto- matic mapping solutions to overcome these challenges.

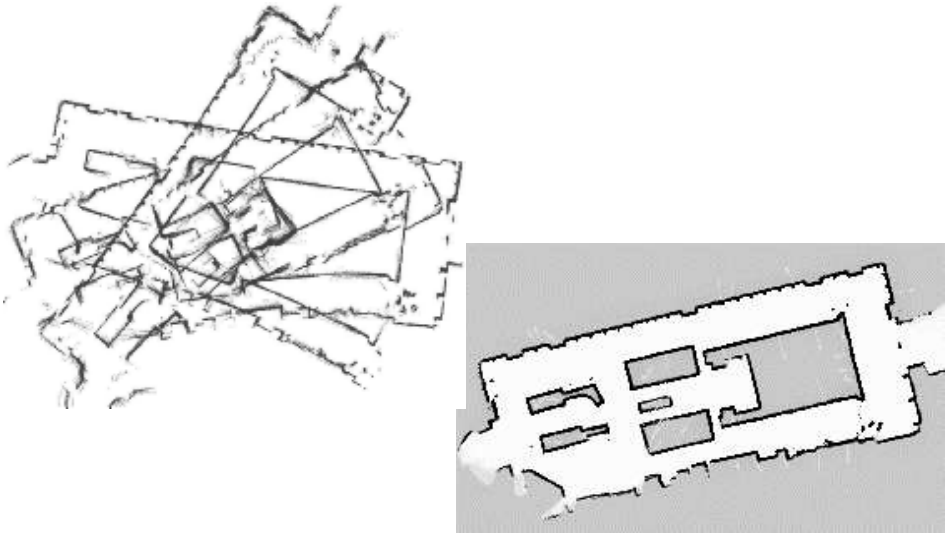


Figure 5.40

Cyclic environments: A naive, local mapping strategy with small local error leads to global maps that have a significant error, as demonstrated by this real-world run on the left. By applying topological correction, the grid map on the right is extracted (courtesy of S. Thrun [142]).

5.8.2.1 Cyclic environments

Possibly the single hardest challenge for automatic mapping to be conquered is to correctly map cyclic environments. The problem is simple: given an environment that has one or more loops or cycles (e.g., four hallways that intersect to form a rectangle), create a glo- bally consistent map for the whole environment.

This problem is hard because of the fundamental behavior of automatic mapping sys- tems: the maps they create are not

perfect. And, given any local imperfection, accumulating such imperfections over time can lead to arbitrarily large *global* errors between a map, at the macrolevel, and the real world, as shown in figure 5.40. Such global error is usually irrelevant to mobile robot localization and navigation. After all, a warped map will still serve the robot perfectly well so long as the local error is bounded. However, an extremely large loop still eventually returns to the same spot, and the robot must be able to note this fact in its map. Therefore, global error does indeed matter in the case of cycles.

In some of the earliest work attempting to solve the cyclic environment problem, Kuipers and Byun [94] used a purely topological representation of the environment, rea- soning that the topological representation only captures the most abstract, most important

features and avoids a great deal of irrelevant detail. When the robot arrives at a topological node that could be the same as a previously visited and mapped node (e.g., similar distin- guishing features), then the robot postulates that it has indeed returned to the same node. To check this hypothesis, the robot explicitly plans and moves to adjacent nodes to see if its perceptual readings are consistent with the cycle hypothesis.

With the recent popularity of metric maps, such as fixed decomposition grid represen- tations, the cycle detection strategy is not as straightforward. Two important features are found in most autonomous mapping systems that claim to solve the cycle detection prob- lem. First, as with many recent systems, these mobile robots tend to accumulate recent per- ceptual history to create small-scale local *submaps* [51, 74, 157]. Each submap is treated as a single sensor during the robot's position update. The advantage of this approach is two- fold. Because odometry is relatively accurate over small distances, the relative registration of features and raw sensor strikes in a local submap will be quite accurate. In addition to this, the robot will have created a virtual sensor system with a significantly larger horizon than its actual sensor system's range. In a sense, this strategy at the very least defers the problem of very large cyclic environments by increasing the map scale that can be handled well by the robot.

The second recent technique for dealing with cycle environments is in fact a return to the topological representation. Some recent automatic mapping systems will attempt to identify cycles by associating a topology with the set of metric submaps, explicitly identi- fying the loops first at the topological level. In the case of [51], for example, the topological level loop is identified by

a human who pushes a button at a known landmark position. In the case of [74], the topological level loop is determined by performing correspondence tests between submaps, postulating that two submaps represent the same place in the envi- ronment when the correspondence is good.

One could certainly imagine other augmentations based on known topological methods. For example, the globally unique localization methods described in section 5.7 could be used to identify topological correctness. It is notable that the automatic mapping research of the present has, in many ways, returned to the basic topological correctness question that was at the heart of some of the earliest automatic mapping research in mobile robotics more than a decade ago. Of course, unlike that early work, today's automatic mapping results boast correct cycle detection combined with high-fidelity geometric maps of the environ- ment.

5.8.2.2 Dynamic environments

A second challenge extends not just to existing autonomous mapping solutions but to the basic formulation of the stochastic map approach. All of these strategies tend to assume that the environment is either unchanging or changes in ways that are virtually insignificant. Such assumptions are certainly valid with respect to some environments, such as, for exam- ple, the computer science department of a university at 3 AM. However, in a great many

cases this assumption is incorrect. In the case of wide-open spaces that are popular gather- ing places for humans, there is rapid change in the free space and a vast majority of sensor strikes represent detection of transient humans rather than fixed surfaces such as the perim- eter wall. Another class of dynamic environments are spaces such as factory floors and warehouses, where the objects being stored redefine the topology of the pathways on a day- to-day basis as shipments are moved in and out.

In all such dynamic environments, an automatic mapping system should capture the *salient* objects detected by its sensors and, furthermore, the robot should have the flexibility to modify its map as to the positions of these salient objects changes. The subject of *con- tinuous mapping*, or mapping of dynamic environments, is to some degree a direct out- growth of successful strategies for automatic mapping of unfamiliar environments. For
example, in the case of stochastic mapping using the credibility factor c_t mechanism, the credibility equation can continue to provide feedback regarding the probability of the exist- ence of various mapped features after the initial map creation process is ostensibly com-
plete. Thus, a mapping system can become a map-modifying system by simply continuing to operate. This is most effective, of course, if the mapping system is real-time and incre- mental. If map construction requires off-line global optimization, then the desire to make small-grained, incremental adjustments to the map is more difficult to satisfy.

Earlier we stated that a mapping system should capture only the *salient* objects detected by its sensors. One common argument for handling the detection of, for instance, humans
in the environment is that mechanisms such as c_t can take care of all features that did not deserve to be mapped in the first place. For example, in [157] the authors develop a system based on a set of local occupancy grids (called *evidence grids*) and a global occupancy grid.
Each time the robot's most recent local evidence grid is used to update a region of the global occupancy grid, extraneous occupied cells in the global occupancy grid are freed if the local occupancy grid detected no objects (with high confidence) at those same positions.

The general solution to the problem of detecting salient features, however, begs a solu- tion to the perception problem in general. When a robot's sensor system can reliably detect the difference between a wall and a human, using, for example, a vision system, then the problem of mapping in dynamic environments will become significantly more straightfor- ward.

We have discussed just two important considerations for automatic mapping. There is still a great deal of research activity focusing on the general map-building and localization problem [22, 23, 55, 63, 80, 134, 147, 156]. However, there are few groups working on the general problem of probabilistic map-building (i.e., stochastic maps) and, so far, a consis- tent and absolutely general solution is yet to be found. This field is certain to produce sig- nificant new results in the next several years, and as the perceptual power of robots improves we expect the payoff to be greatest here.

5. Methodology

1 Motivation

Autonomous robots that are to perform everyday manipulation tasks in human living environments must be able to pick up and place many objects of daily use. To manipulate them, the robots first have to perceive these objects. They are to detect the objects in realistic settings, categorize them, recognize object instances, estimate their 6D poses or even reconstruct their shapes from partial views.

Many algorithms have been developed to solve these problems for different subsets of objects, with varying accuracy and reliability, with different require- ments for computational resources, and under different context conditions. Some of them require prior object models while others can do without, some infer only general categories, others exact instances without the knowledge of the broader categories these objects fall into. The approaches also differ in the type of sensors used, in speed, in that not all of them report 6D poses, in the number of objects they can deal with at once, etc.

The realization of robot perception systems that can perceive the range of ob- jects to be manipulated in a typical human environment with the accuracy and reliability needed for grasping them successfully in real everyday settings poses a very hard and long-term research problem. Robot manipulation tasks are usually restricted to detecting distinctively textured objects, objects with distinct colors or specific shapes, based on the full list of possible objects.

Service robots, such as household robots, perform the same kinds of tasks with the same objects in the same environment over and over again. This enables them to learn and make use of more specific perception mechanisms for the particu- lar objects and environments through *task and environment adaptation* (Horswill, 1995). Task and environment adaptation

thereby enables the robot to better per- ceive the objects in its environment by exploiting its experience and considering only relevant objects.

We consider that perceptions tasks can be intuitively categorized into *active* or *passive*. In the first type of task, a specific object or a set of objects needs to be located and thus the robot has to actively search for them. In the second case, the robot, possibly while performing other tasks, observes the environment or a specific region and identifies the different objects. For example, while setting the table for breakfast, the robot might spot a bottle of wine somewhere. Later, when the robot is to set a table for a lunch that includes a wine beverage, it does not need to actively start searching for a bottle, but instead retrieves the bottle's

location from memory and navigates directly to it.

In both types of perception tasks, some sort of model is needed that describes what constitutes a certain object or an object of a certain type. Since we are most interested in pick-and-place scenarios like completing a table setting (which en- compasses both *active* and *passive* perception), models that aid the robot in grasp- ing are of great importance. While identifying objects based on their appearance works reasonably well (Kragic and Vincze, 2009), a 3D model is required in order to manipulate these objects and it has to be matched to the observed model. Also, as concluded in (Kragic and Vincze, 2009), a problem of current vision systems is robustness and scalability.

In this article we investigate object perception mechanisms for autonomous robots performing everyday manipulation tasks. The perception mechanisms use general perception methods to initialize learning, adapt and specialize their strate- gies as much as possible to the environment and objects it has to use. By this, some of the problems become easier to deal with, thus improve efficiency.

The proposed object perception system maintains an *Object Model Database* that provides information for the detection, recognition, pose estimation and grasp- ing of the objects in the environment. We present a system that can use and build such a database under some assumptions and discuss the scalability issues. Our approach is to start with general detection routines and categorization modules to limit the possibilities for identities of objects and then add more and more details until an acceptable solution can be found. As the most important perception meth- ods, texture-based and 3D perception routines constitute the focus of this article. However, simple additions like considering physical size, color classification and heat-sensors can easily be added to the system's repertoire of examination meth- ods.

The use of multiple detectors is important, as real-world objects, especially products of the same company have very similar appearance (coffee, cans, choco- late, etc.) as illustrated in Figure 1. Using only geometric descriptors is not gen- erally applicable either, as, for example,

mugs, drinks and cereal boxes typically have similar shapes but different appearance. In both cases, the intended use of the object can be significantly different and can be relevant to the task at hand. Ad- ditionally, some objects might be hard to detect with one sensor, but more easily detectable using others (for example semi-transparent or shiny objects), as shown in subsection 7.4.

Though the set of objects of daily use that a personal robot could encounter

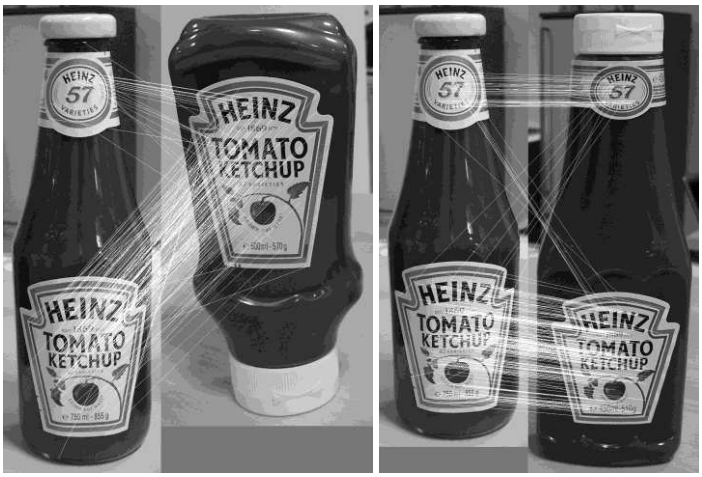


Figure 1: Two examples of good model matches using SIFT features extracted from images. The images however depict three ketchup bottles with three different shapes which makes classification of the objects based on visual appearance challenging.

in its tasks is unlimited, there are certain regularities that can be exploited with respect to the objects' shapes, textures and locations. Therefore, the perception system can adapt itself to a specific set of objects that are usually present in the world while at the same time retaining a certain degree of flexibility with respect to the incorporation of novel objects in its database of models. For example, a new flavor of iced tea should be recognized and manipulated as an instance of a box from its geometry, even though the robot has never seen it before. The knowledge about the iced tea can then be easily enriched with its appearance, provided it can be re-detected using geometric information.

Also, certain information can be incorporated from external sources, for ex- ample a database of semantically labeled visual appearances of objects, which lacks the geometric appearance. In this case, the system should complete the missing information whenever possible. Similarly, in the case of autonomously learned object models, the semantic label or the correction of the assumed object identities could be done externally.

To this end, our system can learn the specific models of the objects it encoun- ters. Since we do not assume that every perceived object in the environment is known, the system can mistakenly assume, for example, that a certain feature is unique to an object when in fact it is not. Our view is, however, that these mistakes are unavoidable during autonomous object learning; they are compensated by the ability to automatically acquire models. Also, initializing the system with external databases (which are rather complete and labeled with semantically rich data, such

as the KIT Object Models Web Database[1]) alleviates this problem significantly.

As an example, let us consider the case where the robot is to fetch a green object. Since this object happens to be the only green one it already knows about, it looks for the first green object it finds while heading towards the last observed position of the object. While it drives around, it observes various other objects that do not match the feature (color) it is looking for. But since navigating around and taking laser scans takes longer than identifying the objects (scans with ade- quate resolution take in the range of seconds), the robot is constantly executing an "identify-all-objects" routine. The latter updates the stored time-stamped position of recognized objects and assigns all unrecognized ones to new entries, assuming they are objects that were previously unknown to it.

As a simple, but effective way of considering different viewing angles, we implemented the following algorithms to aid the robot in integrating sequences of information. If, between subsequent scans, an unrecognized object candidate is found to have the same position as another object in the database, we inherit the class label from the object in the database. In case of e.g. incomplete models, this location based identification enables us to incrementally learn the appearance of previously unknown views.

Mistakes in object detection and updates are possible of course, as in every classification problem, but the system can deal correctly with novel objects, some- thing that is difficult for re-detection-based systems with a fixed set of objects. At- taching the semantic label to learned objects and the correction of mistaken object identities in the database is then performed by the human user.

We are building on, improving and combining several of our earlier works, mainly the ones presented in (Marton et al., 2010a) and (Marton et al., 2010b), and to some extent (Blodow et al., 2009), (Rusu et al., 2009a), (Marton et al., 2009b) and (Rusu et al., 2008b).

We adapted the model fitting and geometric classification to work on the less accurate 3D data coming from the eye-safe Hokuyo UTM-30LX. The number of geometric categories was reduced (small differences like the existence of a handle are hard to detect with inaccurate sensors) and

the results were made more robust by considering each view of each object as a separate class during training, but taking only the identified category into account during testing.

Visual classification was improved to be more scalable by a strategy similar

[1] http://i61p109.ira.uka.de/ObjectModelsWebUI/

to visual bag-of-words from the computer vision field. The approach presented here is taken from the field of document retrieval, where word groups are used in a hierarchical tree setup to allow for fast lookup times. Please see subsection 6.3 for more detail on the combination of visual bag-of-words with document retrieval methods.

In brief, the main contributions of this article are:

- a perception system for object classification, categorization and modeling;
- automatic learning of descriptors for novel objects/views;
- a perception system based on affordable sensors, thus easy to rebuild.

The kinds of limitations of the system (that would require human supervision at some point) are as follows:

- previously unseen objects could be erroneously classified – this effect can, however, be minimized to the combined false classification rate of the de- tectors if the specialization step uses all of them;
  previously unseen faces of object could lead to the classification of a novel object if in the previous step it was not observed in the same location or identified correctly – this is unavoidable, but new views of known objects can be identified, given continuous observations;
- known objects can be miss-classified for example in cases where objects have identically textured faces (some cereal boxes for example) – this issue also occurs in re-detection-based systems, but if the object is continuously observed, and one of the ambiguous faces is not the first one that is seen, misclassification can be avoided.

Some of the above issues can be resolved by an object identity resolution sys- tem like we described in (Blodow et al., 2010) (see also Section 8.3), which main- tains a probabilistically modeled belief state of where certain objects are located given the sensor measurements, however, this in itself is a complex task and falls outside of the scope of this

paper.

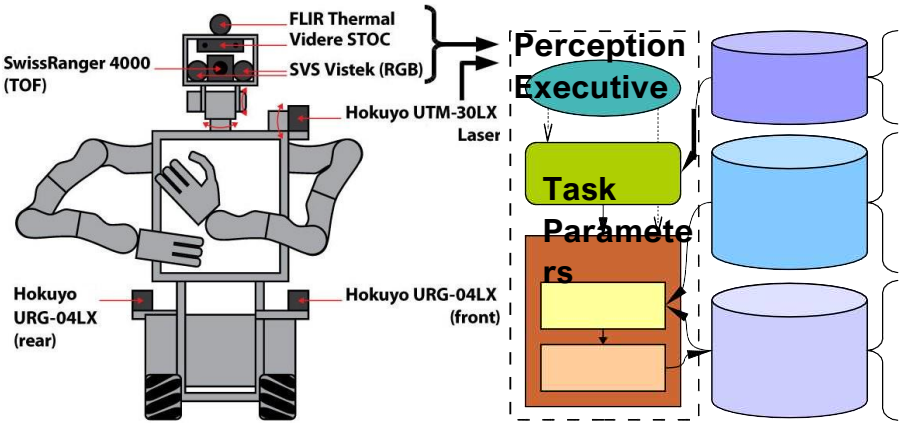
In the next section, we describe the architecture of our system, followed by the presentation of related work through the prism of different domains. Next, we give

technical details on the overall implementation of our system and then describe the components of the different modules in Sections 5 and 6. We evaluate the different functions of our system in Section 7, and in Section 8 we show specific applications that benefit from our perception system. We conclude in Section 9 and present which next steps we planned for improving the system.

2 System Overview

Figure 2 shows the schematic representation of our system, starting from the sen- sors of the robot used for localization and perception of the environment, through the possible *Object Detection Methods*, and the selection of modules from the *Examination Method Library*, to object recognition and learning using the infor- mation stored in the *Object Model Database*.

Our two robots share similar setups, with standard cameras in stereo setups, and a tilting Hokuyo UTM-30LX for obtaining 3D point clouds. Lasers close to ground level provide information for localization. While many other sensors are placed on both robots, these are the ones relevant to the presented system.



ptual Task Execution	
Hierarchical Look-Up of Objects with Matching Features	
Update Matching Object or Insert New Object	
	Segmentation Methods
	Examination Method Library
	Object Model Database

- subtract known background
- planar support
- use of semantic environment map

Image and 3D based methods:

- categorization
- classification
- reconstruction
- data fusion from multiple sources

Holds object information:

- 3D shape
- appearance based feature

s of different views
- last observed position and time

Figure 2: Object processing pipeline architecture: from sensor data to objects.

The images and 3D information coming from the robot's sensors are processed by the *Perception Executive* and the gathered data is interpreted according to the task at hand (searching for a specific object or identifying all objects). First, to limit the search-space for object locations, a set of possible locations is extracted and the corresponding sensor readings (3D clusters, 2D Regions-Of-Interest) are

considered to represent object candidates. These object hypotheses are then pro- cessed as needed (please see Figure 3 and Section 6) in order to associate the percepts to the correct object in the *Object Model Database*.

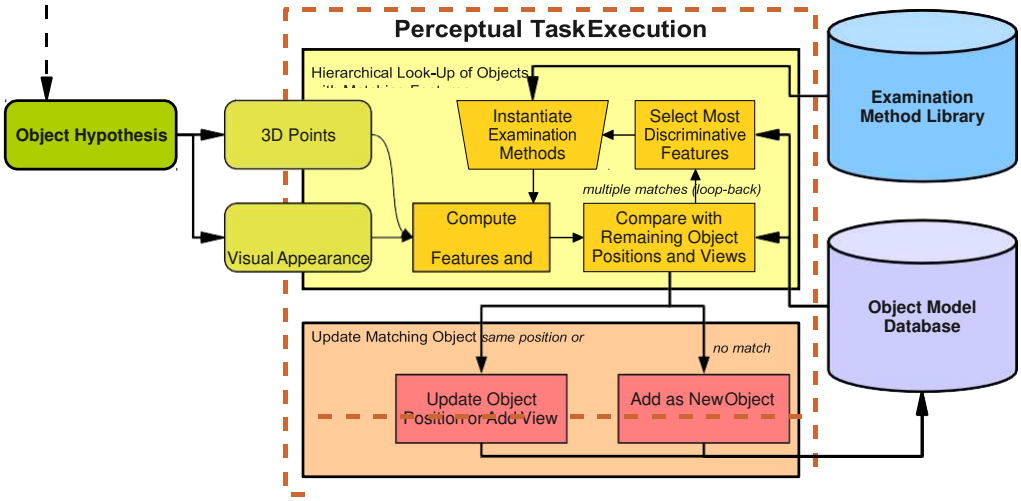


Figure 3: Details of the *Perceptual Task Execution* module's operation. Selecting and combining methods from the *Examination Method Library* to recognize known objects (either based on their stored features, or by co-locality in subsequent cycles), and to update their position or add their new view. The loop-back is not needed if a single object is being searched for. In case new objects were found, all features and representations are computed from the available data in order to be inserted into the *Object Model Database*.

When an object is being sought for, the system selects a set of features, whose values uniquely describe the object amongst all the objects in the database. The same features' values are then computed using the examination modules for each object hypothesis, and the first one that presents matching ones is selected as the target object for e.g. grasping. In the case that no geometric model is associated with the database object in question, we compute it on demand and feed it to the grasp planner.

If computational resources allow all object hypotheses can be checked against all objects in the database – and new objects, or new positions or views of known objects, can be detected. In this case, the features for each object hypothesis are computed one by one, according to a hierarchy, and the possible object identities are filtered in each step. The selection of the feature to be used in each step is hand encoded as of now, but an expansion of the single-object case is envisioned to be extended for finding the most discriminative features in each step.

This process is repeated until either an object is found whose stored features match all observed features, or until there are no matching objects left in the database, signaling that a novel object was observed. In the ambiguous case, when all features were computed and there are still multiple matching objects left from the database, the system takes no action and leaves the object hypothesis unclassified. Note that this is the case only if no visual features are present for the cluster (or stored for the possible matching objects), as visual feature classi- fication assigns each observation to at most one modeled object, as discussed in subsection 6.3.

Since the computation of the features for an object hypothesis is not too ex- pensive, as shown in the corresponding sections, the aim of the procedure is to minimize the number of objects that have to be compared against as drastically as possible in each step, and allow a large number of objects to be handled efficiently. Given enough descriptive features, this method can scale well in the context of ob- jects of daily use in human living environments. We consider this approach to be a move away from bottom-up, rigid pipelines, towards a more flexible setup. This enables the robot to specialize to the current situation, producing shorter process- ing times as not all the methods are needed all the time.

In order to be able to learn more and more about an object, multiple detections with different sensors and from different points of view are needed. To check if two percepts belong to the same object or not, we use a simplified version of the probabilistic framework for identity resolution (Blodow et al., 2010), which is based on positions of objects.

3 Related Work

In the following subsections, we will discuss related approaches in groups repre- senting the main areas of research this work falls into.

3.1 Perception Systems

In (Nakayama et al., 2009), the AI Goggles system is presented, which is a wear- able system capable of describing generic objects in the environment and retriev- ing the memories of them using visual information in real time without any ex- ternal computation resources. The system is also capable of learning new objects

or scenes taught by users, however the training and testing phases are separate. As the core of the system, a high-accuracy and high-speed image annotation and retrieval method supporting online learning are considered. The authors use color higher-order local auto-correlation (Color-HLAC) features and the Canonical Cor- relation Analysis (CCA) algorithm to learn the latent variables. In this work, we present a method for combining multiple features, and discuss how unknown ob- jects can be detected and learned automatically.

Another system optimized for speed is MOPED (Torres et al., 2010). It builds on POSESEQ, a state of the art object recognition algorithm and demonstrates a massive improvement in scalability and latency without sacrificing robustness. The authors achieve this with both algorithmic (different variations of nearest neighbor search) and architectural (SIMD instructions, etc) improvements, with a novel feature matching algorithm, a hybrid GPU/CPU architecture that exploits parallelism at all levels, and an optimized resource scheduler. Since the system considers visual only features, it has certain limitations that can be tackled by considering 3D data as well. Our system uses a different visual classifier, but in theory any method can be used. The main advantage of our approach is in the combination of image and 3D features, and the building of models for grasping and re-detection for new objects.

A system for 3D perception and modeling is presented in (Arbeiter et al., 2010), that can be used to reconstruct a 3D environment or learn models for object recognition on a mobile robot. Both color and time-of-flight cameras are used, and 2D features are extracted from color images and linked to 3D coordinates. Those then serve as input for a modified fastSLAM algorithm for rendering environment maps or object models. While the 3D aspects are not given that much importance, dealing with repeating visual landmarks on an object is handled nicely by the sys- tem. As the focus is on determining correspondences between scans, recognizing when an object is novel and has to be learned is not addressed.

A self-referenced 3D modeler is presented in (Strobl et al., 2009), where the authors demonstrate that an ego-motion algorithm tracking

natural, distinctive fea- tures and a concurrent 3D modeling of the scene is indeed possible. The use of stereo vision, an inertial measurement unit, and robust cost functions for pose es- timation in the system further increase performance. While here the authors build accurate models using a hand-held device, we are using the robot's own sensors and movements to generate views that are similar to what the robot will encounter when it needs to do recognition.

Incremental learning and recognition of objects is done in an unsupervised manner in (Triebel et al., 2010), but the authors focus mainly on furniture pieces, and it is not clear how well multiple objects could be reliably detected without any prior information. Our approach is a semi-supervised one, where the robot generates object hypotheses on its own, but the categorization happens into user- defined classes.

A similar approach to ours is taken by (Hinterstoisser et al., 2010), where a new view of an object is added to the model if it is so different from the stored ones as to cause the matching score to drop below the detection level. As we are combining the image data with 3D information, we effectively eliminate the problem of scale and have a better segmentation of the objects, and therefore require fewer views to be learned.

3.2 Model Fitting

Multiple sensors were used for solving similar tasks, like cameras (Ulrich et al., 2009; Coates et al., 2009), stereo cameras (Fritz et al., 2009; Hillenbrand, 2008), 3D sensors (Steder et al., 2009), and also their combinations to speed up or im- prove results (Klank et al., 2009).

A vision-based grasping system which segments objects on a table and con- structs triangular meshes for them is presented in (Richtsfeld and Vincze, 2008). While the presented method is general and works for many objects, it creates com- plicated models for certain objects, which could be simplified through the usage of geometric primitives. A simplification of the modeling problem is used in Grasp- It (Miller and Allen, 2004), where combinations of geometric shape primitives such as spheres, cylinders, cones and boxes are used to model each object.

A computer vision and machine learning based method is used in (Saxena et al., 2008) to train classifiers that can predict the grasping points in an image. This is then applied to images of unseen objects. To obtain 3D positions of grasp- ing points, the authors use stereo cameras, but their approach works reliably only to the extent provided by the training data. Another issue is the segmentation of objects, since grasp points are

provided with no information about what objects are in the scene and to which of them do the identified points correspond. In (Bone et al., 2008) an accurate line laser and a camera is used to build models and iden- tify grasping points for novel objects with very encouraging results. However the system was tested only on two objects, thus its scalability is not clear.

Available models of complex objects are decomposed into superquadric parts in (Biegelbauer and Vincze, 2007) and (Zhang et al., 2004), and these models are matched to a point cloud. This needs a database of models, and moreover, their decomposition into superquadric components, which is often difficult to obtain. A random sample consensus (RANSAC) based approach for model decomposition is presented in (Schnabel et al., 2007), where a set of 3D geometric primitives (planes, spheres, cylinders, cones and tori) are fit to noisy point clouds. Since the point clouds presented there are complete, the authors do not need to reconstruct the missing parts.

In (Thrun and Wegbreit, 2005) the authors describe a method for detecting and verifying symmetries in point clouds obtained from a single viewpoint, and they project the existing points according to the detected symmetry to obtain the back side. However, using our methods we were able to reconstruct surfaces by approximating them with shape equations and thus generate a complete model, which can be meshed with the required density or sparseness according to the speed and accuracy requirements of our applications.

3.3 Object Classification

There are two principal mainstream lines in the area of the object recognition related research: one aiming at recognition of objects in camera images, and one using 3D depth data acquired through range scanning devices. Combining both of them leads to a hybrid approach and our work falls into this category. Depending on the type of perception data, various different 2D (e.g. (Lowe, 2004)) and 3D (e.g. (Rusu et al., 2008a)) distinctive local features have been developed. Taken individually however, they are still insufficient to solve the full object recognition problem as both are prone to failure in situation where texture-less objects are present or depth data is too noisy or ambiguous. Therefore, different research initiatives have decided to combine sets of local features and cluster them together using different metrics (kernels) in order to be able to infer the global identifiers for objects.

In (Fergus et al., 2003) the authors are proposing an unsupervised scale-invariant learning scheme, in order to detect objects on a wide range of images. Ob- jects therein are modeled as flexible constellations of parts using a probabilis- tic representation for all significant aspects of the object. The work exploits the expectation-maximization algorithm in a maximum-likelihood setting. The

method in (Romea et al., 2009) estimates 6 degrees of freedom (DOF) object poses in cluttered scenes by matching local descriptors to stored models. Since the objects present in household environments are often texture-less, our approach constitutes an important advantage over the above proposed research initiatives, which fail to work in the absence of well textured objects.

Another approach to obtain 3D information directly from camera images is to project computer aided design (CAD) models from a database to the image and search for good matches in the edges domain, as in (Ulrich et al., 2009) for example. While this is a more direct method, it is still dependent on a database of CAD models.

The work in (Ruhnke et al., 2009) uses an iterative matching procedure to merge similar models in an unsupervised manner. However, it is unclear how well the proposed algorithm would generalize to unknown, novel objects. The authors of (Lai and Fox, 2009) perform outdoor laser scans classification combining man- ual labeling and data downloaded from the Internet in an effort to achieve what the authors call domain adaption. While their presented recall curves outperform others, the number of objects is relatively low and household objects are less dis- tinct. In (Steder et al., 2009), the authors investigate the extraction of GOODSAC point features and object recognition from range images that are in turn computed from point cloud data sets. These object models are, as in our case, created from real 3D data but processed using the work in (Ruhnke et al., 2009).

The combination of depth information with camera images is addressed in (Quigley et al., 2009). The authors calculate depth information for each pixel in the scene
by applying laser-line triangulation with a rotating vertical laser and a camera. To obtain high resolution 3D images, each scan requires 6 seconds with an additional 4 seconds spent on post-processing and triangulation. Thus a waiting period of 10 seconds has to be expected before object detection and robot manipulation could be performed.

In the work of (Xue et al., 2009) the grasping of objects modeled in the 3D object modeling center (KIT Object Models Web Database) was

presented. The center employs a digitizer, a turntable and a pair of RGB cameras mounted to a rotating bracket which allows for views from above the scene. At the time being, there are around 110 highly detailed, high-precision objects publicly available. While working with such a system and data would yield high quality results, its downside lies in the fact that the modeling center is rather expensive and cannot be used online, i.e. mounted on a mobile robot for autonomous mapping. In

another initiative, a database (Columbia Grasp Database (Goldfeder et al., 2009)) has been built. The major difference between this work and ours lies in how the models are obtained. The authors created artificial 3D models whereas we acquired our models by scanning real world objects and surfaces, and are thus facing the problem of noisy and cluttered data.

4 Implementation Details

The following sections detail the functioning principles of the building blocks of our system as shown in Figure 2.

Our system is developed within the ROS open-source framework (Robot Op- erating System, www.ros.org), as a collection of modules – "nodes" , that each tackle a specific sub-problem, and can either be run separately or as dynamically loadable libraries – "plugins". Thus, the data flow between nodes can either oc- cur through shared memory, when optimal performance is required, or over the network, which is convenient for decentralized processing.

Using ROS and general examination methods based on PCL (Point Cloud Li- brary, www.pointclouds.org)enables us to be platform independent in the sense that the system can be run on different robots, and results can be inter- changed between them. So far, we have successfully performed our experiments on two different robots, TUM-Rosie (Pangercic et al., 2010; Marton et al., 2010a; Blodow et al., 2010) and the PR2 (Wyrobek et al., 2008).

Currently we employ the following methods to achieve the overall classifica- tion:

- Auxiliary modules (noise removal, kd-tree, octree – see below);
- Local feature estimation: Moving Least Squares (MLS) and Radius-based Surface Descriptor (RSD[2]) – Section 6.1;
- Global (view-based) features: Global RSD (GRSD[3]) feature (Section 6.2),

[2] http://docs.pointclouds.org/trunk/classpcl_1_1_r_s_d_

estimation.html

[3]http://code.cs.tum.edu/indefero/index.php//p/mapping/source/tree/master/pcl_cloud_algos/src/pcl_cloud_algos/global_rsd.cpp

Scale-Invariant Feature Transform (SIFT) (Lowe, 2004) feature using Vo- cabulary Trees[4](Section 6.3);

- Building geometric models: box/cylinder/rotational model fitting and trian- gulation – Section 6.4;
- Categorization using Support Vector Machines (SVM) as described in (Chang and Lin, 2001) and classification using SIFT descriptors with vocabulary trees.

To increase the performance of the 3D data processing steps, we rely on noise removal (Rusu et al., 2008b), and efficient spatial decompositions such as kd-trees and octrees. Noise removal is performed to remove jump edges in the laser scans and to remove points from noisy or low density areas of object clusters. We use kd-trees to find a fixed number of neighbors or all the neighbors in a sphere with a given radius, while octrees are useful for down-sampling and occupancy grids, but also for the verification of the geometric models (Blodow et al., 2009), and for computing object-level features efficiently, as in subsection 6.2.

5 Segmenting Object Hypotheses

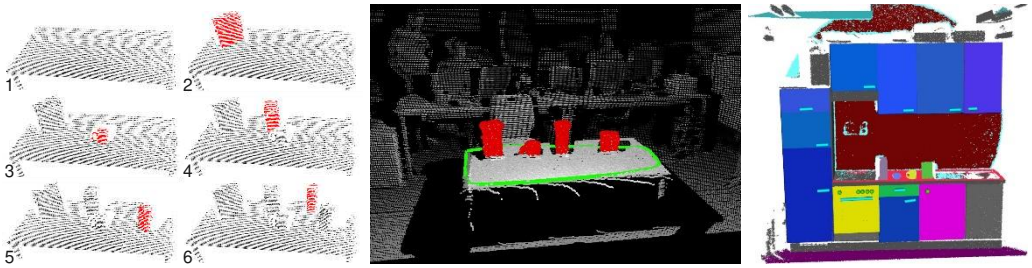


Figure 4: Different object detection methods, from left to right: background subtraction, planar support, and using environment maps.

Approaches to segment camera or point cloud percepts into distinct objects can be roughly split into two paradigms: Locating an instance of a

known object in the scene, using e.g. appearance models or feature descriptors, or segmenting the

[4] http://www.ros.org/wiki/objects_of_daily_use_finder

sensory data without prior knowledge about the objects or their views in order to acquire information about them.

In the first case, a direct search for each appearance can be performed, or a set of feature words can be extracted from the sensor data and matched to a model database as in (Torres et al., 2010; Hinterstoisser et al., 2010). These approaches are usually limited by the fact that for locating one or more of a very large number of possible objects, performance can decrease significantly.

In the second case, as the objects to be detected are unknown, it is possible to segment scenes under certain assumptions, as presented in Figure 4:

- *Background Subtraction:* Supposing successive addition of objects to the scene, one can easily detect successive changes even in cluttered scenes[5].
- *Planar Support:* Supposing a detectable support that assures physical sta- bility allows for segmentation of objects in an euclidean sense, as long as the objects are apart from one another (Rusu et al., 2009a). However, too much clutter is not acceptable in this case.
- *Environment Maps:* Supposing a known 3D environment, such as stored in a Semantic Object Map, makes it possible to define regions of interest in which objects are expected, which can then be processed as in the *Planar Support:* method. This approach is a natural extension to the 3D semantic mapping efforts.

The latter of these alternatives requires obviously the largest amount of knowl- edge about the environment. Since we have a static semantic environment map of our kitchen (Rusu et al., 2008b), we generated a URDF (unified robot description format[6]) map to describe the geometrical and kinetic properties of our environ- ment. Every link in a URDF tree describes a part or reference frame in the map (e.g. walls, furniture items, appliances, doors, handles, etc.), and links are con- nected with joints of varying types, such as fixed for rigid links (such as those between walls and the floor), prismatic (such as drawers), or revolute (for rota- tional joints such as door

hinges).

Applying labels to certain links, such as *counter top*, *drawer*, *door*, *wall*, and *shelf* lets us specify search filters that help pre-segmenting the raw point data

---

[5]More advanced methods include the learning of a foreground/background model.

[6]www.ros.org/wiki/urdf

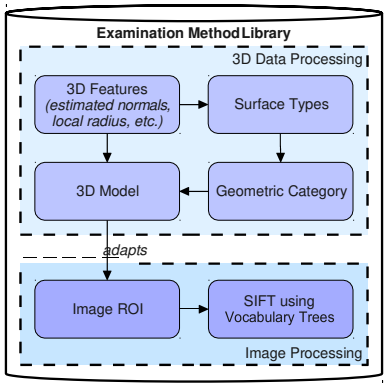


Figure 5: The conceptual repre- sentation of the complete pipeline that can be formed by the examination methods.

into meaningful regions, such as on top of certain counters, or the region in front of doors. The filters can reuse the geometric descriptions of the collision shape model stored in the URDF description and thus adapt the filter routines to each link label instance's distinct properties. These regions are processed by special routines that can e.g. determine the opening angle of a cupboard door or the location of an extended drawer, or simply delete all points that represent walls to reduce further processing time. This results in faster processing times, since only some regions of the raw input data need to be examined, and every specialized processing step is executed solely on the task-relevant data.

For the scope of this publication, objects are located on supporting horizontal structures such as tables, shelves or other furniture, so we focus on object clusters found on them. However, for some of the training and test data acquisition, we fell back to the *Planar Support* method.

An interesting aspect of the tree-like structure of the map is the fact that for arbitrarily aligned rectangular or box-like search spaces, one can transform the in- put data such that the search space becomes axis-aligned. We then identify com- plete subtrees in the map that share the same rotation and cache the transformed point cloud such that all search operations for these links can perform very fast axis-aligned bounding box

tests with a minimal amount of point cloud transfor- mations. A detailed analysis of this map-based pre-segmentation approach falls outside the scope of this paper and will be addressed in a separate publication.

6 Examination Methods

In this section we present the examination methods used by our system, and their possible connections as summarized in Figure 5. Each method has a common in- terface, with initializing, processing, and result validating and returning functions, that enable the easy extension with additional methods if needed. Each method lists the data type it requires and provides, and the parameters are obtained through the parameter server of ROS. We are currently making efforts to enable the def- inition of processing pipelines using configuration files, but as for now the steps are hard-coded.

6.1 Estimating Descriptive Local 3D Features

Since the point cloud coming from the 3D sensor is very noisy, we are using a Moving Least Squares algorithm to smooth it. To speed up the process, and considering that small details are not detected by the laser which has errors in the centimeter range, we have simplified and thus sped up the Robust MLS method described in our previous work (Marton et al., 2009b), which was developed for a more accurate laser. To process the 3D points obtained by the Hokuyo laser, we use the Principle Component Analysis (PCA) approximation of the tangent plane's normal, and re-fit the points and normals using MLS as shown in Figure 6.

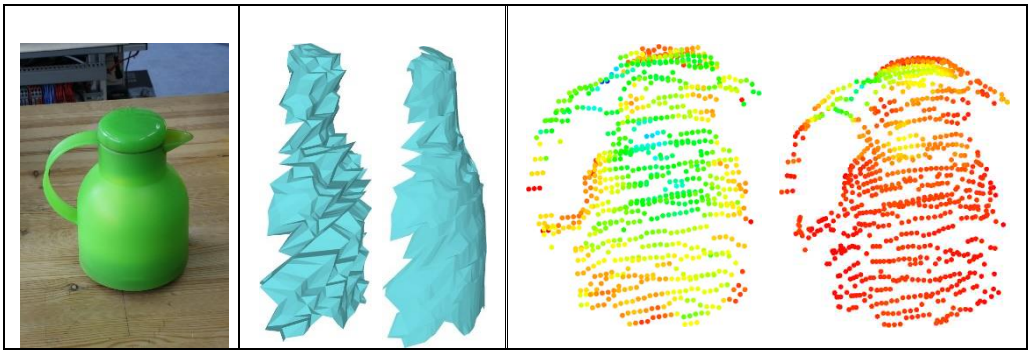


Figure 6: Left: Photo of a teapot. Middle: rendering from the side of the raw scan of the teapot obtained from the laser and the points projected

onto the estimated underlying surface. Right: the estimated surface curvature values before and after MLS.

While estimated surface normals are important, they alone tell little about the

surface's type. The estimation of the curvature using the ratio of eigenvalues of a neighborhood's covariance matrix on the other hand neglects surface normals and we found it to be too inaccurate. As argued in (Marton et al., 2010a), the approx- imated radii of the smallest and biggest fitting curves to a local neighborhood are values with physical meaning, which can be tied directly to the underlying sur- face without the need for classification. We compute this feature (Radius-based Surface Descriptor, or RSD) starting similarly as in the case of spin images with a local support (Johnson, 1997), with the added advantage that we consider the normals of the neighboring points directly, and that we can extract values that intuitively describe the local surface (as shown in Figure 7).

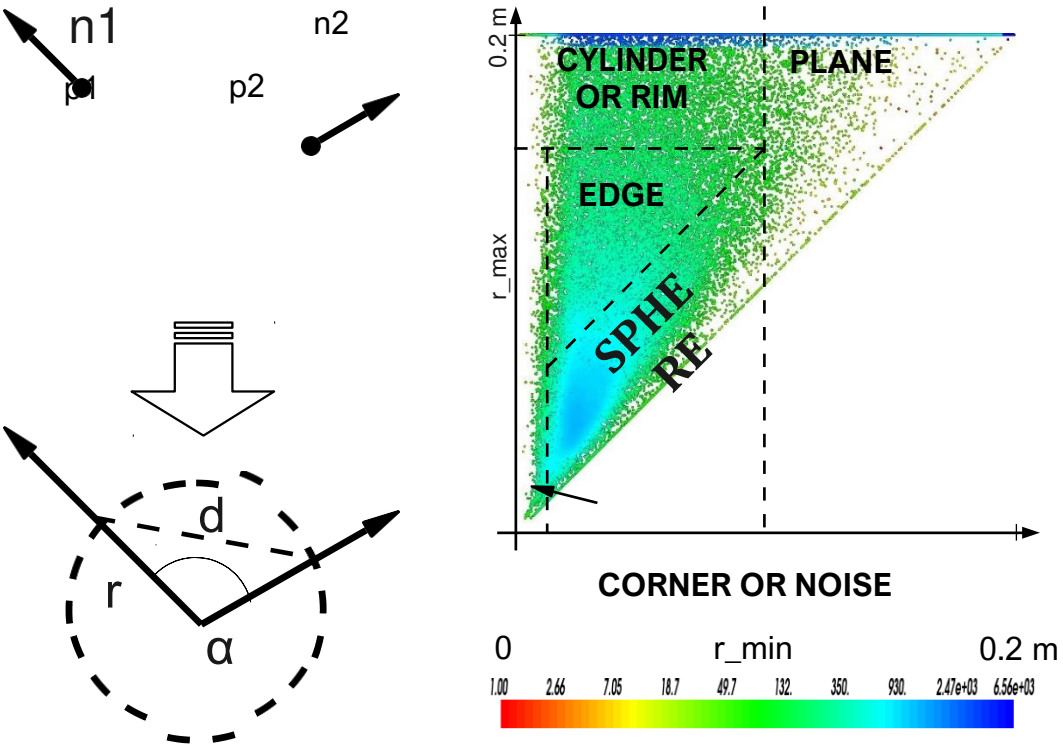


Figure 7: Left: Illustration of how a distance and an angle between normals define a radius as per Equation (1). Right: Identifying same number of surface classes as using FPFH without classification. The plot shows the log density/number of points in a 1 cm range in the feature space of r_{min} and

r_{max} for a variety of objects, with the rough ranges corresponding to certain shapes marked.

If we look at the case of the sphere, for each point all the circles that fit to its neighborhood have the same radius, namely the radius r of the sphere itself. For each of these circles we can write the following relation between the distance d of a point on the sphere from the original point and the angle α between these two

points' (undirected) normals:

$$d_{(\alpha)} = \sqrt{2r} \sqrt{1 - cos(\alpha)} \quad (1)$$

From this we can see that given the distance and the angle between two point normals one can approximate the radius of the circle the point is on.

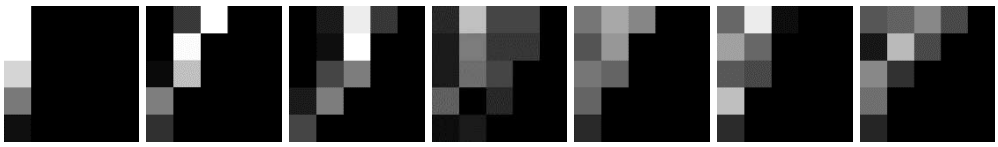


Figure 8: Computation of the Radius-based Surface Descriptors (RSD): 2D histograms of normal angle difference and distance from a reference point to its neighbors (examples are shown from left to right for histograms computed for points on a plane, sphere, corner, edge and three cylinders, where the last two are computed for different points of the corrected teapot scan presented in Figure 6) – from these histograms the physical surface radii can be read as the slopes of the different lines going through the lower left corner.

In the case of an ideal plane, this estimated radius will always be infinite with all neighbors, since they have parallel normals. A point on a cylinder is on mul- tiple circles (ellipses actually), and the radius estimated with different neighbors will vary between the minimum radius (the radius of the cylinder) and the maxi- mum radius (infinity). For corners and edges the estimated radius changes simi- larly as for spheres and cylinders (see Figure 8).

Given a point on a surface along with its neighbors, this minimum and maxi- mum radius can be estimated using the model in Equation (1) by solving the equa- tion system for r given the maximum and minimum angles for different distance intervals. To make the estimation easier, we can exploit the fact that α $[0, \pi/2]$[7] and the Taylor decomposition of Equation (1) is simple:

$$d_{(\alpha)} = r\alpha + r\alpha^3/24 + O(\alpha^5) \quad (2)$$

where $O(\alpha^5)$ indicates the existence of elements with order 5 and upwards. Thus we can assume $d\ r\alpha$ which renders the problem of finding r to a simple regres- s̃i̊on – as seen in Figure 9.

To find the minimum and maximum radius of a point based on its nearest neighbors, we compute the minimum and maximum angle in each

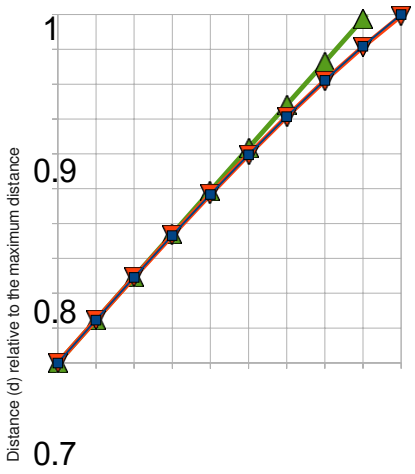
---

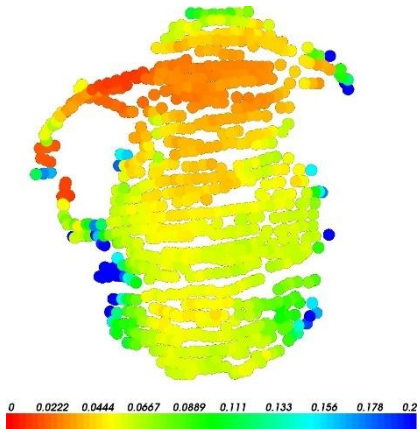
of 5 distance

[7] since we don't assume that normals are oriented consistently we compute α as the angle between the lines defined by the two points and their normals

bins, and compute the corresponding radius value. This introduces a small error (r_{min} is typically under- and r_{max} over-estimated), which we found to be less than 1 cm for all points of a set of 10 synthetic spheres with different surface radii R (for which $r_{min} = r_{min} = R$). For the real scan of the teapot in the right part of Figure 9 we found similar errors (its top cylinder having a radius of 4 cm and the bottom 7 cm).

In the cases where the two normals do not form an equal angle with the dis- tance vector, the algorithm would assume they do and compute the radius like that. However, as the process is repeated for every point pair, and we are only interested in the minimum and maximum radius, these small errors are either evened out, or produce the typical result for curvature computation – i.e. estimated radii on a plane start dropping from very large to very small values the closer the point is to an edge. In the case of RSD this is also aided by the fact that the previous surface and normal estimation step also relies on a neighborhood of points. Please see Figure 14 for an example involving sharp edges (tea box).





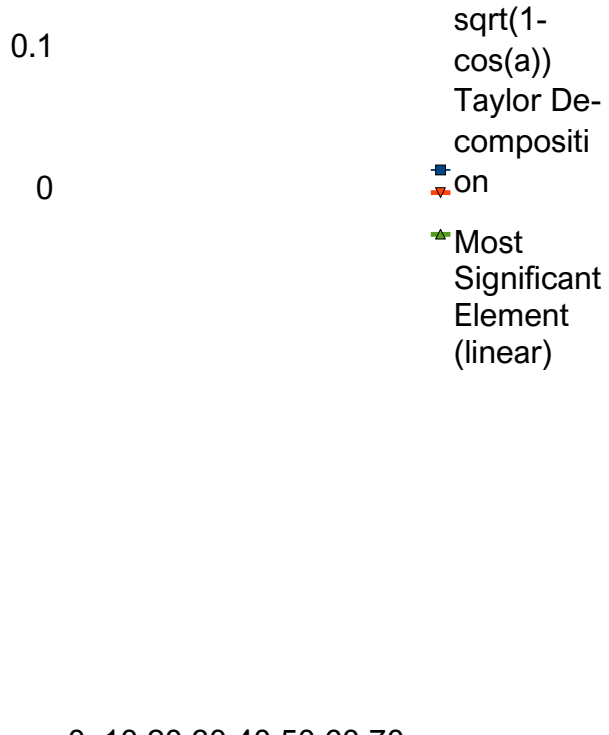


Figure 9: Left: variation of point-to-point distance by angle, measured from the center point of a half-sphere. Right: radius estimation results for a scan of a teapot – values are color-coded from red (radius of 0) to blue (radius of 0.2 m or higher). Note that for visualization purposes the values were capped at 0.2 m, as planar regions have an infinite or very large radius. For the accurate color information we kindly direct the reader to the electronic version of the article.

This feature is easy to compute, while still being very descriptive (see Sub- section 6.2) and does not require consistently oriented normals. Because it is a continuous value that estimates the real minimal metric radius of the curve each

points lies on, it can be used for example as a prior when sampling points to fit different surfaces.

In our case, points with very high estimated minimal radius are preferred when fitting boxes, while they are avoided along with points having a very small radius when searching for rotational models. We incorporated this preference in the ran- dom sampling step for each model fitting algorithm. Since the surfaces scanned with these noisy sensors appear to be "bumpy" even after MLS, the estimated min- imal radius is in some cases smaller than it ideally should be. Still, they provide our RANSAC methods with a fast and accurate weighting for the points, depend- ing on the model to be fit. The inliers are also weighted by how well their radius fits the model.

Based on the minimum and maximum radius it is possible to devise simple heuristic rules that categorize the surface types and enable the computation of a global descriptor, Global Radius-based Surface Descriptor (GRSD), as detailed in the next subsection.

6.2 Object Categorization

We are using a two layered classification scheme for geometric categorization (Marton et al., 2010b), where local surface labels are combined in a global de- scriptor. To speed up the local surface classification, we label voxels directly instead of individual points, as those are needed for the global classification. This reduces the complexity proportionally to the average number of points in a voxel.

In each voxel with a width of 2.5 cm, we compute the minimum and maximum radius and label the surface by successive checks of the radii: planar ($r_{min} > 0.1$), cylindrical (if not planar and $r_{max} > 0.175$), sharp edge/corner or noisy (if not cylindrical and $r_{min} < 0.015$), spherical (if not edge and $r_{max} \ r_{min} < 0.05$) and rim (in the remaining cases). This is a simple and fast way to categorize surfaces (see Figure 7), and it divides the feature space formed by the radii well enough for the computation of a global feature. We picked the local surface labels to have an intuitive significance, but the actual correctness of these labels is not relevant for the global classification.

Once all voxels are annotated locally using a geometric class, our processing pipeline constructs a global feature space that can produce a unique signature for each object cluster. This space is based on the idea that, for a set of labeled

voxels, a global feature can be constructed by observing the relationships between all these local labels (and the encompassed free space). Since the labels represent geometric classes obtained from the classification of RSD descriptors, we call this new feature the Global Radius-based Surface Descriptor (GRSD).

Algorithm 1: GRSD Computation

L = $\{l_1 ... l_m\}$ /* octree leaves encapsulatingthe object */

C = $\{c_{l1} ... c_{lm}\}$ /* the leaf classes generated by *RSD* (e.g. class empty, class plane, etc.) */

H = ZeroMatrix /* transition matrix holding the number of neighboring classes */

foreach $l_i \in L$ **do**

  foreach $l_j \in L$ **do**

    $r_{ij} \leftarrow (l_i, l_j)$ /* create a line segment between l_i and l_j */

    $I = \{l_k | l_k = L \cap r_{ij}\}$ /* get leaves intersected by r_{ij} */

    /* count the class changes between leaves in *I* */

    foreach $l_k \in I$ **do**

      $H[c_{lk}, c_{lk+1}]$ + + /* increase counter */

The complete algorithm is described in Algorithm 1, and Figure 10 shows two sets of feature vectors for different objects generated by the GRSD estimation. We were using the Octomap library (Wurm et al., 2010) to facilitate ray intersection tests in the voxels encompassing the objects.

The computation of GRSD is similar to GFPFH (Rusu et al., 2009b), but

H

we use voxel-based labeling based on RSD, and we sum up the individual his- tograms instead of computing their distribution, to further reduce computational complexity. This way, the complete processing of a cluster (correcting, estimating normals, computing the voxelized RSD values, labeling voxels and constructing the GRSD) takes between 0.2 and 0.5 seconds (depending on object size) on a single core working at 2 GHz.

6.2.1 Selection of Geometric Categories

The geometric categories were selected based on an analysis of the following databases containing objects of daily use: TUM Organizational Principles Database

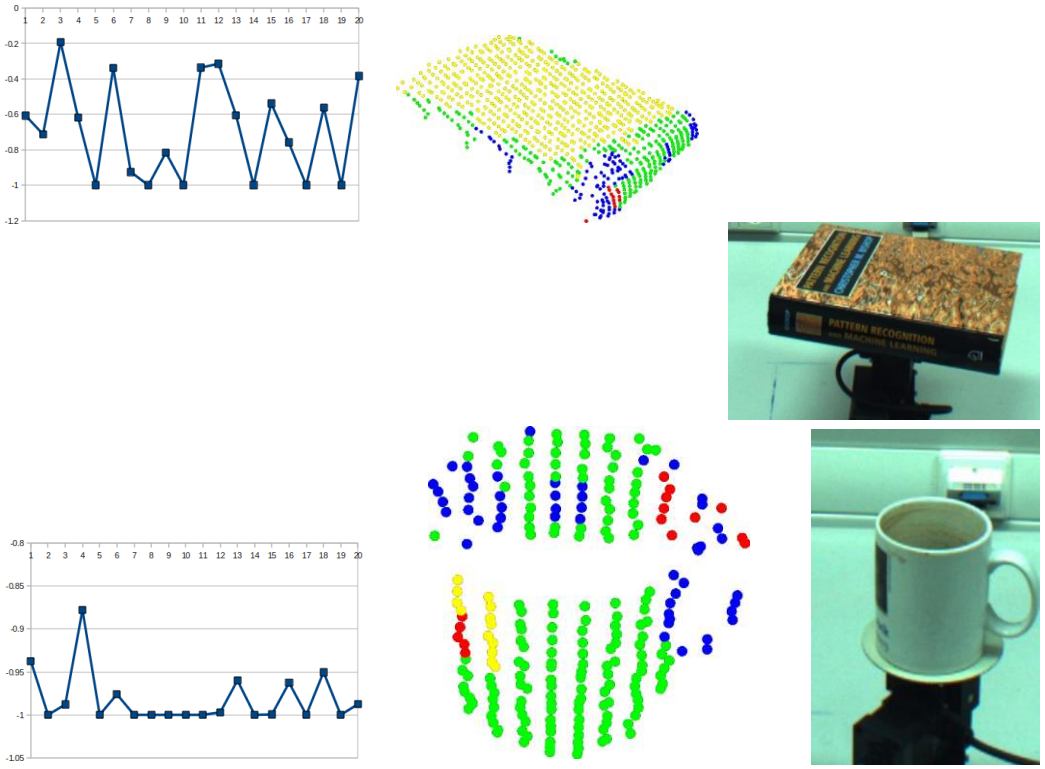


Figure 10: Plots of GRSD histograms (left) and identified surface types based on RSD (middle) for a flat box (i.e. book, upper row) and a cylinder (i.e. mug, bottom row). Left: the GRSD histogram bin values are scaled between -1 and 1 (based on the value ranges found in the training data). Middle: the colors represent the following local surfaces: red

- sharp edge/corner (or noise), yellow - plane, green - cylinder, light blue - sphere (not present), and dark blue - rim (i.e. boundary, transition between surfaces). For the accurate color information we kindly direct the reader to the electronic version of the article.

– **TUM-OPD**[8], A List of Household Objects for Robotic Retrieval Prioritized by People with ALS – **ALS** (Choi et al., 2009), KIT Object Models Web Database

– **KIT**, Semantic 3D Database – **SemanticDB**[9] and Household Objects Database from Willow Garage – **WG**[10].

We selected 6 geometric categories, covering most of the objects a robot could perceive and manipulate in a kitchen, namely: *mug/can sized cylinder*, *flat box*, *pressed-top box (tetrapak)*, *box*, *tube-like cylinder* and *rotationally symmetric*. The statistics done over the different databases (Table 1) reveal a comparably

---

[8] http://ias.in.tum.de/kb/wiki/index.php/Example_images_of_kitchens

[9] http://ias.cs.tum.edu/download/semantic-3d

[10] http://www.ros.org/wiki/household_objects_database

Category	TUM-OPD	ALS	KIT	SemanticDB	WG
mug/can sized cylinder	93	3	26	9	68
flat box	23	5	28	4	3
pressed-top box (tetrapak)	4	0	1	4	1
box	42	4	23	4	2
tube-like cylinder	12	1	6	3	18
rotationally symmetric	46	2	11	0	21
other	99	10	15	11	19

Table 1: Frequencies of the selected categories of objects in different known databases of objects of daily use – based on hand-labeling through visual inspection.

small number of objects (like scissor, money, sunscreen) that do not fall into one of the above categories (154 out of 621, i.e. just under 25%). We included the *other* category into our geometric classification, and merged rotationally symmet- ric objects like teapots, bottles, etc. into it. Since we group each view separately during training, adding very different objects to the *other* category does not make the category more ambiguous for the classifier.

6.2.2 Evaluation

We trained each view of objects from the categories listed above as a different class using an SVM classifier. We used 934 individual views for training, at least 2 objects of each type scanned from different angles. For the final classification only the general category was considered, to avoid mistakes due to similar views, thus the model's accuracy for 174 newly segmented scans of the training objects (i.e. test views) was 93.68%, and we tested it using 205 views of untrained ob- jects and had a success rate of 85.37%. The corresponding confusion matrices in Figure 11 show that we obtained good results for the distinctive categories, and had around

30% confusion rate between two of the categories (between the side views of flat boxes and relatively thin tetrapaks). The existence of similar views between differently shaped objects also make the automatic learning of geometric categories quite difficult, producing analogous results to Figure 1, further support- ing the need for combining perceptual mechanisms.

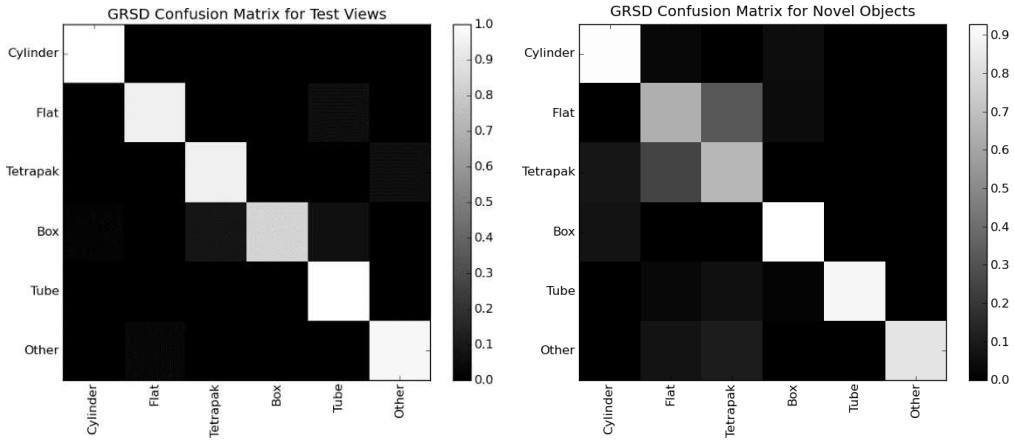


Figure 11: Confusion matrices for new scans of the trained objects (left) and scans of new objects (right) provide the true and false positive normalized statistics for the geometric classes (neglecting the view classification results) as discussed in 6.2.2. Ground truth is shown in rows, and classification results in columns. Please note the different scales in the two plots.

6.3 Visual Feature Classification

The visual feature detection is carried out only on the region of interest (ROI) in the image, in order to avoid false positive matches with the background. To obtain the ROI the robot simultaneously takes a 3D scan and captures an image of the scene in front of it. The robot generates object hypotheses by detecting candidate point clusters in the 3D point cloud acquired by the depth sensor. These object hypotheses are then back-projected into the captured image as regions of interest which are used for detecting and recognizing objects.

For the SIFT-based detection we first determine segments by performing re- gion growing on detected features in image space which typically results in an over-segmentation of the region of interest.

Identifying the object and the image region it belongs to is then performed through methods transferred from document retrieval. In document retrieval tasks as for example in web search we look for the documents that best match a given query term. To do so the search engines compute a frequency statistics for discriminative words, word stems, as a pre-processing step performed on all documents. Given a search term, fast indexing mechanisms quickly search for the documents that are with respect to frequency

particularly relevant for the search term. The application of text retrieval technol- ogy for object matching is promising because it is very mature and the techniques aim at having high recall and precision rates while at the same time being very fast.

The computational idea of textual document retrieval can be mapped to object description matching in the following way. The descriptors computed from the regions of interest that belong presumably to (or are partial views of) the same object are considered to be the search term. The object descriptors of the different views of the relevant objects are the documents of the document retrieval model. Word frequencies are replaced by the frequency of visual object features. Given a large set of objects, represented by their object descriptors, and the feature descriptor of an image region we can then index the objects where the particular features are particularly prominent using the respective methods from document retrieval.

In this paper we apply vocabulary trees for weighted TF-IDF (Term Frequency Inverse Document Frequency, (Robertson, 2004)) indexing, a method used in doc- ument retrieval to find the documents that best fit a given textual user query. In this reformulation of object identification vocabulary trees speed up the retrieval of the matching objects.

The methods for object descriptor matching do not only match a given region descriptor to the large set of object descriptors but they can also learn new object descriptors to be put into the visual object library (see Figure 12).

6.3.1 Vocabulary Tree

The vocabulary tree approach to object classification is based on the bag of words (Sivic et al., 2005) document retrieval methods, that represent the subject of a docu- ment by the frequency in which certain words appear in the text. This technique
has been adapted to visual object classification substituting the words with local descriptors such as SIFT computed on image features. The vocabulary tree of branching factor K and depth L is a tree data structure

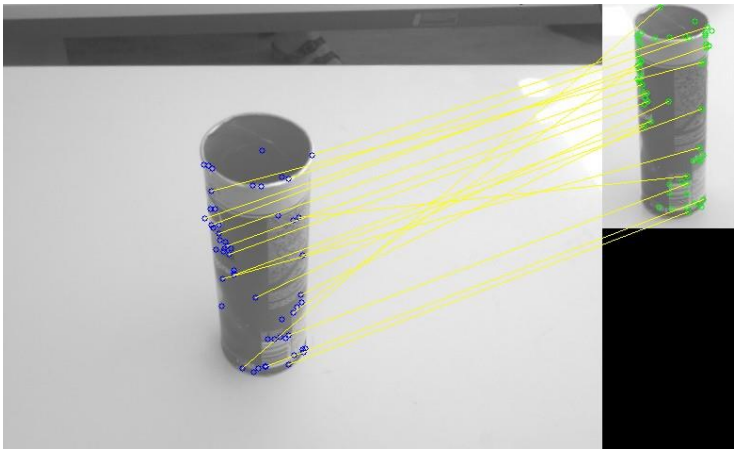
N

N

where the nodes in the tree represent a set of SIFT descriptors. The root node of the vocabulary tree repre- sents the SIFT descriptors of all views of all object models in the library. If a node n in the vocabulary tree represents the set of SIFT descriptors then its children nodes represent the partitioning of into k subsets represented by the children nodes cn_1 . . . cn_k where the SIFT descriptors within a children nodes are similar



(a) View 1 (b) View 2 (c) View 3 (d) View 4 (e) View 5



(f) Successful match against an online learned appearance model

Figure 12: Top row: Different robot views used for online learning of SIFT appearance models. Bottom row: Successful match against an online learned appearance models.

and the ones of different children nodes dissimilar.

Thus by taking a SIFT descriptor *sd* and classifying it hierarchically through the vocabulary tree using the defined distance measure on the SIFT descriptors we quickly find the set of most similar SIFT descriptors in the *Object Model Database* as the leaf nodes whose representative SIFT descriptors have the smallest dis- tances to *sd*. For efficiency *sd* is not compared to all features in a given node, but to the centroid of its features.

The SIFT descriptors in the vocabulary tree also have a reference to the object model they occur in. Thus, when *sd* matches a leaf node it votes for the object models that the SIFT descriptors of the identified leaf belong to.

The children nodes $cn_1 \ldots cn_N$ of are computed by applying k-means clus- tering to the SIFT descriptors of node n. Since the weighted TF-IDF algorithm works on words (equivalent of leaf nodes), we use a vocabulary tree to convert the keypoint descriptors into words, where each word is an integer value correspond-

ing to the number of the leaf node.

Building the database

In our approach we use a similar database (*Object Model Database*) as described in (Nister and Stewenius, 2006). In order to be able to detect objects the database stores only the quantized SIFT features of the images, but not the images them- selves.

Extracting SIFT features

In order to extract the visual SIFT features from the images we use an open- source implementation of the standard SIFT algorithm (Fast SIFT Image Features Library[11]) as initially described by (Lowe, 2004). Each SIFT feature is charac- terized by a 128 dimensional descriptor vector, 2 image coordinates, a scale and an orientation value. In the current implementation we use only the descriptor vectors for the detection process and the image coordinates for visualization.

Generating database documents

After we have the vocabulary tree, we quantize feature descriptors to single words. For every image, we take all SIFT features, we quantize them with the vocabulary tree and we group the resulting words in one document for every image. In this way the document is just the list of all quantized features corresponding to a single image.

Populating and training the database

After we have all image documents, we insert them into a specialized database as proposed in (Nister and Stewenius, 2006). The database is then trained with the weighted TF-IDF algorithm. After that it can be queried with documents gen- erated from the input camera images in order to find the best database matches between objects in the image and the objects in the database. The database docu- ments along with specific database information can be stored in a binary format in order to allow for fast loading of the database. Additional information like image file names, textures and feature coordinates are also saved for the visualization purposes.

The whole detection process is implemented as a single ROS node, which receives an image coming from the camera and outputs the most probable matches from the database.

[11] http://sourceforge.net/projects/libsift/



Figure 13: Example of matching SIFT features for different objects from the imported database.

6.3.2 Retrieving Object Models

In order to find an object in the received image we have to generate a database document in the same way as described in the section above. We first extract the SIFT features from the received image and we quantize the descriptor vectors to words with the vocabulary tree. A single document is formed from all words of the input image and we can query the database with it. The database returns the best *N* matches with their respective scores *s* (between 0 and 2, where 0 is best and 2 is worst). A score is calculated by comparing the two documents (d_1, d_2) in a scoring function realized as an L2-norm metric:

$$s(d_1, d_2) = \frac{d_1}{\|d_1\|} - \frac{d_2}{\|d_2\|} \qquad (3)$$

$$2 \qquad 2 \quad 2$$

Above mentioned [0 . . . 2] interval is obtained through metric scaling.

6.4 3D Model Reconstruction

In this subsection, we will lay out the 3D reconstruction methods we integrated in order to create complete object model hypotheses that can be used in grasping scenarios. The focus in this subsection does not lie on

millimeter-accurate mod- eling of clusters, which is realistically not possible in a lot of cases due to sensor noise and large variability in the objects we encounter. Instead, we strive to im- prove the models generated from point cloud data in two aspects: First, we want the resulting reconstruction to be smooth – especially when compared to a simple meshing approach, where sensor noise causes problems in the grasp planner, e.g. when computing contact points and forces. On the other hand, we want to gen- erate a working hypothesis concerning the backside of an object perceived only

partially. Otherwise, grasp analysis might suggest a pinch grasp on the edge of the cluster, even though the object continues smoothly towards the unobserved back.

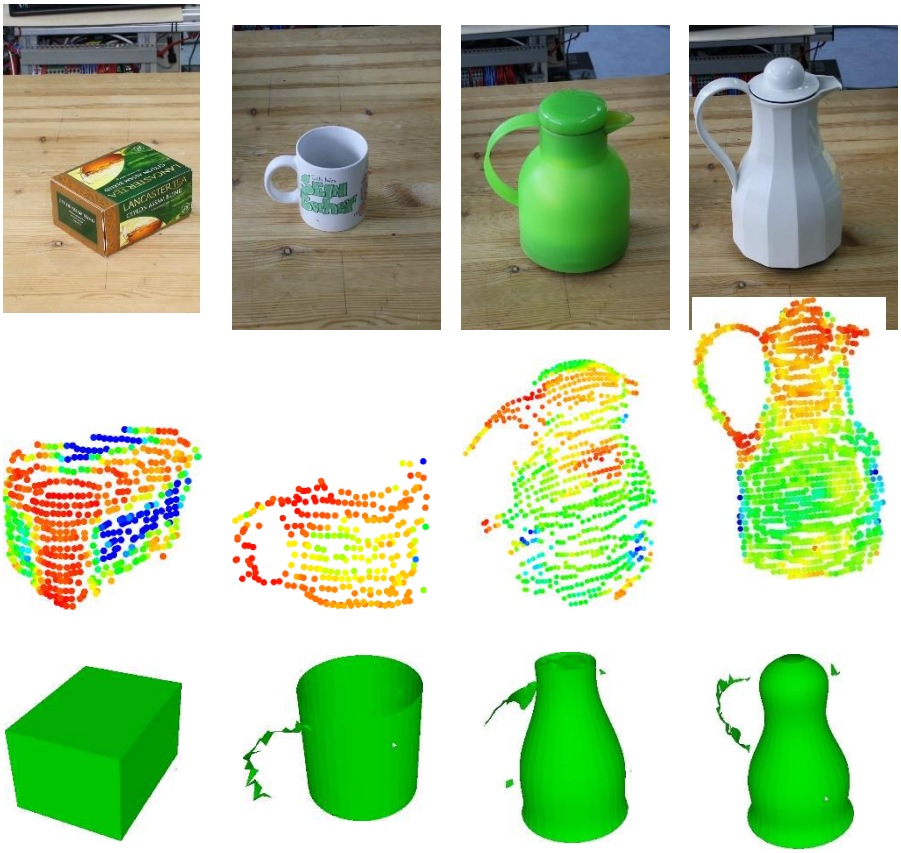


Figure 14: Automatic reconstruction of objects of different types. Top row shows the estimated r_{min} values color-coded by the legend of Figure 9 (red 0, blue 0.2 m or higher).

As of now, reconstruction algorithms have been developed for boxes, cylin- ders and rotational objects (Marton et al., 2010a), as shown in Figure 14. We chose these models because they are inherently symmetrical and

occur frequently in a large number of objects (as shown in Table 1). In the case of cylinders and rotational objects, a rotation axis is determined, which allows for simple comple- tion of the object model, and for boxes, the hidden surfaces can also be retrieved relatively straight-forward.

If a fitted model does not have at least 75% of the points as inliers, we assume that it has a more complex shape, and a simple triangulation is performed as a fall

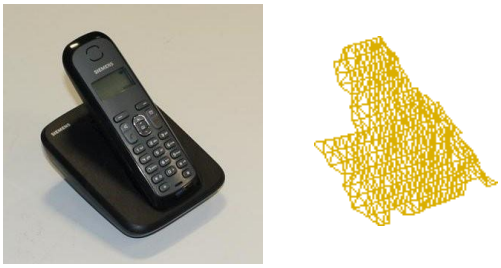


Figure 15: Triangulation of an object where no good model could be fitted.

back. Also in the other cases, the outliers of the geometric models are triangu- lated and added to the final model as a mesh, e.g. to model additional geometric features of an object, such as handles. Figure 15 shows such an example. As the majority of grasp planning methods use triangular meshes, the geometric shapes are decomposed in triangles as well, but we are exploring the usage of the shapes directly for more optimized grasping.

6.4.1 Box and Cylinder Fitting

The decision on selecting the appropriate fitting model is based on our previous work on footprint analysis (Marton et al., 2009a), but in addition, we rely on the 3D normals to robustly detect a circular or a rectangular footprint, also during the 3D model fitting and validation step. This enables our method to make the best choice regarding what model it should choose.

If a sufficient number of points have a minimum radius greater than 0.1 m they highly likely lie on planar surfaces and we set out to find the best fitting box to the cluster. Unlike in (Marton et al., 2009a), we fit a rectangular model directly to the points having normals perpendicular to the “up” axis (as we assume boxes to be statically stable, i.e. standing on one of their sides) and maximize the number of inliers in a RANSAC loop. This direct approach estimates the box orientation and outperforms the line detection and merging and/or box detection based on PCA. Since we assume a

correct segmentation and noise is removed in pre-processing, the box dimensions are computed from the oriented bounding box, and a final inlier count is performed.

For cylinders, we use a RANSAC approach which is based on the observation that on a cylinder surface, all normals are orthogonal to the cylinder axis, and intersect it. We consider the two lines defined by two sample points and their cor- responding normals as two skew lines, and the shortest connecting line segment

as the axis. Determining the radius is then a matter of computing the distance of one of the sample points to the axis.

The quality of box and cylinder fittings for two objects is presented in Fig- ure 16. Further tests on a total of 267 measurements of 5 objects show an average fitting error in 5.31 mm with a standard deviation (STD) of 2.69 mm for the ex- tensions of boxes, and slightly less for cylinder radii, an average of 2.85 mm and STD of 1.23 mm. Both are well below the original sensor noise of roughly 1 cm. The average error for the axis orientation for cylinders was 1.11 degrees with a STD of 0.34 degrees. By enforcing an upright position for the axes results are more robust, but it is not mandatory.

Compared to the original approach presented in (Marton et al., 2009a) we can see a clear increase in accuracy. On the same scans, the obtained average errors for box extents was 10.6 mm with STD 6.05 mm. Results did not improve (nor worsen) for cylinders as the methods have the same basis, but instead we are now reconstructing them in arbitrary poses, while the original approach assumed all objects to be upright (and would consider lying cylinders to be boxes).

6.4.2 Estimation of Rotational Surfaces

To reconstruct surfaces of revolution, we employ a RANSAC-based two-step ap- proach as described in (Blodow et al., 2009). In the first step, a rotation axis is estimated from sample points by minimizing a function over the line-to-line distances between the axis and the lines defined by each sample point and its cor- responding normal. This is based on the observation that for a rotational object, a line constructed from a point and corresponding normal intersects the symmetry axis, similar to the cylinder case. The contour line is then estimated in the second step (see below).

Let $(\boldsymbol{a}, \dot{\boldsymbol{a}})$ denote the axis, defined by a point $\boldsymbol{a}$ and a direction vector $\dot{\boldsymbol{a}}$ and let $(\boldsymbol{p}_i, \boldsymbol{n}_i)$ denote the line defined by the ith sample point and its corresponding normal vector. Then, we minimize the following function over $\boldsymbol{a}$ and $\dot{\boldsymbol{a}}$:

$$\sum^{m}$$

$$\sum_{i=0} d_{l,l}((\boldsymbol{a}, \dot{\boldsymbol{a}}), (\boldsymbol{p_i}, \dot{\boldsymbol{n}}_i))^2, \quad (4)$$

where $d_{l,l}$ stands for the line-to-line distance. This can be solved using a non- linear optimizer like Levenberg-Marquardt.

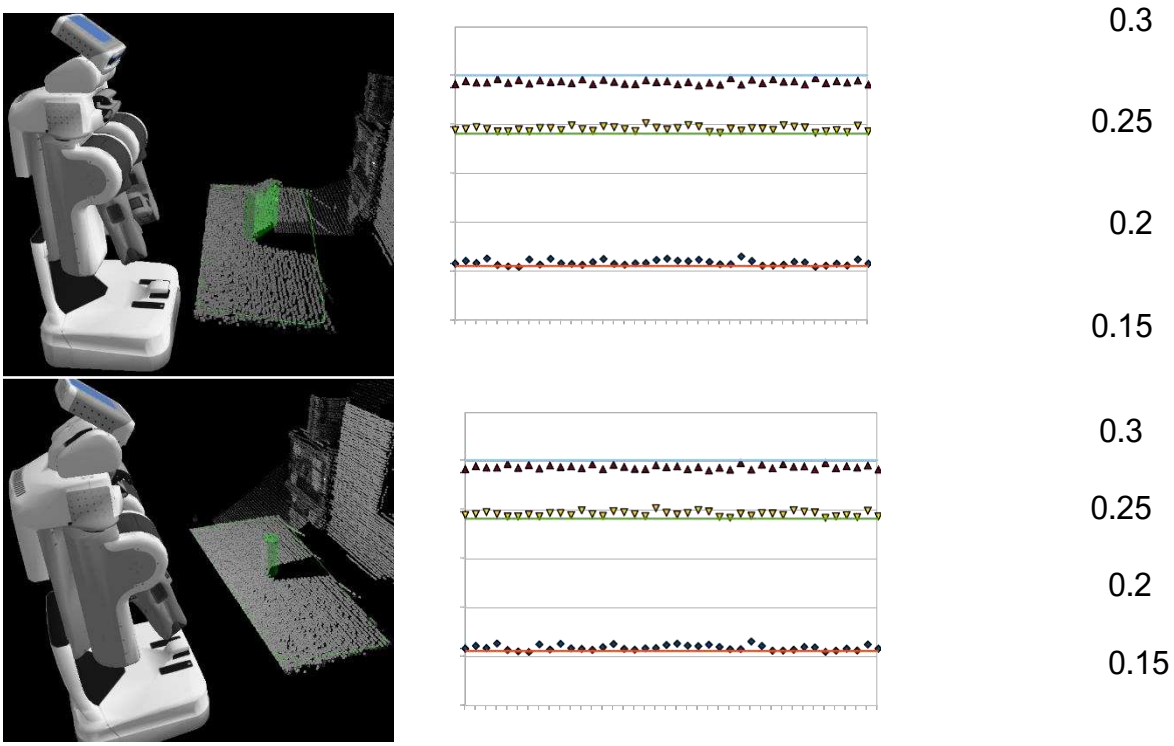


Figure 16: Left: visualization of the setup used for measuring fitting errors. The model and the table are marked with green. Right: errors in estimated box width (W), height (H), thickness (T) – and in estimated cylinder radius (R) and angle to the vertical (A). Graph shows these errors for subsequent scanning and fitting (X axis shows the scan number).

Once an axis has been found, the original sample points are

transformed into a 2D coordinate system such that the rotation axis coincides with the X axis. Every point $\boldsymbol{p}_i$ is projected onto a point $\boldsymbol{p}_{i,2D}$ whose x coordinate is defined as its position along the rotation axis, and whose y coordinate represents the point-to- line distance between $\boldsymbol{p}_i$ and $(\boldsymbol{a}, \dot{\boldsymbol{a}})$.

We then employ a polynomial fitting step based on least-squares minimization to fit a preliminary contour line to the projected (2D) sample points. This contour line is then used to determine which points are inliers to the rotational model, and the polynomial is refitted using these inliers. This can be repeated until changing the polynomial coefficients does not increase the number of inliers. An example of this 2D problem is shown in Figure 17-right for the teapot data-set used earlier. Note that the contour curve is not influenced by the outliers that come from the

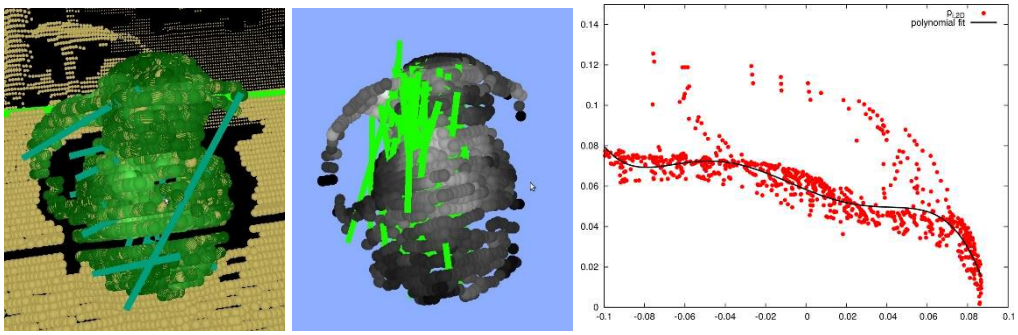


Figure 17: The axis estimation step fails consecutively if surface points and normals con- tain too much noise (left). Re-sampling the points results in usable axis estimates (mid- dle). Fitting a polynomial function to generate the contour curve given the axis (right).

handle, for example.

Unfortunately, the axis estimation step as presented in (Blodow et al., 2009) is sensitive to higher degrees of noise – which we encounter in less accurate 3D sensors. The minimization function (4) becomes less smooth and the optimization gets “stuck” in local optima most of the time, as can be seen in Figure 17-left. We alleviated this problem by performing the rotational estimation on smooth resampled point cloud data, which has much less noise and more accurate normals (see Figure 17-middle). Figure 18 shows the analysis of errors in determining the axis during rotation estimation.

This smoothing step alleviates some of the problems originating from noisy normal estimation, but it also loses detail on the underlying surface, such as sharp creases. The fitting step can therefore not be guaranteed to capture the ground truth model. In our experiments, the standard deviation of the distances of points to the model were found to be in the order of magnitude of 2 millimeters with a correlation coefficient of r^2 0.95, however this only represents how well the model fits the data points, not the underlying actual geometry. For a more detailed analysis, we have to refer to the previous work on these methods as the signif- icance of the herein presented fitting methods lies in the *completion* of partially perceived

models, and not in quality of fit of the reconstruction methods.

Although the rotational estimation is less accurate and computationally more intensive than fitting of simple models, it is an important addition in order to be able to deal with more types of objects (see Table 1). More accurate results can be obtained by increasing the expected probability of a successful fit for RANSAC

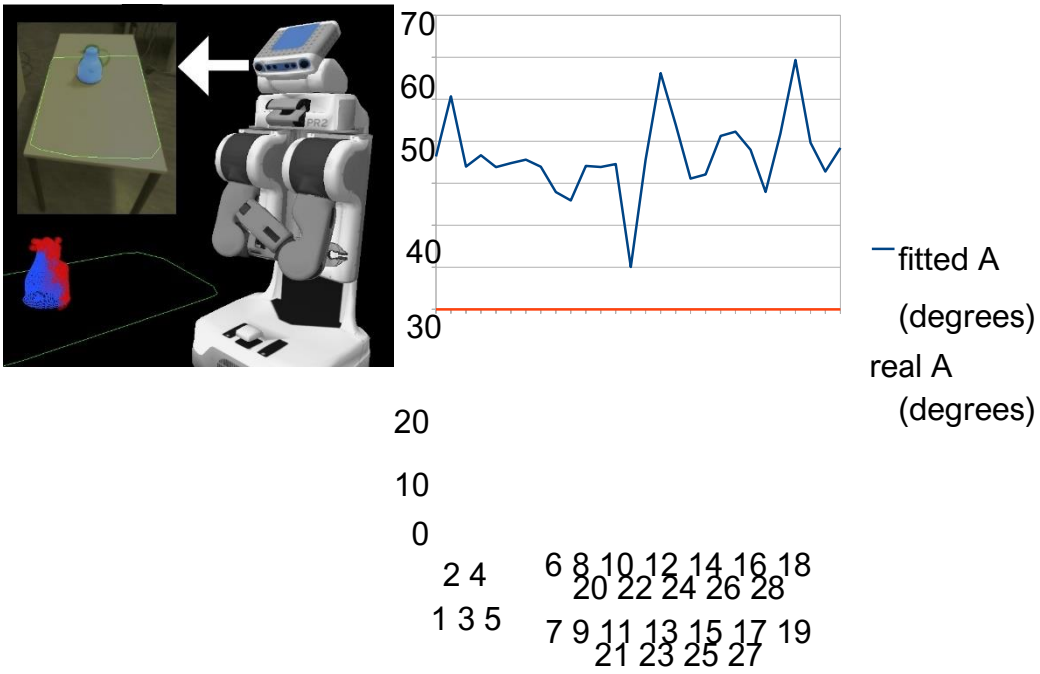


Figure 18: Left: visualization of the setup used for measuring fitting errors. Right: errors in the angle of the estimated axis to the vertical (A) if no visibility checks are performed in the optimization loop. The inaccurate fits are penalized by their parts in free space, and the more accurate ones, like at iteration 14, will be selected as the final model. Graph shows these errors for subsequent axis optimization steps, at different RANSAC iterations.

(and thus the maximum number of iteration), but in order to obtain results under 2 seconds, axis estimation errors around 10 degrees were obtained.

In the case of box-like and cylindrical models, the fact that the dimensions are computed accurately from partial views already validates the use of this approach for grasp planning. As presented in subsection 8.1 having a hypothesized back side makes a significant difference when the goal is to pre-compute realistic stable grasps. Similarly, this is an

important consideration for rotational objects as well, as considering only the scanned points does not give any estimate on how large the object is, and what the back side looks like. While we do not claim to have solved every problem related to this uncertainty, approaches such as ours or (Thrun and Wegbreit, 2005) provide useful hypotheses. Their full evaluation, apart from the fit quality presented above, is on our agenda, including the acquisition of enough ground truth data, performing and evaluating the grasping, and the analysis the effect of fitting errors on grasp stability when compared to reactive blob-grasping.

7 System Evaluation

To facilitate object recognition we maintain a database of the descriptors provided by the examination methods. It is important to note that not all descriptors must be

known due to the multi-modal nature and incremental learning feature of our sys- tem. The view-dependent variations of the descriptors are stored for every view, thus each view of an object that has been observed can be looked up and compared with object hypotheses during classification. The examination methods described in Section 6 provide the following data on a per-view basis: the geometric cat- egory, the appearance descriptors, the model parameters and the 3D mesh. The timestamped position of the last detection is saved for each object as well.

The following subsections detail the object identification and model learning modules using the above presented subsystems. We have to differentiate between the two operating modes, *active* perception, when a given object needs to be lo- cated, and *passive* perception of objects for updating their locations and models. Since the *Object Model Database* does not have to contain every possible infor- mation about the objects, it is straightforward to look for a novel object by adding a new entry that contains known information about the object.

7.1 Object Classification

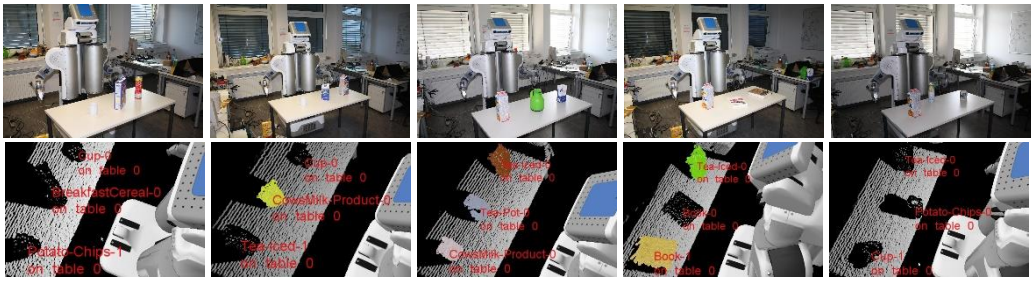
For object classification, we are iterating over the examination methods (Figure 5), run them on the object hypotheses to extract features (if possible), and search through the list of available objects in the database to see which ones present the same or similar features. The objects that match are kept in the list for the next iteration of comparison, and the others are removed. This way, at each step the set of possible objects gets reduced. As discussed before, if multiple objects remain after checking all features, the result is deemed ambiguous, whereas if at a given step no objects match, we consider the object hypothesis to be a novel object.

Due to the combination of features, an object candidate is accepted only if it matches all of the checked features. This reduces the effect of miss-classifications, at the cost of observed views of known objects presenting previously unobserved features being added as new objects (unless observed in subsequent scans). As the preceding process is started only if the object was not identified in the previ- ous step (otherwise the previous

identity is assumed), new views can be learned through continuous observation of an object as presented in the next subsection.

Figure 19 shows results we obtained with our first batch of 12 testing objects. For the 3D categorization we used the model trained as described in subsection 6.2 and the visual features were trained on the SemanticDB database as detailed in subsection 6.3. The tests show correct classifications, achieved in under 2 seconds

(not considering acquisition time). Please see the next subsections for a more detailed evaluation.



(a) Cup-0 BreakfastCereal-0 Potato-Chips-1

(b) Cup-0 CowsMilk-Product-0 Tea-Iced-1

(c) Tea-Iced-0 Tea-Pot-0 CowsMilk-Product-0

(d) Tea-Iced-0 Book-0 Book-1

(e) Tea-Iced-0 Potato-Chips-0 Cup-1

Figure 19: Classification results for a variety of objects. Clusters shown in random colors.

7.2 Detection based on Known Views

In order to evaluate our approach as a whole, we performed the detection and recognition test in our kitchen laboratory (see left column of Figure 20). The test was carried out on a total number of 13 objects located at 4 different scenes (de- noted with Scene 1 ... Scene 4 and depicted in top-down order in the right column of Figure 20). The robot was programmed to navigate to each of the scenes and capture point cloud and image from several different views by traversing along the free paths around the scenes. The basic *Planar Support* approach could not have been applied

for the scenes 2 and 4 as the supporting planes are too high, thus impossible to scan with either of our robots.

The vocabulary tree and corresponding database with descriptors were trained and built from images from the SemanticDB database and 10 more images of products from the http://www.germandeli.com web site. The parameters *K* and *L* were both set to 5, resulting in a 1 minute training time of the datas- base for the 65000 features extracted from 170 images. In this configuration the querying for 1 object cluster took 50 ms. Setting the score value of the database retrieval mechanism to the experimentally determined value of 1.0 enables us to classify all measurements that exceed this value as unknown.

Scene	**#Views/#Known**	**#Failures**	**#Unknown**	**Success** (with/no unknowns)
Scene 1	52/42	0	10	80.8%/100%
Scene 2	11/11	5	0	54.5%/54.5%
Scene 3	24/24	2	0	91.6%/91.6%
Scene 4	12/12	0	0	100%/100%
Total	99/89	7	10	82.8%/92.1%

Table 2: Detection of objects and identification of unknown views using SIFT with vo- cabulary trees.

The SIFT based detection performed well (Table 2), and geometric classifi- cation achieved just below 80% success rate (lower than in the experiments de- scribed earlier due to partial occlusions). The final database that was built con- tained some mistakes, as mentioned in the Motivation section, which required a few manual corrections, but all in all constituted less effort than manually adding all descriptors of all new object to the database using a rotating table for example.

7.3 Improved Detection through Incremental Learning

In the case that an object hypothesis is detected in the same position in the map's coordinate frame in subsequent scans we assume it is the same object as the one identified previously. For the localization we use an AMCL-based framework combining robot's odometry and laser readings in a priori built 2D map (Pfaff et al., 2006). Localization's absolute error margin lies at 0.02 m on average, thus we consider object hypotheses to represent the same object if they are not further away than 0.05 m. If the subsequent call of the examination methods returns no matching views for the given object hypothesis, we store the current observation as a new view of the object in the *Object Model Database*, as shown in Figure 21.

To demonstrate the capability of our system to acquire new object

models on the fly we set up the Scene 1 with one unknown object (green milk box) which in fact generated all 10 un-classified views reported in the first row of Table 2. Knowing that these don't match anything in the database, we can introduce them as new object models. The assumption we are making here is that the scene re- mains static, thus the cluster cloud and the defined ROI at the given 3D position in the world coordinate frame contain the same object.

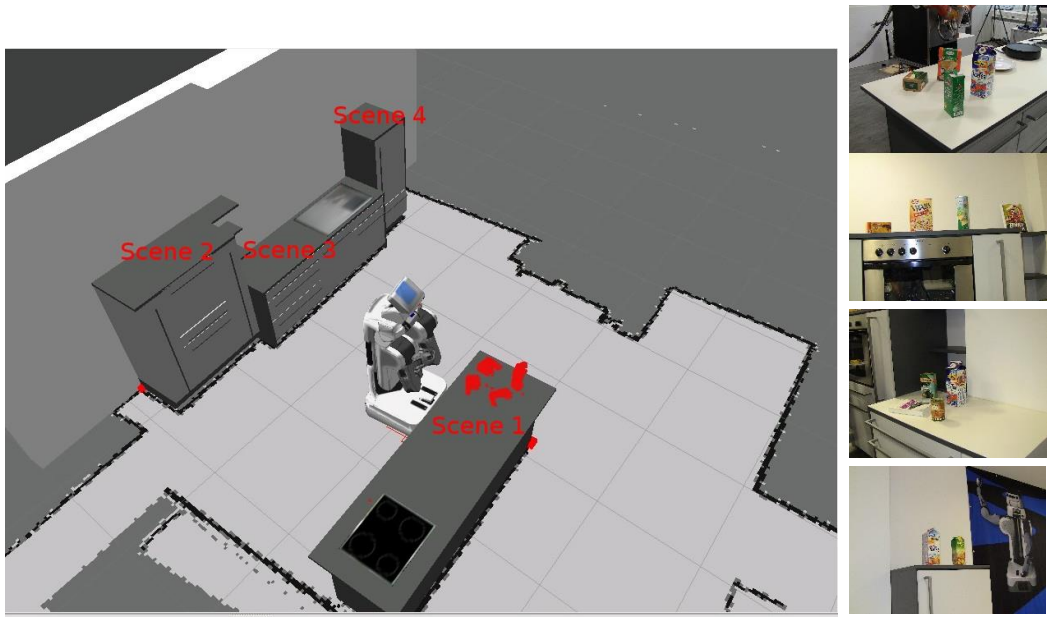


Figure 20: We performed the final evaluation test on a total number of 13 objects located at 4 different scenes in our kitchen lab (denoted with Scene 1 ... Scene 4 and depicted in top-down order in the right column). The robot was programmed to navigate to each of the scenes and capture point cloud and image from several different views by traversing along the free paths around the scenes.

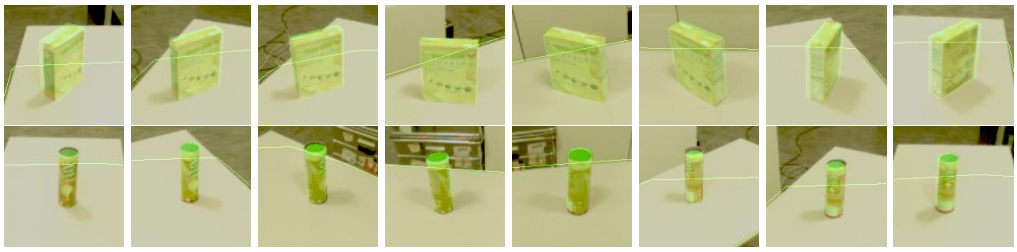


Figure 21: Different views of a cereal box (top) and chips (bottom) learned by multiple observations in the same position. The green line represents the

table's edge as detected by the laser (some parts of it were not scanned), and the green markers denote the projections of the fitted 3D model onto the image data.

After this we performed another test run on the Scene 1 with the re-trained tree

Scene 1	**#Views/#Know n**	**#Failure s**	**#Unknow n**	**Success** (with/no unknowns)
Before	52/42	0	10	**80.8%**/100%
After	52/52	2	0	**96.2%**/96.2%

Table 3: Improved detection for Scene 1 from Figure 20 before and after the vocabulary tree was re-trained and database rebuilt with the templates for green milk box. All in all, more views got the correct label.

and the updated database of SIFT descriptors and were able to reduce the number of not detected objects down to 2 as shown in Table 3.

Since most large databases (e.g. germandeli website) offer only single pictures of objects, incremental learning is an important feature for a perception system that needs to develop over time.

7.4 Discussion on Failure Cases

All five SIFT-based classification failures for the Scene 2 (Table 2, 2nd row and right column of Figure 20) are attributed to the reflective metallic surface of a green chips cylinder which made the classification using SIFT descriptors practi- cally impossible in that position. And this is where the power of our multi-modal system becomes apparent. It was in all 6 measurements of this object for this scene correctly categorized as a *tube-like cylinder*, which for the set of our 13 ob- jects, made the *Perceptual Task Execution* to decide for the instance of green chips cylinder object – the only tall cylindrical object in the *Object Model Database*. In a database with a larger number of objects containing e.g. multiple tall cylinders correct geometric categorization would still provide a significant hint at the possi- ble class instance. Additionally, the geometric model reconstruction provides the means for eventual manipulation.

8 Application Scenarios

Object reconstruction, categorization and classification routines are useful for ad- dressing many practical problems in robotics. Despite these not being the main focus of the paper we would like to briefly present the use cases for the herein



Figure 22: Grasping of the localized ob- ject by the PR2 robot. The fitted geomet- ric model was triangulated and fed into the grasp planning software to obtain grasping points.

presented system.

8.1 Grasping of Objects using Reconstructed Object Models

For grasping applications the geometric models are the most important ones, thus if an object needs to be grasped the geometric reconstruction has to be performed as denoted in Section 6.4. The triangulated meshes are then compared with stored models which have already grasp points computed, and if no good enough match was found they are uploaded into the GraspIt simulator (Goldfeder et al., 2009) where they get annotated with large sets of stable, pre-computed grasp points. Computed grasps can then be performed by the robot, as shown in Figure 22

Grasps for novel objects (e.g. Figure 15) are computed for the specific robot (PR2 and TUM-Rosie) at run-time from 3D sensor data, using heuristics based on both the overall shape of the object and its local features. We kindly refer the reader to the papers by (Hsiao et al., 2010) and (Marton et al., 2010a) to obtain more information for PR2 and TUM-Rosie calculation of grasps.

The major benefit of the reconstruction routine is that grasps are calculated by also considering the backsides of the objects as depicted in Figure 23. From the contacts, the grasp quality is statically estimated using the algorithm presented in (Ferrari and Canny, 1992).

8.2 CAD Model-based Object Detection

It is possible to perform 6 DOF position retrieval of objects on camera images using state-of-the-art 3D shape model matching technique that simulates the 2D

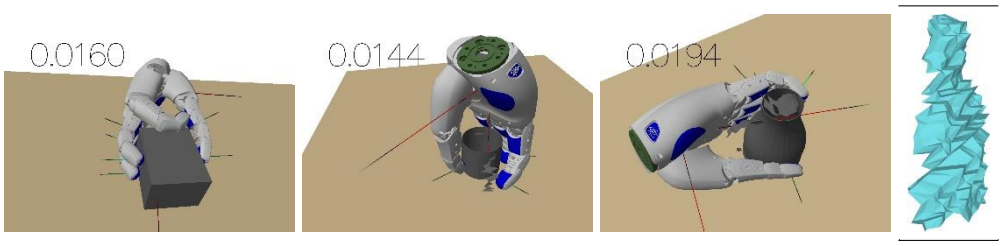


Figure 23: Grasp planning application for TUM-Rosie. Note that we used incomplete scans of the objects, like the one depicted in the last image (side view of triangulated mesh) and were still able to generate good grasps for the back sides.

appearance of the objects in a shape model generation phase (Ulrich et al., 2009). The routine gets object candidates (clusters of points) projected onto the corre- sponding 2D image from point cloud data and then uses a priori built CAD model of objects to perform the actual detection. The strength of this approach is in that the CAD models do not need to be modeled manually but can rather be generated by the robot itself in the desired complexity (see Figure 24 for the matching results and Section 6.4 for the generation of CAD models).



Figure 24: Locating the resultant geometric model in an image by TUM-Rosie robot.

These meshes can be used by other systems as well, even if the

sensors and their setup is different, as detailed in the next subsection. If, however, the model is built by the agent performing the CAD model-based detection, the known pose of the model can be used to avoid unnecessary matching of very different poses in the image.

8.3 Lifelong Dynamic World State Mapping

The multi-modal nature of our perception system lends itself to applications for entity resolution (Blodow et al., 2010), where we proposed to use Markov Logic Networks to probabilistically track trajectories of objects over time using recon- structions and refinement algorithms as described in section 6. In principle, the approach computes similarity measures between different object cluster observa- tions from compatible object descriptions (e.g. category and/or sift descriptors) and uses the full power of first-order logic with the probabilistic semantics of graphical models in order to deal with difficult conditions such as partial observ- ability of the robot's environment, incomplete object descriptions, and a dynamic environment in which we can not observe the human and his actions directly.

8.4 Platform independent Perception System

Our perception system has been validated on 2 state-of-the-art service robots, the TUM-Rosie and the PR2 thanks to the general infrastructure provided by ROS. Since the only prerequisite is to have the kinematic chains between the 2D cameras and the 3D tilting laser sensor calibrated with the precision under 5 mm (which still assures correct region-of-interest extraction for the purpose of tabletop ma- nipulation), our system can thus be ported to any arbitrary robot platform that meets the above requirement. As most of the testing for this article was done on the PR2 robot, Figures 23 and 24 verify our claims for the TUM-Rosie robot by showing the execution of the shape-based matching algorithm and grasp planning respectively.

While our two robots have similar sensor setups, cameras and lasers are widely used on different platforms as well, and many 3D sensors provide accuracy similar to or better than the Hokuyo UTM-30LX, ensuring the usability of the presented methods.

9 Conclusions and Future Work

In this paper, we presented a system for automatic and efficient object detec- tion and modeling that can improve its models with limited human supervision. The high number and variety of objects in a typical household require the use of

multiple descriptive features and the automatic construction of geometric mod- els for performing robotics tasks autonomously, as we believe that having an object database built by the robot itself in the operating environment (or adapt- ing/enriching a seed database) is more suitable to deal with the specifics of the environment efficiently than a generic approach of providing as many models of as many objects as possible.

In the future, we plan to automatically learn the objects' geometric classes and learn which geometric model type fits best to each class. Also, after multiple views of an object are obtained, the geometric models should be re-adapted into a consistent global model which should be used to locate the object in the scenes. In this way, for example the handle of a cup could be assumed to be in the back even without being detected.

Probably the most important and exciting application of the presented methods would be the extension of the system with more features and the integration with the system presented in Section 8.3. This advancement would bring household assistants one step closer to being truly autonomous when it comes to dealing with objects of daily use. The integration of multiple sources of information, prior knowledge, and combining multiple observations seem to be natural ways to achieve robust performance.

Application of speed and separation monitoring method in human- robot collaboration: industrial case study

G. Belingardi
(DIMEAS), Politecnico di Torino, Torino, Italy, giovanni.belingardi@polito.it

S. Heydaryan
(DIMEAS), Politecnico di Torino, Torino, Italy,sahar.heydaryan@polito.it

P. Chiabert
(DIGEP), Politecnico di Torino, Torino, Italy, *paolo.chiabert@polito.it*

Abstract

Application of human-robot-collaboration techniques in automotive industries has many advantages on productivity, production quality, and workers' ergonomy, however workers' safety aspects play the key role during this collaboration. In this paper, results of the ongoing research about the development of a manufacturing cell for the automotive brake disc assembly that is based on the human-robot collaboration are presented. Operational speed and worker-robot separation monitoring methodology (SSM) as one of the available method to reduce the risk of injury according to the ISO technical specification 15066 on collaborative robot in sharing space with human, has been applied. Virtual environment simulation has been used, considering

different percentages of robot maximum speed, to determine the SSM algorithm parameters for estimating the minimum protective distance between the robot and operator. Using ISO/TS 15066 and virtual environment simulation, the minimum separation distance between operator and robot has been estimated. Using human-robot collaboration along with the safety issues specified by SSM system has increased the safety of operation and reduced the operator fatigue during the assembly process.

Key words: *speed and separation monitoring, human-robot collaboration, assembly, worker safety*

1. INTRODUCTION

Nowadays, application of robots in manufacturing industries, especially in assembly lines, has dramatically increased. Many industries such as automotive, airplane and electronic, have been automated up to more than fifty percent although there are still many challenges in implementing this type of automation that are unsolved [1]. In production line, many tasks have repetitive nature such as welding, painting, and handling heavy and fragile objects. The idea of human-robot collaboration is to fill the gap between the manually and fully automated processes. Collaborative robots can be a solution to help an operator while performing the un-complex repetitive tasks. Robots can help the operator to share tasks and increase productivity. Which can lead to improvement of efficient and safe performance [2]. Human-robot collaboration in the same time and space has many hazards and risks related to operator safety. For this purpose, Occupational health and safety organizations rely on national and international standards to provide guidance for maintaining safety in the working environment. The International Organization for Standardization (ISO) Technical Specification (TS) 15066 has listed four different scenarios for increasing safety with industrial collaborative robot [3].

1. **SMS:** the first method is safety-rated, monitored stop method that requires a software or device to pause the function of robot when the worker is coming closer to the robot in order to prevent dangerous motion.
2. **HG:** The second is hand guiding method moving robot system by hand-operated device to transmit motion commands.
3. **SSM:** The third method is speed and separation monitoring which is increasing safety by specifying the minimum protective distance between a robot and an operator in the collaboration work space.
4. **PFL:** The fourth method is called power force limiting, it allows the contact between on operator and a robot, but the requirement is the control of robot momentum to avoid any injury and pain.

Recently, virtual environment plays an important role in designing various facility of production lines by providing analysis of difficult visualized situation. The implementation of virtual environment helps to reduce the risk of changing the production planning when unexpected dangerous situations are detected; also, it can help to improve cost, process time and process ergonomic safety [4].

The authors of [5] focused on the third method from ISO (TS) to investigate a set of metrics for SSM algorithm and to discover the collision avoidance path

based on consideration of safety criteria, sensor uncertainty and variable control factors in robots. The aim of the research activity described in [6] is to present a new method for designing and optimizing hybrid reconfigurable systems. The reconfigurability is addressed by a clear task decomposition between robots and operators. Virtual environment simulation has been used to consider different scenarios of reconfigurability in working station to enhance the operator awareness and reduce the risk of injuries. In the research reported in [7] authors applied virtual environment to implement manufacturing tasks for building aerospace composite parts. This paper has two goals, one of them is short-term goal which is to enhance the behavior of human while collaborating with robot inside the virtual environment. The second goal is long-term goal, they investigated how to improve acceptability of Human-Robot collaboration (H-R-C) and to improve relevant collaborative conditions by means of virtual environment. The aim of the research reported in [8] is to obtain a collaborative procedure that results to be more fluent and acceptable for humans in case of teamwork with robot. The obtained results show an improved collaboration between human and robot and a reduction of stop-and-go command during collaboration. For this purpose, firstly they simulated the collaborative environment with robot and human then tested for confirmation in real world (laboratory environment). They use the virtual environment and train the operator how to behave with robot.

There are many commercial codes available to simulate the manufacturing production line but the most popular one is Tecnomatix which is developed by Siemens company. This code is divided into different segments to design and complete tasks. The segment of Plant Simulation software is aimed to create digital models of production lines and to investigate different potential scenarios for system layouts. The duty of JACK software is to analyze the ergonomic effects on human; another segment is used for programming robots, manufacturing process and analyzing the product assembly process is called Process Simulate software. In this paper Process Simulate segment has been used to simulate the process of the brake disk assembly.
In this research, SSM system along with the virtual environment simulation have been applied to determine the minimum separation distance of human and robot in the brake disc assembly work station.

2. SIMULATION PROCEDURE

In the production line, the assembly of brake disk is performed in several steps. The objective of this study is to reproduce a workbench assembly in one cell, at a laboratory scale, where the human-robot collaboration is introduced. It should be clear that when a human operator and a robot are working together in the same work place, the risk of collision between them is high if not appropriately controlled. In any case is higher with respect to the usual organization where human operator and robot are not working together in the same work

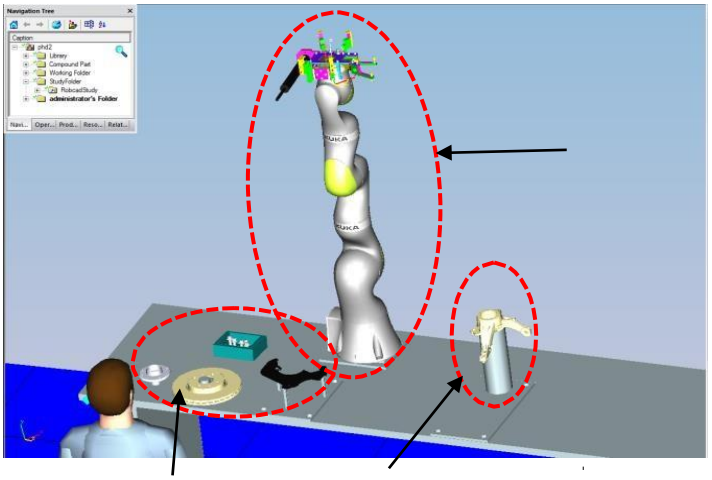
place. Therefore, these models allow us to make experience of this relatively new manufacturing environment and further to develop some optimization. The assembly area is divided in two main parts as shown in Fig 1.

(a) Picking area: the zone of the workbench in which the robot picks up the various components for the assembly.

(b) Assembly Area: where the upright and bearing are placed and fixed to allow the assembly.

The complete task on the workbench is subdivided between an operator and a robot as shown in Fig 1. The components to be assembled are placed on the workbench to supply what is needed to perform the assembly tasks and the appropriate tools for the operator. The components located on the workbench consist of the screw kits, the hub kit, the dust protection kit, the brake disc kit and the tip kits.

In order to develop a solution that is acceptable for this human-robot collaboration procedure, which has always a high risk of risk of collision, a kuka robot has been considered in this research. The Kuka lbr iiwa robot has the capability of working with an operator, simultaneously performing multiple tasks. Moreover and importantly in the collision avoidance perspective, this robot has a quick response in the case of dangerous situations. The Kuka (LBR IIWA R820 14") robot, as illustrated in Fig.1 (c), is used. This robot is characterized by a maximum range of 820 mm and a payload of 14 kg. The name LBR stands for "Leichtbauroboter" that means lightweight robot with 7 degrees of freedom. It is located in front of the workbench. This robot has precise system of sensors placed on each axis.



(a) Picking Area (b) Assembly Area (c) KUKA robot

Figure 1. Workbench area

The aim of applying H-R-C in this procedure is to help the human operator by applying a robot for improving the ergonomic by reducing the workload. This aim is relevant since the part weight is high and might cause muscular pain after repetitive tasks. Due to the fact that, during the manual assembly of brake disc, each operator should lift 5 kg brake disc and each operator works 8 hours in one shift, and during this time he

assembles around 160 brake discs, at the end of the day he results to have lift more than 800 kg. Using H-R- C will reduce considerably the burden lifted by an operator, since the heavy loads are now managed by the robot.

Tasks have been allocated to human and robot as following: The assembly job is started with taking the dust protection from picking area by robot and inserting it on the upright and bearing on the assembly area. Then the 3 M6 screws are taken from the screw kit in picking area by the operator and are tightened on the plate by the robot. In the third step, hub is taken from picking area by the operator then inserted on the dust protection. In the fourth step, the brake disc is picked from picking area and inserted on the hub in the assembly area by the robot. In the last step 2 screws are taken by the operator from screw kit then the robot tights on the brake disc in the assembly area. It is also important to mention that, safety cameras are placed in workstation to detect any kind of human and robot motions during their interactions.

These activities have been reproduced in Process Simulate. In the following lines, it will be described how to use and import the models into the Process Simulate. The parts are subdivided in two categories in virtual environment: The first category is related to those parts without any movement called static link, in other words they are stationary parts like; screw, brake disk, hub, dust protection as shown in Fig 2. The second category contains the dynamic parts, i.e. those parts that have movements such as gripper, robot,…,etc. Process Simulate works just with COJT format, and the only format that can be converted to COJT is JT format, it is necessary to produce parts in JT format in CATIA or NX

3. SPEED AND SEPARATION MONITORING METHOD

This paper focuses on the third human-robot collaborative scenario: “speed and separation monitoring” (SSM). In order to preserve a static safe separation distance between the robot and a human walking around the collaborative workspace, the SSM method offers a reasonable solution. The purpose of this method is to measure continuously the separation distance between the robot and the operator and compare with the so called authorized (worker protective) distance. Using the SSM method when the separation distance tends to reduce below the authorized distance, the robot stops any kind of motion. The robot initiates again its movement when the separation distance becomes equal or greater than the authorized distance [3,9]. SSM can offer reasonable solutions in order to preserve a safe separation distance between the robot and a human walking around the collaborative workspace. SSM can be implemented both under static and dynamic conditions. The Static SSM method considers constant the human and the robot speeds. While the dynamic SSM method can consider variable speeds. In this research, the static SSM method has been used to reduce potential risks in human-robot collaboration. The equation for calculating the minimum protective distance in human- robot collaboration in ISO/TS 15066 [3] is the extended version of the one which is defined in ISO 13855 [10] for determining the protective distance for immobile machines.

According to the ISO/TS15066 [3], the minimum protective distance, (S), at time (t0) is given by equation (1).

in the first step.

$$v_S(\quad)dt \quad C \quad Z_S \quad Z_R \quad (1)$$

$$\quad\quad t0$$

1. Snap Ring
2. Bearing
3. Up-Right
4. Dust protection
5. Screws for dust protection (3*M6)
6. Hub
7. Screw for brake disk (2*M8)
8. Break disk

□ □ □ □*t*0
□ □ □*t*0 □*tR* □



Figure 2. CAD part in NX software

Where (VR) is speed of the robot in the direction of the human, (VH) is the directed speed of human in the collaborative workspace in the direction of the moving part of the robot, (VS) is the speed of robot in the stopping path, from activation of the stop command until the robot has stopped. (TS) is the time to stop the robot motion. (TR) is the robot responding time in case of the operator presence. Where the part of body can intrude into the sensing area before it is detected, the uncertainty disturbance of boundary distance to exception of operator reach is (C). The uncertainty of robot position and operator position (sensor) are respectively (ZR) and (ZS) [3,9].

It is important to mention that, VR is the robot gripper velocity and (VH) is the manikin center velocity. The SSM stopping diagram is presented in Fig 3. The total time for stopping the motion of the robot is the summation of the sensor detection time (td), the robot reaction time (tr) and the robot stopping time (ts). The hazard area is therefore representative of the authorized stopping distance. As soon as the authorized stopping distance is calculated, a stop signal will be sent to the robot control system [11].

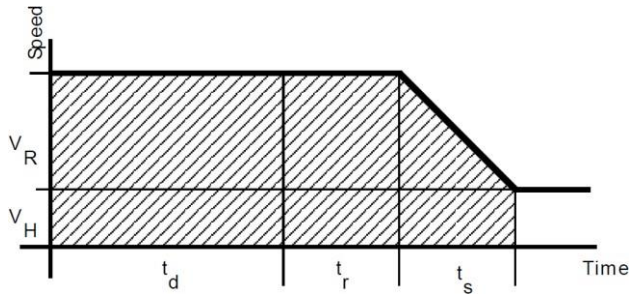


Figure 3. SSM Stopping Diagram [10].

human speed (VH) is variable between 1600 mm/s to 2000 mm/s., In order to be on the safer side, the worst- case value of human speed (VH = 2000 mm/s) is selected based on ISO/TS 13855 [10] and the maximum velocity of Kuka robot (VR= 250mm/s) is considered. The term (B) which is the robot stopping distance and the term (TS) are calculated based on equations (3) and (4) respectively mentioned in ISO 10218-1 [12] as following:

$B = V^2 / 2a$

T_S □ V_R / a

R

(3)

(4)

4. RESULTS AND DISCUSSION

Tecnomatix Process Simulate software presents simulation results in two modes as illustrated in Fig 4. and Fig 5.: (a) time-based simulation and (b) event- based simulation. Typically, time-based simulation consists of resources, products and operations whereas for event-based simulation signals should be defined. In time-based simulation, the sequence of operation and period of time process should be defined. In time-based simulation, it is essential to define the kinematic motion for non-stationary parts during the compose of manufacturing process. The main difference between

In the equations (3) and (4), (a) is the worst-case deceleration value of the robot during the stopping procedure.

While equations (2)-(4) seem to be simple, a quite complex procedure should be done to determine these values. The two parameters (TR) and (a) should be determined through the simulation procedure and the formula presented by [3]. According to [9] the stopped position of the robot (Ps,i), which is the summation of robot distance traveled while the sensors are detecting plus the distance traveled while the robot stopping begins, is calculated for different percentages (i%) of robot maximum velocity (VR) as following :

time-based and event-based is that, in event-based

P = (V

V^2

$T + \frac{R,100}{}$

V^2

+ P) - (V T)i - ($\frac{R,100}{}$)i2

(5)

simulation the logic box is constructed based on the

s,i

R,100 R $2a$

s,100 R,100 R $2a$

operations sequence. This means that, this method uses the signal-based logic to define the sequence of operations, in this paper event-based method is used.

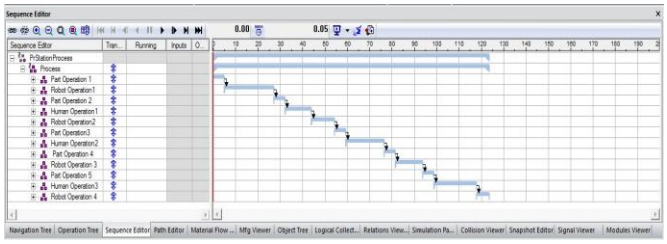


Figure 4. Gantt chart of operations in time based mode

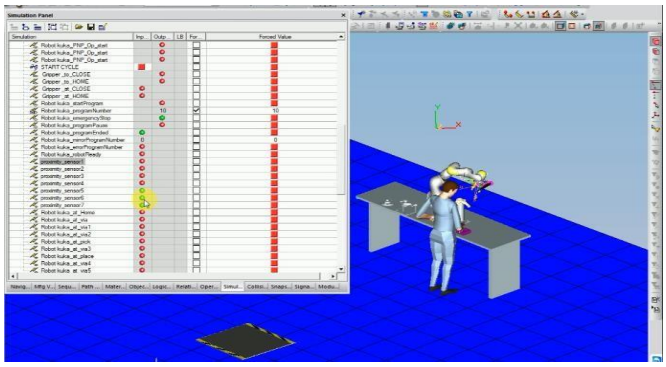


Figure 5. Virtual environment in event-based mode

Equation (2), which is a linearized form of the equation (1), has been used to calculate the minimum allowable robot-human distance under static condition [9,11].

$$S = (VHTR + VHTS) + (VRTR) + B + C \qquad (2)$$

In order to calculate the minimum separation distance, all the necessary factors should be determined. The

Having determined the stopping position of robot regarding to different percentages of robot maximum speed through the simulation, it is possible to implement the regression analysis in equation (5) to determine the values of the variables of (TR) and (a). Then the robot stopping distance and stopping time (TS) can be easily calculated based on equation (3) and (4). However, to calculate (C), which is the summation of robot and human position uncertainty and intrusion distance safety margin for separation, is a quite challenging process. One way to calculate (C) based on ISO 13855 [10] is throughout consideration of the worst- case scenario, which is implemented in this research following what reported in [3,9]. According to the above- mentioned description, the parameters of equation (2) are determined as in Table 1. The minimum separation distance is calculated as below:

Table 1. Parameters of minimum separation distance equation

VH = 2 m/s
VR = 0.25 m/s
TR= 0.41 s
TS = 0.5 s
B = 0.0625 m

C = 0.9 m

S ≈ ((2*0.41) + (0.25*0.5)) + (0.25*0.41) + (0.0625) + (0.9) = 2m

The schematic of human-robot interaction in the assembly cell is shown in the Fig 7.

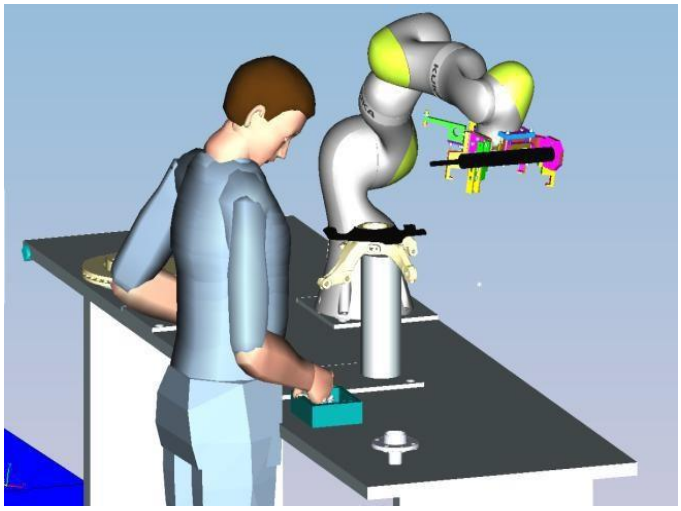


Figure 7. Human-Robot collaboration in assembly cell

Sensors in the working area calculate the distance (D) between the robot and any moving or movable object all the time as soon as (D) becomes less than the minimum separation distance (S). Sensors will issue a stop signal to the robot and the robot stops any kind of motion as shown in Figs 8., and 9.; again, when (D) becomes larger than the minimum authorized distance the robot begins to work.

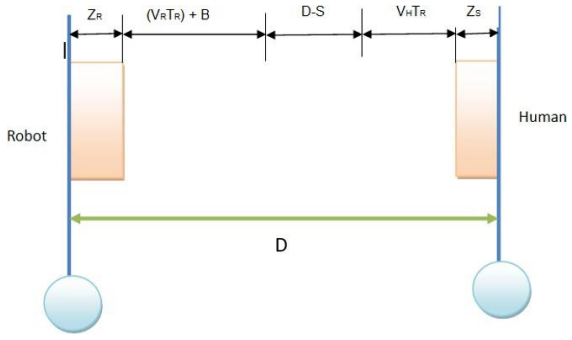


Figure 8. The SSM issues a stop when (D – S) < 0 [10]

Figure 9. Human-Robot collaboration with SSM method

5. CONCLUSIONS

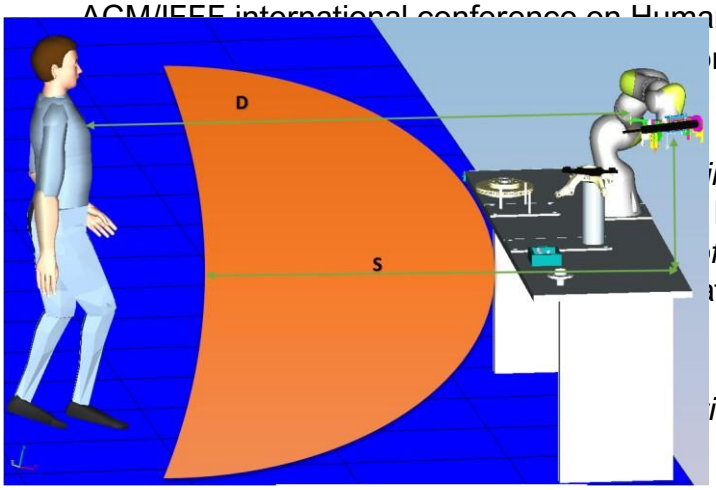
The application of the Human- Robot Collaboration is becoming more and more interesting in the manufacturing industry and in particular in the automotive plants, for the advantages that can come

in terms of safety and ergonomic issues for the workers and in terms of quality for the products. In this paper, SSM system, as one of the available evaluation procedure in ISO/TS 15066, has been applied to increases safety in assembly cell using human-robot collaboration.
One of the relevant point for worker safety, when he is working in the same workplace of the robot, is to assure in any circumstances that the minimum separation distance is maintained.
According to the ISO/TS 15066, a linearized formula has been used to determine the minimum separation distance between the robot and human in assembly work station. In order to determine the parameters, necessary to estimate the robot stopping distance (B), robot stopping time (TS) and human-robot position uncertainties (C), a virtual environment tool has been used to simulate the assembly process through different percentages of robot maximum speed with respect to different stopping position of robot. The minimum allowable separation distance between human and robot has been calculated as 2 meters. As soon as the distance between the robot and human becomes less than the authorized separation distance, the robot stops working and when the distance returns to be larger than that the authorized one, again the robot begins to work. However always there is a need to use the most advanced sensors in the working area where human- robot cooperation takes place in order to reduce any risk of injuries.

6. REFERENCES

[1] Bartscher, S. (2012), *"Innovation Product Division".* BMW Group, personal communication.

[2] Unhelkar, V.V., Siu, H.C., and Shah, J.A. (2014), *"Comparative performance of human and mobile robotic assistants in collaborative fetch-and-deliver tasks",* Proceedings of the ACM/IEEE international conference on Human- robot interaction, pp. 82-89.



n, ISO/TS 15066 (2016), Robots and Robotic

A. (2009), *"Innovative assembly process for irtual environment",* International Journal on No. 2, pp. 93-101.

f Speed and Separation Monitoring in Shared ation Science and Engineering, Vol. 10, No. 2,

Pini, F., and Vergnano, A. (2012), *"Hybrid ion*

through virtual prototyping and digital manufacturing tools", International Journal on Interactive Design and Manufacturing, Vol. 6 No. 1, pp. 17-27.

[7] Matsas, E. and Vosniakos, G.C. (2017),*" Design of a virtual reality training system for human–robot collaboration in manufacturing tasks"*, International Journal on Interactive Design and Manufacturing, Vol. 11, No. 2, pp. 139-153.

[8] Hoffman, G. and Breazeal, C. (2007), *"Effects of anticipatory action on human robot teamwork efficiency, fluency, and perception of team",*

Proceedings of the ACM/IEEE international conference on Human-robot interaction, pp. 1–8.

[9] Marvel, J.A. and Norcross, R. (2017), *"Implementing speed and separation monitoring in collaborative robot workcells",* Robotics and Computer Integrated Manufacturing, Vol. 44, pp. 144–155.

[10] International Organization for Standardization, ISO13855 (2010), Safety of Machinery– Positioning of Safe guards with Respect to the Approach Speeds of Parts of the Human Body.

[11] Szabo, S.S., Shackleford, W.P., Norcross, R. and

Marvel, J.A. (2012), *"A Testbed for Evaluation of Speed and Separation Monitoring in a Human Robot Collaborative Environment"*. U.S Department of Commerce National Institute of Standards and Technology.

[12] Intelligent Systems Division Gaithersburg, MD 20899-8230, International Organization for Standardization, ISO10218-1 (2011), Robots and Robotic Devices – Safety Requirements – Part 1: Robots.

6. Related Work & Discussion

6 Planning and Navigation

6.1 Introduction

This book has focused on the elements of a mobile robot that are critical to robust mobility: the kinematics of locomotion; sensors for determining the robot's environmental context; and techniques for localizing with respect to its map. We now turn our attention to the robot's cognitive level. Cognition generally represents the purposeful decision-making and execution that a system utilizes to achieve its highest-order goals.

In the case of a mobile robot, the specific aspect of cognition directly linked to robust mobility is *navigation competence*. Given partial knowledge about its environment and a goal position or series of positions, navigation encompasses the ability of the robot to act based on its knowledge and sensor values so as to reach its goal positions as efficiently and as reliably as possible. The focus of this chapter is how the tools of the previous chapters can be combined to solve this navigation problem.

Within the mobile robotics research community, a great many approaches have been proposed for solving the navigation problem. As we sample from this research background it will become clear that in fact there are strong similarities between all of these approaches even though they appear, on the surface, quite disparate. The key difference between vari- ous navigation architectures is the manner in which they decompose the problem into smaller subunits. In section 6.3 below, we describe the most popular navigation architec- tures, contrasting their relative strengths and weaknesses.

First, however, in section 6.2 we discuss two key additional competences required for mobile robot navigation. Given a map and a goal location, *path planning* involves identi- fying a trajectory that will cause the robot to reach the goal location when executed. Path planning is a strategic problem-solving competence, as the robot must decide what to do over the long term to achieve its goals.

The second competence is equally important but occupies the opposite, tactical extreme. Given real-time sensor readings, *obstacle avoidance* means modulating the trajectory of the robot in order to avoid collisions. A great variety of approaches have demonstrated compe- tent obstacle avoidance, and we survey a number of these approaches as well.

6.2 Competences for Navigation: Planning and Reacting

In the artificial intelligence community planning and reacting are often viewed as contrary approaches or even opposites. In fact, when applied to physical systems such as mobile robots, planning and reacting have a strong complementarity, each being critical to the other's success. The navigation challenge for a robot involves executing a course of action (or plan) to reach its goal position. During execution, the robot must react to unforeseen events (e.g., obstacles) in such a way as to still reach the goal. Without reacting, the plan- ning effort will not pay off because the robot will never physically reach its goal. Without planning, the reacting effort cannot guide the overall robot behavior to reach a distant goal
– again, the robot will never reach its goal.

An information-theoretic formulation of the navigation problem will make this comple- mentarity clear. Suppose that a robot M at time i has a map M_i and an initial belief state b_i . The robot's goal is to reach a position p while satisfying some temporal constraints:
$loc_G(R) = p ; g \le n$. Thus the robot must be at location p at or before timestep n.

Although the goal of the robot is distinctly physical, the robot can only really sense its belief state, not its physical location, and therefore we map the goal of reaching location p to reaching a belief state b_G , corresponding to the belief that $loc_G(R) = p$. With this for- mulation, a plan q is nothing more than one or more trajectories from b_i to b_G . In other words, plan q will cause the robot's belief state to transition from b_i to b_G if the plan is executed from a world state consistent with both b_i and M_i .

Of course the problem is that the latter condition may not be met. It is entirely possible that the robot's position is not quite

consistent with b_i , and it is even likelier that M_i is either incomplete or incorrect. Furthermore, the real-world environment is dynamic. Even
if M_i is correct as a single snapshot in time, the planner's model regarding how M changes over time is usually imperfect.

In order to reach its goal nonetheless, the robot must incorporate new information gained during plan execution. As time marches forward, the environment changes and the robot's sensors gather new information. This is precisely where reacting becomes relevant. In the best of cases, reacting will modulate robot behavior locally in order to correct the planned- upon trajectory so that the robot still reaches the goal. At times, unanticipated new infor- mation will require changes to the robot's strategic plans, and so ideally the planner also incorporates new information as that new information is received.

Taken to the limit, the planner would incorporate every new piece of information in real time, instantly producing a new plan that in fact reacts to the new information appropri- ately. This theoretical extreme, at which point the concept of planning and the concept of reacting merge, is called *integrated planning and execution* and is discussed in section 6.3.4.3.

A useful concept throughout this discussion of robot architecture involves whether par- ticular design decisions sacrifice the system's ability to achieve a desired goal whenever a solution exists. This concept is termed *completeness*. More formally, the robot system is *complete* if and only if, for all possible problems (i.e., initial belief states, maps, and goals), when there exists a trajectory to the goal belief state, the system will achieve the goal belief state (see [27] for further details). Thus when a system is incomplete, then there is at least one example problem for which, although there is a solution, the system fails to generate a solution. As you may expect, achieving completeness is an ambitious goal. Often, com- pleteness is sacrificed for computational complexity at the level of representation or rea- soning. Analytically, it is important to understand how completeness is compromised by each particular system.

In the following sections, we describe key aspects of planning and reacting as they apply to a mobile robot path planning and obstacle avoidance and describe how representational decisions impact the potential completeness of the overall system. For greater detail, refer to [21, 30, chapter 25].

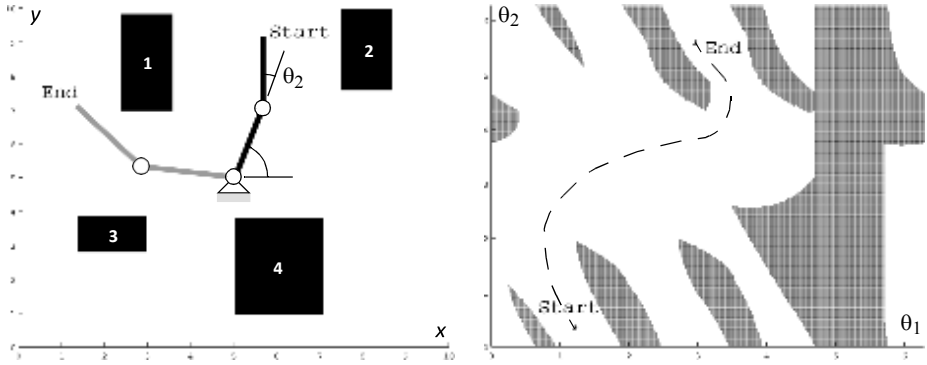
6.2.1 Path planning

Even before the advent of affordable mobile robots, the field of path-planning was heavily studied because of its applications in the area of industrial manipulator robotics. Interest- ingly, the path planning problem for a manipulator with, for instance, six degrees of free- dom is far more complex than that of a differential-drive robot operating in a flat environment. Therefore, although we can take inspiration from the techniques invented for manipulation, the path-planning algorithms used by

mobile robots tend to be simpler approximations owing to the greatly reduced degrees of freedom. Furthermore, industrial robots often operate at the fastest possible speed because of the economic impact of high throughput on a factory line. So, the dynamics and not just the kinematics of their motions are significant, further complicating path planning and execution. In contrast, a number of mobile robots operate at such low speeds that dynamics are rarely considered during path planning, further simplifying the mobile robot instantiation of the problem.

Configuration space. Path planning for manipulator robots and, indeed, even for most mobile robots, is formally done in a representation called *configuration space*. Suppose that a robot arm (e.g., SCARA robot) has k degrees of freedom. Every state or configuration of the robot can be described with k real values: $q_1, \ldots, q_k$. The k-values can be regarded as a point p in a k-dimensional space called the configuration space C of the robot. This description is convenient because it allows us to describe the complex 3D shape of the robot with a single k-dimensional point.

Now consider the robot arm moving in an environment where the workspace (i.e., its physical space) contains known obstacles. The goal of path planning is to find a path in the



a) *b)*

Figure 6.1

Physical space (a) and configuration space (b): (a) A two-link planar robot arm has to move from the configuration *start* to *end*. The motion is thereby constraint by the obstacles 1 to 4. (b) The corre- sponding configuration space shows the free space in joint coordinates (angle θ_1 and θ_2) and a path that achieves the goal.

physical space from the initial position of the arm to the goal position, avoiding all colli- sions with the obstacles. This is a difficult problem to visualize and solve in the physical space, particularly as *k* grows large. But in configuration space the problem is straightfor-

ward. If we define the *configuration space obstacle O* as the subspace of *C* where the

robot arm bumps into something, we can compute the free space $F = C - O$ in which the robot can move safely.

Figure 6.1 shows a picture of the physical space and configuration space for a planar robot arm with two links. The robot's goal is to move its end effector from position *start* to *end*. The configuration space depicted is 2D because each of two joints can have any posi- tion from 0 to 2 . It is easy to see that the solution in C-space is a line from *start* to *end* that remains always within the free space of the robot arm.

For mobile robots operating on flat ground, we generally represent robot position with three variables (x, y, θ), as in chapter 3. But, as we have seen, most robots are nonholo- nomic, using differential-drive systems or Ackerman steered systems. For such robots, the nonholonomic constraints limit the robot's velocity $(\dot{x}, \dot{y}, \dot{\theta})$ in each configuration (x, y, θ). For details regarding the construction of the appropriate *free space* to solve such path-planning problems, see [21, p. 405].

In mobile robotics, the most common approach is to assume for path-planning purposes that the robot is in fact holonomic, simplifying the process tremendously. This is especially common for differential-drive robots because they can rotate in place and so a holonomic path can be easily mimicked if the rotational position of the robot is not critical.

Furthermore, mobile roboticists will often plan under the further assumption that the robot is simply a *point*. Thus we can further reduce the configuration space for mobile robot path planning to a 2D representation with just x - and y -axes. The result of all this simpli- fication is that the configuration space looks essentially identical to a 2D (i.e., flat) version of the physical space, with one important difference. Because we have reduced the robot to a point, we must inflate each obstacle by the size of the robot's radius to compensate. With this new, simplified configuration space in mind, we can now introduce common tech- niques for mobile robot path planning.

Path-planning overview. The robot's environment representation can range from a con- tinuous geometric description to a decomposition-based geometric map or even a topolog- ical map, as described in section 5.5. The first step of any path-planning system is to transform this possibly continuous environmental model into a discrete map suitable for the chosen path-planning algorithm. Path planners differ as to how they effect this discrete decomposition. We can identify three general strategies for decomposition:

1. Road map: identify a set of routes within the free space.
2. Cell decomposition: discriminate between free and occupied cells.
3. Potential field: impose a mathematical function over the space.

The following sections present common instantiations of the road map and cell decom- position path-planning techniques, noting in each case whether completeness is sacrificed by the particular representation.

6.2.1.1 Road map path planning

Road map approaches capture the connectivity of the robot's free space in a network of 1D curves or lines, called *road maps*. Once a road map is constructed, it is used as a network of road (path) segments for robot motion planning. Path planning is thus reduced to con- necting the initial and goal positions of the robot to the road network, then searching for a series of roads from the initial robot position to its goal position.

The road map is a decomposition of the robot's configuration space based specifically on obstacle geometry. The challenge is to construct a set of roads that together enable the robot to go anywhere in its free space, while minimizing the number of total roads. Gener- ally, completeness is preserved in such decompositions as long as the true degrees of free- dom of the robot have been captured with appropriate fidelity. We describe two road map approaches below that achieve this result with dramatically different types of roads. In the case of the *visibility graph*, roads come as close as possible to obstacles and resulting paths are minimum-length solutions. In the case of the *Voronoi diagram*, roads stay as far away as possible from obstacles.

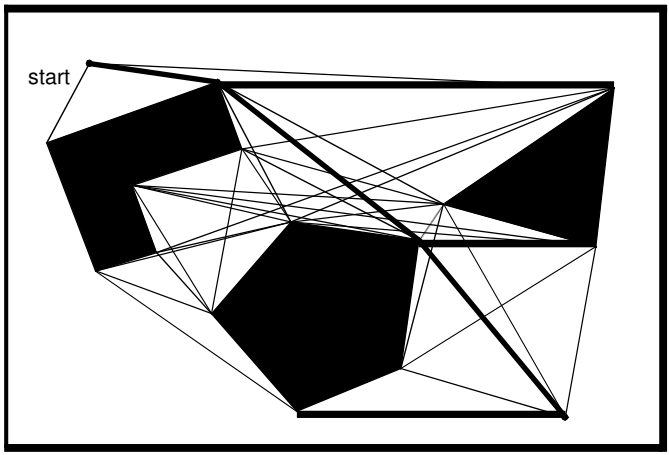


Figure 6.2

Visibility graph [21]. The nodes of the graph are the initial and goal points and the vertices of the con- figuration space obstacles (polygons). All nodes which are visible from each other are connected by straight-line segments, defining the road map. This means there are also edges along each polygon's sides.

Visibility graph. The visibility graph for a polygonal configuration space *C* consists of edges joining all pairs of vertices that can see each other (including both the initial and goal positions as vertices as well). The unobstructed straight lines (roads) joining those vertices are obviously the shortest distances between them. The task of the path planner is thus to find the shortest path from the initial position to the goal position along the roads defined by the visibility graph (figure 6.2).

Visibility graph path planning is moderately popular in mobile robotics, partly because implementation is quite simple.

Particularly when the environmental representation describes objects in the environment as polygons in either continuous or discrete space, the visibility graph search can employ the obstacle polygon descriptions readily.

There are, however, two important caveats when employing visibility graph search. First, the size of the representation and the number of edges and nodes increase with the number of obstacle polygons. Therefore the method is extremely fast and efficient in sparse environments, but can be slow and inefficient compared to other techniques when used in densely populated environments.

The second caveat is a much more serious potential flaw: the solution paths found by visibility graph planning tend to take the robot as close as possible to obstacles on the way

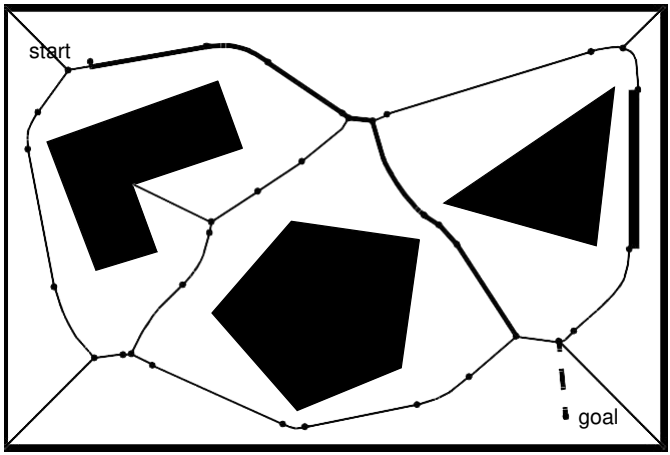


Figure 6.3

Voronoi diagram [21]. The Voronoi diagram consists of the lines constructed from all points that are

equidistant from two or more obstacles. The initial q_{init} and goal q_{goal} configurations are mapped into the Voronoi diagram to q'_{init} and q'_{goal} , each by drawing the line along which its distance to the boundary of the obstacles increases the fastest. The direction of movement on the Voronoi dia-

gram is also selected so that the distance to the boundaries increases fastest. The points on the Voronoi diagram represent transitions from line segments (minimum distance between two lines) to parabolic segments (minimum distance between a line and a point).

to the goal. More formally, we can prove that visibility graph planning is *optimal* in terms of the length of the solution path. This powerful result also means that all sense of safety, in terms of staying a reasonable distance from obstacles, is sacrificed for

this optimality. The common solution is to grow obstacles by significantly more than the robot's radius, or, alternatively, to modify the solution path after path planning to distance the path from obstacles when possible. Of course such actions sacrifice the optimal-length results of vis- ibility graph path planning.

Voronoi diagram. Contrasting with the visibility graph approach, a Voronoi diagram is a complete road map method that tends to maximize the distance between the robot and obstacles in the map. For each point in the free space, compute its distance to the nearest obstacle. Plot that distance in figure 6.3 as a height coming out of the page. The height increases as you move away from an obstacle. At points that are equidistant from two or more obstacles, such a distance plot has sharp ridges. The Voronoi diagram consists of the edges formed by these sharp ridge points. When the configuration space obstacles are poly- gons, the Voronoi diagram consists of straight and parabolic segments. Algorithms that

find paths on the Voronoi road map are complete just like visibility graph methods, because the existence of a path in the free space implies the existence of one on the Voronoi diagram as well (i.e., both methods guarantee completeness). However, the path in the Voronoi dia- gram is usually far from optimal in the sense of total path length.

The Voronoi diagram has an important weakness in the case of limited range localiza- tion sensors. Since this path-planning algorithm maximizes the distance between the robot and objects in the environment, any short-range sensor on the robot will be in danger of fail- ing to sense its surroundings. If such short-range sensors are used for localization, then the chosen path will be quite poor from a localization point of view. On the other hand, the vis- ibility graph method can be designed to keep the robot as close as desired to objects in the map.

There is, however, an important subtle advantage that the Voronoi diagram method has over most other obstacle avoidance techniques: *executability*. Given a particular planned path via Voronoi diagram planning, a robot with range sensors, such as a laser rangefinder or ultrasonics, can follow a Voronoi edge in the physical world using simple control rules that match those used to create the Voronoi diagram: the robot maximizes the readings of local minima in its sensor values. This control system will naturally keep the robot on Voronoi edges, so that Voronoi-based motion can mitigate encoder inaccuracy. This inter- esting physical property of the Voronoi diagram has been used to conduct automatic map- ping of an environment by finding and moving on unknown Voronoi edges, then constructing a consistent Voronoi map of the environment [59].

6.2.1.2 Cell decomposition path planning

The idea behind cell decomposition is to discriminate between geometric areas, or cells, that are free and areas that are occupied by objects. The basic cell decomposition path-planning algorithm can be summarized as follows [30]:

- Divide *F* into simple, connected regions called “cells”.
- Determine which opens cells are adjacent and construct a “connectivity graph”.
- Find the cells in which the initial and goal configurations lie and search for a path in the connectivity graph to join the initial and goal cell.
- From the sequence of cells found with an appropriate searching algorithm, compute a path within each cell, for example, passing through the midpoints of the cell boundaries or by a sequence of wall-following motions and movements along straight lines.

An important aspect of cell decomposition methods is the placement of the boundaries between cells. If the boundaries are placed as a function of the structure of the environment, such that the decomposition is lossless, then the method is termed *exact cell decomposition*. If the decomposition results in an approximation of the actual map, the system is termed

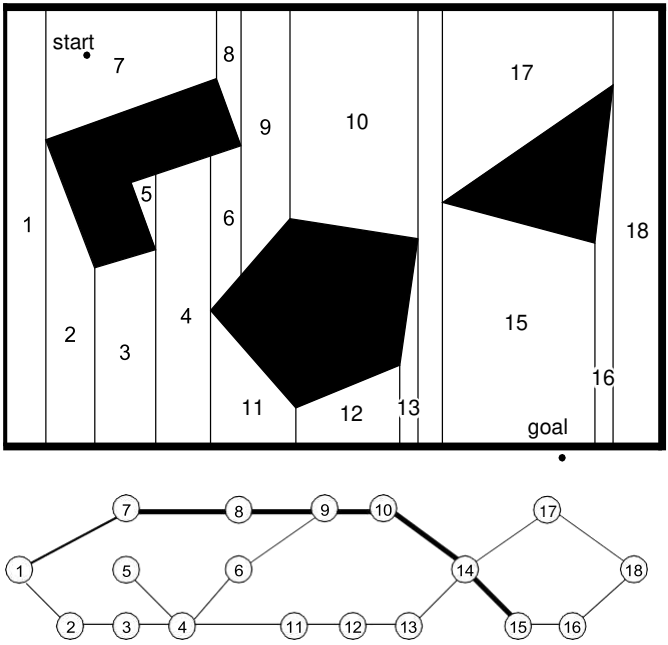


Figure 6.4

Example of exact cell decomposition.

approximate cell decomposition. In section 5.5.2 we described these decomposition strate- gies as they apply to the design of map representation for localization. Here, we briefly summarize these two cell decomposition techniques once again, providing greater detail about their advantages and disadvantages relative to path planning.

Exact cell decomposition. Figure 6.4 depicts exact cell decomposition, whereby the boundary of cells is based on geometric criticality. The resulting cells are each either com-

pletely free or completely occupied, and therefore path planning in the network is complete, like the road map based methods above. The basic abstraction behind such a decomposition is that the particular position of the robot within each cell of free space does not matter; what matters is rather the robot's ability to traverse from each free cell to adjacent free cells. The key disadvantage of exact cell decomposition is that the number of cells and, there- fore, overall path planning computational efficiency depends upon the density and com- plexity of objects in the environment, just as with road mapbased systems. The key advantage is a result of this same correlation. In environments that are extremely sparse, the number of cells will be small, even if the geometric size of the environment is very

large. Thus the representation will be efficient in the case of large, sparse environments. Practically speaking, due to complexities in implementation, the exact cell decomposition technique is used relatively rarely in mobile robot applications, although it remains a solid choice when a lossless representation is highly desirable, for instance to preserve complete- ness fully.

Approximate cell decomposition. By contrast, approximate cell decomposition is one of the most popular techniques for mobile robot path planning. This is partly due to the pop- ularity of grid-based environmental representations. These grid-based representations are themselves fixed grid-size decompositions and so they are identical to an approximate cell decomposition of the environment.

The most popular form of this, shown in figure 5.15 of chapter 5, is the fixed-size cell decomposition. The cell size is not dependent on the particular objects in an environment at all, and so narrow passageways can be lost due to the inexact nature of the tessellation. Practically speaking, this is rarely a problem owing to the very small cell size used (e.g., 5 cm on each side). The great benefit of fixed size cell decomposition is the low computa- tional complexity of path planning.

For example, NF1, often called *grassfire*, is an efficient and simple-to-implement tech- nique for finding routes in such fixed-size cell arrays [96]. The algorithm simply employs wavefront expansion from the goal position outward, marking for each cell its distance to the goal cell [79]. This process continues until the cell corresponding to the initial robot position is reached. At this point, the path planner can estimate the robot's distance to the goal position as well as recover a specific solution trajectory by

simply linking together cells that are adjacent and always closer to the goal.

Given that the entire array can be in memory, each cell is only visited once when looking for the shortest discrete path from the initial position to the goal position. So, the search is linear in the number of cells only. Thus complexity does not depend on the sparseness and density of the environment, nor on the complexity of the objects' shapes in the environ- ment. Formally, this grassfire transform is simply breadth-first search implemented in the constrained space of an adjacency array. For more information on breadth-first search and other graph search techniques, refer to [30].

The fundamental cost of the fixed decomposition approach is memory. For a large envi- ronment, even when sparse, this grid must be represented in its entirety. Practically, due to the falling cost of computer memory, this disadvantage has been mitigated in recent years. The Cye robot is an example of a commercially available robot that performs all its path planning on a 2D 2 cm fixed-cell decomposition of the environment using a sophisticated grassfire algorithm that avoids known obstacles and prefers known routes [42].

Figure 5.16 of chapter 5 illustrates a variable-size approximate cell decomposition method. The free space is externally bounded by a rectangle and internally bounded by

three polygons. The rectangle is recursively decomposed into smaller rectangles. Each decomposition generates four identical new rectangles. At each level of resolution only the cells whose interiors lie entirely in the free space are used to construct the connectivity graph. Path planning in such adaptive representations can proceed in a hierarchical fashion. Starting with a coarse resolution, the resolution is reduced until either the path planner iden- tifies a solution or a limit resolution is attained (e.g, k • size of robot). In contrast to the exact cell decomposition method, the approximate approach can sacrifice completeness, but it is mathematically less involving and thus easier to implement. In contrast to the fixed- size cell decomposition, variable-size cell decomposition will adapt to the complexity of the environment, and therefore sparse environments will contain appropriately fewer cells, consuming dramatically less memory.

6.2.1.3 Potential field path planning

Potential field path planning creates a field, or gradient, across the robot's map that directs the robot to the goal position from multiple prior positions (see [21]). This approach was originally invented for robot manipulator path planning and is used often and under many variants in the mobile robotics community. The potential field method treats the robot as a point under the influence of an artificial potential field $U(q)$. The robot moves by follow- ing the field, just as a ball would roll downhill. The goal (a minimum in this space) acts as an attractive force on the robot and the obstacles act as peaks, or repulsive forces. The superposition of all forces is applied to the robot, which, in most cases, is assumed to be a point in the configuration space (see figure 6.5). Such an artificial potential field smoothly guides the

robot toward the goal while simultaneously avoiding known obstacles.

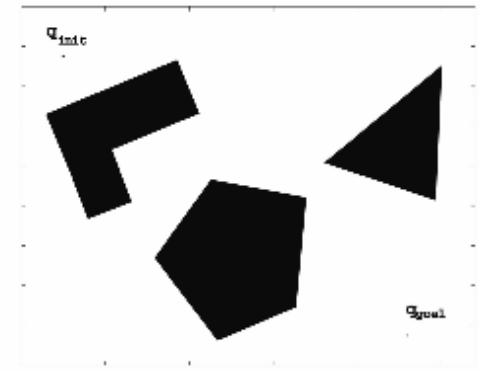
It is important to note, though, that this is more than just path planning. The resulting field is also a control law for the robot. Assuming the robot can localize its position with respect to the map and the potential field, it can always determine its next required action based on the field.

The basic idea behind all potential field approaches is that the robot is attracted toward the goal, while being repulsed by the obstacles that are known in advance. If new obstacles appear during robot motion, one could update the potential field in order to integrate this new information. In the simplest case, we assume that the robot is a point, thus the robot's orientation θ is neglected and the resulting potential field is only 2D (x, y). If we assume a differentiable potential field function $U(q)$, we can find the related artificial force $F(q)$ acting at the position q $= (x, y)$.

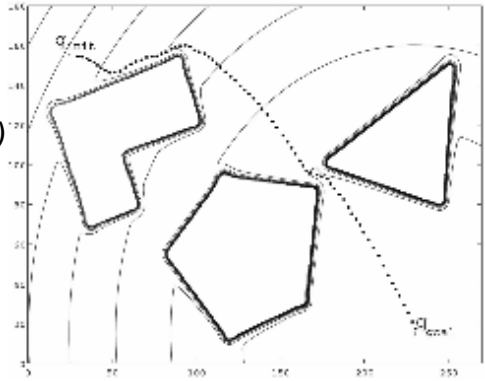
$$F(q) = -\nabla U(q) \tag{6.1}$$

where $\nabla U(q)$ denotes the gradient vector of U at position q.

a)



b)



c)

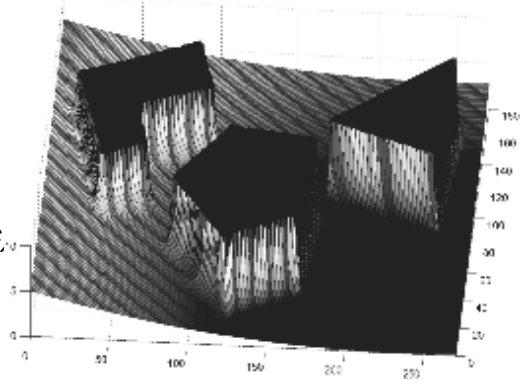


Figure 6.5

Typical potential field generated by the attracting goal and two obstacles (see [21]). (a) Configuration of the obstacles, start (top left) and goal (bottom right). (b) Equipotential plot and path generated by the field. (c) Resulting potential field generated by the goal attractor and obstacles.

$$\nabla U = \begin{bmatrix} \frac{\partial U}{\partial x} \\ \frac{\partial U}{\partial y} \end{bmatrix} \tag{6.2}$$

The potential field acting on the robot is then computed as the sum of the attractive field of the goal and the repulsive fields of the obstacles:

$$U(q) = U_{att}(q) + U_{rep}(q) \tag{6.3}$$

Similarly, the forces can also be separated in a attracting and repulsing part:

$$\begin{aligned} F(q) &= F_{att}(q) - F_{rep}(q) \\ &= -\nabla U_{att}(q) - \nabla U_{rep}(q) \end{aligned} \tag{6.4}$$

Attractive potential. An attractive potential can, for example, be defined as a parabolic function.

$$U_{att}(q) = \frac{1}{2} k_{att} \|q - q_{goal}\|^2 \tag{6.5}$$

where k_{att} is a positive scaling factor and $\rho_{goal}(q)$ denotes the Euclidean distance $\|q - q_{goal}\|$. This attractive potential is differentiable, leading to the attractive force F_{att}

$$F_{att}(q) = -\nabla U_{att}(q) \tag{6.6}$$

$$= -k_{att} \cdot \nabla \rho_{goal}(q) \tag{6.7}$$

$$\nabla \rho_{goal}(q) \tag{6.8}$$

$$= -k_{att} \cdot (q - q_{goal})$$

that converges linearly toward 0 as the robot reaches the goal.

Repulsive potential. The idea behind the repulsive potential is to generate a force away from all known obstacles. This repulsive potential should be very strong when the robot is close to the object, but should not influence its movement when the robot is far from the object. One example of such a repulsive field is

$$U_{rep}(q) = \begin{cases} \frac{1}{2} k_{rep} \left(\frac{1}{\rho(q)} - \frac{1}{\rho_0} \right)^2 & \text{if } \rho(q) \leq \rho_0 \\ 0 & \text{if } \rho(q) > \rho_0 \end{cases} \quad (6.9)$$

where k_{rep} is again a scaling factor, $\rho(q)$ is the minimal distance from q to the object and ρ_0 the distance of influence of the object. The repulsive potential function U_{rep} is positive or zero and tends to infinity as q gets closer to the object.

If the object boundary is convex and piecewise differentiable, $\rho(q)$ is differentiable everywhere in the free configuration space. This leads to the repulsive force F_{rep} :

$$F_{rep}(q) = -\nabla U_{rep}(q) \quad (6.10)$$

$$= \begin{cases} k_{rep} \left(\frac{1}{\rho(q)} - \frac{1}{\rho_0} \right) \frac{q - q_{obstacle}}{\rho^2(q) \rho(q)} & \text{if } \rho(q) \leq \rho_0 \\ 0 & \text{if } \rho(q) > \rho_0 \end{cases}$$

The resulting force $F(q) = F_{att}(q) + F_{rep}(q)$ acting on a point robot exposed to the attractive and repulsive forces moves the robot away from the obstacles and toward the goal (see figure 6.5). Under ideal conditions, by setting the robot's velocity vector proportional to the field force vector, the robot can be smoothly guided toward the goal, similar to a ball rolling around obstacles and down a hill.

However, there are some limitations with this approach. One is local minima that appear dependent on the obstacle shape and size. Another problem might appear if the objects are concave. This might lead to a situation for which several minimal distances $d(q)$ exist, resulting in oscillation between the two closest points to the object, which could obviously sacrifice completeness. For more detailed analyses of potential field characteristics, refer to [21].

The extended potential field method. Khatib and Chatila proposed the extended poten- tial field approach [84]. Like all potential field methods this approach makes use of attrac- tive and repulsive forces that originate from an artificial potential field. However, two additions to the basic potential field are made: the *rotation potential field* and the *task potential field*.

The rotation potential field assumes that the repulsive force is a function of the distance from the obstacle and the orientation of the robot relative to the obstacle. This is done using

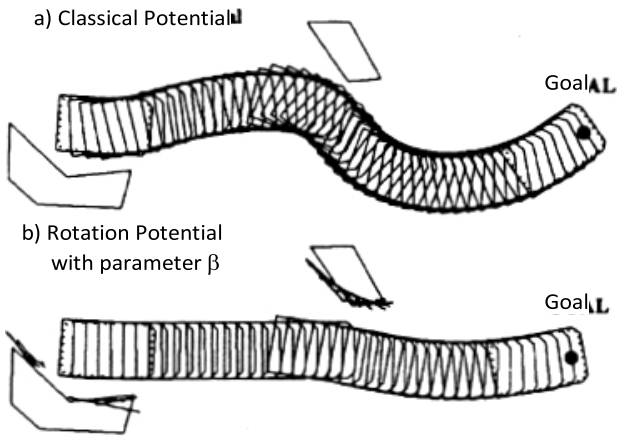


Figure 6.6

Comparison between a classical potential field and an extended potential field. Image courtesy of Raja Chatila [84].

a gain factor which reduces the repulsive force when an obstacle is parallel to the robot's direction of travel, since such an object does not pose an immediate threat to the robot's trajectory. The result is enhanced wall following, which was problematic for earlier imple- mentations of potential fields methods.

The task potential field considers the present robot velocity and from that it filters out those obstacles that should not affect the near-term potential based on robot velocity. Again a scaling is made, this time of all obstacle potentials when there are no obstacles in a sector named Z in front of the robot. The sector Z is defined as the space which the robot will sweep during its next movement. The result can be smoother trajectories through space. An example comparing a classical potential field and an extended potential field is depicted in figure

6.6.

A great many variations and improvements of the potential field methods have been pro- posed and implemented by mobile roboticists [67, 111]. In most cases, these variations aim to improve the behavior of potential fields in local minima while also lowering the chances of oscillations and instability when a robot must move through a narrow space such as a doorway.

Potential fields are extremely easy to implement, much like the grassfire algorithm described in section 6.2.1.2. Thus it has become a common tool in mobile robot applica- tions in spite of its theoretical limitations.

This completes our brief summary of the path-planning techniques that are most popular in mobile robotics. Of course, as the complexity of a robot increases (e.g., large degree of

freedom nonholonomics) and, particularly, as environment dynamics becomes more signif- icant, then the path-planning techniques described above become inadequate for grappling with the full scope of the problem. However, for robots moving in largely flat terrain, the mobility decision-making techniques roboticists use often fall under one of the above cat- egories.

But a path planner can only take into consideration the environment obstacles that are known to the robot in *advance*. During path execution the robot's actual sensor values may disagree with expected values due to map inaccuracy or a dynamic environment. Therefore, it is critical that the robot modify its path in real time based on actual sensor values. This is the competence of *obstacle avoidance* which we discuss below.

6.2.2 Obstacle avoidance

Local obstacle avoidance focuses on changing the robot's trajectory as informed by its sen- sors during robot motion. The resulting robot motion is both a function of the robot's cur- rent or recent sensor readings *and* its goal position and relative location to the goal position. The obstacle avoidance algorithms presented below depend to varying degrees on the exist- ence of a global map and on the robot's precise knowledge of its location relative to the map. Despite their differences, all of the algorithms below can be termed obstacle avoid- ance algorithms because the robot's local sensor readings play an important role in the robot's future trajectory. We first present the simplest obstacle avoidance systems that are used successfully in mobile robotics. The Bug algorithm represents such a technique in that only the most recent robot sensor values are used, and the robot needs, in addition to current sensor values, only approximate information

regarding the direction of the goal. More sophisticated algorithms are presented afterward, taking into account recent sensor history, robot kinematics, and even dynamics.

6.2.2.1 Bug algorithm

The Bug algorithm [101, 102] is perhaps the simplest obstacle avoidance algorithm one could imagine. The basic idea is to follow the contour of each obstacle in the robot's way and thus circumnavigate it.

With Bug1, the robot fully circles the object first, then departs from the point with the shortest distance toward the goal (figure 6.7). This approach is, of course, very inefficient but guarantees that the robot will reach any reachable goal.

With Bug2 the robot begins to follow the object's contour, but departs immediately when it is able to move directly toward the goal. In general this improved Bug algorithm will have significantly shorter total robot travel, as shown in figure 6.8. However, one can still construct situations in which Bug2 is arbitrarily inefficient (i.e., nonoptimal).

A number of variations and extensions of the Bug algorithm exist. We mention one more, the Tangent Bug [82], which adds range sensing and a local environmental represen-

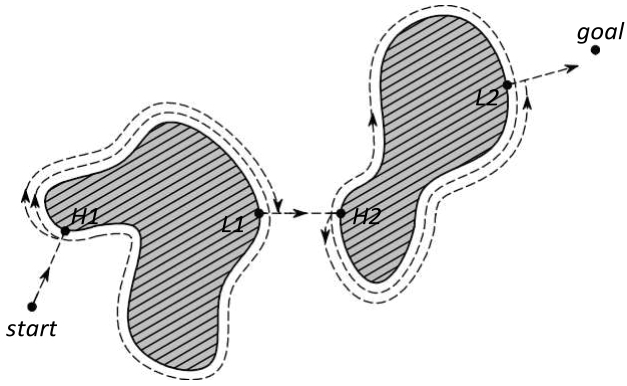


Figure 6.7

Bug1 algorithm with H1, H2, hit points, and L1, L2, leave points [102].

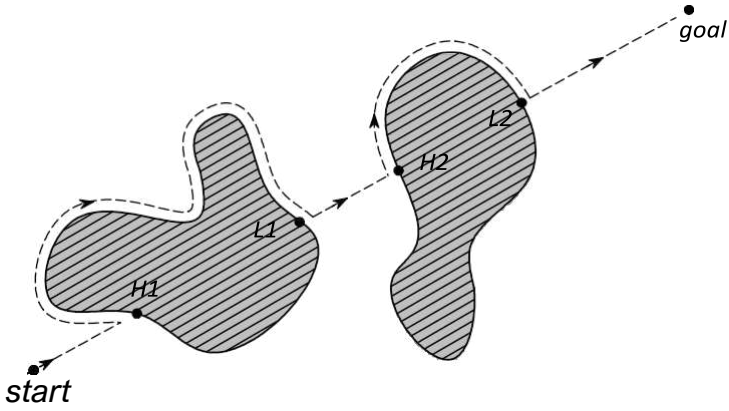


Figure 6.8

Bug2 algorithm with H1, H2, hit points, and L1, L2, leave points [102].

tation termed the local tangent graph (LTG). Not only can the robot move more efficiently toward the goal using the LTG, it can also go along shortcuts when contouring obstacles and switch back to goal seeking earlier. In many simple environments, Tangent Bug approaches globally optimal paths.

Practical application: example of Bug2. Because of the popularity and simplicity of Bug2, we present a specific example of obstacle avoidance using a variation of this tech- nique. Consider the path taken by the robot in figure 6.8. One can characterize the robot's motion in terms of two states, one that involves moving toward the goal and a second that involves moving around the contour of an obstacle. We will call the former state GOAL- SEEK and the latter WALLFOLLOW. If we can describe the motion of the robot as a function of its sensor values and the relative direction to the goal for each of these two states, and if we can describe when the robot should switch between them, then we will have a practical implementation of Bug2. The following pseudocode provides the highest-level control code for such a decomposition:

```
public void bug2(positiongoalPos){
 boolean atGoal = false;


 while( ! atGoal){

    position robotPos = robot.GetPos(&sonars);

    distance goalDist = getDistance(robotPos, goalPos);

   angle goalAngle = Math.atan2(goalPos, robotPos)-robot.GetAngle();
    velocity forwardVel, rotationVel;


    if(goalDist < atGoalThreshold){
       System.out.println("At Goal!");
       forwardVel = 0;
```

```
      rotationVel = 0;
      robot.SetState(DONE);
      atGoal = true;
    }
    else{
      forwardVel =ComputeTranslation(&sonars);
      if(robot.GetState() == GOALSEEK){
        rotationVel =ComputeGoalSeekRot(goalAngle);
        if(ObstaclesInWay(goalAngle, &sonars))
          robot.SetState(WALLFOLLOW);
      }
      if(robot.GetState() == WALLFOLLOW){
        rotationVel =
        ComputeRWFRot(&sonars);
        if( ! ObstaclesInWay(goalAngle,&sonars))
          robot.SetState(GOALSEEK);
      }
    }
    robot.SetVelocity(forwardVel, rotationVel);
  }
}
```

In the ideal case, when encountering an obstacle one would choose between left wall fol- lowing and right wall following depending on which direction is more promising. In this simple example we have only right wall following, a simplification for didactic purposes that ought not find its way into a real mobile robot program.

Now we consider specifying each remaining function in detail. Consider for our pur- poses a robot with a ring of sonars placed radially around the robot. This imagined robot will be differential-drive, so that the sonar ring has a clear "front" (aligned with the forward direction of the robot). Furthermore, the robot accepts motion commands of the form shown above, with a rotational velocity parameter and a translational velocity parameter. Mapping these two parameters to individual wheel speeds for each of the two differential drive chassis' drive wheels is a simple matter.

There is one condition we must define in terms of the robot's sonar readings, Obsta- clesInWay(). We define this function to be true whenever any sonar range reading in the direction of the goal (within 45 degrees of the goal direction) is short:

```
private boolean ObstaclesInWay(angle goalAngle, sensorvals
    sonars) { int minSonarValue;
    minSonarValue=MinRange(sonars, goalAngle
                                -(pi/4),goalAngle+(pi/4));
    return (minSonarValue < 200);
} // end ObstaclesInWay() //
```

Note that the function ComputeTranslation() computes

translational speed whether the robot is wall-following or heading toward the goal. In this simplified example, we define translation speed as being proportional to the largest range readings in the robot's approximate forward direction:

```
private int ComputeTranslation(sensorvals
    sonars) { int minSonarFront;

    minSonarFront = MinRange(sonars, -pi/4.0,
    pi/4.0); if (minSonarFront < 200) return 0;

    else return (Math.min(500, minSonarFront - 200));

} // end ComputeTranslation() //
```

There is a marked similarity between this approach and the potential field approach described in section 6.2.1.3. Indeed, some mobile robots implement obstacle avoidance by treating the current range readings of the robot as force vectors, simply carrying out vector addition to determine the direction of travel and speed. Alternatively, many will consider short-range readings to be repulsive forces, again engaging in vector addition to determine an overall motion command for the robot.

When faced with range sensor data, a popular way of determining rotation direction and speed is to simply subtract left and right range readings of the robot. The larger the differ- ence, the faster the robot will turn in the direction of the longer range readings. The follow- ing two rotation functions could be used for our Bug2 implementation:

```
private int ComputeGoalSeekRot(angle
    goalAngle) { if (Math.abs(goalAngle) <
    pi/10) return 0; else return (goalAngle *
    100);

} // end ComputeGoalSeekRot() //


private int ComputeRWFRot(sensorvals
    sonars) { int minLeft, minRight,
    desiredTurn; minRight =
    MinRange(sonars, -pi/2, 0); minLeft =
    MinRange(sonars, 0, pi/2);

    if (Math.max(minRight,minLeft) < 200) return (400);

                                          // hard left turn

    else {

        desiredTurn = (400 - minRight) * 2;

        desiredTurn = Math.inttorange(-400, desiredTurn,
        400); return desiredTurn;

    } // end else

} // end ComputeRWFRot() //
```

Note that the rotation function for the case of right wall following combines a general avoidance of obstacles with a bias to turn right when there is open space on the right, thereby staying close to the obstacle's contour. This solution is certainly not the best solu- tion for implementation of Bug2. For example, the wall follower could do a far better job by mapping the contour locally and using a PID control loop to achieve and maintain a specific distance from the contour during the right wall following action.

Although such simple obstacle avoidance algorithms are often used in simple mobile robots, they have numerous shortcomings. For example, the Bug2 approach does not take into account robot kinematics, which can be especially important with nonholonomic robots. Furthermore, since only the most recent sensor values are used, sensor noise can have a serious impact on real-world performance. The following obstacle avoidance tech- niques are designed to overcome one or more of these limitations.

6.2.2.2 Vector field histogram

Borenstein, together with Koren, developed the vector field histogram (VFH) [43]. Their previous work, which was concentrated on potential fields [92], was abandoned due to the method's instability and inability to pass through narrow passages. Later, Borenstein, together with Ulrich, extended the VFH algorithm to yield VFH+ [150] and VFH*[149].

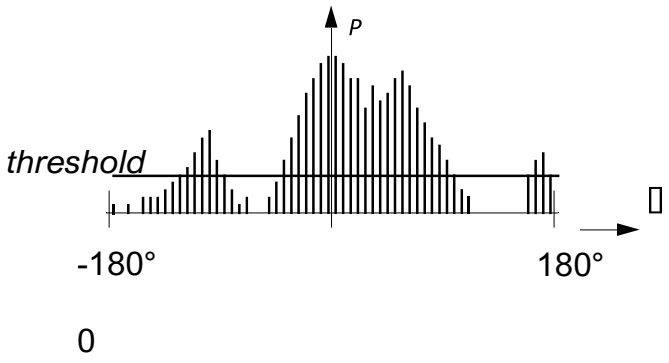


Figure 6.9

Polar histogram [93].

One of the central criticisms of Bug-type algorithms is that the robot's behavior at each instant is generally a function of only its most recent sensor readings. This can lead to unde- sirable and yet preventable problems in cases where the robot's instantaneous sensor read- ings do not provide enough information for robust obstacle avoidance. The VFH techniques overcome this limitation by creating a local map of the environment around the robot. This local map is a small occupancy grid, as described in section 5.7 populated only by relatively recent sensor range readings. For obstacle avoidance, VFH generates a polar histogram as shown in figure 6.9. The x-axis represents the angle at which the obstacle was found and the y-axis represents the probability *P* that there really is an obstacle in that direction based on the occupancy grid's cell

values.

From this histogram a steering direction is calculated. First all openings large enough for the vehicle to pass through are identified. Then a cost function is applied to every such candidate opening. The passage with the lowest cost is chosen. The cost function *G* has three terms:

$$G = a \cdot \text{target\_direction} + b \cdot \text{wheel\_orientation} + c \cdot \text{previous\_direction} \quad (6.11)$$

target_direction = alignment of the robot path with the goal;

wheel_orientation = difference between the new direction and the current wheel orien- tation;

previous_direction = difference between the previously selected direction and the new direction.

The terms are calculated such that a large deviation from the goal direction leads to a big cost in the term “target direction”. The parameters a , b , c in the cost function G tune the behavior of the robot. For instance, a strong goal bias would be expressed with a large value for a . For a complete definition of the cost function, refer to [92].

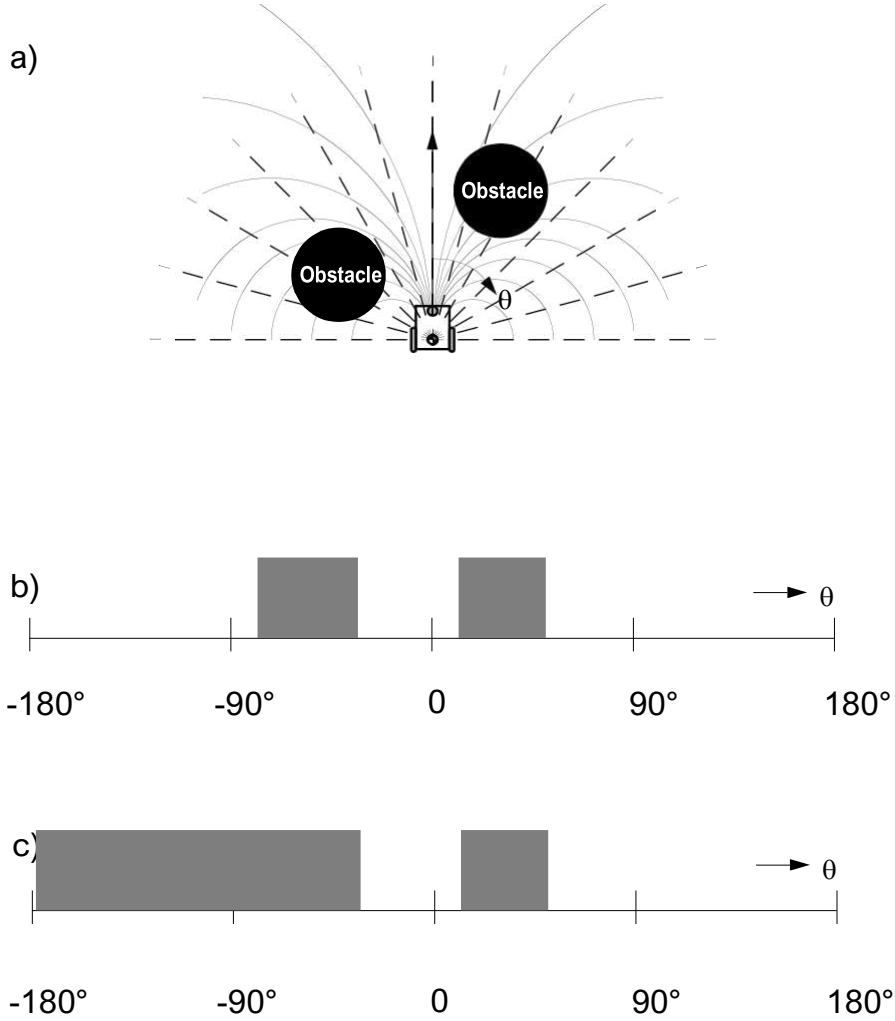


Figure 6.10

Example of blocked directions and resulting polar histograms [54]. (a) Robot and blocking obstacles.

(b) Polar histogram. (b) Masked polar histogram.

In the VFH+ improvement one of the reduction stages takes into account a simplified model of the moving robot's possible trajectories based on its kinematic limitations (e.g., turning radius for an Ackerman vehicle). The robot is modeled to move in arcs or straight lines. An obstacle thus blocks all of the robot's allowable trajectories which pass through the obstacle (figure 6.10a). This results in a masked polar histogram where obstacles are enlarged so that all kinematically blocked trajectories are properly taken into account (fig- ure 6.10c).

6.2.2.3 The bubble band technique

This idea is an extension for nonholonomic vehicles of the elastic band concept suggested by Khatib and Quinlan [86]. The original elastic band concept applied only to holonomic

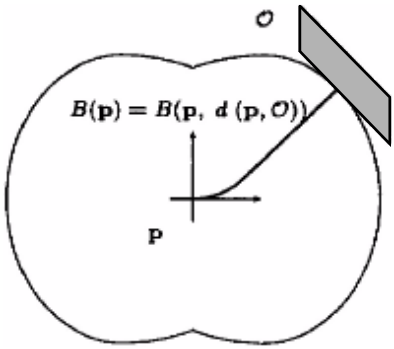


Figure 6.11

Shape of the bubbles around the vehicle (courtesy of Raja Chatila [85]).

vehicles and so we focus only on the bubble band extension made by Khatib, Jaouni, Cha- tila, and Laumod [85].

A *bubble* is defined as the maximum local subset of the free space around a given con- figuration of the robot that which can be traveled in any direction without collision. The bubble is generated using a simplified model of the robot in conjunction with range infor- mation available in the robot's map. Even with a simplified model of the robot's geometry, it is possible to take into account the actual shape of the robot when calculating the bubble's size (figure 6.11). Given such bubbles, a band or string of bubbles can be used along the trajectory from the robot's initial position to its goal position to show the robot's expected free space throughout its path (see figure 6.12).

Clearly, computing the bubble band requires a global map and a global path planner. Once the path planner's initial trajectory has been computed and the bubble band is calcu- lated, then

modification of the planned trajectory ensues. The bubble band takes into account forces from modeled objects and internal forces. These internal forces try to mini- mize the "slack" (energy) between adjacent bubbles. This process, plus a final smoothing operation, makes the trajectory smooth in the sense that the robot's free space will change as smoothly as possible during path execution.

Of course, so far this is more akin to path optimization than obstacle avoidance. The obstacle avoidance aspect of the bubble band strategy comes into play during robot motion. As the robot encounters unforeseen sensor values, the bubble band model is used to deflect the robot from its originally intended path in a way that minimizes bubble band *tension*.

An advantage of the bubble band technique is that one can account for the actual dimen- sions of the robot. However, the method is most applicable only when the environment con- figuration is well-known ahead of time, just as with off-line path- planning techniques.

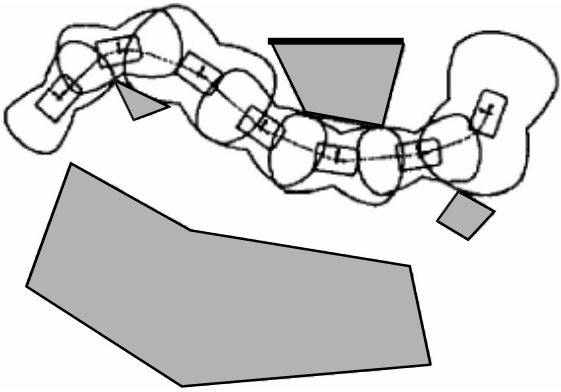


Figure 6.12

A typical bubble band (Courtesy of Raja Chatila [85]).

6.2.2.4 Curvature velocity techniques

The basic curvature velocity approach. The curvature velocity approach (CVM) from Simmons [135] enables the actual kinematic constraints and even some dynamic con- straints of the robot to be taken into account during obstacle avoidance, which is an advan- tage over more primitive techniques. CVM begins by adding physical constraints from the robot and the environment to a velocity space. The velocity space consists of rotational velocity ω and translational velocity v, thus assuming that the robot only travels along arcs of circles with curvature $c = \omega / v$.

Two types of constraints are identified: those derived from the robot's limitations in acceleration and speed, typically $-v_{max} \leq v \leq v_{max}$, $-\omega_{max} \leq \omega \leq \omega_{max}$; and, second, the

constraints from obstacles blocking certain v and ω values due to their positions. The obstacles begin as objects in a Cartesian grid but are then transformed to the velocity space by calculating the distance from the robot position to the obstacle following some constant curvature robot trajectory, as shown in figure 6.13. Only the curvatures that lie within c_{min} and c_{max} are considered since that curvature space will contain all legal trajectories.

To achieve real-time performance the obstacles are approximated by circular objects and the contours of the objects are divided into few intervals. The distance from an endpoint of an interval to the robot is calculated and in between the endpoints the distance function is assumed to be constant.

The final decision of a new velocity (v and ω) is made by an objective function. This function is only evaluated on that part of the velocity space that fulfills the kinematic and

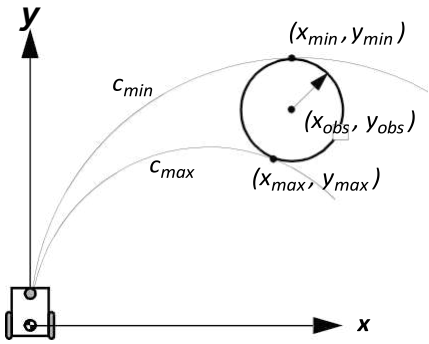


Figure 6.13

Tangent curvatures for an obstacle (from [135]).

dynamic constraints as well as the constraints due to obstacles. The use of a Cartesian grid for initial obstacle representation enables straightforward sensor fusion if, for instance, a robot is equipped with multiple types of ranging sensors.

CVM takes into consideration the dynamics of the vehicle in useful manner. However a limitation of the method is the circular simplification of obstacle shape. In some environ- ments this is acceptable while, in other environments, such a simplification can cause seri- ous problems. The CVM method can also suffer from local minima since no *a priori* knowledge is used by the system.

The lane curvature method. Ko and Simmons presented an improvement of the CVM which they named the lane curvature method, (LCM) [87] based on their experiences with the

shortcomings of CVM. CVM had difficulty guiding the robot through intersections of corridors. The problems stemmed from the approximation that the robot moves only along fixed arcs, whereas in practice the robot can change direction many times before reaching an obstacle.

LCM calculates a set of desired lanes, trading off lane length and lane width to the clos- est obstacle. The lane with the best properties is chosen using an objective function. The local heading is chosen in such way that the robot will transition to the best lane if it is not in that lane already.

Experimental results have demonstrated better performance as compared to CVM. One caveat is that the parameters in the objective function must be chosen carefully to optimize system behavior.

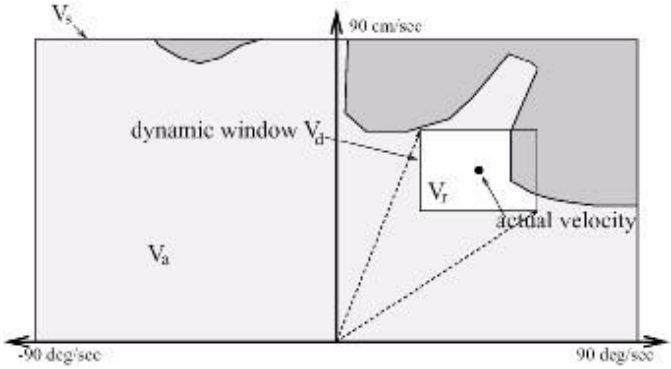


Figure 6.14

The dynamic window approach (courtesy of Dieter Fox [69]). The rectangular window shows the possible speeds □*v*□ □□ and the overlap with obstacles in configuration space.

6.2.2.5 Dynamic window approaches

Another technique for taking into account robot kinematics constraints is the dynamic window obstacle avoidance method. A simple but very effective dynamic model gives this approach its name. Two such approaches are represented in the literature. The dynamic window approach [69] of Fox, Burgard, and Thrun, and the global dynamic window approach [44] of Brock and Khatib.

The local dynamic window approach. In the local dynamic window approach the kine- matics of the robot is taken into account by searching a well-chosen velocity space. The velocity space is all possible sets of tuples (*v* , □) whe

re v is the velocity and ω is the

angular velocity. The approach assumes that robots move only in circular arcs representing each such tuple, at least during one timestamp.

Given the current robot speed, the algorithm first selects a *dynamic window* of all tuples (v, ω) that can be reached within the next sample period, taking into account the acceler- ation capabilities of the robot and the cycle time. The next step is to reduce the *dynamic window* by keeping only those tuples that ensure that the vehicle can come to a stop before hitting an obstacle. The remaining velocities are called admissible velocities. In figure 6.14, a typical dynamic window is represented. Note that the shape of the dynamic window is rectangular, which follows from the approximation that the dynamic capabilities for trans- lation and rotation are independent.

A new motion direction is chosen by applying an objective function to all the admissible velocity tuples in the dynamic window. The objective function prefers fast forward motion,

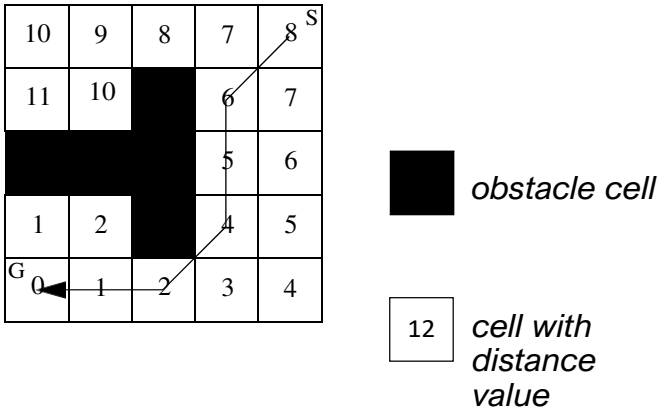


Figure 6.15

An example of the distance transform and the resulting path as it is generated by the NF1 function. S, start; G, goal.

maintenance of large distances to obstacles and alignment to the goal heading. The objec- tive function *O* has the form

$$O = a \cdot heading(v) + b \cdot velocity(v) + c \cdot dist(v) \quad (6.12)$$

heading = Measure of progress toward the goal location;

velocity = Forward velocity of the robot

encouraging fast movements; dist = Distance to the

closest obstacle in the trajectory.

The global dynamic window approach. The global dynamic window approach adds, as the name suggests, global thinking to the algorithm presented above. This is done by adding *NF1*, or grassfire, to the objective function *O* presented above (see section 6.2.1.2 and figure 6.15). Recall that NF1 labels the cells in the occupancy grid with the total distance L to the goal. To make this faster the global dynamic window approach calculates the NF1 only on a selected rectangular region which is directed from the robot toward the goal. The width of the region is enlarged and recalculated if the goal cannot be reached within the constraints of this chosen region.

This allows the global dynamic window approach to achieve some of the advantages of global path planning without complete a priori knowledge. The occupancy grid is updated from range measurements as the robot moves in the environment. The NF1 is calculated for every new updated version. If the NF1 cannot be calculated due to the fact that the robot is surrounded by obstacles, the method degrades to the dynamic window approach. This keeps the robot moving so that a possible way out may be found and NF1 can resume.

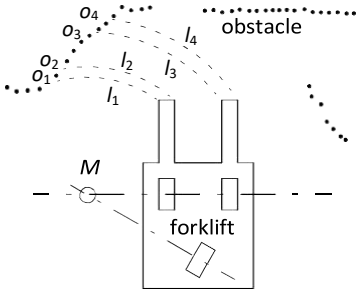


Figure 6.16

Distances l_i resulting from the curvature i_C , when the robot rotates around *M* (from [128]).

The global dynamic window approach promises real-time, dynamic constraints, global thinking, and minimal free obstacle avoidance at high speed. An implementation has been demonstrated with an omnidirectional robot using a 450 MHz on-board PC. This system produced a cycle frequency of about 15 Hz when the occupancy grid was 30 × 30 m with a 5 cm resolution. Average robot speed in the tests was greater than 1 m/s.

6.2.2.6 The Schlegel approach to obstacle avoidance

Schlegel [128] presents an approach that considers the dynamics as well as the actual shape of the robot. The approach is adopted for raw laser data measurements and sensor fusion using a Cartesian grid to represent the obstacles in the environment. Real-time performance is achieved by use of precalculated lookup tables.

As with previous methods we have described, the basic assumption is that a robot moves in trajectories built up by circular arcs, defined as curvatures i_C . Given a certain curvature i_C Schlegel calculates the distance l_i to collision between a single obstacle point (x, y) in the Cartesian grid and the robot, depicted in figure 6.16. Since the robot is allowed to be any shape this calculation is time consuming and the result is therefore precalculated and stored in a lookup table.

For example, the search space window V_S is defined for a differential-drive robot to be all the possible speeds of the left and right wheels, v_r, v_l . The dynamic constraints of the robot are taken into account by refining V_S to only those values which are reachable within the next timestep, given the present robot motion. Finally, an objective function chooses the best speed and direction by trading off goal direction, speed, and distance until colli- sion.

During testing Schlegel used a wavefront path planner. Two robot chassis were used, one with synchro-drive kinematics and one with tricycle kinematics. The tricycle-drive

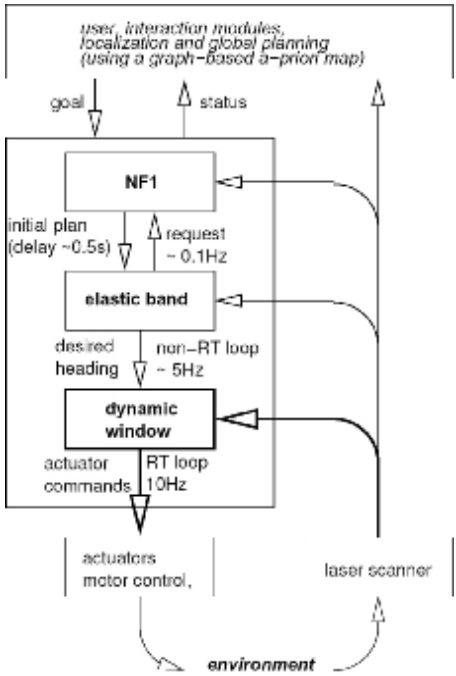


Figure 6.17

Flow diagram of the ASL approach [122].

robot is of particular interest because it was a forklift with a complex shape that had a sig- nificant impact on obstacle avoidance. Thus the demonstration of reliable obstacle avoidance with the forklift is an impressive result. Of course, a disadvantage of this approach is the potential memory requirements for the lookup table. In their experiments, the authors
used lookup tables of up to 2.5 Mb m Cartesian grid with a resolution using a 6 □ 6 of
10 cm and 323 different curvatures.

6.2.2.7 The ASL approach

The Autonomous Systems Lab (ASL) at the Swiss Federal Institute of Technology devel- oped an obstacle avoidance method [122] for a mass exhibition [132] that required mobile robots to move through dense crowds and also ensure a certain flow of visitors. It merges three approaches in order to have a system that moves smoothly without stopping for replanning and is able to carefully nudge its way through when it is safe to do so. It is a local path-planning and obstacle avoidance method receiving its input in the form of way- points from higher levels. An overview is given in figure 6.17.

Local path planning is performed by NF1. The resulting path is converted to an elastic band that does not take into account kinematics, taking advantage of the fact that the octag- onal robot used in the exhibition can turn on the spot most of the time. This keeps path updates as simple as possible. An enhanced dynamic window then takes care of moving the robot along the path.

6.2.2.8 Nearness diagram

Attempting to close a model fidelity gap in obstacle avoidance methods, the nearness dia- gram (ND) [107] can be considered to have some similarity to a VFH but solves several of its shortcomings, especially in very cluttered spaces. It was also used in [108] to take into account more precise geometric, kinematic, and dynamic constraints. This was achieved by breaking the problem down into generating the most promising direction of travel with the sole constraint a circular robot, then adapting this to the kinematic and dynamic constraints of the robot, followed by a correction for robot shape if is noncircular (only rectangular shapes were supported in the original publication). Global reasoning was added to the approach and termed the global nearness diagram (GND) in [110], somewhat similar to the GDWA extension to the DWA, but based on a workspace representation (instead of con- figuration space) and updating free space in addition to obstacle information.

6.2.2.9 Gradient method

Realizing that current computer technology allows fast recalculation of wavefront propa- gation techniques, the gradient method [89] formulates a grid-based global path planning that takes into account closeness to obstacles and allows generating continuous interpola- tions of the gradient direction at any given

point in the grid. The NF1 is a special case of the proposed algorithm, which calculates a navigation function at each timestep and uses the resulting gradient information to drive the robot toward the goal on a smooth path and not grazing obstacles unless necessary.

6.2.2.10 Adding dynamic constraints

Attempting to address the lack of dynamic models in most of the obstacle avoidance approaches discussed above, a new kind of space representation was proposed by Minguez, Montano, and Khatib in [109]. The ego-dynamic space is equally applicable to workspace and configuration space methods. It transforms obstacles into distances that depend on the braking constraints and sampling time of the underlying obstacle avoidance method. In combination with the proposed spatial window (PF) to represent acceleration capabilities, the approach was tested in conjunction with the ND and PF methods and gives satisfactory results for circular holonomic robots, with plans to extend it to nonholonomic, noncircular architectures.

6.2.2.11 Other approaches

The approaches described above are some of the most popularly referenced obstacle avoid- ance systems. There are, however, a great many additional obstacle avoidance techniques in the mobile robotics community. For example Tzafestas and Tzafestas [148] provide an overview of fuzzy and neurofuzzy approaches to obstacle avoidance. Inspired by nature, Chen and Quinn [56] present a biological approach in which they replicate the neural net- work of a cockroach. The network is then applied to a model of a four-wheeled vehicle.

The Liapunov functions form a well known theory that can be used to prove stability for nonlinear systems. In the paper of Vanualailai, Nakagiri, and Ha [153] the Liapunov func- tions are used to implement a control strategy for two-point masses moving in a known environment. All obstacles are defined as antitargets with an exact position and a circular shape. The antitargets are then used when building up the control laws for the system. How- ever, this complex mathematical model has not been tested on a real-world robot to our knowledge.

6.2.2.12 Overview

Table 6.1 gives an overview on the different approaches for obstacle avoidance.

Table 6.1

Overview of the most popular obstacle avoidance algorithms

	model fidelity	>	other requisites	se	↩ e	perfor mance	

B			method
Tangent	Bug 2	Bug1 [101	
poi	poin	po	shap
			kinem
			dyna
loc	local	lo	
local			lo
			glo
			path
ran ge	tactil e	tactile	
			cy
			archite
efficient	ineffici	very	remar

Table 6.1

Overview of the most popular obstacle avoidance algorithms

Bubble		Vector Field Histogram			method	
Bubble ban	Elastic ban	VF H*	VFH +	VF H		
C-	C-	cir	cir	simplistic	shap	model fidelity
exact		ba	ba		kinem	
		simplist	simplist		dyna	
loc	globa	essentiall	loc	loc	view	
		histogra	histogra	histogram	lo	other requisites
polygo	polygo				glo bal	
requir	requir				path plann	
		sonars	sonars	ran ge	sensors	
variou s	variou s	nonholon omic	nonholon omic	synchro- drive	tested robots	
		6 … 242	6	27 ms	cy	perfor mance
		66 MHz,	66 MHz,	20 MHz,	archite	
		fewer	local	local	remar	

Table 6.1

Overview of the most popular obstacle avoidance algorithms

Dynamic		Curvature velocity		method	
Global dynami	Dynamic window	Lane curvatur	Curvature velocity		
circ	circl	cir	circle	shap	model fidelity
(holonom	exa	ex	exact	kinem	
bas	basi	ba	basic	dyna	
glo	local	loc	local	view	
	obstacle line	histogram	histogram	lo	other requisites
C-space				glo	
NF				path	
180° FOV SCK	24 sonars ring, 56 infrared	24 sonars ring,	24 sonars ring,	sensors	
holonomic (circular)	synchro-drive (circular)	synchro-drive	synchro-drive	tested robots	
6.7 ms	250	125 ms	125 ms	cy	performance
450 MHz,	486	200 MHz,	66 MHz,	archite	
turning into	local	local	local	remar	

Table 6.1

Overview of the most popular obstacle avoidance algorithms

Ot					meth od	
Gradient method	Global nearness	Nearness diagram	ASL appr	Schl ogo		
circle	circle (but	circle (but	polygon	polygon	shap	model fidelity
exact	(holono	(holonom	exact	exact	kinem	
basic			basic	basic	dyna	
global	glo	loc	local	global	view	
	gri		grid		lo	other requisites
local	NF			grid	glo bal	
fused			graph	wavefront	path plann	
180° FOV distance	180° FOV SCK	180° FOV SCK	2x 180° FOV SCK	360° FOV	sensors	
nonholo nomic	holonom ic	holonomic (circular)	differential drive	synchrodrive (circular),	tested robots	
100 ms (core			100 ms (core		cy	perfor mance
266 MHz,			380 MHz,		archite	
		local	turning into	allows shape	remar	

6.3 Navigation Architectures

Given techniques for path planning, obstacle avoidance, localization, and perceptual inter- pretation, how do we combine all of these into one complete robot system for a real-world application? One way to proceed would be to custom-design an application-specific, mono- lithic software system that implements everything for a specific purpose. This may be efficient in the case of a trivial mobile robot application with few features and even fewer planned demonstrations. But for any sophisticated and long-term mobile robot system, the issue of mobility architecture should be addressed in a principled manner. The study of *nav- igation architectures* is the study of principled designs for the software modules that con- stitute a mobile robot navigation system. Using a well-designed navigation architecture has a number of concrete advantages:

6.3.1 Modularity for code reuse and sharing

Basic software engineering principles embrace software modularity, and the same general motivations apply equally to mobile robot applications. But modularity is of even greater importance in mobile robotics because in the course of a single project the mobile robot hardware or its physical environmental characteristics can change dramatically, a challenge most traditional computers do not face. For example, one may introduce a Sick laser rangefinder to a robot that previously used only ultrasonic rangefinders. Or one may test an existing navigator robot in a new environment where there are obstacles that its sensors cannot detect, thereby demanding a new path-planning representation.

We would like to change part of the robot's competence without

causing a string of side effects that force us to revisit the functioning of other robot competences. For instance we would like to retain the obstacle avoidance module intact, even as the particular ranging sensor suite changes. In a more extreme example, it would be ideal if the nonholonomic obstacle avoidance module could remain untouched even when the robot's kinematic struc- ture changes from a tricycle chassis to a differential-drive chassis.

6.3.2 Control localization

Localization of robot control is an even more critical issue in mobile robot navigation. The basic reason is that a robot architecture includes multiple types of control functionality (e.g., obstacle avoidance, path planning, path execution, etc.). By localizing each function- ality to a specific unit in the architecture, we enable individual testing as well as a princi- pled strategy for control composition. For example, consider collision avoidance. For stability in the face of changing robot software, as well as for focused verification that the obstacle avoidance system is correctly implemented, it is valuable to localize all software related to the robot's obstacle avoidance process. At the other extreme, high-level planning and task-based decision-making are required for robots to perform useful roles in their

environment. It is also valuable to localize such high-level decision-making software, enabling it to be tested exhaustively in simulation and thus verified even without a direct connection to the physical robot. A final advantage of localization is associated with learn- ing. Localization of control can enable a specific learning algorithm to be applied to just one aspect of a mobile robot's overall control system. Such targeted learning is likely to be the first strategy that yields successful integration of learning and traditional mobile robot- ics.

The advantages of localization and modularity prove a compelling case for the use of principled navigation architectures.

One way to characterize a particular architecture is by its decomposition of the robot's software. There are many favorite robot architectures, especially when one considers the relationship between artificial intelligence level decision making and lower-level robot control. For descriptions of such high-level architectures, refer to [2] and [26]. Here we concentrate on navigation competence. For this purpose, two decompositions are particu- larly relevant: temporal decomposition and control decomposition. In section 6.3.3 we define these two types of decomposition, then present an introduction to *behaviors*, which are a general tool for implementing control decomposition. Then, in section 6.3.4 we present three types of navigation architectures, describing for each architecture an imple- mented mobile robot case study.

6.3.3 Techniques for decomposition

Decompositions identify axes along which we can justify discrimination of robot software into distinct modules.

Decompositions also serve as a way to classify various mobile robots into a more quantitative taxonomy. *Temporal decomposition* distinguishes between real- time and non real-time demands on mobile robot operation. *Control decomposition* identi- fies the way in which various control outputs within the mobile robot architecture combine to yield the mobile robot's physical actions. Below we describe each type of decomposition in greater detail.

6.3.3.1 Temporal decomposition

A temporal decomposition of robot software distinguishes between processes that have varying real-time and non real-time demands. Figure 6.18 depicts a generic temporal decomposition for navigation. In this figure, the most real-time processes are shown at the bottom of the *stack*, with the highest category being occupied by processes with no real- time demands.

The lowest level in this example captures functionality that must proceed with a guar- anteed fast cycle time, such as a 40 Hz bandwidth. In contrast, a quasi real-time layer may capture processes that require, for example, 0.1 second response time, with large allowable worst-case individual cycle times. A tactical layer can represent decision-making that

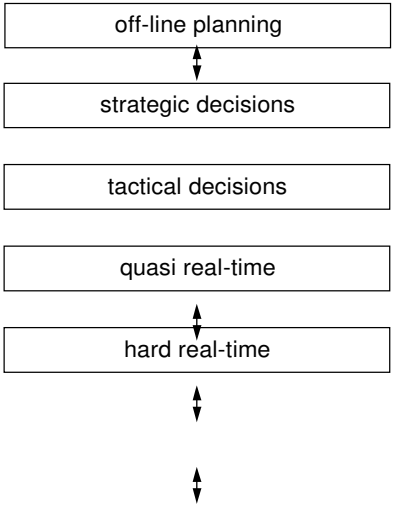


Figure 6.18

Generic temporal decomposition of a navigation architecture.

affects the robot's immediate actions and is therefore subject to some temporal constraints, while a strategic or off-line layer represents decisions that affect the robot's behavior over the long term, with few temporal constraints on the module's response time.

Four important, interrelated trends correlate with temporal decomposition. These are not set in stone; there are exceptions. Nevertheless, these general properties of temporal decompositions are enlightening:

Sensor response time. A particular module's sensor response time can be defined as the amount of time between acquisition of a sensor-based event and a corresponding change in the output

of the module. As one moves up the stack in figure 6.18 the sensor response time tends to increase. For the lowest-level modules, the sensor response time is often limited only by the raw processor and sensor speeds. At the highest-level modules, sensor response can be limited by slow and deliberate decision-making processes.

Temporal depth. Temporal depth is a useful concept applying to the temporal window that affects the module's output, both backward and forward in time. *Temporal horizon* describes the amount of look ahead used by the module during the process of choosing an output. *Temporal memory* describes the historical time span of sensor input that is used by the module to determine the next output. Lowest-level modules tend to have very little temporal depth in both directions, whereas the deliberative processes of highest-level modules make use of a large temporal memory and consider actions based on their long-term consequences, making note of large temporal horizons.

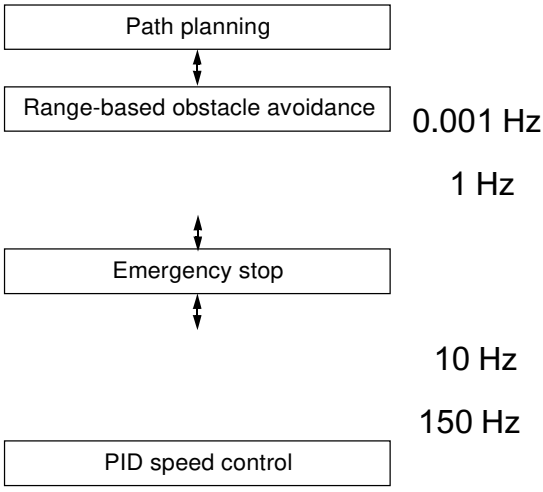


Figure 6.19

Sample four-level temporal decomposition of a simple navigating mobile robot. The column on the right indicates realistic bandwidth values for each module.

Spatial locality. Hand in hand with temporal span, the spatial impact of layers increases dramatically as one moves from low-level modules to high-level modules. Real-time mod- ules tend to control wheel speed and orientation, controlling spatially localized behavior. High-level strategic decision-making has little or no bearing on local position, but informs global position far into the future.

Context specificity. A module makes decisions as a function not only of its immediate inputs but also as a function of the robot's context as captured by other variables, such as the

robot's representation of the environment. Lowest-level modules tend to produce out- puts directly as a result of immediate sensor inputs, using little context and therefore being relatively context insensitive. Highest-level modules tend to exhibit very high context spec- ificity. For strategic decision-making, given the same sensor values, dramatically different outputs are nevertheless conceivable depending on other contextual parameters.

An example demonstrating these trends is depicted in figure 6.19, which shows a tem- poral decomposition of a simplistic navigation architecture into four modules. At the lowest level the PID control loop provides feedback to control motor speeds. An emergency stop module uses short-range optical sensors and bumpers to cut current to the motors when it predicts an imminent collision. Knowledge of robot dynamics means that this module by nature has a greater temporal horizon than the PID module. The next module uses longer- range laser rangefinding sensor returns to identify obstacles well ahead of the robot and make minor course deviations. Finally, the path planner module takes the robot's initial and goal positions and produces an initial trajectory for execution, subject to change based on actual obstacles that the robot collects along the way.

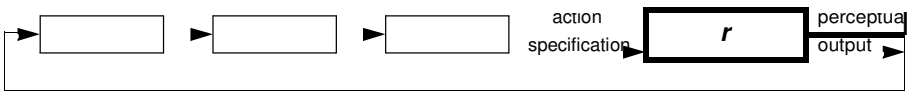


Figure 6.20

Example of a pure serial decomposition.

Note that the cycle time, or bandwidth, of the modules changes by orders of magnitude between adjacent modules. Such dramatic differences are common in real navigation architectures, and so temporal decomposition tends to capture a significant axis of variation in a mobile robot's navigation architecture.

6.3.3.2 Control decomposition

Whereas temporal decomposition discriminates based on the time behavior of software modules, control decomposition identifies the way in which each module's output contrib- utes to the overall robot control outputs. Presentation of control decomposition requires the evaluator to understand the basic principles of discrete systems representation and analysis. For a lucid introduction to the theory and formalism of discrete systems, see [17, 71].

Consider the robot algorithm and the physical robot instantiation (i.e., the robot form and its environment) to be members of an overall system whose connectivity we wish to examine. This overall system *S* is comprised of a set *M* of modules, each module *m* con- nected to other modules via inputs

and outputs. The system is *closed*, meaning that the input of every module m is the output of one or more modules in M. Each module has precisely one output and one or more inputs. The one output can be connected to any number of other modules inputs.

We further name a special module r in M to represent the physical robot and environ-

ment. Usually by r we represent the physical object on which the robot algorithm is

intended to have impact, and from which the robot algorithm derives perceptual inputs. The module r contains one input and one output line. The input of r represents the complete action specification for the physical robot. The output of r represents the complete percep- tual output to the robot. Of course the physical robot may have many possible degrees of freedom and, equivalently, many discrete sensors. But for this analysis we simply imagine the entire input/output vector, thus simplifying r to just one input and one output. For sim- plicity we will refer to the input of r as O and to the robot's sensor readings I. From the point of view of the rest of the control system, the robot's sensor values I are inputs, and the robot's actions O are the outputs, explaining our choice of I and O.

Control decomposition discriminates between different types of control pathways through the portion of this system comprising the robot algorithm. At one extreme, depicted in figure 6.20 we can consider a perfectly linear, or sequential control pathway.

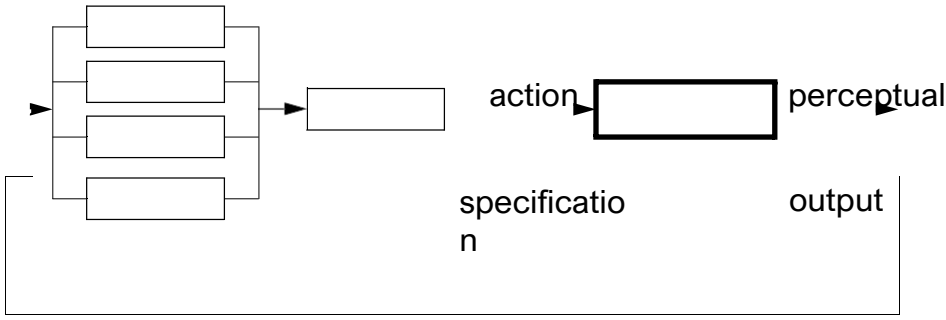


Figure 6.21

Example of a pure parallel decomposition.

Such a serial system uses the internal state of all associated modules and the value of the robot's percept *I* in a sequential manner to compute the next robot action *O* . A pure serial architecture has advantages relating to predictability and verifiability. Since the state and outputs of each module depend entirely on the inputs it receives from the module upstream, the entire system, including the robot, is a single well-formed loop. Therefore, the overall behavior of the system can be evaluated using well-known discrete forward simulation methods.

Figure 6.21 depicts the extreme opposite of pure serial control, a fully parallel control architecture. Because we choose to define *r* as a module with precisely one input, this par- allel system includes a special module *n* that provides a single output for the consumption of *r* . Intuitively, the fully parallel system distributes responsibility for the system's control output *O* across multiple modules, possibly simultaneously. In a pure sequential system, the control flow is a linear sequence through a string of modules.

Here, the control flow contains a *combination* step at which point the result of multiple modules may impact *O* in arbitrary ways.

Thus parallelization of control leads to an important question: how will the output of each component module inform the overall decision concerning the value of *O*? One simple combination technique is temporal switching. In this case, called *switched parallel*, the system has a parallel decomposition but at any particular instant in time the output *O* can be attributed to one specific module. The value of *O* can of course depend on a differ- ent module at each successive time instant, but the instantaneous value of *O* can always be determined based on the functions of a single module. For instance, suppose that a robot has an obstacle avoidance module and a path-following module. One switched control implementation may involve execution of the path-following recommendation whenever the robot is more than 50 cm from all sensed obstacles and execution of the obstacle avoid- ance recommendation when any sensor reports a range closer than 50 cm.

The advantage of such switched control is particularly clear if switching is relatively rare. If the behavior of each module is well understood, then it is easy to characterize the behavior of the switched control robot: it will obstacle avoid at times, and it will path- follow other times. If each module has been tested independently, there is a good chance the switched control system will also perform well. Two important disadvantages must be noted. First, the overall behavior of the robot can become quite poor if the switching is itself a high-frequency event. The robot may be unstable in such cases, switching motion modes so rapidly as to dramatically devolve into behavior that is neither path-following nor obsta- cle avoiding. Another disadvantage of switched control is that the robot has no path-follow- ing bias when it is obstacle avoiding (and vice versa). Thus in cases where control *ought* to mix recommendations from among multiple modules, the switched control methodology fails.

In contrast, the much more complex *mixed parallel* model allows control at any given time to be shared between multiple modules. For example, the same robot could take the obstacle avoidance module's output at all times, convert it to a velocity vector, and combine it with the path-following module's output using vector addition. Then the output of the robot would never be due to a single module, but would result from the mathematical com- bination of both modules outputs. Mixed parallel control is more general than switched control, but by that token it is also a more challenging technique to use well. Whereas with switched control most poor behavior arises out of inopportune switching behavior, in mixed control the robot's behavior can be quite poor even more readily. Combining multi- ple recommendations mathematically does not guarantee an outcome that is globally

supe- rior, just as combining multiple vectors when deciding on a swerve direction to avoid an obstacle can result in the very poor decision of straight ahead. Thus great care must be taken in mixed parallel control implementations to fashion mixture formulas and individual module specifications that lead to effective mixed results.

Both the switched and mixed parallel architectures are popular in the behavior-based robotics community. Arkin [2] proposes the *motor-schema* architecture in which *behaviors* (i.e., modules in the above discussion) map sensor value vectors to motor value vectors. The output of the robot algorithm is generated, as in mixed parallel systems, using a linear combination of the individual behavior outputs. In contrast, Maes [103, 104] produces a switched parallel architecture by creating a *behavior network* in which a behavior is chosen discretely by comparing and updating activation levels for each behavior. The subsumption architecture of Brooks [45] is another example of a switched parallel architecture, although the active model is chosen via a suppression mechanism rather than activation level. For a further discussion, see [2].

One overall disadvantage of parallel control is that verification of robot performance can be extremely difficult. Because such systems often include truly parallel, multithreaded implementations, the intricacies of robot-environment interaction and sensor timing

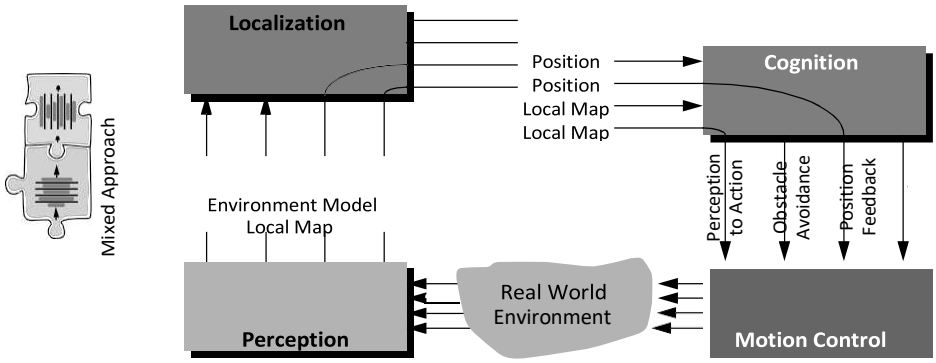


Figure 6.22

The basic architectural example used throughout this text.

required to properly represent all conceivable module-module interactions can be difficult or impossible to simulate. So, much testing in the parallel control community is performed empirically using physical robots.

An important advantage of parallel control is its biomimetic aspect. Complex organic organisms benefit from large degrees of true parallelism (e.g., the human eye), and one goal of the parallel control community is to understand this biologically common strategy and leverage it to advantage in robotics.

6.3.4 Case studies: tiered robot architectures

We have described temporal and control decompositions of robot architecture, with the common theme that the roboticist is always composing multiple modules together to make up that architecture. Let us turn again toward the overall mobile robot navigation task with this understanding in mind. Clearly, robot behaviors play an important role at the real-time levels of robot

control, for example, path-following and obstacle avoidance. At higher tem- poral levels, more tactical tasks need to modulate the activation of behaviors, or modules, in order to achieve robot motion along the intended path. Higher still, a global planner could generate paths to provide tactical tasks with global foresight.

In chapter 1, we introduced a functional decomposition showing such modules of a mobile robot navigator from the perspective of information flow. The relevant figure is shown here again as figure 6.22.

In such a representation, the arcs represent aspects of real-time and non real-time com- petence. For instance, obstacle avoidance requires little input from the localization module and consists of fast decisions at the cognition level followed by execution in motion con- trol. In contrast, PID position feedback loops bypass all high-level processing, tying the perception of encoder values directly to lowest-level PID control loops in motion control.

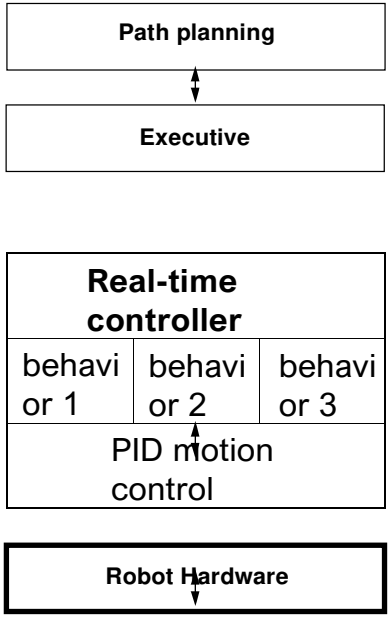


Figure 6.23

A general tiered mobile robot navigation architecture based on a temporal decomposition.

The trajectory of arcs through the four software modules is provides temporal information in such a representation.

Using the tools of this chapter, we can now present this same architecture from the per- spective of a temporal decomposition of functionality. This is particularly useful because we wish to discuss the interaction of strategic, tactical, and real-time processes in a naviga- tion system.

Figure 6.23 depicts a generic tiered architecture based on the approach of Pell and col- leagues [120] used in designing an

autonomous spacecraft, *Deep Space One*. This figure is similar to figure 6.19 in presenting a temporal decomposition of robot competence. How- ever, the boundaries separating each module from adjacent modules are specific to robot navigation.

Path planning embodies strategic-level decision-making for the mobile robot. Path planning uses all available global information in non real time to identify the right sequence of local actions for the robot. At the other extreme, *real-time control* represents compe- tences requiring high bandwidth and tight sensor-effector control loops. At its lowest level, this includes motor velocity PID loops. Above those, real-time control also includes low- level behaviors that may form a switch or mixed parallel architecture.

In between the path planner and real-time control tiers sits the *executive*, which is responsible for mediating the interface between planning and execution. The executive is responsible for managing the activation of behaviors based on information it receives from the planner. The executive is also responsible for recognizing failure, saving (placing the robot in a stable state), and even re-initiating the planner as necessary. It is the executive in

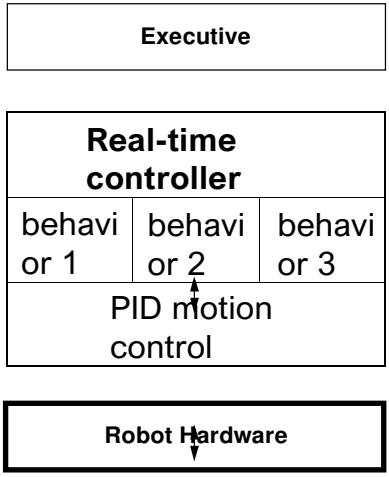


Figure 6.24

A two-tiered architecture for off-line planning.

this architecture that contains all tactical decision-making as well as frequent updates of the robot's short-term memory, as is the case for localization and mapping.

It is interesting to note the similarity between this general architecture, used in many specialized forms in mobile robotics today, and the architecture implemented by Shakey, one of the very first mobile robots, in 1969 [115]. Shakey had *LLA* (low-level actions) that formed the lowest architectural tier. The implementation of each LLA included the use of sensor values in a tight loop just as in today's behaviors. Above that, the middle architec- tural tier included the *ILA* (intermediate-level actions), which would activate and deactivate LLA as required based on perceptual feedback during execution. Finally, the topmost tier

for Shakey was *STRIPS* (Stanford Research Institute Planning System), which provided global look ahead and planning, delivering a series of tasks to the intermediate executive layer for execution.

Although the general architecture shown in figure 6.23 is useful as a model for robot navigation, variant implementations in the robotics community can be quite different. Below, we present three particular versions of the general tiered architecture, describing for each version at least one real-world mobile robot implementation. For broader discussions of various robot architectures, see [26].

6.3.4.1 Off-line planning

Certainly the simplest possible integration of planning and execution is no integration at all. Consider figure 6.24, in which there are only two software tiers. In such navigation architectures, the executive does not have a planner at its disposal, but must contain a priori all relevant schemes for traveling to desired destinations.

The strategy of leaving out a planner altogether is of course extremely limiting. Moving such a robot to a new environment demands a new instantiation of the navigation system,

and so this method is not useful as a general solution to the navigation problem. However such robotic systems do exist, and this method can be useful in two cases:

Static route-based applications. In mobile robot applications where the robot operates in a completely static environment using a route-based navigation system, it is conceivable that the number of discrete goal positions is so small that the environmental representation can directly contain paths to all desired goal points. For example, in factory or warehouse settings, a robot may travel a single looping route by following a buried guidewire. In such industrial applications, path-planning systems are sometimes altogether unnecessary when a precompiled set of route-based solutions can be easily generated by the robot program- mers. The Chips mobile robot is an example of a museum robot that also uses this architec- ture (118). Chips operates in a unidirectional looping track defined by its colored landmarks. Furthermore, it has only twelve discrete locations at which it is allowed to stop. Due to the simplicity of this environmental model, Chips contains an executive layer that directly caches the path required to reach each goal location rather than a generic map with which a path planner could search for solution paths.

Extreme reliability demands. Not surprisingly, another reason to avoid on-line planning is to maximize system reliability. Since planning software can be the most sophisticated portion of a mobile robot's software system, and since in theory at least planning can take time exponential to the complexity of the problem, imposing hard temporal constraints on successful planning is difficult if not impossible. By computing all possible

solutions off- line, the industrial mobile robot can trade versatility for effective constant-time planning (while sacrificing significant memory of course). A real-world example of off-line plan- ning for this reason can be seen in the contingency plans designed for space shuttle flights. Instead of requiring astronauts to problem-solve on-line, thousands of conceivable issues are postulated on Earth, and complete conditional plans are designed and published in advance of the Shuttle flights. The fundamental goal is to provide an absolute upper limit on the amount of time that passes before the astronauts begin resolving the problem, sacrificing a great deal of ground time and paperwork to achieve this performance guarantee.

6.3.4.2 Episodic planning

The fundamental information-theoretic disadvantage of planning off-line is that, during run-time, the robot is sure to encounter perceptual inputs that provide information, and it would be rational to take this additional information into account during subsequent exe- cution. Episodic planning is the most popular method in mobile robot navigation today because it solves this problem in a computationally tractable manner.

As shown in figure 6.25, the structure is three-tiered as in the general architecture of figure 6.23. The intuition behind the role of the planner is as follows. Planning is compu-

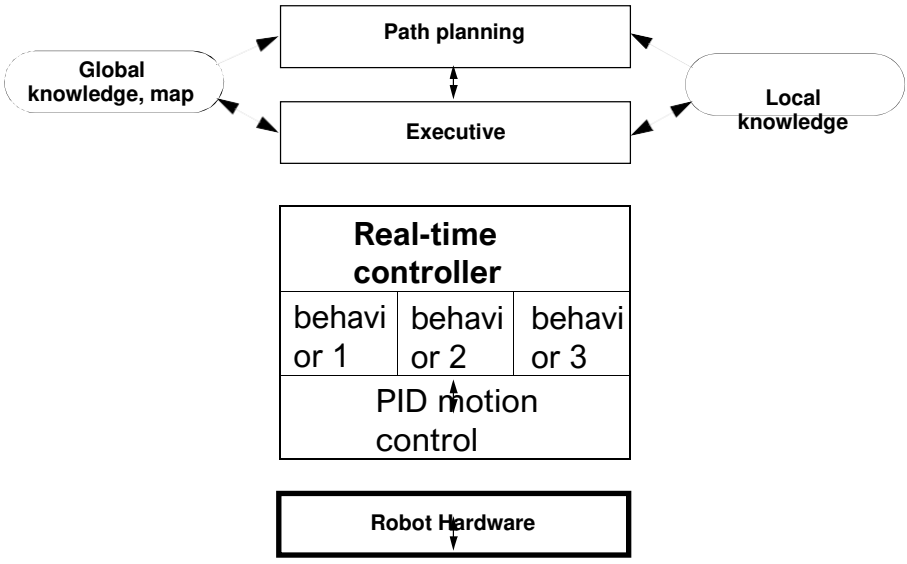


Figure 6.25

A three-tiered episodic planning architecture.

tationally intensive, and therefore planning too frequently would have serious disadvan- tages. But the executive is in an excellent position to identify when it has encountered enough information (e.g., through feature extraction) to warrant a significant change in strategic direction. At such points, the executive will invoke the planner to generate, for example, a new path to the goal.

Perhaps the most obvious condition that triggers replanning is detection of a blockage on the intended travel path. For example, in [129] the path-following behavior returns fail- ure if it fails to make progress for a number of seconds. The executive receives this failure notification, modifies the short-term occupancy grid

representation of the robot's surround- ings, and launches the path planner in view of this change to the local environment map.

A common technique to delay planning until more information has been acquired is called *deferred planning*. This technique is particularly useful in mobile robots with dynamic maps that become more accurate as the robot moves. For example, the commer- cially available Cye robot can be given a set of goal locations. Using its grassfire breadth- first planning algorithm, this robot will plot a detailed path to the closest goal location only and will execute this plan. Upon reaching this goal location, its map will have changed based on the perceptual information extracted during motion. Only then will Cye's execu- tive trigger the path planner to generate a path from its new location to the next goal loca- tion.

The robot Pygmalion implements an episodic planning architecture along with a more sophisticated strategy when encountering unforeseen obstacles in its way [36, 122]. When the lowest-level behavior fails to make progress, the executive attempts to find a way past the obstacle by turning the robot 90 degrees and trying again. This is valuable because the

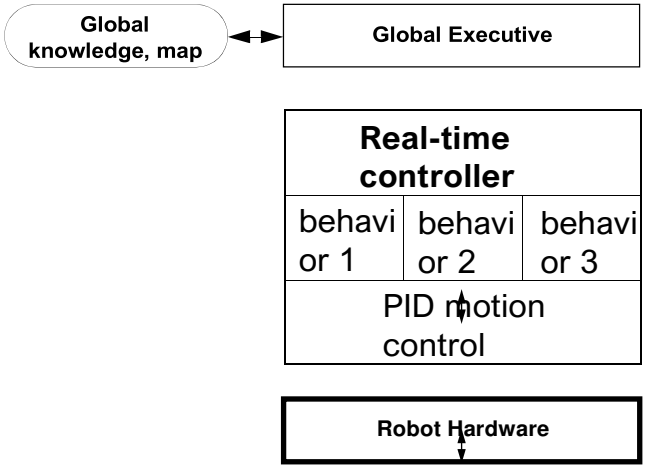


Figure 6.26

An integrated planning and execution architecture in which planning is nothing more than a real-time execution step (behavior).

robot is not kinematically symmetric, and so servoing through a particular obstacle course may be easier in one direction than the other.

Pygmalion's environment representation consists of a continuous geometric model as well as an abstract topological network for route planning. Thus, if repeated attempts to clear the obstacle fail, then the robot's executive will temporarily cut the topological con- nection between the two appropriate nodes and will launch the planner again, generating a new set of waypoints to the goal. Next, using recent laser rangefinding data as a type of local map (see figure 6.25), a geometric path planner

will generate a path from the robot's current position to the next waypoint.

In summary, episodic planning architectures are extremely popular in the mobile robot research community. They combine the versatility of responding to environmental changes and new goals with the fast response of a tactical executive tier and behaviors that control real-time robot motion. As shown in figure 6.25, it is common in such systems to have both a short-term local map and a more strategic global map. Part of the executive's job in such dual representations is to decide when and if new information integrated into the local map is sufficiently nontransient to be copied into the global knowledge base.

6.3.4.3 Integrated planning and execution

Of course, the architecture of a commercial mobile robot must include more functionality than just navigation. But limiting this discussion to the question of *navigation* architectures leads to what may at first seem a degenerate solution.

The architecture shown in figure 6.26 may look similar to the off-line planning architec- ture of figure 6.24, but in fact it is significantly more advanced. In this case, the planner tier has disappeared because there is no longer a temporal decomposition between the executive

and the planner. Planning is simply one small part of the executive's nominal cycle of activities.

The idea of speeding up planning to the point that its execution time is no longer signif- icant may seem wishful. However, using specific algorithms in particular environments, such reductions in the cost of planning have been demonstrated. Consider the work of Stentz [139]. Stentz designed a mobile robot control architecture for a large off-road vehi- cle traveling over partially known terrain at high speeds. Using advanced caching tech- niques from computer science, Stentz optimized a grassfire path-planning algorithm called *D** so that global path planning would be possible *within* the basic control loop of the exec- utive.

The result, depicted in figure 6.26, is an architecture in which the local and global representations are the same, and in which the executive has all global planning functionality required for the problem built in. The advantage of this approach is that the robot's actions at every cycle are guided by a global path planner, and are therefore optimal in view of all of the information the robot has gathered. Of course, the method is computationally chal- lenging and will not be practical in more complex environments until processor speeds increase even further. It also has basic limits of applicability as the size of the environment increases, but this has not yet been a barrier when applying this method to real-world sce- nario sizes.

The somewhat recent success of an integrated planning and execution method, *D**, underlines the fact that the designer of a robot navigation architecture must consider not only all aspects of the robot and its environmental task but must also consider the state of processor and memory technology. We expect that mobile robot architecture design is sure to remain an active area of innovation for years to come. All forms of technological progress, from robot sensor inventions to microprocessor speed increases, are likely to cat- alyze new revolutions in mobile robot architecture as previously unimaginable tactics become realities.

Chapter 1 Introduction

At the base of most complex tasks in mobile robotics lies a hard perception problem. The correct interpretation of raw sensor data is often a crucial part when one aims at applications in the real world. A robot must be able to understand its surrounding, in order to work in it and interact with it. Most people would only call a robot intelligent, if it is able to perceive what is happening around it. Perception is a prerequisite for the ability to take only suitable actions and to react to changes. Without appropriate sensors a robot is very restricted in what it can achieve and is only able to work at very specific tasks. In highly controlled environments, like they exists for automated assembly, external sensors might be omitted. If all required parts are always placed at the same locations, a robotic arm can perform tasks in a scripted open- loop fashion. But without such an environment that is specifically designed for the robot, it is indispensable to equip a robot with appropriate sensors and provide it with the means to understand their input. The topic of this thesis is therefore the interpretation of low-level sensor

information and its application in high-level tasks.

The most popular sensors in robotics are arguably visual cameras and range sensors. Cam- eras provide rich information about the environment and have proven their potential in the scientific literature. In addition to that, human beings are able to solve a myriad of tasks based on vision. They have the advantage of being passive sensors with theoretically infinite range. Yet, visual information also has serious shortcomings. The data typically depends strongly on the illumination and spatial information can only be acquired indirectly because of the miss- ing depth information. Range sensors, on the other hand, provide accurate spatial information about the environment. Their measurements are often independent of the illumination of the environment since they typically are active sensors. Laser scanners, for example, send out nu- merous pulses of infrared light in different directions and determine the distances to the closest obstacles on the beams, based on the time it takes the light to return to the receiver. While this means that such sensors have a very restricted range, typically less than 100 m, much less prior knowledge is necessary to reason about the acquired data since it directly represents the geom- etry of the surrounding. This is an important property in navigation tasks, which, by nature, are elementary for mobile platforms. In this work, we mainly focus on 3D perception, meaning the analysis and interpretation of 3D range scans. We discuss several problems in the area of 3D perception and present solutions that are based on the extraction of features from the raw sensor data, which can then be used efficiently to solve the task at hand. “Feature” is a very broad term in this context. It means that we extract some compressed but useful information from the input data, which is easier to interpret than the raw measurements. We can then build upon these features to address complex 3D perception problems. During this thesis, we intro- duce several innovative algorithms and techniques that advance the state of the art in robotic

perception. We elaborate systems that enable us to address complex 3D perception problems and focus on realistic experimental evaluations with real world data.

This thesis is organized as follows: In the remainder of this chapter, we summarize our sci- entific contributions and present open-source software resulting from our work. In Chapter 2, we introduce some basic knowledge that is elementary for the remaining content. The subse- quent chapters then address specific perception problems in the context of mobile robotics.

In Chapter 3, we discuss the benefits of point feature extraction, meaning the detection of interesting areas (keypoint detection) and the description of such areas in a condensed fashion (descriptor calculation). Such features are used for similarity checks between chunks of data and are applicable to numerous purposes. We introduce an approach for point feature extraction from 3D range data, namely the Normal Aligned Radial Feature (NARF). See Figure 1.1(a) for a visual motivation.

The subsequent chapters (Chapter 4 to Chapter 7) present systems that build upon the NARF technology to solve different perceptual problems that all require an efficient way to find similar structures in 3D range scans. Finding such similarities is necessary to establish connections between parts of the scans. In Chapter 4, we describe an object recognition system that finds the positions of known objects in 3D scenes. Using point feature correspondences, the system determines similar areas between a scene and a point cloud model of an object. Based on the correspondences, our method calculates potential positions of the object in the scene. The candidate positions are then evaluated based on spatial verification. This means that our system compares the actual perception with what it expects to see and calculates a score based on how well that fits. See Figure 1.1(b) for a visual motivation. In Chapter 5, we introduce an approach for unsupervised model learning. We elaborate how we can build point cloud object models in an automated fashion. Our system only needs an unordered collection of scans that show different perspectives of one or multiple objects. See Figure 1.1(c) for a visual motivation. Chapter 6 presents a method for environment modeling based on surface primitives. Our system is able to represent 3D scenes based on a small dictionary of surface patches. This follows the observation that typical 3D environments consist of recurring structures, like flat surfaces, edges, corners, etc.. We learn a dictionary of such structures and use it to represent the scene, leading to a semantically rich representation, with low memory requirements. See Figure 1.1(d) for a visual motivation. In Chapter 7, we descuss a system for place recognition. This system matches a query scan against a database of scans and determines if the scan comes from a new location or one already in the database. In the latter case it also determines the relative pose between the scans. See Figure 1.1(e) for a visual motivation.

Up to this point, the mentioned approaches all require a method to detect similar structures in homogeneous sensor data, specifically to find corresponding areas between 3D range scans. In contrast to this, we describe a system that needs to find correspondences between heterogeneous data types, specifically between 3D range data and visual images, in Chapter 8. We present an approach for mapping under consideration of aerial images, meaning we discuss how one can track the position of a robot in an aerial image and use this information to build highly accurate, globally consistent maps. See Figure 1.1(f) for a visual motivation. In addition, we discuss how such maps can be used to evaluate SLAM approaches.

In Chapter 9, we introduce a complete robotic system in form of a pedestrian assistant robot, designed to navigate in highly populated urban environments. The system is built up from a multitude of different modules, to each of which apply high robustness requirements to ensure the robustness of the entire system. Our core contribution to this system is a module that analyzes the low-level sensor data and provides high-level information to the other modules, specifically, it performs a traversability analysis and obstacle detection in 2D and 3D laser data.

See Figure 1.1(g) for a visual motivation. We conclude this work in Chapter 10 and give an outlook on future work.

1.1 Scientific Contributions

This thesis describes several contributions to the area of robotic perception based on 3D range data. In summary, we propose

- a novel point feature type for 3D range scans. The Normal Aligned Radial Feature (NARF) provides a method for keypoint extraction and descriptor calculation (Chapter 3);
- an approach for object recognition in 3D range scans, based on existing point cloud mod- els of objects (Chapter 4);
- a new method to learn point cloud models of objects from an unordered collection of 3D range scans in an unsupervised fashion, whereas our main contribution is the detection and evaluation of overlapping areas, which is essential for building consistent models (Chapter 5);
- a system for 3D environment modeling based on a dictionary of surface primitives, which can be learned either on the same or on another collection of scans. Our main contribution here is the efficient detection of similar structures (Chapter 6);
- a technique for place recognition based on 3D range data, which calculates a score for a query scan and potential matches in the database, together with the relative transformation (Chapter 7);
- a SLAM algorithm, which incorporates aerial images as prior information, whereas our main contribution was in the matching of the sensor data against the aerial image. In this context we also present a metric to compare the performance of different SLAM approaches (Chapter 8);
- an existing, real world navigation system for a robotic pedestrian assistant, whereas our main contribution is in the traversability analysis and obstacle detection (Chapter 9).

1.2 Contributions to Open-Source Software

We released several of the algorithms described in this thesis as open-source software. We believe that open-source software is a great possibility to enable realistic comparisons between different approaches and to speed up the research process since the wheel does not have to be invented again and again by different researchers. We invite others to use our software, to extend it, and to adapt it to their needs.

- Contributions to the Point Cloud Library (PCL)
  http://www.pointclouds.org
  We contributed different classes to this open source library. Mainly the following parts:
  - The range image class: We provide as class that implements range images (see Section 2.3.2) and implements specific functionalities for them.

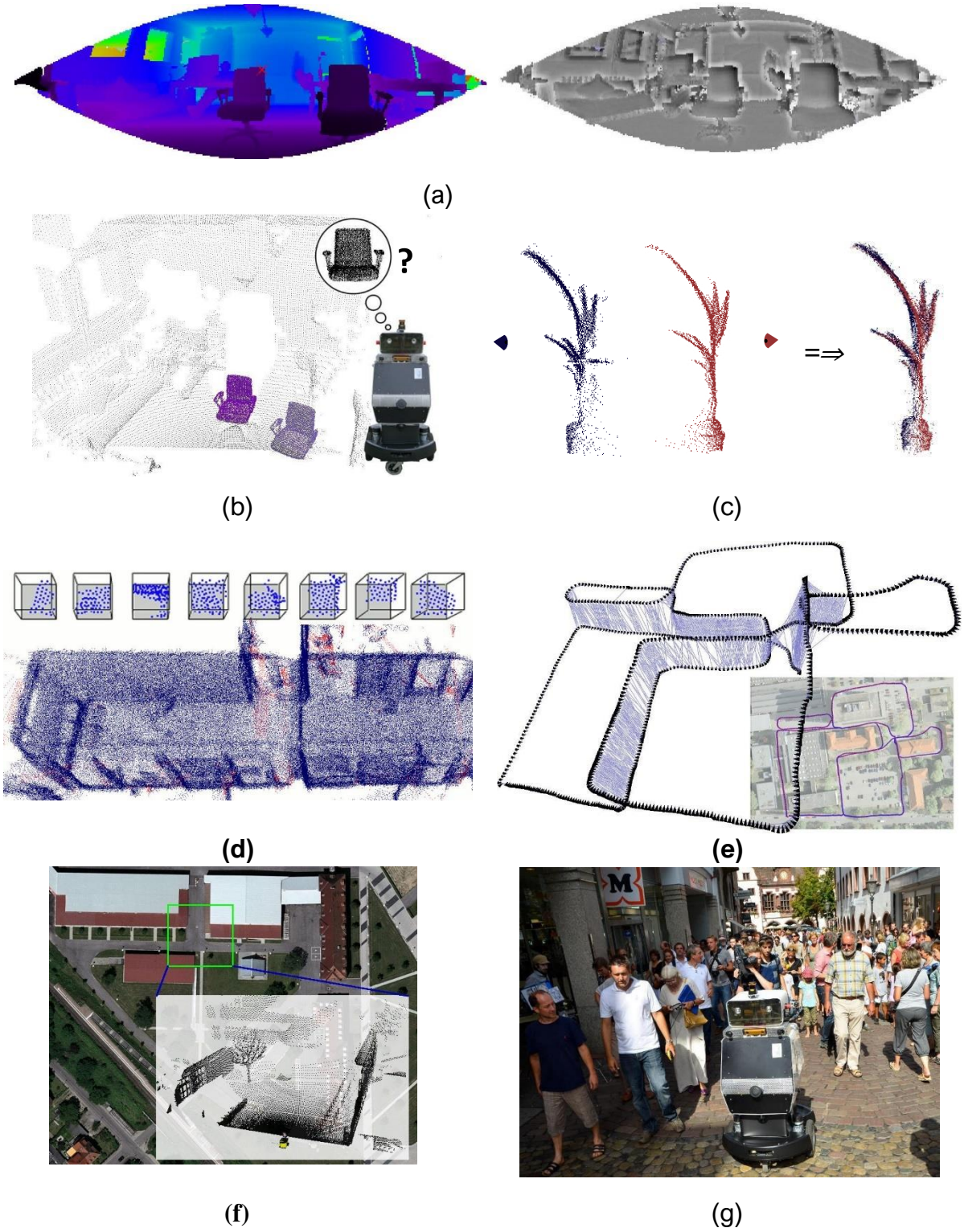


Figure 1.1: Motivational images for the different chapters of this thesis. **(a)**: On the left we marked the top right corner of a chair. On the right, the similarity values of all other points according to our feature descriptor are shown, from least similar (white) to most similar (black). Upper right rectangular corners are clearly distinguished. **(b)**: Object recognition. A robot looking for the model of a chair in a point cloud of an office scene. **(c)**: Unsupervised model learning. Two different views of a plant were found in different scans and automatically merged into a new model. **(d)**: Modeling with surface primitives. A point cloud of a corridor, built up from a collection of surface patches, whereas eight of these are shown on the top. **(e)**: Place recognition. The trajectory of a dataset of 3D scans on a Google Earth (c Google) image, overlain with loop closures

found by our system. **(f)**: Mapping under consideration of aerial images. The robot matches its current 3D scan against the aerial image (c Google) to determine its position. **(g)**: Our pedestrian assistant robot moving in a crowded inner city area.

  - The range image border extractor: This class finds traversals from foreground to background in range images. See Section 3.1.
  - The NARF keypoint detector: This class extracts NARF keypoints from range im- ages. See Section 3.2.
  - The NARF descriptor: This class calculates NARF descriptors for points in range images. See Section 3.3.

  This software was initially developed in collaboration with Radu B. Rusu, Kurt Konolige, and Wolfram Burgard. Since then, a number of PCL developers provided patches and bug fixes, for which we are grateful.
- SLAM evaluation toolkit
  http://ais.informatik.uni-freiburg.de/slamevaluation
  We created this tool to enable other researchers to evaluate the accuracy of their SLAM algorithms, given a set of manually verified relations. See Section 8.3.2 for additional details on the metric used for the evaluation. Next to the software library, we offer such relations for some frequently used public datasets on the website and also provide a num- ber of complete datasets we captured ourselves. This framework is a joint effort with Rainer Kümmerle, Christian Dornhege, Michael Ruhnke, Giorgio Grisetti, Cyrill Stach- niss, Alexander Kleiner, and Wolfram Burgard.

1.3 Publications

The content of this thesis is based on papers published in international journals and confer- ence proceedings and one patent. In the remainder of this section, these publications are listed chronologically.

Journal Articles:

- R. Kümmerle, B. Steder, C. Dornhege, A. Kleiner, G. Grisetti, and W. Burgard. Large scale graph-based SLAM using aerial images as prior information. *Journal of Auto- nomous Robots*, 30(1):25–39, 2011
- R. Kümmerle, B. Steder, C. Dornhege, M. Ruhnke, G. Grisetti, C. Stachniss, and A. Klei- ner. On measuring the accuracy of SLAM algorithms. *Journal of Autonomous Robots*, 27(4):387–407, 2009

Patents:

- Y. Asahara, T. Yamamoto, M. Van Loock, B. Steder, G. Grisetti, and W. Burgard. Object recognition method, object recognition apparatus, and autonomous mobile robot, 2012. (PATENT)

Conferences, Workshops, and Book Chapters:

- R. Kümmerle, M. Ruhnke, B. Steder, C. Stachniss, and W. Burgard. A navigation system for robots operating in crowded urban environments. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2013. Accepted for Publication

- M. Ruhnke, B. Steder, G. Grisetti, and W. Burgard. 3D environment modeling based on surface primitives. In *Towards Service Robots for Everyday Environments*, pages 281–

  300. Springer Berlin/Heidelberg, 2012

- B. Steder, M. Ruhnke, S. Grzonka, and W. Burgard. Place recognition in 3D scans using a combination of bag of words and point feature based relative pose estimation. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2011

- B. Steder, R. B. Rusu, K. Konolige, and W. Burgard. Point feature extraction on 3D range scans taking into account object boundaries. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2011

- B. Steder, R. B. Rusu, K. Konolige, and W. Burgard. NARF: 3D range image features for object recognition. In *Workshop on Defining and Solving Realistic Perception Problems in Personal Robotics at the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2010

- M. Ruhnke, B. Steder, G. Grisetti, and W. Burgard. Unsupervised learning of compact 3D models based on the detection of recurrent structures. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2010 **Finalist Best Paper Award**

- B. Steder, G. Grisetti, and W. Burgard. Robust place recognition for 3D range data based on point features. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2010

- B. Steder, G. Grisetti, M. Van Loock, and W. Burgard. Robust on-line model-based object detection from range images. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2009

- W. Burgard, C. Stachniss, G. Grisetti, B. Steder, R. Kümmerle, C. Dornhege, M. Ruhnke, A. Kleiner, and J. D. Tardós. A comparison of SLAM algorithms based on a graph of relations. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2009

- R. Kümmerle, B. Steder, C. Dornhege, A. Kleiner, G. Grisetti, and W. Burgard. Large scale graph-based SLAM using aerial images as prior information. In *Proc. of Robotics: Science and Systems (RSS)*, 2009

- M. Ruhnke, B. Steder, G. Grisetti, and W. Burgard. Unsupervised learning of 3D object models from partial views. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2009

This thesis does not report on the following publications, which were written during my time as a research assistant:

- Journal Articles
  - B. Steder, G. Grisetti, C. Stachniss, and W. Burgard. Visual SLAM for flying vehi- cles. *IEEE Transactions on Robotics*, 24(5):1088–1093, 2008

- Conferences and Workshops

  - B. Steder, G. Grisetti, S. Grzonka, C. Stachniss, and W. Burgard. Estimating con- sistent elevation maps using down-looking cameras and inertial sensors. In *Proc. of the Workshop on Robotic Perception on the Int. Conf. on Computer Vision Theory and Applications*, 2008
  - B. Steder, G. Grisetti, S. Grzonka, C. Stachniss, A. Rottmann, and W. Burgard. Learning maps in 3D using attitude and noisy vision sensors. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2007
  - B. Steder, A. Rottmann, G. Grisetti, C. Stachniss, and W. Burgard. Autonomous navigation for small flying vehicles. In *Workshop on Micro Aerial Vehicles at the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2007

1.4 Collaborations

During the creation of this thesis, we collaborated with a number of other researchers. We will now summarize these collaborations for the individual chapters of this work.

- Chapter 3: The system presented in this chapter is the result of a summer internship at Willow Garage, Inc (CA, USA). It was created in collaboration with Radu B. Rusu, Kurt Konolige, and Wolfram Burgard. See [130, 131] for related publications.

- Chapter 4: An initial implementation of the system presented in this chapter was the result of a collaboration with Giorgio Grisetti, Mark Van Loock, and Wolfram Burgard. It was published in [5] and [127]. Yet, the system and results we present in this chapter are based on a new implementation, which only shares some basic concepts with the original one. These results, which are the outcome of a collaboration with Giorgio Grisetti, Radu
  B. Rusu, Kurt Konolige, and Wolfram Burgard, were previously unpublished.

- Chapter 5: The system presented in this chapter is the result of a collaboration with Michael Ruhnke, Giorgio Grisetti, and Wolfram Burgard. The work was done in the context of Michael Ruhnke's diploma thesis, which was co-supervised by the author of this PhD thesis. We present the whole system for completeness, yet our main contribution of Chapter 5 is described in Section 5.1.4. While the author of this PhD thesis took part in the creation of the whole system and supported the work, the remaining content of this chapter was mostly handled by others. See [104] for the related publication.

- Chapter 6: The system presented in this chapter is the result of a collaboration with Michael Ruhnke, Giorgio Grisetti, and Wolfram Burgard. While we present the whole system for completeness, only Section 6.3 can be claimed as work been done in the con- text of this thesis. The author of this thesis took part in the

creation of the whole system and supported the work, yet the remaining content of this chapter was mostly handled by others. See [105, 106] for related publications.

- Chapter 7: The system presented in this chapter is the result of a collaboration with Giorgio Grisetti, Michael Ruhnke, Slawomir Grzonka, and Wolfram Burgard. See [123, 129] for related publications.

- Chapter 8: The system presented in this chapter is the result of a collaboration with Rainer Kümmerle, Christian Dornhege, Alexander Kleiner, Giorgio Grisetti, and Wolfram Burgard. While we present the whole system for completeness, only Section 8.1.3 and Section 8.1.4 can be claimed as work been done in the context of this thesis. The author of this thesis took part in the creation of the whole system, preparing and performing the experiments, and general support of the work, yet the remaining content of this chapter was mostly handled by others. See [17, 72, 73, 74] for related publications.
- Chapter 9: The system presented in this chapter is the result of a collaboration with Rainer Kümmerle, Michael Ruhnke, Cyrill Stachniss, and Wolfram Burgard. It describes work that was done in the in the context of an EU project (FP7-231888-EUROPA). While we present the whole navigation system for completeness, only Section 9.2.3 can be claimed as work been done in the context of this thesis. The author of this thesis took part in the creation of the whole system, preparing and performing the experiments, and general support of the work, yet the remaining content of this chapter was mostly handled by others. See [71] for the related publication.

Chapter 2

Basics and Notations

In this chapter, we present the notations and basic concepts used throughout the remainder of this work. We describe mathematical notations in Section 2.1, abbreviations in Section 2.2, some commonly used data structures in Section 2.3, and give details about SLAM in Section 2.4.

2.1 Mathematical Notations

In the remainder of this thesis we will use the following notations in mathematical equations:

Example(s)	Type	Description of notation
$x = 1.0$, $\delta = 0.3$ $\alpha = 12^\circ$	Scalar	Lower case latin and greek letters, with α, β, and γ reserved for angles
$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \end{bmatrix}$	Vector	Bold lower case latin letters
$R = \begin{bmatrix} x_{11} & x_{12} & \dots \\ x_{21} & x_{22} & \dots \\ \vdots & \vdots & \ddots \end{bmatrix}$	Matrix	Upper case latin and greek letters
$S = \{\mathbf{x}_1, \mathbf{x}_2, \dots\}$	Set	Upper case calligraphic letters
$\mathrm{M} = \{M_1, M_2, \dots\}$ $\mathrm{P} = \{P_1, P_2, \dots\}$	Set of sets	Fraktur symbols
$\lvert x \rvert$	Absolute value	A scalar enclosed by $\lvert\ \rvert$
$\lvert\mathbf{x}\rvert = \begin{vmatrix} x_1 \\ x_2 \\ \vdots \end{vmatrix} = 2$	Size	A set or a vector enclosed by $\lvert\ \rvert$
$\lVert\mathbf{x}\rVert_p = \sqrt[p]{\sum_{i=1}^{\lvert\mathbf{x}\rvert} \lvert x_i \rvert^p}$ $\lVert\mathbf{x}\rVert_1 = \sum_{i=1}^{\lvert\mathbf{x}\rvert} \lvert x_i \rvert$ $\lVert\mathbf{x}\rVert_2 = \sqrt{\sum_{i=1}^{\lvert\mathbf{x}\rvert} x_i^2}$	L^p-norm	A vector enclosed by $\lVert\ \rVert_p$, whereas $\lVert\ \rVert_1$ is the Manhattan norm and $\lVert\ \rVert_2$ is the Euclidean norm

2.2 Abbreviations

We will use the following abbreviations in this thesis:

Abbreviation	Meaning	Brief description
BIC	**B**ayesian **I**nformation **C**riterion	A decision criterion for model learning methods
BoW	**B**ag **o**f **W**ords	An approach that describes data compactly based on a dictionary
g2o	**G**eneral **G**raph **O**ptimizer	An open source software for graph optimization problems
CAD	**C**omputer-**A**ided **D**esign	CAD programs are typically used to create 3D models
EUROPA	**EU**ropean **RO**botic **P**edestrian **A**ssistant	An EC-funded research project
GOODSAC	**GOOD SA**mple **C**onsensus	A sample consensus algorithm
GPS	**G**lobal **P**ositioning **S**ystem	A system for global localization based on satellite signals
ICP	**I**terative **C**losest **P**oint	A registration algorithm
IMU	**I**nertial **M**easurement **U**nit	A sensor measuring orientations
MCL	**M**onte **C**arlo **L**ocalization	A localization method based on a particle filter
NARF	**N**ormal **A**ligned **R**adial **F**eature	A 3D point feature type
NARVP	**N**ormal **A**ligned **R**ange **V**alue **P**atch	A 3D descriptor vector
PCA	**P**rincipal **C**omponent **A**nalysis	A method to find eigenvalues and eigenvectors for a matrix
PCL	The **P**oint **C**loud **L**ibrary	An open source software for point cloud processing
RANSAC	**RAN**dom **SA**mple **C**onsensus	A sample consensus algorithm
RBPF	**R**ao-**B**lackwellized **P**article **F**ilter	A filtering method applicable to SLAM
RGB-D	**R**ed, **G**reen, **B**lue, and **D**epth	RGB-D sensors capture color and range images
SIFT	**S**cale-**I**nvariant **F**eature **T**ransform	A 2D point feature type
SLAM	**S**imultanious **L**ocalization **A**nd **M**apping	A mapping procedure for mobile robots

SURF	**S**peeded **U**p **R**obust **F**eature	A 2D point feature type

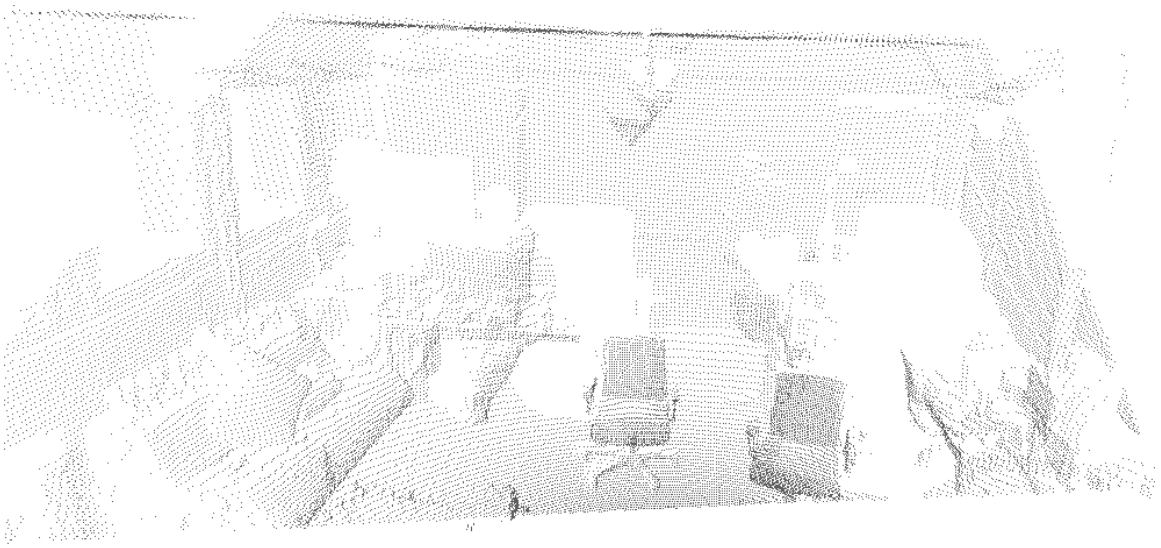


Figure 2.1: Visualization of an example point cloud showing an office scene.

2.3 Data Structures

The following data structures are frequently used in this work.

2.3.1 Point Clouds

Point clouds are a very common data structure to describe a set of unordered 3D points:

$$P = \{\mathbf{p}_1, \mathbf{p}_2, \ldots\} \quad \text{with } \mathbf{p} = \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} \quad \forall\, \mathbf{p} \in P \wedge p_x, p_y, p_z \in \mathbb{R} \tag{2.1}$$

See Figure 2.1 for a visualization of an example point cloud showing an office scene.

2.3.2 Range Images

Range images, also called *depth images*, are 2D representations of 3D scenes. Every pixel in a range image captures the distance to the closest obstacles as seen from one point in space, whereas the pixel position determines the direction of the beam. See Figure 2.2 for an example range image. There are different projection models to create a range image. They can be based on a projection plane, like normal visual camera images, but these lack the possibility to represent scenes that cover more than 180°. Therefore, spherical projections are popular, like they are used for maps showing the whole globe. Here one important parameter of the range image is the angular resolution α, whereas $\alpha = 1°$ means that the angle between the beams represented by neighboring pixels is 1°.

If a 3D scan is captured from one point in space, i.e., the sensor does not move while the 3D points are generated, a range image is a very good representation of the acquired data. It closely resembles the original sensor data and preserves not only information about the 3D end points, but also about the measured free space, maximum range readings, and unknown space. Every point in a range image also represents a 3D point, which is why range images can interchangeably be use as 2D images and 3D point

clouds. Since computer vision is a very successful research area, it is reasonable to transfer concepts to the area of range-based

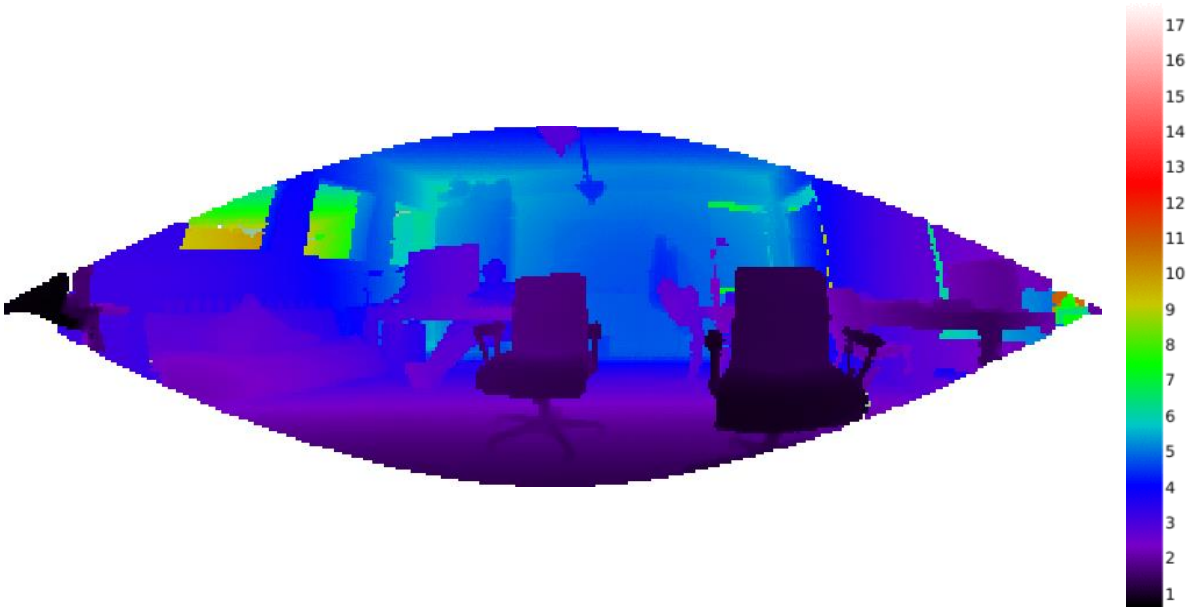


Figure 2.2: Visualization of an example range image representing the same scene as shown in Figure 2.1. The legend on the right shows which color corresponds to a specific measured range in meters. Every

pixel represents a beam going from the known sensor position in a direction that is defined by the position of the pixel in the image. The range image in this example has an angular resolution of $\alpha = 0.5^{\circ}$. The 3D endpoint can be calculated from the known beam and the measured range.

3D perception. Range images offer this possibility while still keeping the advantages of range scans. A range image can be calculated efficiently from a point cloud using a *z*-buffer [41], as long as the original sensor position is known.

An implementation of a range image class which was written in the context of this thesis is available under an open source license. See Section 1.2 for a list of contributions to open source software.

2.4 SLAM

SLAM means Simultaneous Localization and Mapping and describes one of the basic problems in robotics, namely how to use a mobile robot and its sensors to create a model of the environ- ment. One can distinguish two sub-elements in this context, *mapping with known poses* and *localization*. Mapping with known poses addresses the problem, how the sensor information can be used to create a model of the environment under the assumption that the position of the robot is known for all time steps. Localization is the ability to determine the position of the robot given a model of the environment. Even if these sub-parts are solved, SLAM remains a hard problem since the two elements mutually depend on each other. To integrate new sensor data into the model, the robot needs to know its position, but to determine its position, it needs the model.

There exists a number of different approaches to address the SLAM problem.

Approaches based on different variants of Kalman filters are very popular [39, 77, 120, 137] because they estimate a fully correlated posterior about landmark maps and robot poses. Unfortunately, they make strong assumptions about the motion model and the sensor noise. If these are violated, the filters tend to diverge [66, 144]. Particle filters [50, 90] are another possibility, where the probability distribution is approximated by a limited number of samples (particles). While

such approaches can closely approximate arbitrary distributions for large numbers of particles, their runtime requirements grow linearly with this number which makes them computation- ally demanding in complex applications. In recent years, least square error minimization ap- proaches [42, 51, 55, 70, 84, 96] have become very popular. These approaches try to optimize an error function for spatial constraints between robot poses to acquire a maximum likelihood trajectory. They have some relevance for the content of this work, mainly for Chapter 7, Chap- ter 8, and Chapter 9. We will therefore present their basic concepts in the following section.

2.4.1 Graph-based SLAM

One possibility to describe a SLAM problem is the so called *graph-based SLAM* formulation. Here, the individual robot poses of a trajectory are represented as nodes in a graph and the edges between the nodes represent spatial constraints between them. In general, these spatial constraints either represent incremental motions (odometry, scan matching, etc.) or detected *loop closures*. Loop closures are defined as multiple traversals of the same area and can be detected from overlapping observations. They are necessary to counter the error accumulation of the purely incremental constraints. Under this graph-based formulation, a solution to the SLAM problem is a configuration of the nodes which minimizes the error introduced by the constraints, thereby computing the configuration of the nodes that is most likely. Let $\mathbf{x}_i$ be node i in a graph, whereas each node contains the parameters to represent a robot pose. Let $\mathbf{z}_{ij}$ and Ω_{ij} be the mean and the information matrix of an observation, which describe a mea- surement of node j seen from node i. Let $\mathbf{e}(\mathbf{x}_i, \mathbf{x}_j, \mathbf{z}_{ij})$ be a function that computes the error between the current measurement $\mathbf{z}_{ij}$ and the expected measurement based on $\mathbf{x}_i$ and $\mathbf{x}_j$. We now define an error function for the complete graph configuration $\mathbf{x}$ as

$$\chi(\mathbf{x}) = \sum_{i,j} \mathbf{e}(\mathbf{x}_i, \mathbf{x}_j, \mathbf{z}_{ij})^T \Omega_{ij} \mathbf{e}(\mathbf{x}_i, \mathbf{x}_j, \mathbf{z}_{ij}), \tag{2.2}$$

which leads to the following optimization problem:

$$\tilde{\mathbf{x}} = \operatorname*{argmin}_{\mathbf{x}} \chi(\mathbf{x}). \tag{2.3}$$

This means that we have to find a graph configuration $\tilde{\mathbf{x}}$ that minimizes the error of all the observations. This is equivalent to finding the maximum a-posteriori configuration given the
measurements = $\operatorname{argmax}_{\mathbf{x}} p(\mathbf{x}|\mathbf{z})$ since $\chi(\mathbf{x})$ is actually the corresponding negative log-
$\tilde{\mathbf{x}}$ likelihood.

In this work, we solve such optimization problems with one of the publicly available soft- ware libraries that were designed for this purpose. Specifically we use the *General Graph Optimizer* (g^{2}o) [70].

Chapter 3

NARF: A Point Feature for 3D Range Data

The evaluation of similarity between different structures is highly relevant in many perception tasks since it is a prerequisite to recognize something as previously seen. In this chapter we address this topic by means of point feature extraction from 3D range data. We present the general concept of such features, their benefits, and how their application can typically be divided into keypoint extraction, descriptor calculation, and matching steps. Subsequently, we present a novel point feature de- veloped throughout this work, namely the Normal Aligned Radial Feature (NARF). It operates on range images and has the special property that it explicitly considers the boundaries of objects as a source of information. We present experiments in which we analyze the individual components with respect to their repeatability and matching capabilities. Following this chapter, we will present several systems that build upon this point feature type to address hard perception problems.

* * * *

Many perception tasks in the area of robotics, like object recognition, place recognition, map- ping, or localization, require the ability to find similar parts in different sets of sensor readings. A popular method is to extract *point features*, which describe a chunk of data in a form that can be used to efficiently perform comparisons between different regions. In 2D (e.g., photos) or 3D (e.g., range scans) perception, such features are usually *local* around a point. In this context this means that for a given point in the scene its immediate vicinity is used to determine the cor- responding feature. We will refer to these kinds of features as *point features*. The extraction and usage of point features from the raw sensor data can typically be divided into several subtasks. In the point feature extraction phase the system tries to find *keypoints*/*interest point*s. This describes the identification of informative (interesting) positions in the data that are useful as point features. These are areas with well defined positions that allow a robust detection under noisy conditions or sensor viewpoint changes. This means that a keypoint should be found in the same place again, if the corresponding area is reobserved. This condition is also known as *repeatability*. The next step is the calculation of *descriptor*s/*description vector*s. These describe the area around a keypoint in form of a set of numbers $\mathbf{d} = d_1, \ldots, d_n$ with $d_i \in \mathbb{R}$. This description should encode the information in the vicinity of the point in a way that allows for easy similarity checks between different descriptors. Therefore, the

description has to be as unique as possible, while still being robust to noise and changes in the sensor position. Last comes the matching phase. After the features have been extracted from the data, they can be used to find similar structures. For this purpose, the descriptors of different features can be compared in a pair-wise fashion. Typically the descriptors allow the *descriptor distance* to be

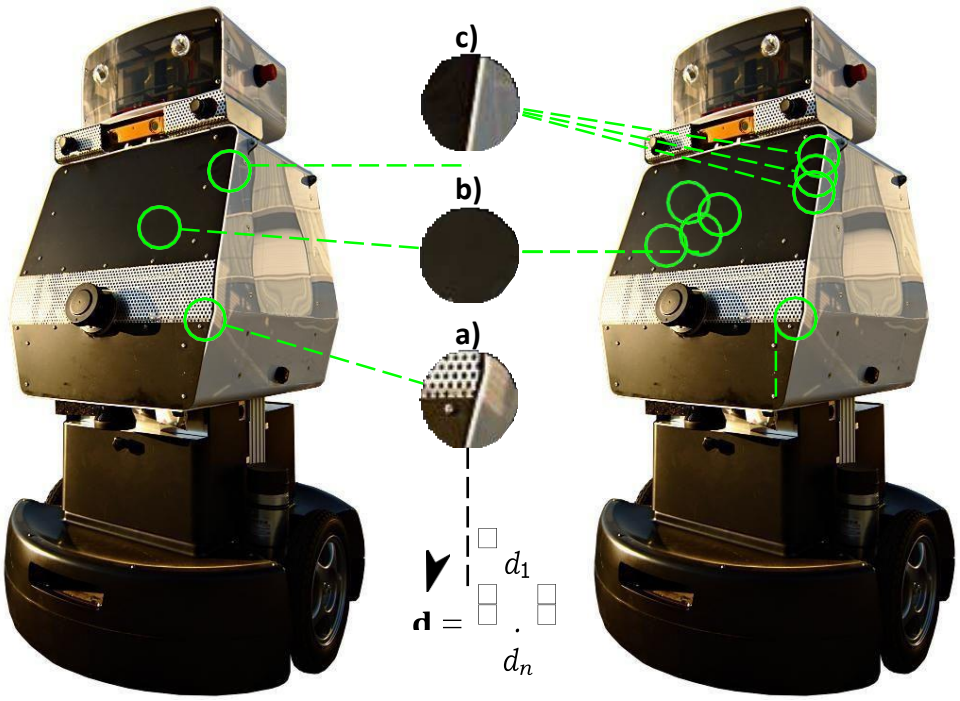


Figure 3.1: Left: Example image with marked feature positions. **Middle**: Scaled up versions of the areas covered by the features. **Right**: Examples for positions, where the features could be found because of a similar structure. Feature **a)** has a clear match. The descriptor **d**, which can be used to determine such a match, is calculated from the image patch surrounding the keypoint in this two dimensional example case. The structures **b)** and **c)** are not unique in their immediate neighborhood and therefore have multiple candidate matches.

calculated by means of a simple norm (e.g., Euclidean Distance or Manhattan Distance). A low descriptor distance implies a high similarity.

Figure 3.1 gives a visual example for the feature extraction on a 2D camera image. Only the structure marked with a) is a good keypoint since its position is clearly defined in a local neighborhood. The structure marked with b) can be moved around arbitrarily on the dark surface and will still lead to very similar descriptors. The structure marked with c) has a line structure which leads to the situation that its position is clearly defined in one dimension, but not in the other one. Slight movements along the line will lead to very similar descriptors. The property describing this is called *self-similarity*. Feature a) is the only one with a low self similarity, because even slight changes in its position will lead to very different descriptors.

Important advantages of keypoints are that they substantially reduce the search space and the computation time required to find correspondences between two scenes. In addition, they focus the computation on areas that are more likely relevant for the matching process.

In 2D computer vision, many successful point features have been developed in the past. Some of the most popular approaches in this area are the SIFT (Scale Invariant Feature Trans- form) keypoint detector and descriptor [83], the SURF (Speeded Up Robust Features) keypoint detector and descriptor [9], the FAST (Features from Accelerated Segment Test) keypoint de- tector [102], and more recently the BRIEF

(Binary Robust Independent Elementary Feature) descriptor [18] and the BRISK (Binary Robust Invariant Scalable Keypoints) keypoint detector and descriptor [79]. These and other methods have been used by researchers all over the world to detect known objects [83] or places [25] in images, create classifiers for object types [118] or scene types [24], implement visual SLAM systems [30], and many other tasks that require a

similarity measure for images or image parts. Important properties for point feature types are often *rotational invariance* and *scale invariance*, meaning that correspondences should also be found if the observations differ regarding the orientation and size of the patches. Yet, there are also applications where these properties are not desirable and should therefore be optional.

Instead of 2D vision data, we focus on feature extraction from 3D range scans in this work. Compared to cameras, 3D sensors provide depth information and are often less sensitive to lighting conditions (e.g., laser sensors). In addition, scale information is directly available. Yet, geometry alone is often less expressive in terms of object uniqueness. Compared to vision, there has been surprisingly little research for keypoint extraction in raw 3D data in the past. We will focus on single range scans, as obtained with 3D laser range finders or camera based depth sensors, where the data is incomplete and dependent on a viewpoint. We chose range images as the way to represent the data since they reflect this situation and enable us to adapt ideas from computer vision. See Section 2.3.2 for more details on range images.

In the remainder of this chapter, we present the Normal Aligned Radial Feature (NARF), a novel keypoint extraction method combined with a feature descriptor for points in 3D range data. The NARF extraction procedure has been designed with two specific goals in mind:

- The selected points are supposed to be in positions where the surface is stable (to ensure a robust estimation of the normal) and where there are sufficient changes in the immediate vicinity to ensure the repeatability of the extraction procedure.
- Since we focus on partial views, we want to make use of object boundaries, meaning the outer shapes of objects seen from a certain perspective. The outer forms are often rather unique so that their explicit use in the keypoint extraction and the descriptor calculation can be expected to make the overall process more robust. For this purpose, we also present a method to extract those boundaries.

The implementation of our method is available under an open-source license. See Section 1.2 for a list of contributions to open source software.

The remainder of this chapter is organized as follows. We will first introduce a method to find object boundaries in range images in Section 3.1. Subsequently, we will describe the NARF keypoint extraction method in Section 3.2 and the NARF descriptor in Section 3.3. Fol- lowing this, we will present experimental results in Section 3.4. We will discuss related work in Section 3.5 and conclude this chapter in Section 3.6.

3.1 Boundary Extraction on Range Images

One important requirement for our feature extraction procedure is the explicit handling of boundaries in the range data. Boundaries typically appear as non-continuous traversals from foreground to background. In this context there are mainly three different kinds of points that we are interested in detecting:

- *Object boundaries* are the outermost visible points still belonging to an object. These points are relevant for the feature extraction because they define the outer form of an object.
- *Shadow boundaries* are points in the background that adjoin occlusions. The

detection of these points is important since the feature extraction should be prevented from using them as relevant structure. Keypoints on the shadow boundaries are not useful because their position changes with the perspective of the sensor.

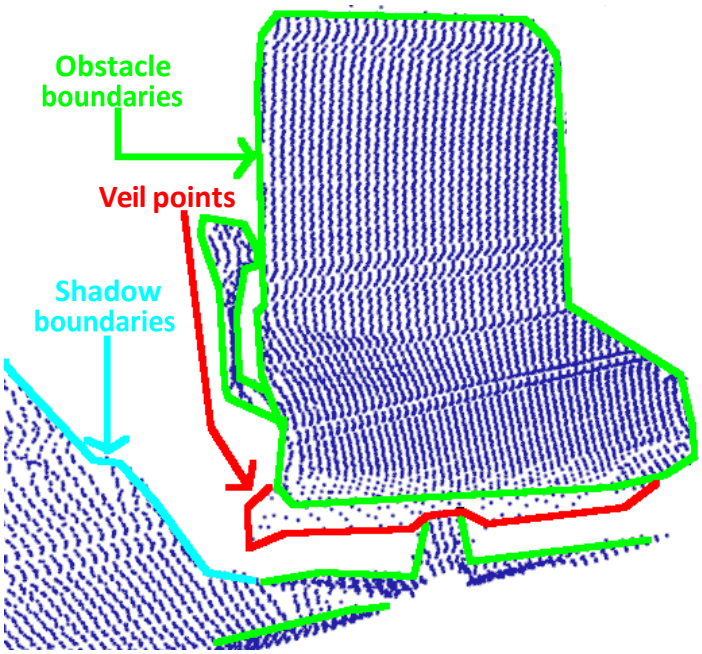


Figure 3.2: Visual example for the different kinds of boundary points in 3D range scans. Obstacle boundaries mark a traversal from foreground to background, whereas shadow boundaries mark a traversal from background to foreground. Veil points typically lie between the former two and are outliers since their range values are interpolated between foreground and background.

- *Veil points* are interpolated points between the obstacle boundaries and the shadow bound- aries. Veil points are a typical phenomenon in 3D range data obtained by laser range scanners. Their correct classification is obviously is important since they should not be incorporated in the feature extraction process.

Figure 3.2 shows an example of the different types of boundary points described above.

3.1.1 Boundary Extraction Overview

There are different indicators that are useful for the detection of those boundaries in a range image, like acute impact angles or changes of the normals. In our practical experiments, we found that the most significant indicator, which is also very robust against noise and changes in resolution, is a change in the distance between neighboring points. We will use this feature to classify boundaries according to the following steps. For every image point consider its local neighborhood and

- employ a heuristic to find the typical 3D distance to neighboring points that are not across a boundary.
- use this information to calculate a score for how likely it is that this point is part of a boundary,
- identify the class to which this boundary point belongs, and

- perform non-maximum suppression to find the exact boundary position.

3.1.2 *The Boundary Extraction Algorithm*

Please note that since we are using range images, every point has a 2D position (the position of the pixel in the image) and a 3D position (the measured position in the world coordinate frame). The same applies for distances between points. We will use the term "2D distance" to refer to the distance between the pixels in the image and we will use the term "3D distance" to refer to the Euclidean distance between the 3D points.

First we analyze every image point and apply a heuristic to find out what 3D distance a certain point typically has to its 2D neighbors, which belong to the same surface. To detect this we use a concept similar to the idea of Bilateral Filtering [4]. For each point $\mathbf{p}_i$ in the range image we select all neighboring points $\{\mathbf{n}_1, \ldots, \mathbf{n}_{s^2}\}$ that lie in the square of size s, centered around $\mathbf{p}_i$. Then we calculate their 3D distances $\{d_0, \ldots, d_{s^2}\}$ to $\mathbf{p}_i$ and sort this set in increasing order to get $\{d'_0, \ldots, d'_{s^2}\}$. Assuming that at least a certain number c of the points lie on the same surface as $\mathbf{p}_i$, we select $\hat{d} = d'_c$ as a typical distance to $\mathbf{p}_i$-s neighbors, that do not include points beyond a boundary. In our implementation we selected $s = 5$ and $c = \frac{(s+1)}{2}^2 = 9$, which would be the highest c to still get a correct value for a point lying on the tip of a right angle corner. In the next step we calculate four scores for every image point, describing the probability of having a boundary on the top, left, right, or bottom. We will only explain the procedure for the direction to the right as the other three are carried out accordingly. Let $\mathbf{p}_{x,y}$ be the 3D point at the pixel position x, y in the image. We calculate the average 3D position of some of its neighbors on the right as

$$\mathbf{p}_{\text{right}} = \frac{1}{m_p} \sum_{i=1}^{m_p} \mathbf{p}_{x+i,y}, \tag{3.1}$$

where m_p is the number of points used for the calculation of the average (3 in our implemen- tation). We take this average instead of just the neighbor $\mathbf{p}_{x+1,y}$ to account for noise and the possible existence of veil points. Next, we calculate the 3D distance $d_{\text{right}} = ||\mathbf{p}_{x,y} - \mathbf{p}_{\text{right}}||_2$ and a score based on the quotient of d_{right} and $\hat{d}$ as

$$s_{\text{right}} = \max\left(0, 1 - \frac{\hat{d}}{d_{\text{right}}}\right). \tag{3.2}$$

This gives us a value in $[0, 1)$, where high values indicate a substantial increase between the typ- ical neighbor distance and the distance to the points on the right, indicating a probable boundary. Many sensors return information about far ranges, meaning distances that are outside of their measuring capabilities. An integration of these far range pixels in the range image is useful because a traversal to far ranges typically marks a boundary. We therefore give such traversals a score of 1.0. Next, we apply a smoothing operation on the score values to achieve continuous boundaries and avoid disruptions originating from sensor noise.

To determine if a given point $\mathbf{p}_{x,y}$ belongs to the foreground or to the background, we have to check if its range value is lower or higher than the range of $\mathbf{p}_{\text{right}}$. A lower value indicates an obstacle boundary, a higher value indicates a shadow boundary. For

all points $\mathbf{p}_{x,y}$ that are potential obstacle boundaries, we now search for a corresponding shadow boundary to the right, selecting the one with the highest score in a maximum 2D distance (3 pixels in our implementation). Depending on the score s_{shadow} of this potential shadow boundary we slightly decrease s_{right} according to

$$s'_{\text{right}} = \max(0.9, 1 - (1 - s_{\text{shadow}})^3)\, s_{\text{right}}. \tag{3.3}$$

We thereby reduce the score by up to 10% for small values of s_{boundary}, meaning areas where no clear shadow boundary is apparent.

In a last step, we check if s'_{right} is above a threshold (0.8 in our implementation) and if it is a maximum regarding $\mathbf{p}_{x-1,y}$ and $\mathbf{p}_{x+1,y}$. If this is the case, we mark $\mathbf{p}_{x,y}$ as an obstacle boundary, its counterpart from above as a shadow boundary, and all pixels in between as veil

points. Figure 3.3(a) and (c) display an example of the output of this procedure. In this figure, the different kinds of boundary points are color coded.

Now that we have the information about the boundaries, we can use it in the keypoint ex- traction procedure, which will be explained in the next section.

3.2 The NARF Keypoint Detector

The detection of keypoints is an important step to reduce the search space for feature extraction and focus the attention on significant structures. We have the following requirements for our keypoint extraction procedure:

- It must take information about boundaries and the surface structure into account.
- It must select positions that can be reliably detected even if the object is observed from another perspective.
- The points must be on positions that provide stable areas for normal estimation or the descriptor calculation in general.

We will now first give an overview over our algorithm and how we approach the points above and then present the procedure in detail.

3.2.1 NARF Keypoint Detection Overview

Stable keypoints need significant changes of the surface in a local neighborhood to be robustly detected in the same place even if observed from different perspectives. This typically means that there are substantially different dominant directions of the surface changes in the area. To capture this, we

- consider the local neighborhood of every image point and calculate a score how much the surface changes at this position and determine a dominant direction for this change, also incorporating the information about boundaries,
- look at the dominant directions in the surrounding of each image point and calculate an interest value that represents both how much these directions differ from each other and how much the surface in the point itself changes (meaning how stable it is),
- perform smoothing on the interest values, and
- perform non-maximum suppression to find the final interest points.

The most important parameter of this process is the *support size* ρ, which is the diameter of the sphere around the keypoint, which includes all points whose dominant directions were used for the calculation of the interest value. This is the same value that will later be used to determine which points will be considered in the calculation of the descriptor. Choosing the value of ρ

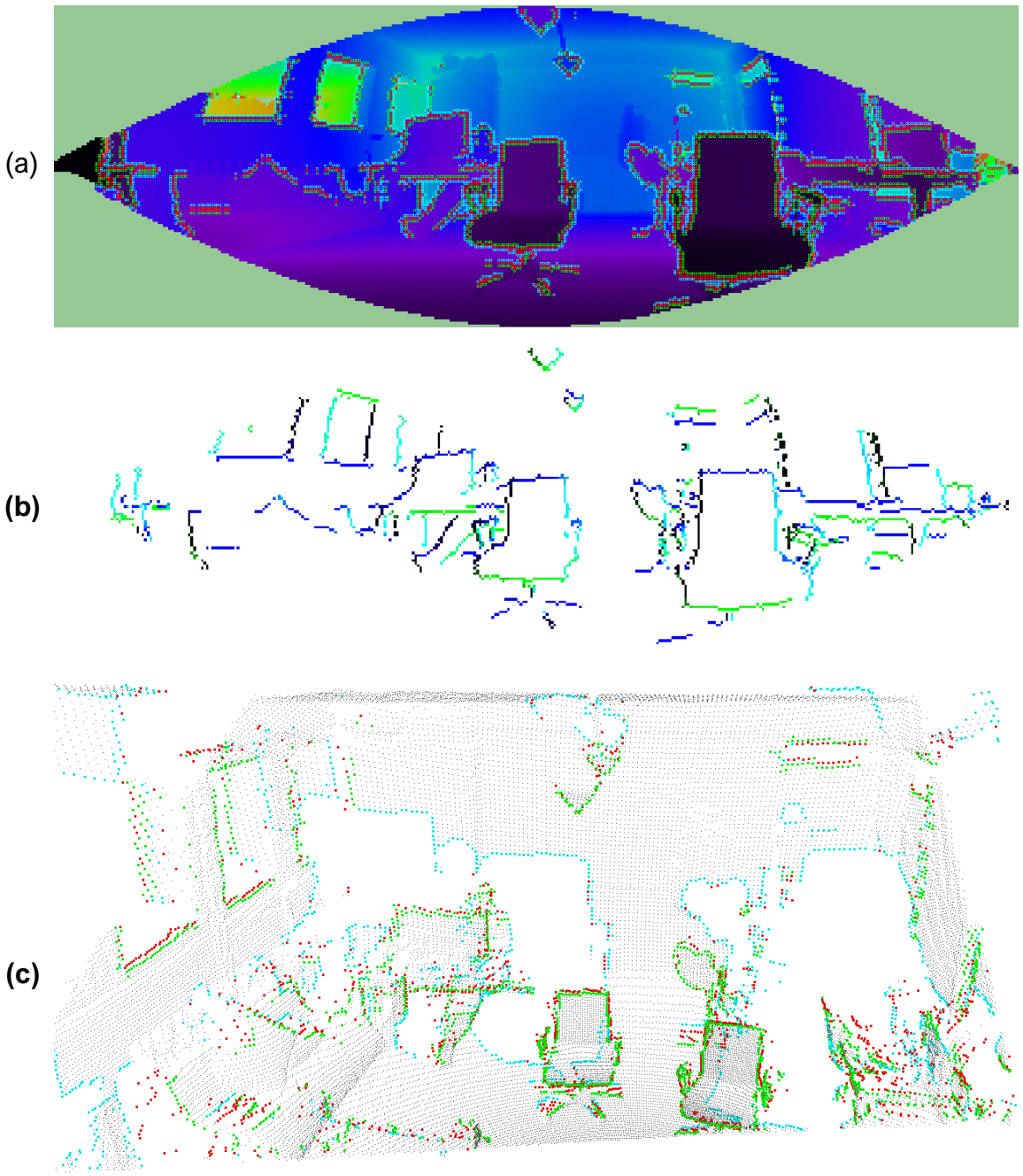


Figure 3.3: (a): Visualization of an example for the boundary extraction procedure overlain on a range image. Green points mark object boundaries, red points mark veil points and turquoise points mark shadow boundaries. **(b):** Visualization of the directions of the boundaries with blue to the top, turquoise to the right, green to the bottom and

black to the left. **(c):** The same visualization as in (a), but showing the point cloud corresponding to the range image.

depends a lot on the size of the structures that we want to find. In general, the higher the value, the more points are used to calculate the feature, which therefore becomes more stable. But in the context of object recognition it should be smaller than the object itself to have some robustness against partial occlusion. We found in our experiments that 25% of the average object size is a reasonable value. For objects of very different sizes it might be necessary to use multiple scales.

3.2.2 The NARF Keypoint Detection Algorithm

We start by calculating the directions of the boundaries we extracted in the previous step. For each point we know if it has a boundary on its top, right, left, or bottom. Thereby every boundary pixel already encodes the direction of the boundary in steps of 45°. Please note that the estimation of this quantity can be improved by averaging over multiple boundary pixels in a local neighborhood. Additionally, we use range images in spherical coordinates, which look distorted when visualized in 2D. If the estimation would be done directly in the range image space, this distortion would influence the calculation of 2D directions in the image itself. To prevent this, we perform the calculations on the 3D points corresponding to the pixels, using local normals. We estimate the normals using Principal Component Analysis (PCA) on a local 2D neighborhood of the points, where we disregard neighbors with 3D distances above $2\hat{d}$ (see Section 3.1.2).

Since we also want to consider the changes on the surfaces that are not related to boundaries, we calculate the principal curvature directions at each point, which gives us the principal direction and the magnitude λ (the largest eigenvalue) of the curvature. Every point in the image $\mathbf{p}_i$ gets an associated main direction $\mathbf{v}$, which is the boundary direction in every boundary point (see Figure 3.3(b)) and the principal direction in every other point. All of these directions get a weight w that is 1 for every boundary point and $1 - (1 - \lambda)^3$ for every other point (this expression scales the magnitude upwards, while keeping it in $[0, 1)$). Figure 3.4(b) shows an example of the values of w for the range image shown in Figure 3.4(a).

Up to now, all the calculations were done on fixed 2D pixel radius surroundings. From now on, the actual 3D support size ρ will be used. As long as enough points are available inside of the sphere with diameter ρ, this will make the method invariant to resolution, viewing distance and non-uniform point distributions.

For every image point $\mathbf{p}$ we consider all its neighbors $\mathcal{N} = \{\mathbf{n}_1, \mathbf{n}_2, \ldots\}$ that are inside of the support size (meaning that the 3D distance to $\mathbf{p}$ is below $\frac{\rho}{2}$) and do not have a boundary in between. Each $\mathbf{n}$ has a main direction $\mathbf{v}_\mathbf{n}$ and a weight $w_\mathbf{n}$. To get back to 2D direction vectors, which helps us reduce the influence of noise from the normal estimation, we project the directions onto a plane perpendicular to the viewing direction (the line from the sensor to $\mathbf{p}$). This leads to a one dimensional angle $\beta_\mathbf{n}$ for each $\mathbf{n}$.

Since two opposite directions do not define a unique position and since the principle curvature analysis does not provide a unique direction, we transform the angles in the following way:

$$\bar{\beta}' = \begin{cases} 2\,(\beta - 180^\circ) & \text{for } \beta > 90^\circ \\ 2\,(\beta + 180^\circ) & \text{for } \beta <= -90^\circ \end{cases} \tag{3.4}$$

which is a value $(-180^\circ, 180^\circ]$, but a difference of 180° of the resulting angles is actually just a difference of 90° in the original angles, which is where the wrap-around occurs now. We furthermore smooth all the weights and angles by applying a bounded Gaussian kernel.

We now define the interest value $i(\mathbf{p})$ of 3D point $\mathbf{p}$ as follows:

$$i_1(\mathbf{p}) = \min_{\mathbf{n}\in N} \left(1 - w_{\mathbf{n}} \max\left(0, 1 - \frac{10\,\|\mathbf{p}-\mathbf{n}\|}{\rho}\right)\right) \tag{3.5}$$

$$f(\mathbf{n}) = w_{\mathbf{n}} \left(1 - \frac{2\,\|\mathbf{p}-\mathbf{n}\|}{\rho} - \frac{1}{2}\right) \tag{3.6}$$

$$i_2(\mathbf{p}) = \max_{\mathbf{n}_1,\mathbf{n}_2\in N} \left(f(\mathbf{n}_1)\, f(\mathbf{n}_2)\left(1 - |\cos(\beta'_{\mathbf{n}1} - \beta'_{\mathbf{n}2})|\right)\right) \tag{3.7}$$

$$i(\mathbf{p}) = i_1(\mathbf{p})\, i_2(\mathbf{p}) \tag{3.8}$$

The Term i_1 scales the interest value i downwards, if $\mathbf{p}$ has neighboring points with high weights close by, which indicates strong surface changes. The term thereby satisfies our desired property to put keypoints only on locally stable surface positions. The term i_2 increases the interest value if there is a pair of neighbors with very different and strong main directions in the vicinity. After i is calculated in every image point, we perform an additional smoothing of the values over the image.

In a final step we now select all maxima of i above a threshold as keypoints. See Fig- ure 3.4(c) and (d) for an example, where the values of i for two different values of ρ are visual- ized and the keypoints are marked. Note, how the keypoints are in the corners of the chairs for a small support size, whereas they move more to the middle for higher values.

We are now able to extract keypoints on a 3D scene. Our next goal is to create descriptors for these keypoints.

3.3 The NARF Descriptor

Feature descriptors characterize the area around a keypoint in a way that makes efficient com- parison regarding similarity possible. Our goals in the development for the NARF descriptor were as follows:

1. The descriptor should captures the existence of occupied and free space, so that parts on the surface and also the outer shape of an object can be described.

2. The descriptor should be robust against noise on the keypoint position.

3. The descriptor should enable us to extract a unique local coordinate frame at the point.

To address issue 3, one has to eliminate six degrees of freedom, namely the 3D position and the 3D angle of the transformation between the original coordinate frame and the coordinate frame of the feature. If we choose the position of the point feature as the origin, only the three angles remain. The normal vector at the point can be used to eliminate two more degrees of freedom, which leaves the rotation around the normal

to be determined. While many feature descriptors in 3D are invariant to the rotation around the normal (like Spin Images [65]), or even the complete 3D orientation [108], it is helpful to have the information about this orientation available for multiple reasons. For one, it might be desirable to be able to not use a unique ori- entation, e.g., if we only search for correspondences with a fixed patch orientation, as in the case of a wheeled robot searching for correspondences between its map and the environment. An invariance regarding the robot's roll might unnecessarily increase the size of the search space since the robot will operate at roll angle zero most of the time. On the other hand, in cases

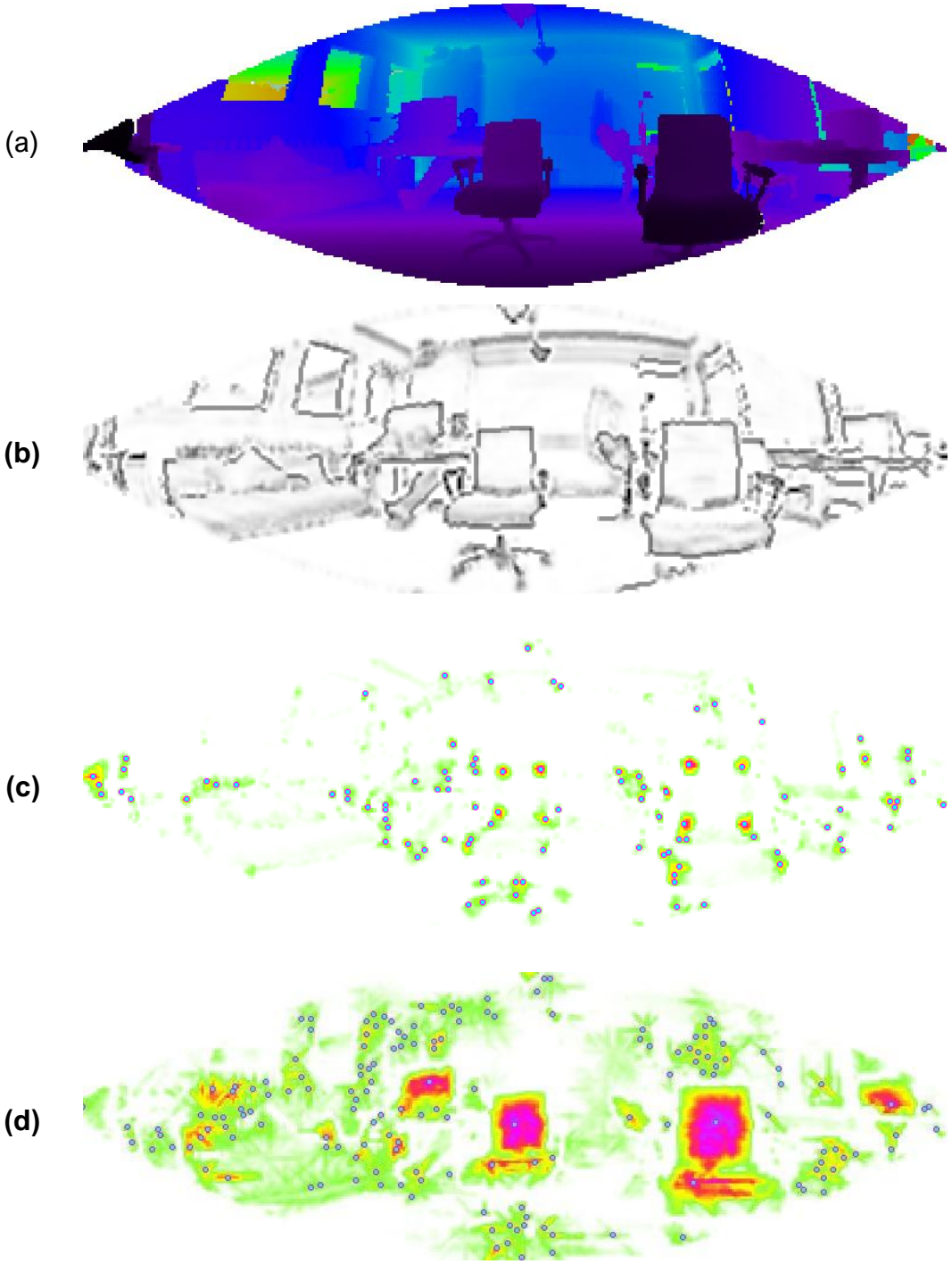


Figure 3.4: The keypoint extraction procedure. **(a)**: Range image of our example scene. **(b)**: Surface change scores according to boundaries and principle curvature. **(c)**: Interest values with marked key- points for a support size of 20 cm. Note how the corners of the chairs are detected as keypoints at this scale. **(d)**: Interest values with

marked keypoints for a support size of 1 m. Note how the whole surfaces of the backrests of the chairs contain one keypoint at this scale.

where the unique orientation is used, it enables additional filtering for consistent local coordi- nate frames between features. The NARF descriptor enables us to extract a unique orientation around the normal. The underlying idea is similar to what is done in SIFT [82] and SURF [9]. Yet, unlike its 2D siblings, this orientation together with the normal defines a complete 6DOF transformation at the position of the keypoint.

3.3.1 NARF Descriptor Calculation Overview

To compute the NARF descriptor in a keypoint, we perform the following steps. Please consider Figure 3.6(a,b) for a visual example of the process.

1. We calculate a normal aligned range value patch in the point, which is a small range image with the observer looking at the point along the normal and a size determined by the support size.
2. We overlay a star pattern onto this patch, where each beam corresponds to one value in the final descriptor. This value captures how much the pixels under the beam change.
3. We extract a unique orientation from the descriptor,
4. We shift the descriptor according to this value to make it invariant to the rotation.

Steps 3 and 4 are optional, if complete rotational invariance is not necessary, as explained above. In principal, the normal aligned range value patch (NARVP) mentioned in step 1 can also be used as a descriptor. Its advantage is that it actually preserves most of the information about the original 3D structure, apart from the resolution. Yet the procedure described in step 2 is supposed to increase the robustness of the descriptor regarding noise, especially regarding the keypoint position. We will show this advantage in the experimental evaluation (see Section 3.4).

3.3.2 The NARF Descriptor Algorithm

As mentioned above, the calculation of the NARF descriptor builds upon a normal aligned range value patch (NARVP) extracted from the feature position in the scan. Such a NARVP can be calculated by creating a local coordinate system with the origin in the keypoint position, the *z*-axis facing in the normal direction and *y* being oriented according to the upright vector
in the world coordinate frame. We then transform all points within the support $\frac{\rho}{2}$ (see radius ρ

Section 3.2.1) into this coordinate frame. The resulting *x* and *y* coordinates define the cell of the descriptor in which a point falls, and the minimum over all *z* values is the value of a cell. A cell where no 3D points falls into gets the maximum value of $\frac{\rho}{2}$. See Figure 3.5 for a visualization
of the process.

The normal vector at the feature point is calculated using PCA on all points that will be used to calculate the descriptor to maximize its stability. The size of the image patch should be high enough to keep enough descriptive structure, but low enough to not surpass the typical resolution of the scan. We chose a size of 10 10 pixels in our

experiments. To prevent problems in areas where the resolution of the original scan is low, interpolation between cells or usage of ray tracing is necessary. In the next step we put a Gaussian blur onto the patch. Then we project a star shaped pattern onto it (see Figure 3.6(b)). Every beam from the star pattern corresponds to one value in the descriptor **d**. We chose **d** = 36 for our experiments, which means 10° between consecutive beams. For each beam we select the set of cells $c_0, \ldots, c_m$ that the beam crosses, with c_0 being the middle of the patch and the rest ordered according to

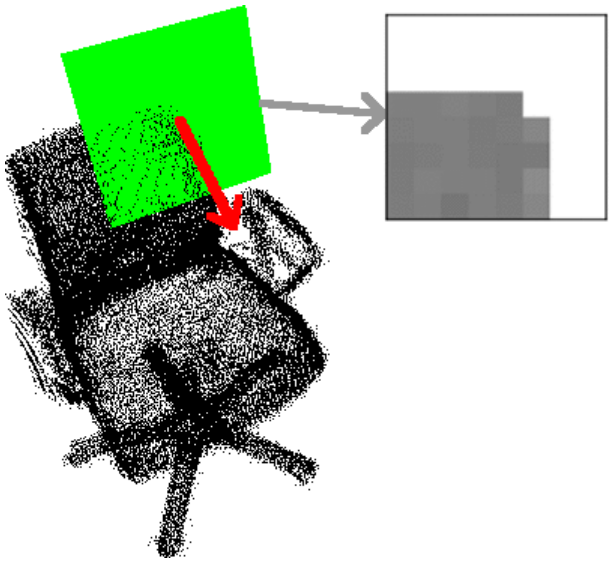


Figure 3.5: Here we visualize, how the Normal Aligned Range Value Patch (NARVP) is calculated. The green patch is centered around the keypoint and oriented according to the normal vector (the red arrow) at the point. The cells of the NARVP, which can be seen on the top right, receive values according to the minimum distance of the points that fall into them.

the distance to c_0. We define the value d_i of the descriptor cell corresponding to the i-th beam as

$$w(c_j) = 2 - \sum_{m=1}^{2} \frac{\|c_j - c_0\|}{\rho} \tag{3.9}$$

$$d'_i = \frac{\sum_{j=0}^{m-1} (w(c_j)\,(c_{j+1} - c_j))}{\sum_{j=0}^{m-1} w(c_j)} \tag{3.10}$$

$$d_i = \frac{\operatorname{atan2}\left(d'_i, \frac{\rho}{2}\right)}{180^\circ}, \tag{3.11}$$

where $w(c_j)$ is a distance-based weighting factor that weights the middle of the patch with 2 and decreases to 1 towards the outer edges of the patch. The basic intuition for d'_i isthe following. The closer to the center a change in the surface is, and the stronger the change is, the more the beam value will deviate from 0. The step from d'_i to d_i is for normalization purposes and scales every cell to [-0.5, 0.5]. Please consider Figure 3.6(a) and (b). At the bottom of (b) the
final descriptor is visualized and the arrows mark corresponding beams. Beams that lie on a flat surface have low values, whereas beams going over the boundary have high values.

Up to now the descriptor was not invariant to the rotation around the normal. We

now try to find one or more unique orientations for the descriptor. For this purpose we discretize the possible 360° into a number of bins and create a histogram. The value for a histogram cell
corresponding to angle β is

$$h(\beta) = \frac{1}{2} + \frac{1}{2|\mathbf{d}|} \sum_{i=1}^{|\mathbf{d}|} d_i \left(1 - \frac{|\beta - \gamma_i|}{180^\circ}\right)^2, \qquad (3.12)$$

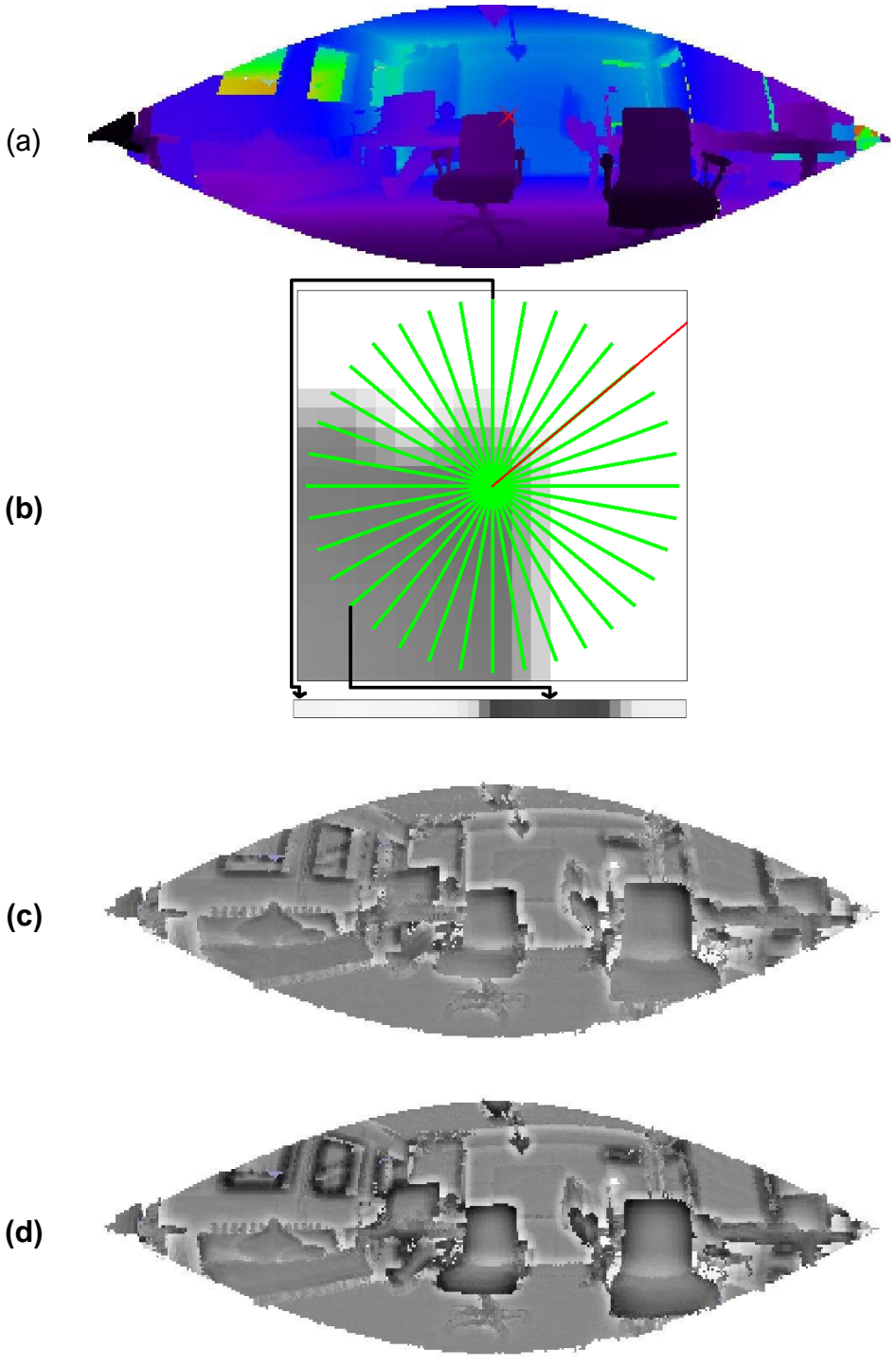


Figure 3.6: (a): An example range image. The red cross marks the position where we extracted the NARF descriptor. **(b)**: Visualization how the descriptor is calculated. The background image behind the star pattern shows a normal aligned range value patch (NARVP) of the top right corner of chair. The actual descriptor is visualized below. Each of the 36 cells of the descriptor corresponds to one of the green beams visualized in the patch, with two of the correspondences marked with arrows. The top most beam corresponds to the first descriptor value and then revolves clock-wise. The additional

red beam pointing to the top right shows the extracted dominant orientation. The intuition is that the more the structure below the beam changes, the higher the corresponding descriptor value. **(c)**: The descriptor distances to every other point in the scene (the brighter the higher the value). Note that mainly top right rectangular corners get low values. **(d)**: Same as (c), but for the rotationally invariant version. In this case, all rectangular corners get low values.

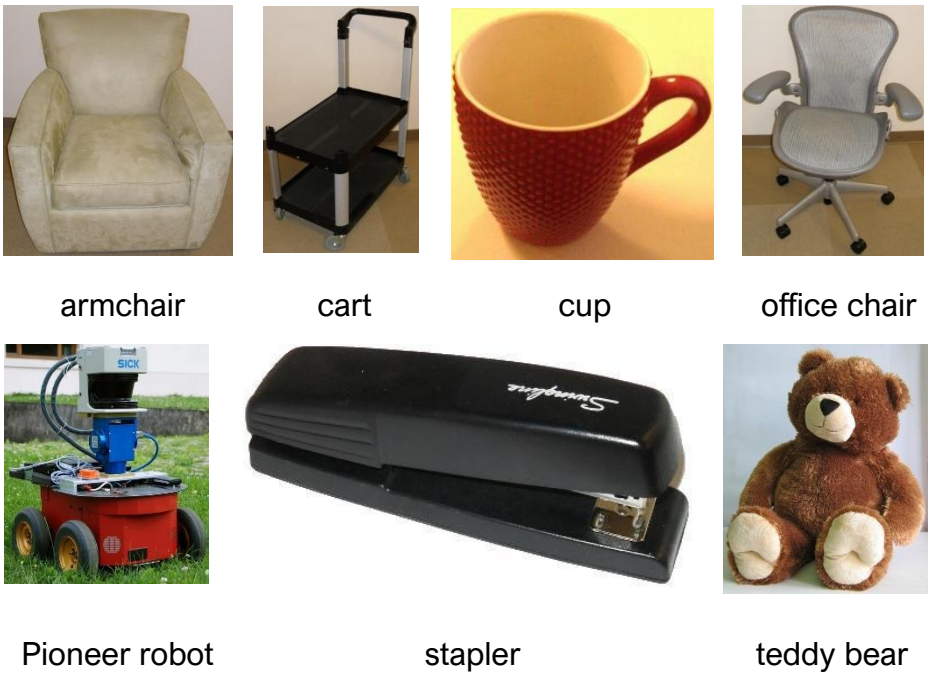


Figure 3.7: Objects used for the experiments. For each of these objects we obtained a complete point cloud model.

where γ_i is the angle corresponding to the i-th descriptor cell. We select the histogram bin with the maximum as the dominant orientation of the patch. If there is another cell with a value above 80% of the maximum, we create a second feature with this orientation. We can now shift the descriptor to create the rotationally invariant version.

The resulting descriptors can now easily be compared using standard distance functions. We chose the Manhattan distance divided by the number of cells in the descriptor, which normalizes the distance to values in [0, 1]. This means the so called *descriptor distance* of two NARF descriptors $\mathbf{d}$ and $\mathbf{d}'$ (with $|\mathbf{d}| = |\mathbf{d}'|$) is defined as:

$$\delta(\mathbf{d}, \mathbf{d}') = \frac{||\mathbf{d} - \mathbf{d}'||_1}{|\mathbf{d}|} = \frac{1}{|\mathbf{d}|} \sum_{i=0}^{|\mathbf{d}|} |d_i - d'_i|. \tag{3.13}$$

Figure 3.6(c) visualizes all the descriptor distances to the selected point in the range image without the rotational invariance. Figure 3.6(d) shows the same for the version with rotational invariance. While the first clearly distinguishes upper right corners from other structures, the latter distinguishes all right angle corners from the other structures, without focus on a specific alignment.

3.4 Experiments

We will now present our experimental evaluation of the NARFs, in which we analyze the key- point stability and the matching performance of the descriptors.

3.4.1 *Keypoint Stability*

In a first experiment, we analyze how stable the position of the keypoints is relative to changes in scale (distance to the object) and viewing angle. For this purpose we selected seven object models in the form of complete point clouds. These models represent very different kinds of

objects and include furniture-sized objects and tabletop objects. See Figure 3.7 for a list of the models. To be able to evaluate objects of very different sizes together, we scaled all the models to a bounding sphere with a diameter of 1.0 m. We used $\rho = 0.25$ m for the support size, which is large enough to cover most of the significant structure of the models, but low enough to account for partial occlusions. For each of these objects we simulated 50 noiseless views from different angles and distances around the object. Then we simulated 100 different views with additional noise on the point positions. See Figure 3.8(c) for example views with marked keypoint positions. We then compared the keypoint positions on the first set of views with each view of the other set.

We calculate the repeatability of the keypoint extraction according to the method proposed by Unnikrishnan [145], which is a 3D extension of a commonly used method in the computer vision literature [88]. For every keypoint $\mathbf{k}_1$ in one view of the object, we search for the closest keypoint $\mathbf{k}_2$ in another view and calculate the ratio of the intersection between the two spheres with radius ρ around them as

$$\text{repeatability} = 1 - \frac{3\,\|\mathbf{k}_1 - \mathbf{k}_2\|_2}{4\rho} + \frac{1}{16}\left(\frac{\|\mathbf{k}_1 - \mathbf{k}_2\|_2}{\rho}\right)^3. \tag{3.14}$$

For the calculation of the repeatability we only take unoccluded points into account, meaning we check for every point if it is influenced by self occlusion regarding the current pair of views, and reject it if this is the case.

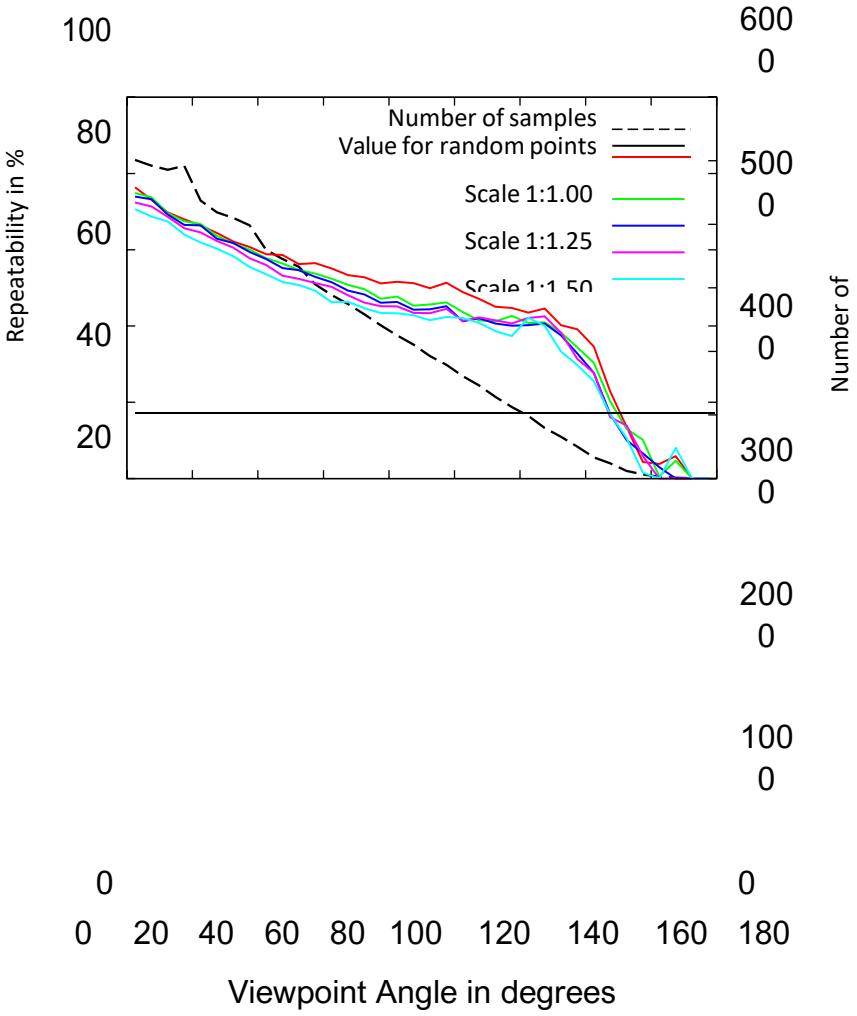
Figure 3.8(a) shows the result of the cross comparison of the positions of the keypoints of all the objects. The scores are shown dependent on the angular viewpoint change and the difference in observing distance (leading to a difference in scale in the range images). While a high change in scale obviously has a negative influence on the keypoint stability (see the different plots for different scales), this influence seems minor compared to angular changes, which is why we will omit changes in scale from now on. The dashed black line shows how many samples were available for the calculation of the averages. This value naturally decreases with the angle because increasingly fewer points are actually visible from both perspectives. The solid black line close to the bottom represents the repeatability value for a random point on the object surface, thereby defining a minimum value below which the results are not meaningful anymore. This plot shows that the keypoint extraction is relatively stable over a wide area of viewpoint changes. For changes below 20° the keypoints share about 70% of their support size on average and about 55% for changes below 60°.

Figure 3.8(b) shows the same analysis, but for every model individually. The cup shows an unusual behavior compared to the rest. This results from the fact that it is rotationally symmetric over a wide range, making the keypoint position much more dependent on the viewpoint. The increase in value between 100° and 160° is caused by keypoints on the left boundary of the cup again being extracted when seen from the other side on the right boundary. The cart, the chair and the pioneer robot models have higher values for the dashed lines. This mainly indicates that a higher number of keypoints was extracted on these models, leading to the situation that random points on the surface are closer to the keypoint positions on average.

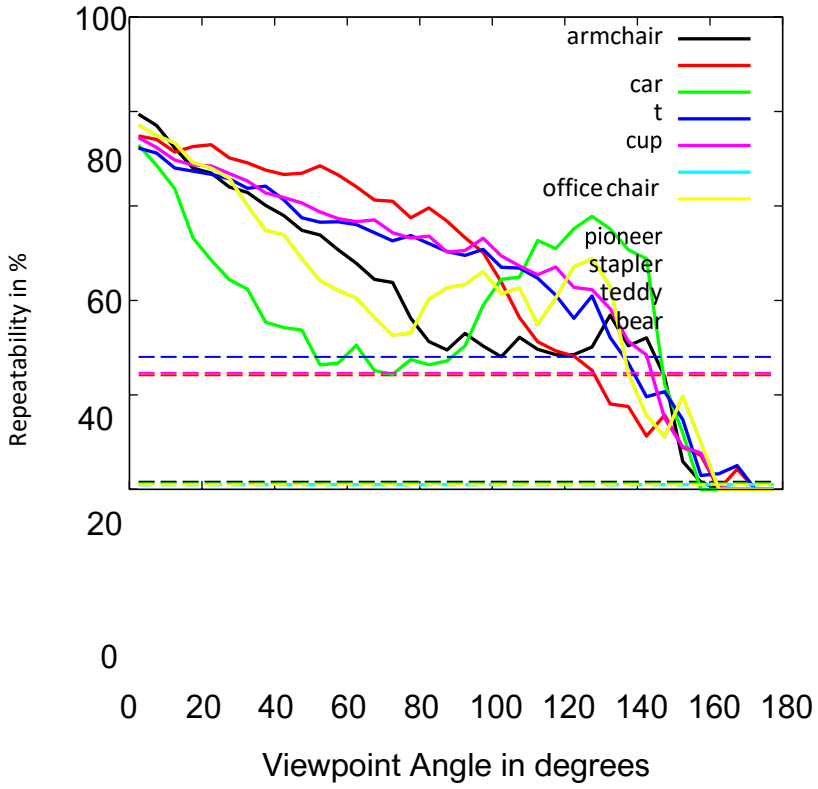
3.4.2 Matching Capability

In another experiment we tested the capability of the NARF-descriptors to match features from an object model to the features in a scene containing the model. For this purpose we collected 10 scenes with a 3D laser range finder in a typical cluttered office environment and 10 scenes with a stereo camera system of tabletop scenes. We artificially added our models, including

(a)


Repeatability in %
Number of
Number of samples
Value for random points
Scale 1:1.00
Scale 1:1.25
Scale 1:1.50
100
80
60
40
20
0
600
500
400
300
200
100
0
0 20 40 60 80 100 120 140 160 180
Viewpoint Angle in degrees

(b)


Repeatability in %
armchair
car
t
cup
office chair
pioneer
stapler
teddy
bear
100
80
60
40
20
0
0 20 40 60 80 100 120 140 160 180
Viewpoint Angle in degrees

(c)

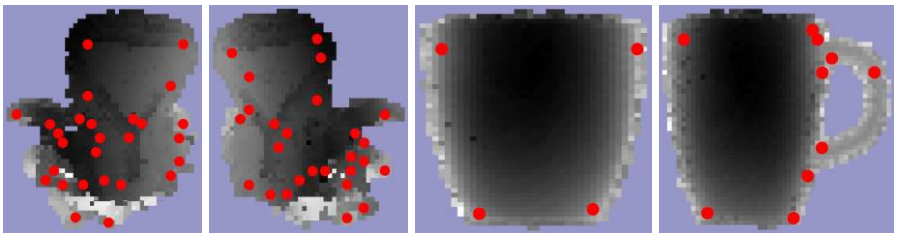


Figure 3.8: (a): This graph shows how repeatable the keypoint detection is regarding differences in the viewpoint angle and differences in the scale between two views of the object (the five solid plots). A scale of 1:1 means the object was seen at the same distance, 1:2 means at double the distance. The dashed line shows how many samples were available for averaging. The number goes down with increased angle since less and less pairs of points are actually visible from both perspectives. The solid black line follows from the average distance of a random point on the models surface to the next keypoint, giving a minimum value for the repeatability, below which the value is meaningless. **(b)**: Same as (a), but per object model. The dashed lines with constant value correspond to the solid black line in the plot in (a). **(c)**: Examples of the simulated views used to create (a) showing two views of the backside of an office chair and two views of a cup. The keypoints extracted on the views are marked in the images.

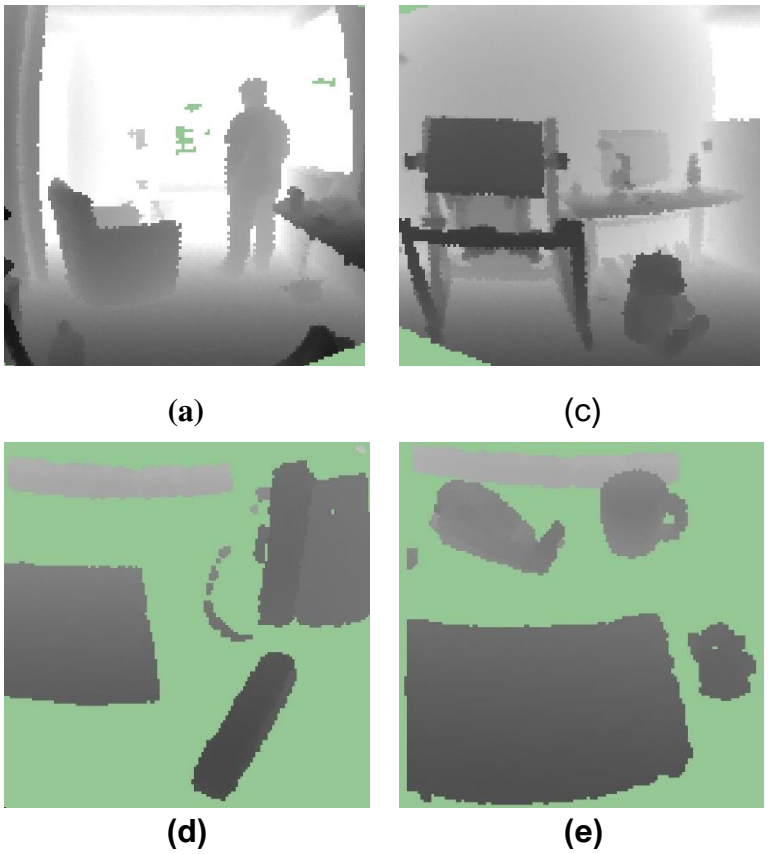


Figure 3.9: Examples for the created scenes. **(a)**: Office scene with armchair. **(b)**: Office scene with teddy bear. **(c)**: Tabletop scene with stapler. **(d)**: Tabletop scene with cup.

noise on the 3D point positions, to these scenes - the armchair, cart, chair, robot, and the teddy bear to the office scenes and the stapler and the cup to the tabletop scenes, thereby creating 70 scenes with one known object in each. In this process, the objects could appear in arbitrary yaw and x,y positions, while the height was restricted to the floor/tabletop and roll and pitch were in their natural orientations. The latter were chosen to be able to test our descriptor also without the rotational invariance. For the tabletop scenes, the table plane itself was removed, as it is often done in tabletop object recognition. Figure 3.9 shows some examples of the created scenes.

To match the objects against the scenes, we sampled poses from our models that differed from the ground truth poses between 0° and 50°.

The resulting numbers of true positives and false positives are summarized in Figure 3.10(a) as ROC (Relative Operating Characteristic) curves, which plot the true positives rate against the false positives rate. In this context that means that the bottom left represents a maximum descriptor distance of 0.0 and for the top right a maximum descriptor distance of 1.0. The general interpretation of these curves is that an approach is better, the higher the curve is above the diagonal. Please note the logarithmic scales, which highlight the area with a low number of false positives. The absolute number of false positives is much higher than the absolute number of true positives, which makes areas with a high ratio of false positives less useful. The thicker plots mark areas, where the number of true positives to false positives is lower than 1:10. The plot labeled as 'NARFs

with all points' is for our NARF feature descriptors extracted at every image point, without using the keypoints. The plot labeled as 'NARFs with keypoints' shows the performance of the NARF features extracted only at the keypoint positions. The plot labeled as 'NARFs rot. inv. with keypoints' is for the rotationally invariant version of the NARF

descriptor. The plot labeled as 'NARVPs with keypoints' is using the Normal Aligned Range Value Patches directly as the descriptor, by just considering them as one-dimensional vectors of range values.

Those curves show that the keypoints are a definite improvement compared to random point positions. The rotationally variant version of the NARF descriptors naturally performs better than the rotationally variant version, because of the reduces search space. Yet, both the rotation- ally invariant version and the rotationally variant version of the NARF descriptor outperform the NARVP descriptors, which confirms our choice to do the additional computation (see Sec- tion 3.3.1).

To evaluate if the system can be used for object recognition, we used the extracted feature matches to actually calculate object poses. Since every feature encodes a local 6DOF coordi- nate frame, one feature match is enough to calculate an object position and orientation. We calculated a pose for every match with a descriptor distance below 0.05. Figure 3.10(b) shows the average number of correct poses versus false poses for one object and one scene. An object pose was classified as correct if its error compared to the true pose was below 0.3 times the object radius in translation and below 15° in rotation. For angular differences below 20° there are typically 2 correct poses versus 10 false poses. It would be up to a spatial verification step to reject those false positives. We will address this step in Chapter 4.

Figure 3.10(c) shows the true poses per used object model. The armchair performs best since it mostly consists of large rectangles that do not change much with moderate changes in the viewpoint. The cup model performs worst, which is due to the low number of keypoints on the model and the symmetrical structure. Yet, the position of the cup, disregarding the orientation, can still be found, as the plot labeled *cup wrong rot* shows.

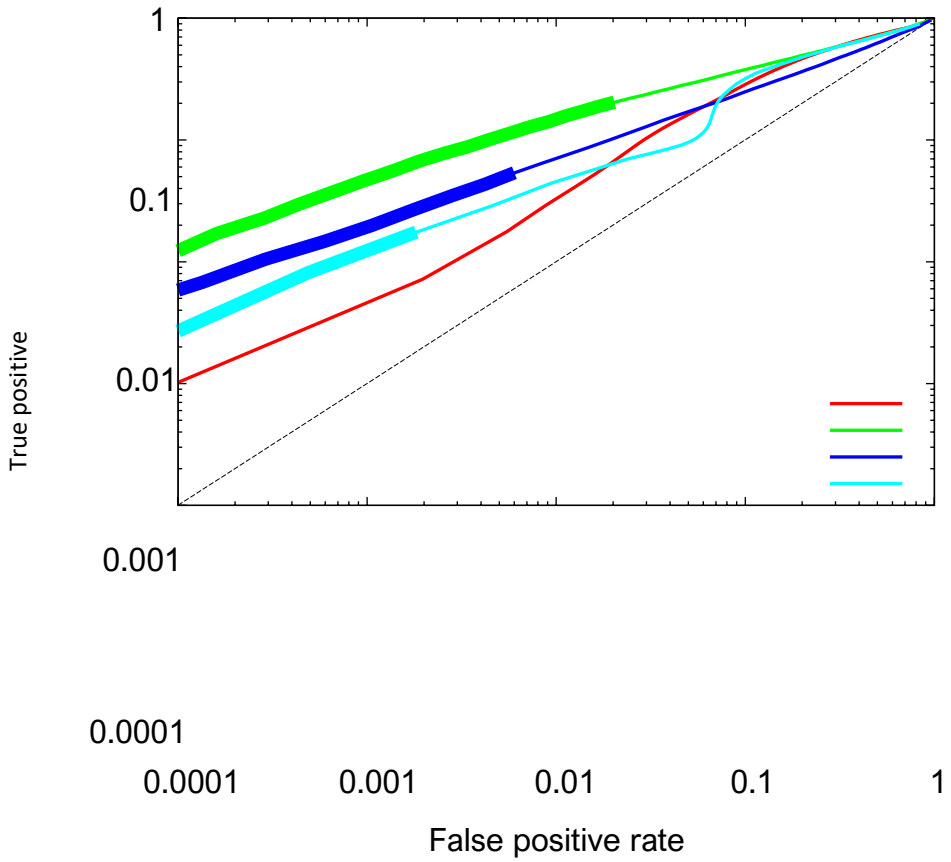
We also tested at which position in the set of matches (ordered by descriptor distance) the first correct object pose typically occurs. Table 3.1 gives an overview per object depending on the viewpoint angle difference. The first number in the table tells the average position in the set of matches (ordered by descriptor distance) where the first correct object pose occurs. The second is the rate with which a correct position was found. As can be seen, if we disregard the cup, which is a special case as explained above, for viewpoint changes up to 15°, the first correct pose can typically be found within 3 trials in the 80% where a correct pose could be found. For the rotationally invariant version of NARF, this is slightly worse. For viewpoint changes up to 15°, the first correct pose can typically be found within 10 trials in the 70% where a correct pose could be found.

3.4.3 Runtime

The point clouds for the office scenes consist of 11,500 points on average. We used an angular resolution of $\alpha = 0.4°$ for the range image creation, which took 18.1 ms on average to compute. The average runtime for the boundary detection was 22.9 ms, for the interest point extraction 27 ms, and for the feature descriptor calculation with 104 features on average 2.9 ms.

The point clouds for the tabletop scenes had 88,395 points on average. We used an angular resolution of $\alpha = 0.2^{\circ}$ for the range image creation, which took 26.4 ms on average to compute. The average runtime for the boundary detection was 5.41 ms, for the interest point extraction

6.9 ms, and for the feature descriptor calculation with 48 features on average 1 ms.


1
0.1
0.01
0.001
0.0001
True positive
0.0001
0.001
0.01
0.1
1
False positive rate

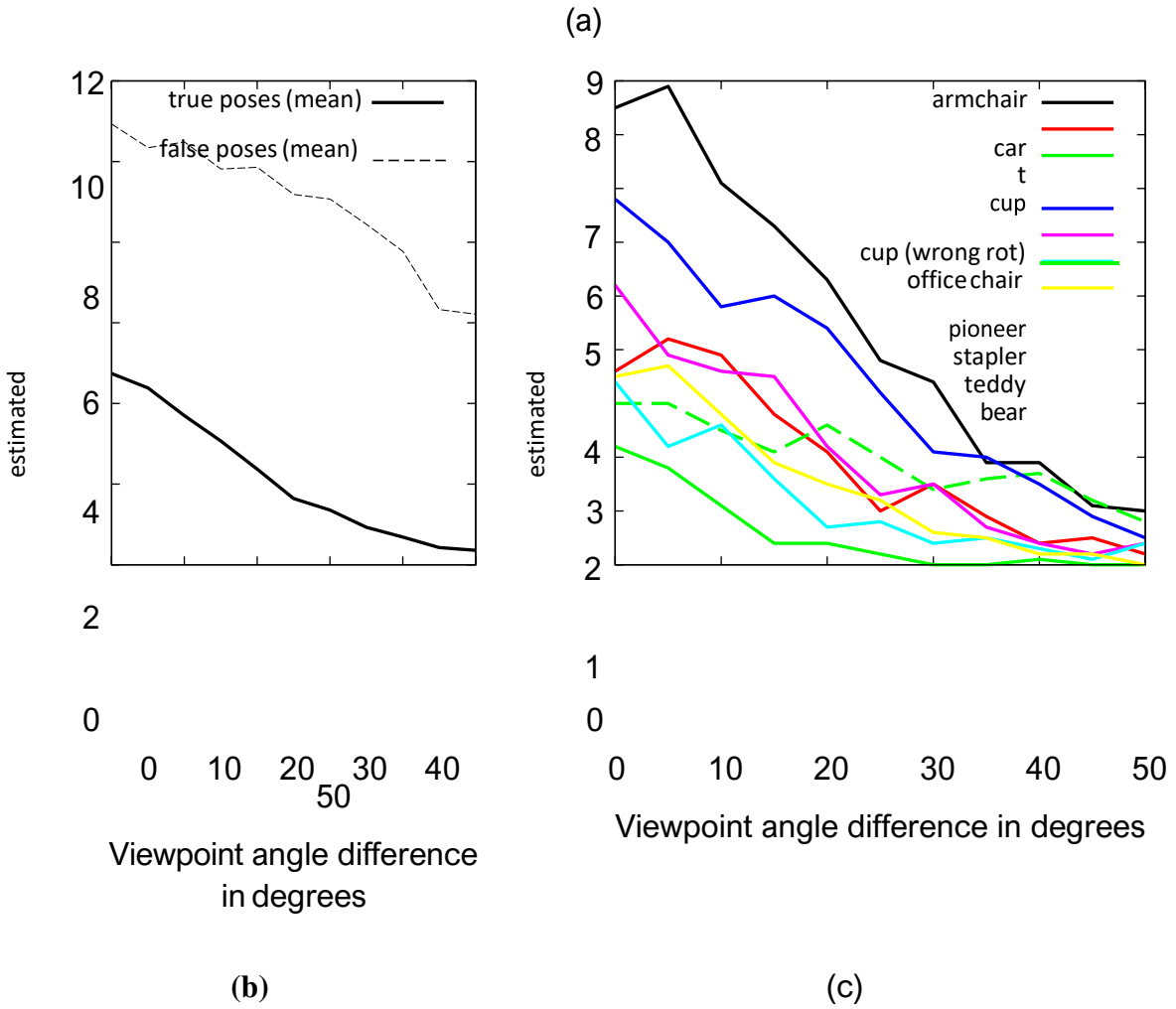


Figure 3.10: (a): This graph shows ROC curves for feature matches where the threshold for the descrip- tor distance increases from the bottom left to the top right. Please note that the axes are logarithmic. The higher a line in this graph, the better is the performance. The parts of the plots that are printed thicker mark the areas, where the ratio between true positives and false positives, in absolute numbers, is better than 1:10. **(b)**: The average number of true/false poses extracted from the feature matches for all objects. **(c)**: The average number of true poses extracted from the feature matches per object. To account for the symmetry of the cup, the plot labeled as 'cup (wrong rot)' gives the number of poses that are correct regarding (*x, y, z*)-position, but not necessarily regarding orientation.

viewpoint angle	armchair	cart	cup	office chair	pioneer robot	stapler	teddy bear
0°	1.6, 100%	1.2, 90%	1.3, 100%	1.2, 90%	1.9, 100%	1.2, 90%	2.1, 0.8
5°	1.1, 100%	2.6, 80%	1.5, 80%	1.6, 80%	1.6, 70%	2.9, 90%	1.2, 100%
10°	1.6, 100%	1.3, 90%	2, 50%	1.7, 90%	1.2, 80%	2.8, 80%	2, 90%
15°	1.8, 100%	2.4, 80%	1, 30%	1.1, 80%	2.9, 80%	4, 80%	2.9, 80%
20°	1.1, 100%	3.3, 90%	1.5, 20%	1.2, 90%	4.2, 60%	2.6, 50%	4.6, 70%
25°	1.8, 90%	3.5, 40%	2, 20%	3.3, 60%	1.5, 60%	3.8, 60%	2.3, 60%
30°	3.4, 90%	6.8, 60%	−, 0%	2, 60%	5, 60%	4, 30%	1, 30%
35°	2, 80%	4.8, 60%	−, 0%	2.2, 40%	6.4, 50%	3, 40%	3, 30%
40°	2.4, 70%	2.3, 30%	6, 10%	3, 30%	2.7, 30%	3, 20%	1, 10%
45°	3.6, 70%	5.7, 30%	−, 0%	1.5, 20%	5.5, 20%	8, 10%	1, 10%
50°	5, 70%	5.5, 20%	−, 0%	2.5, 20%	2.3, 30%	2.7, 30%	−, 0%

viewpoint angle	armchair	cart	cup	office chair	pioneer robot	stapler	teddy bear
0°	2.9, 90%	6.4, 80%	4.6, 70%	1.1, 80%	5.6, 90%	3, 90%	8, 70%
5°	5.1, 90%	2, 60%	1.4, 70%	2.1, 70%	7.3, 70%	3.7, 70%	3.1, 80%
10°	3.1, 90%	2.8, 80%	3.7, 70%	4.2, 80%	7, 80%	4.6, 70%	2.7, 70%
15°	2.8, 90%	6.9, 70%	1, 10%	2.9, 70%	15, 70%	6.2, 60%	22, 70%
20°	3.4, 90%	12, 60%	1, 10%	12, 80%	15, 60%	3.6, 50%	21, 40%
25°	5.2, 90%	14, 20%	12, 20%	5.2, 40%	5.7, 30%	11, 30%	15, 50%
30°	12, 80%	13, 50%	−, 0%	8, 30%	12, 40%	4.5, 20%	15, 30%
35°	7.3, 70%	18, 30%	35, 10%	18, 30%	27, 50%	1, 10%	10, 20%
40°	5.2, 60%	14, 20%	−, 0%	8.5, 20%	55, 30%	−, 0%	14, 20%
45°	9.4, 50%	7.3, 30%	−, 0%	4, 20%	36, 20%	4, 20%	5, 10%
50°	13, 40%	4, 10%	−, 0%	19, 10%	4, 10%		−, 0

Table 3.1: Position in the set of matches where the first correct object pose typically occurs and the recall rate for the individual objects. The table on **top** shows the results for the NARF version without rotational invariance and the table on **bottom** for the version with rotational invariance.

3.5 Related Work

Two of the most popular systems for extracting keypoints and creating stable descriptors in the area of 2D computer vision are SIFT (Scale Invariant Feature

Transform) [82] and SURF (Speeded Up Robust Features) [9]. The keypoint detection and the descriptors are based on local gradients and a unique orientation for the image patch is extracted to achieve rotational invariance. Our approach operates on 3D data instead of monocular camera images. Many of the general concepts of SIFT and SURF, such as the usage of gradients and the extraction of a unique orientation, are useful concepts to be transfered to feature extraction approaches for 3D range scans.

There are also approaches that use features originally designed for vision tasks on special types of images that are calculated from 3D range data. Zhuang *et al.* [154] extract SIFT features on *Bearing Angle Images* [114], which were extracted from range images of the 3D scenes. Li and Olson [81] also create visual images from LIDAR data. They use images from the 2D visualization of laser range data from the birds eye perspective. Their method is usable in 2D and 3D, although a 2D projection of the points is performed in the 3D case. Approaches like these make it possible to use well established feature extraction methods from the vision sector,

However, since those features have not been specifically designed for this purpose, they can not make use of range specific information like the known scale and also are more susceptible to viewpoint changes.

One of the most popular descriptors for 3D data is the *Spin Image* presented by Johnson [65], which is a 2D representation of the surface surrounding a 3D point and is computed for ev- ery point in the scene. A variant of Spin Images, *Spherical Spin Images* [107], improves the comparison by applying nearest neighbor search using the linear correlation coefficient as the equivalence classes of Spin Images. To efficiently perform the comparison of features, the authors furthermore compress the descriptor. Spin Images do not explicitly take empty space (e.g., beyond object boundaries) into account. For example, for a square plane the Spin Images for points in the center and the corners would be identical, while the feature described in this chapter is able to discriminate between such points.

Li and Gusko [80] presented a keypoint detector and descriptor, which they use for surface alignment. They use a scale-space representation of the input data and use the locations of level difference extrema as keypoints. The descriptors are based on the normal field near the surface. Compared to us, this approach focuses mainly on high quality 3D scans with little noise.

An object detection approach based on silhouettes extracted from range images was pre- sented by Stiene *et al.* [132]. The proposed features are based on a fast Eigen-CSS method and a supervised learning algorithm. This is similar to our work in the sense that the authors also try to make explicit use of boundary information. By restricting the system to boundaries only, however, valuable information regarding the structure of the objects is not considered. Addi- tionally, the extraction of a single descriptor for the complete silhouette makes the system less robust to occlusions.

Huang et al. [63] developed a system for automatic reassembly of broken 3D solids. For this purpose the authors also extract boundaries on the 3D structures, in this case to detect sets of faces. The method detects cycles of edges based on surface curvature. The detected surface parts are then matched to find corresponding fragment parts. In our application the boundary detection finds changes from foreground to background and uses this outer shape in the keypoint detection and matching, compared to dividing the 3D structure into faces that are then matched individually.

Many approaches compute descriptors exhaustively in every data point or use simple sam- pling methods [65, 43, 86], thereby introducing an unnecessary overhead. Gelfand *et al.* [46] present an approach to global registration using one-dimensional Integral Volume Descriptors (IVD) estimated at certain keypoints in the data. These keypoints are extracted using a self similarity approach in the IVD space, meaning the descriptor of a point is compared to the descriptors of its neighbors to determine areas where there is a significant change. While this method for keypoint extraction explicitly takes the descriptor into account, it becomes imprac- tical for more complex descriptors, which are more expensive to extract. Unnikrishnan [145] presents a keypoint extraction method with automatic scale detection in unorganized 3D point clouds. This approach, however, does not consider any viewpoint related information and does not attempt to place keypoints in stable positions. Rusu *et al.* [111] present a method for select- ing keypoints based on their *persistence* in growing point neighborhoods. Given a Point Feature Histogram space [110], multiple descriptors are

estimated for several different radii, and only the ones similar between one radius step and the next are kept. The method described in this chapter is by several orders of magnitude faster in the estimation of the keypoints.

Recently, mainly thanks to the new availability of cheap 3D sensors like the Microsoft Kinect or the Asus Xtion Pro Live, the area of feature extraction on 3D range data has got- ten more focus in the community, resulting in a few new developments. The *SHOT* (Signature

of Histograms of OrienTations) descriptor [139] uses a set of local histograms based on normals that are then grouped together to form the descriptor. The descriptor is made viewpoint invari- ant using an unique local 3D reference frame. Zhong [153] presented the *ISS* (Intrinsic Shape Signature), which provides a simple keypoint extraction method and a descriptor calculation method. The keypoint extraction is based on an analysis of the eigenvalues of the point scatter matrix of the spherical neighborhood of a point. This measures the variation of the surface in the area and typically returns a high number of keypoints. The descriptor calculation is also based on a local 3D reference frame extracted from the structure and is a histogram of the weighted 3D occupation in its support.

Bo *et al.* [13] use a method based on sparse coding for object recognition and classification with an RGB-D sensor. Sparse coding is based on the idea that surfaces are described using a linear combination of dictionary entries. The dictionary consists of a number of previously learned surface attributes. Compared to NARFs this approach does not purely rely on depth information, but also incorporates vision information.

Behley *et al.* [10] evaluated the performance of different histogram descriptors regarding the classification (ground, vegetation, etc.) of points in urban 3D datasets, collected with dif- ferent laser based sensors. They also proposed their own descriptor in this context, a spectral histogram. This descriptor is similar to the SHOT descriptor [139] regarding the construction and performed favorably in the classification scenario.

3.6 Conclusions

In this chapter, we presented a novel approach to feature extraction in range images, namely the Normal Aligned Radial Feature (NARF). These point features are generated at locations where the surface is mostly stable but changes significantly in the immediate vicinity. The approach makes explicit use of boundary information, thereby encoding information about the outer shape of an object, which is often very descriptive. In practical experiments, we analyzed the perfor- mance of the keypoints and descriptors regarding repeatability and matching capability. We showed that the keypoints are highly repeatable even for substantial viewpoint differences and that the descriptors enable a robust matching of similar structures. We integrated the NARF keypoint extraction and the descriptor calculation into the Point Cloud Library (PCL) [109] as open source, to enable other researchers to use it. In the following chapters we will present several approaches that build upon the results of this chapter to address complex 3D percep- tion problems in the area of robotics, like object recognition, environment modeling, and place recognition. These systems could not have provided such high quality results without a strong underlying feature type.

Chapter 4

Object Recognition using 3D Range Data

Object recognition is the process to detect known objects and estimate their poses in a perceived scene. The ability to robustly perform object recognition is especially relevant in the area of service robotics. Consider a robot that is supposed to perform pick-and-place tasks. It has to identify the object in question and provide an accu- rate pose estimate for the subsequent grasping task. In this chapter we present an approach that, given a point cloud model of an object, uses point correspondences found with NARFs to determine possible poses in the scene where the object could be present. We introduce a novel algorithm that makes this calculation of candidate object poses from a set of feature matches possible in a very efficient manner. Next, our system performs spatial verification tests that score these candidate poses. We present a novel method that makes a robust evaluation like this possible, by analyz- ing how well the actual sensor measurements fit the assumption that the object is present at a certain position. In practical experiments we show that our method is able to recognize objects reliably, even in substantially cluttered scenes. In addition we demonstrate that our system can use a partial view of an object as a model, if a complete point cloud model is not available. Since all evaluated object poses rely on the feature matching process, the experiments presented in this chapter also extend the insights we got about the NARFs in Chapter 3, proving their reliability and enhancing the understanding of their properties.

* * * *

The ability of a robot to perform high level tasks like tidying up a room, fetching something for the user, or assembling a product from an unorganized set of parts requires many different non-trivial abilities. One of those is the capability to find objects that are relevant for the current task. For example, to execute commands like “please bring a chair” or “please wait at the table”, the robot needs to identify the existence and the 6DOF pose of the corresponding object in the scene.

In computer vision, object recognition has been a popular research topic for a long time. It was, e.g., the first application of the popular SIFT algorithm [82]. Compared to the approaches from computer vision, we focus on the recognition of objects in 3D range data, based on object models in the form of 3D point clouds. The high quality spatial information in 3D scans is advantageous for determining highly accurate object poses. Additionally, 3D sensors often have a wider field of view than cameras, enabling us to find objects in a larger area without an active search. Yet, 3D structures are

typically much less distinctive than textures in vision. Object recognition in 3D scans must therefore be especially robust regarding high numbers of false

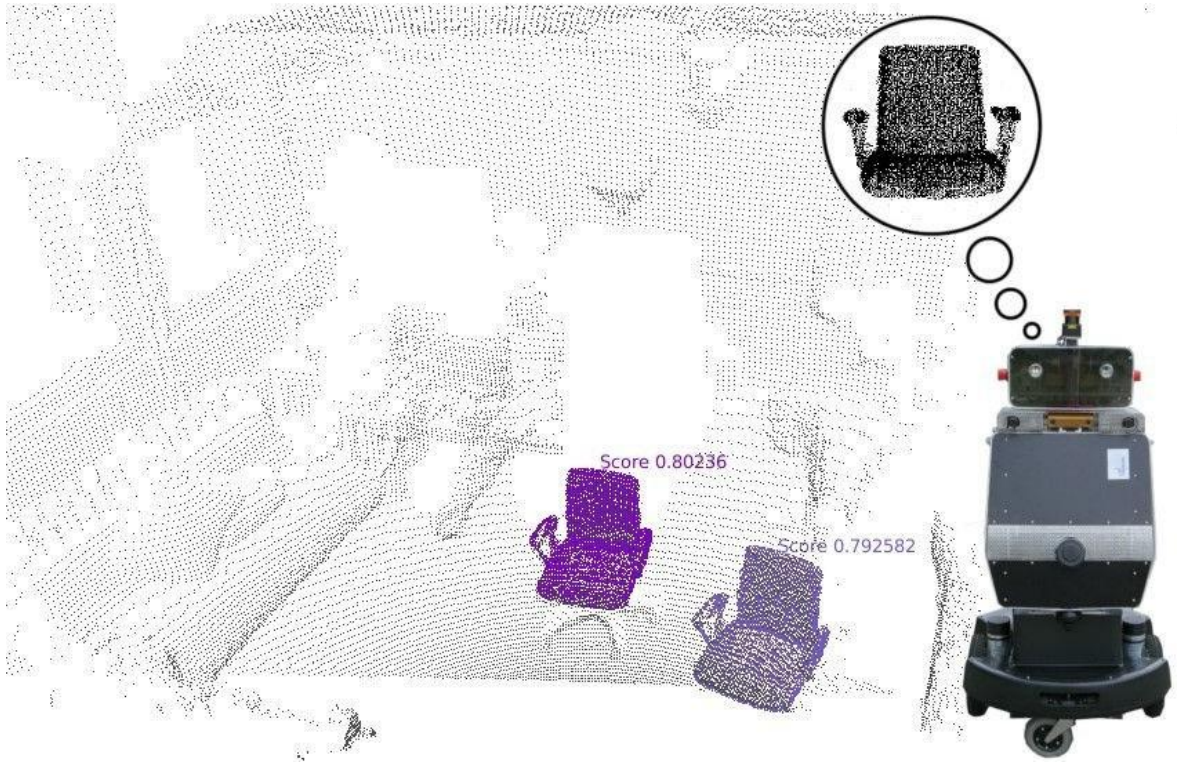


Figure 4.1: Motivational image for the object recognition system, showing a robot searching for the model of a chair in a point cloud of an office scene. Our system found two of these chairs with a certainty given by the score printed above them, whereas the score is between 0 (uncertain) and 1 (certain).

feature matches. For this purpose we will present a new algorithm to calculate potential object poses from a set of feature matches and will introduce two novel methods for the evaluation of such candidate poses based on spatial verification. This means we evaluate how well the actual data fits the predicted measurements assuming a correct pose estimate.

The calculation of the pose estimates and the evaluation step are mostly independent. In principle, one could sample from the 6DOF search space for every potential object and then verify those candidates. Since this is intractable we use the point features we presented in Chapter 3. We compare NARFs extracted from the model with NARFs extracted from the scene and calculate candidate object poses from the matches. Figure 4.1 shows an example of a scene we already used in Chapter 3. Here the two chairs in the foreground were correctly identified by our method.

The remainder of this chapter is organized as follows: we will first discuss the structure of our object database in Section 4.1. After this, we will explain how feature matches are determined between the object database and the current 3D scan in Section 4.2. Following this, we will discuss, how possible object poses can be calculated from sets of matching features in Section 4.3. In Section 4.4 we will explain

how we evaluate these potential object poses to reject false positives. Section 4.5 briefly explains, how the approach handles multiple objects in the database. After this we present our experimental evaluation in Section 4.6. We will finish this chapter with a discussion about related work in Section 4.7 and a conclusion in Section 4.8.

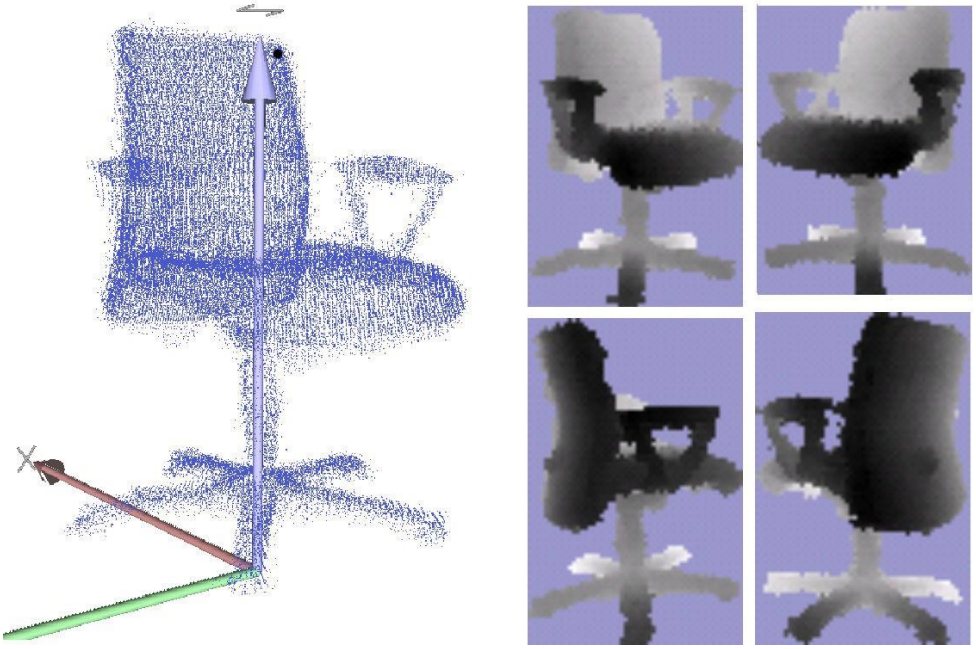


Figure 4.2: Point cloud model of a chair and four range images sampled from different perspectives around the model.

4.1 The Object Database

Our object database consists of a set of object models. For each of these models our system expects either a complete point cloud or a range scan from a single perspective. In the latter case the original sensor position must also be known and a correct detection is only possible if the object is seen from the same side. For a single view, we calculate a range image (see Section 2.3.2) from the perspective defined by the given viewpoint. For the complete point cloud model on the other hand, we sample different view points around the object and calculate range images for these perspectives. Figure 4.2 shows an example for this procedure. For each of the sampled views we then extract NARFs and store them in the model structure.

4.2 Feature Matching

The scene can be provided to our system either directly as a range image, or in form of a point cloud with a given view point. The latter is then used to compute a range image (see Section 2.3.2). From the scene range image we then extract NARFs (see Chapter 3) and compare them to the features extracted from the model views.

Let $N^S = \{\mathbf{d}^S_1, \mathbf{d}^S_2, \ldots, \mathbf{d}^S_{|N^S|}\}$ be the set of NARF descriptors for the current scene and let $N^M = \{\mathbf{d}^M_1, \mathbf{d}^M_2, \ldots, \mathbf{d}^M_{|N^M|}\}$ be the set of NARF descriptors for the model. For each descriptor $\mathbf{d}^S \in N^S$ in the current scene and each descriptor $\mathbf{d}^M \in N^M$ in the model, we compute the descriptor distance $\delta(\mathbf{d}^S, \mathbf{d}^M)$ according to Equation 3.13. Based on this, we determine the set of potential correspondences $C = \{\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_{|C|}\}$, which are defined

as all feature index pairs between the scene and the model where the descriptor distance is below a
given threshold δ_{max}:

$$C = \{\mathbf{c} \quad | \quad \mathbf{c} = (i,j)^T \text{ with } \delta(\mathbf{d}_i^S, \mathbf{d}_j^M) \leq \delta_{max} \wedge \mathbf{d}_i^S \in N^S \wedge \mathbf{d}_j^M \in N^M\} \quad (4.1)$$

To efficiently perform this comparison process, we store all feature description vectors in a *k*d-tree. To find the neighbors in the feature space, we apply the best-bins-first technique proposed by Lowe [82] to handle the high dimensionality of the vectors.

In the following we will use the feature correspondences to calculate candidate transfor- mations.

4.3 Candidate Poses from Feature Matches

Each NARF encodes its own local coordinate frame. Therefore, a single feature correspondence is enough to determine the potential 6DOF pose of the object in the scene as the difference be- tween the feature coordinate frames. Another possibility is to use several correspondences to calculate a candidate pose. If we use only the point positions and the normals, then two feature matches are enough. If we use only the point positions alone, then we need to know at least three correct feature correspondences [60]. While using an increasing number of correspondences re- duces the influence of noise on the estimate, there are also significant disadvantages: the search space size increases exponentially with the number of used correspondences and if the object model provides only a small number of keypoints or if it is partially occluded in the scene, there may only exist a small number of correct feature matches for a certain instance in the scene. To exploit the advantages of the different sized sets of correspondences, we add candidate poses from single correspondences, from correspondence pairs, and from correspondence triples. Yet we only consider a fixed number n_C of the best candidate poses for each of these possibilities. To decide which sets of the correspondences are used to determine candidate poses, we sort the set of correspondences C according to the respective descriptor distances to get the ordered set of correspondences $C' = \{\mathbf{c}'_1, \mathbf{c}'_2, \ldots, \mathbf{c}'_{|C|}\}$. For single correspondences we just select the first n_C elements from C' to calculate candidate poses. For two or three correspondences we work through the sets in the order defined by the loop structures in Algorithm 1. In this way, we obtain the best correspondences (according to the descriptor distance) in an order that prevents the search from getting stuck if correspondences with low descriptor distances are actually false matches. We discard or accept the correspondence sets based on some simple heuristics. For example, we check if the metric distances between the three feature points in the scene are similar to the distances of the feature points in the model and we enforce a minimum distance between them. Additionally, we do not use triples of features that lie on a line. Correspondence sets that pass these quick tests are then used to generate candidate object poses, until n_C such poses were created for pairs and triples of correspondences respectively.

In the next step, the elements of the resulting set of candidate object poses are evaluated regarding how well they fit the actual observation. Our method to perform this evaluation will be discussed in Section 4.4.

Please note that the idea behind this procedure, to select a small set of correspondences to calculate a transformation and then verify it in a second step, is

borrowed from sample consen- sus algorithms like RANSAC [40]. Or in this case rather GOODSAC [87], where the samples are not selected randomly but according to a quality measure (the descriptor distance in our case). Yet, we do not actually use a sample consensus method since or spatial verification method (see Section 4.4) does not rely on the original feature matches to ensure that we can also find an object with even only one matching feature.

Algorithm 1 Nested loops that determine the order in which pairs or triples of correspondences are selected to find candidate object poses.

Selection scheme for pairs of correspondences:

FOR ($j = 2; j \leq |C|; j = j +$
1**) FOR (**$i = 1; i < j; i = i$
$+ 1$**)**

$pair = (c_i, c_j)$

. . .

This generates the sequence (c'_1, c'_2), (c'_1, c'_3), (c'_2, c'_3), (c'_1, c'_4), (c'_2, c'_4), (c'_3, c'_4), . . .

Selection scheme for triples of correspondences:

FOR ($k = 3; k \leq |C|; k = k +$
1**) FOR (**$j = 2; j < k; j = j$
$+ 1$**)**

FOR ($i = 1; i < j; i = i + 1$**)**

$triple = (c_i, c_j, c_k)$

. . .

This generates the sequence (c'_1, c'_2, c'_3), (c'_1, c'_2, c'_4), (c'_1, c'_3, c'_4), (c'_2, c'_3, c'_4), (c'_1, c'_2, c'_5), . . .

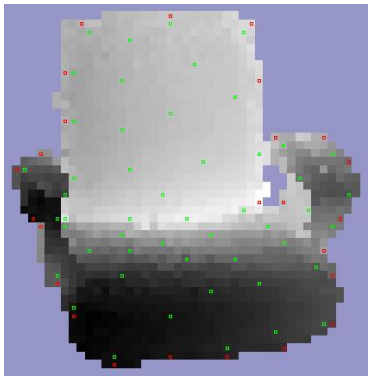


Figure 4.3: This image shows one of the sampled views of a point cloud model of a chair with the validation points marked in color. The green points are on the regular surface of the chair, whereas the red points are on the boundary.

4.4 Spatial Verification

Our spatial verification procedure is divided in two independent steps. At first a fast approx- imate method to reduce the number of potential object poses in a runtime efficient way and second a slower more accurate method to further reduce the number

of false positives.

4.4.1 *Validation Points*

The first method is based on a restricted number of surface points that are compared between scene and model regarding the candidate transformation in question. We will call these points *validation points*. The procedure works as follows:

As already done for the feature extraction procedure in Section 4.1, we sample different views of the object to simulate different perspectives from which the object could have been seen. For each of these views we select a fixed number (50 in our implementation) of uniformly

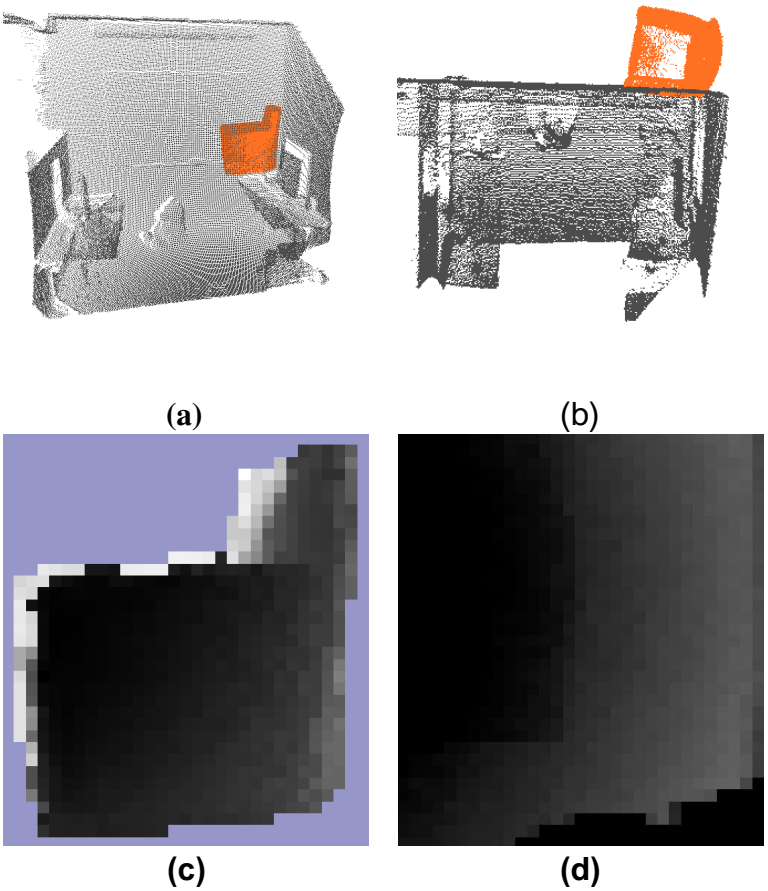


Figure 4.4: Example why considering boundaries is important **(a)** and **(b)** show different perspectives of an example scene with a falsely detected armchair model in a wall. **(c)** shows how the perspective of the object should look like and **(d)** shows what the part in the actual scene looks like. Considering only the range measurements on the surface of the object would lead to a high score, since the flat part fits well onto the wall. An analysis of the outer shape on the other hand would reveal this as a false positive.

distributed 3D points from the surface of the object and a fixed number (25 in our implemen- tation) of uniformly distributed 3D points from the boundary of the object. The boundary is defined as those pixels in the range image of the model that have at least one unoccupied pixel as a neighbor. We call these points the validation points, whereas we distinguish between the surface validation points V_{Surface} and the boundary validation points V_{Boundary}. These points need to be extracted only once during the model creation. This process therefore does not influence the runtime of the object detection process. See Figure 4.3 for an example view of a model of a chair with marked positions of the extracted validation points. We distinguish between these two types of points to increae the robustness of the evaluation. While it is of course im- portant that the surface structure matches, the outer shape of the object is also a strong feature and should therefore be considered in the evaluation. If we only consider the surface, a flat object might be matched into a wall, where the surface fits perfectly, but the outer shape is not recognizable anymore. Figure 4.4 shows an example for this situation.

Please note that the set of simulated views does not have to be identical with the one used for feature extraction. This is because the size of the set for the feature extraction influences the runtime of the object search since more features have to be matched. Therefore we try to keep this set small, meaning we sample in larger angular steps. We use 36° in our implementation, meaning we have a maximum error of 18° between the sampled view and the actual observation. It is up to the feature descriptor, to have enough view point invariance to enable matching under

these conditions. We will analyze this property of the NARF descriptor in Section 4.6.1. For the validation points on the other hand, the number of views does not influence the runtime of the matching procedure, which is why we use a much denser sampling of 10° here.

Let $T \in O$ be the current candidate transformation we want to evaluate. We consider the validation points $V = \{\mathbf{p}^V_1, \ldots, \mathbf{p}^V_{|V|}\}$, which are sampled from the range image whose viewpoint is most similar to the one defined by T. Please note that we currently do not distinguish between V_{Surface} and V_{Boundary}, since they are mostly treated the same. We will indicate the differences below. We map the selected validation points into the scene using T and receive the set of transformed validation points $V' = \{\mathbf{p}^{V'}_1, \ldots, \mathbf{p}^{V'}_{|V|}\}$. We then project each $\mathbf{p}^{V'} \in V'$ into the scene range image and consider a local pixel neighborhood around it (with a radius of two pixels in our implementation), which contains the points $S_{V'} = \{\mathbf{p}^S_{V'1}, \ldots, \mathbf{p}^S_{V'|S_{V'}|}\}$.

Here appears the only difference between the treatment of the surface validation points V_{Surface} and the boundary validation points V_{Boundary}. For the surface validation points, $S_{V'}$ really contains all points from the local 2D neighborhood in the range image. For the boundary validation points, we only consider those points from the local neighborhood for $S_{V'}$ that are also boundary points. We use the method introduced in Section 3.1.2 to detect these boundary points. Please note that the boundary classification was already performed during the NARF extraction phase. Therefore its usage in the spatial evaluation does not influence the runtime.

If the candidate transformation is correct, then the range values of the original points in the range image of the scene and the range values of the transformed validation points should be the same. Let $r_S(\mathbf{p})$ be the range value of a point $\mathbf{p}$ in the range image of the scene. Based on this we calculate the following score $s_V(\mathbf{p}^{V'})$ and weight $w_V(\mathbf{p}^{V'})$ for a single transformed validation point $\mathbf{p}^{V'} \in V'$.

$$s_V(\mathbf{p}^{V'}) = \max_{\mathbf{p}^S_{V'} \in S_{V'}} \begin{cases} 1 - \frac{|r(\mathbf{p}^S_{V'}) - r(\mathbf{p}^{V'})|}{\varepsilon_r} & : |r(\mathbf{p}^S_{V'}) - r(\mathbf{p}^{V'})| < \varepsilon_r \\ 0 & : \text{else} \end{cases}, \tag{4.2}$$

whereas

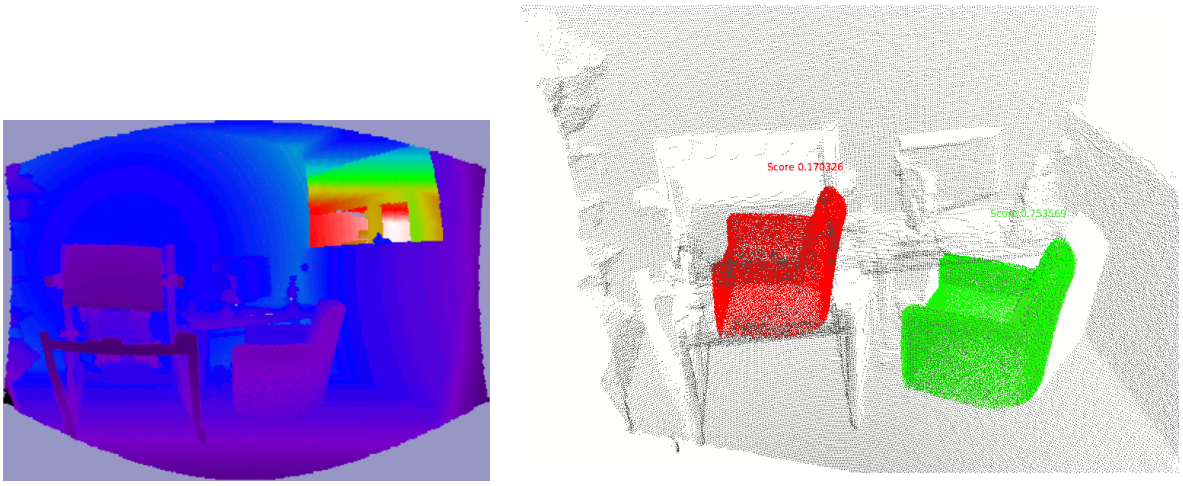
$$w_V(\mathbf{p}^{V'}) = \begin{cases} \frac{1}{|S_{V'}|} \sum_{\mathbf{p}^S_{V'} \in S_{V'}} d_{2D}(\mathbf{p}^S_{V'}, \mathbf{p}^{V'}) \cdot |r(\mathbf{p}^S_{V'}) - r(\mathbf{p}^{V'})| & : \forall\, \mathbf{p}^S_{V'} \in S_{V'} : w_{\text{seeThrough}} = 1 \\ 1 & : \text{else} \end{cases} \tag{4.3}$$

$$s_V(\mathbf{p}^{V'}) = 1 - \frac{d_{2D}(\mathbf{p}^S_{V'}, \mathbf{p}^{V'})}{d^{\max}} \cdot \frac{r(\mathbf{p}^S_{V'}) - r(\mathbf{p}^{V'})}{\varepsilon_r} \tag{4.4}$$

Here, ε_r is the maximum error between the range values, below which the validation

point can still be considered to be at the right place. If the range difference between the two points is within the error bound, i.e., $|r(\mathbf{p}^{sV'}) - r(\mathbf{p}^{V'})| < \varepsilon_r$, then the point pair receives a positive score $s_V(\mathbf{p}^{sV'}, \mathbf{p}^{V'}) \in (0, 1]$, which is the relative range error, weighted by the 2D distance between the two pixel positions in the range image $d_{2D}(\mathbf{p}^{sV'}, \mathbf{p}^{V'})$. If the pixel position is identical, d_{2D} is 0 and therefore the left factor in Equation 4.3 is 1. If d_{2D} reaches the maximum allowed 2D distance d_{2D}^{max} ($2\sqrt{2}$ in our implementation), then the left factor in Equation 4.3 is 0.75. If the difference in range between the points is above ε_r, the score for the validation point $s_V(\mathbf{p}^{V'})$ is 0. Yet, there are differences for the weight $w_V(\mathbf{p}^{V'})$ of the validation point. Normally, this weight is 1. But if all neighboring points in the scene have a substantially higher range, i.e., $r(\mathbf{p}^{sV'}) - r(\mathbf{p}^{V'}) > \varepsilon_r \quad \forall \mathbf{p}^{sV'} \in V'$, then we increase $w_V(\mathbf{p}^{V'})$ to $w_{seeThrough}$ (5 in our implementation) since this situation means the sensor actually measured a point behind the object, effectively looking through it, which is a strong indicator for a wrong pose estimate.

The score for the complete set of validation points has a value between 0.0 and 1.0 and is defined as



(a) (b)

Figure 4.5: (a): Range image of an example scene. **(b)**: The scene as a point cloud with two found armchairs. The red one is a false positive, whereas the green one is a true positive.

$$\tilde{s}_V(V') = \frac{\sum_{\mathbf{p}^{V'} \in V'} w_V(\mathbf{p}^{V'})\, s_V(\mathbf{p}^{V'})}{\sum_{\mathbf{p}^{V'} \in V'} w_V(\mathbf{p}^{V'})}, \tag{4.5}$$

whereas we combine the scores for the surface validation points and the boundary validation points to

$$\hat{s}_V(V') = \tilde{s}_V(V'_{\text{surface}})\, \tilde{s}_V(V'_{\text{boundary}}). \tag{4.6}$$

This score is already a good indicator for the quality of an object candidate and can be used as an initial filter. We can reject poses with a low validation point score. In addition, we typically receive multiple instances of an object at very similar locations. This is caused by different sets of correspondences, which all originate from the same object in the scene. Therefore, we prune the solutions by keeping only the ones with the highest score for all objects that are found in a local area. The validation point method has the advantage of being very fast for a reasonable number of validation points. Yet since we only sampled possible perspectives, this is an approximate method. We will perform an additional, more expensive verification for the remaining object poses.

4.4.2 View Simulation

In the last step we presented a method to score candidate poses based on a set of precomputed points. In this section we present a method to evaluate a candidate position based on a simulated view of the object. For this purpose we calculate a low-resolution range image of the model given the virtual camera position described by the

candidate pose. If the match is valid, the result should be similar to the corresponding part of the range image of the scene.

Figure 4.5 shows an example scene with two detected instances of a model of an armchair, whereas one of these detections is a false positive and one is a true positive. Figure 4.6(a) shows the simulated view of the object for the true positive and Figure 4.6(b) shows the corresponding part of the scene. One can see, that these two images look very similar. Figure 4.6(c) shows a combination of the other two images, in form of a range image of the scene with the object point

cloud projected into it. There are only very minor visual differences between Figure 4.6(b) and Figure 4.6(c). Figure 4.7(a),(b), and (c) show the corresponding images for the false positive, where the armchair was wrongly matched with a part of a table. Visually it is obvious that this is not a correct match.

We now calculate a score based on these images. This process is mostly equivalent to the validation point scoring described in Section 4.4.1. The main difference is that we use all valid pixels from the object view for the comparison instead of the validation points. In addition, we introduce a new error term based on the gradients of the range image pixels (see Figure 4.6(d)-
(i) and Figure 4.7(d)-(i)). For every point we now not only evaluate if the measured range value fits the expected range value (see Equation 4.3), but we also compare, in an equivalent fashion, if the gradients in x and y direction between the original perception and the simulated version are similar. In our example this means we compare Figure 4.6(e) with Figure 4.6(f) and Figure 4.6(h) with Figure 4.6(i). Equivalently for Figure 4.7. This enforces an even stricter likeness of the surface.

Figure 4.6(j) and Figure 4.7(j) show the individual scores each pixel received, whereas the true positive received an overall score of 0.75 and the false positive received an overall score of 0.17.

This step is computationally more expensive than the validation point scoring because we have to compute new range images from the model and perform several costly operations on them. However, we found that this leads to more reliable scores. We reduce the influence of the higher computational needs by first filtering out results with low validation point scores. The view simulation is then typically only performed on a small number of potential object poses.

4.5 Searching for Multiple Objects

In the previous section, we described how to detect one type of object in a scene. The approach can be straightforwardly extended to detect multiple models. In this case, we store all the features of all the models in the *k*d-tree and save from which view of of a certain model model it was extracted. Once the potential correspondences are found, we determine the possible models from the associated range images. We finally execute the previously described procedure once for every candidate model. In the best case this makes the object search complexity logarithmic in the number of object models. The average and worst case however are linear.

4.6 Experimental Evaluation

We will now present experiments to evaluate our approach. In these experiments we will com- pare poses our system calculated with previously determined ground truth poses. In this evalu- ation, we consider an object pose as correct, if no point on the model is further than one tenth of the diameter of the model away from its true position. The runtime results provided in the experiments were achieved on an Intel I7 quad core machine.

4.6.1 *Single View Models*

In a first experiment we evaluate a situation, where no complete point cloud model is available. As discussed in Section 4.1, our system can also use a single view of an object as the model. To test how well the object recognition works in this case, we captured 80 3D scans of an office

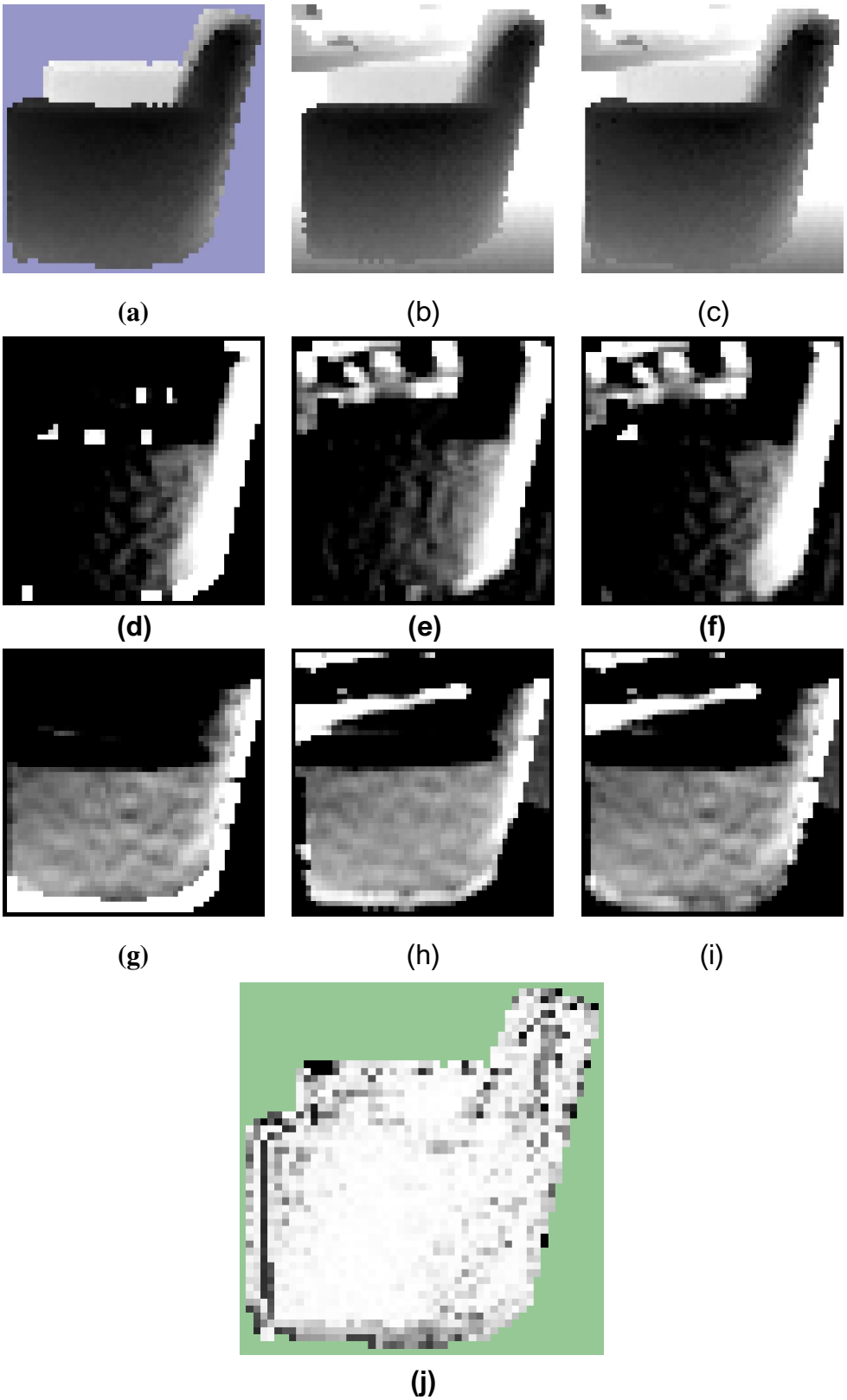


Figure 4.6: View simulation example for the true positive shown in Figure 4.5. **(a)**: The simulated view of the object. **(b)**: The corresponding part of the scene. **(c)**: The part of the scene with the object simulated inside. **(d)**: The gradients of (a) in x-direction. **(e)**: The gradients of (b) in x-direction. **(f)**: The gradients of (c) in x-direction. **(g)**: The gradients of (a) in y-direction. **(h)**: The gradients of (b) in y-direction. **(i)**: The gradients

of (c) in y-direction. **(j)**: The scores for every pixel from black (score 0) to white (score 1). Irrelevant pixels are marked green.

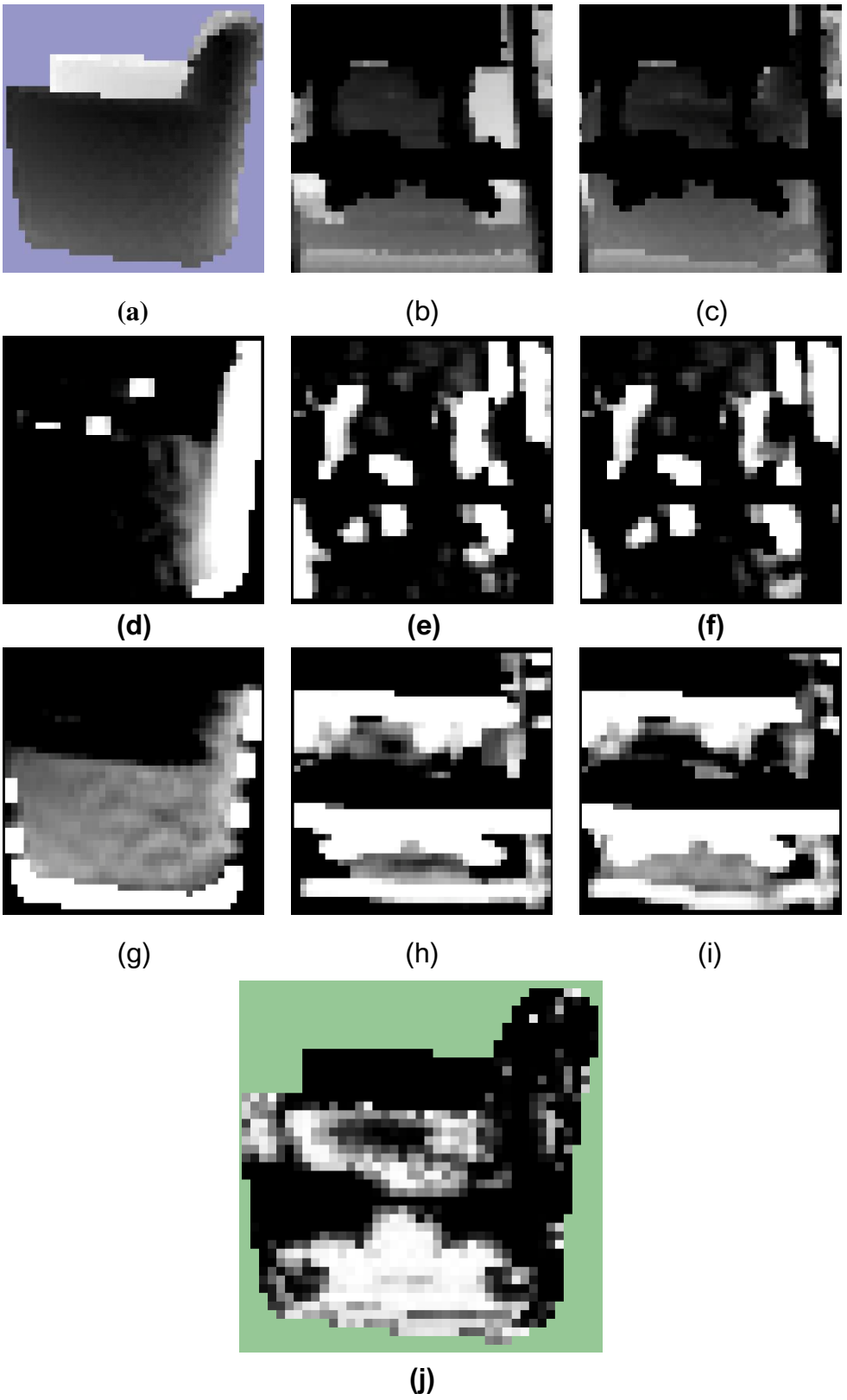


Figure 4.7: View simulation example for the false positive shown in Figure 4.5. **(a)**: The simulated view of the object. **(b)**: The corresponding part of the scene. **(c)**: The part of the scene with the object simulated inside. **(d)**: The gradients of (a) in x-direction. **(e)**: The gradients of (b) in x-direction. **(f)**: The gradients of (c) in x-direction. **(g)**: The gradients of (a) in y-direction. **(h)**: The gradients of (b) in y-direction. **(i)**: The gradients

of (c) in y-direction. **(j)**: The scores for every pixel from black (score 0) to white (score 1). Irrelevant pixels are marked green.



Figure 4.8: Example images from a dataset of a rotating chair, both as color images and range images.

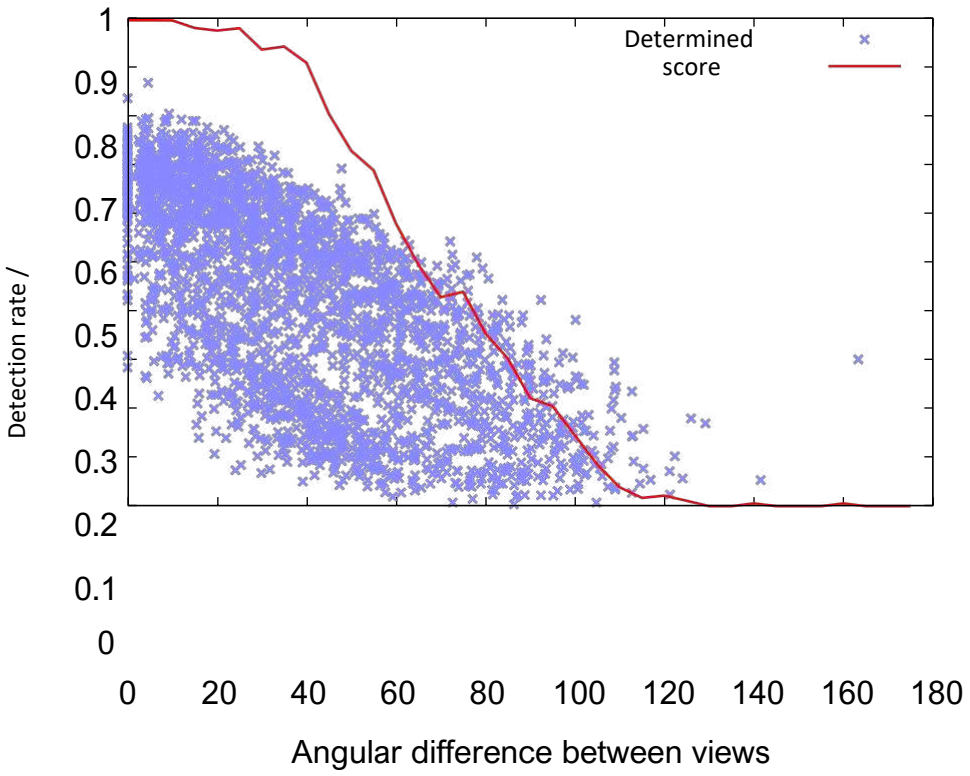


Figure 4.9: This plot shows experimental results regarding the detection of an object, in this case an office chair, for which we do not have a complete point cloud model, but a single view of the object instead. We show the detection rate for different angular differences in the perspective and also the determined scores for the successfully detected object poses.

chair from equally distributed perspectives around it. We used a Microsoft Kinect for this pur- pose. See Figure 4.8 for example images. We manually labeled the true positions of the object in the scans to acquire close-to-ground-truth data. We then used each of the 80 perspectives as a single-view object model and tried to detect the chair

in the entire collection of scenes. The re- sult is summarized in Figure 4.9. The plot shows the rate with which the correct object position was found, depending on the angular difference between the perspectives. One can see that the detection rate is above 90% for differences below 40°. For higher angular differences it drops quickly. The plot also shows the determined validation point scores (see Section 4.4.1) for the individual detection processes where a correct pose could be found. Typical validation point scores for small angular differences are between 0.3 and 0.8. The lower values typically occur for unfavorable perspectives of the chair, meaning side views where the backrest is visible at a steep angle. Please note that we did not perform the view simulation step (see Section 4.4.2) for this experiment since this process explicitly requires a complete model.

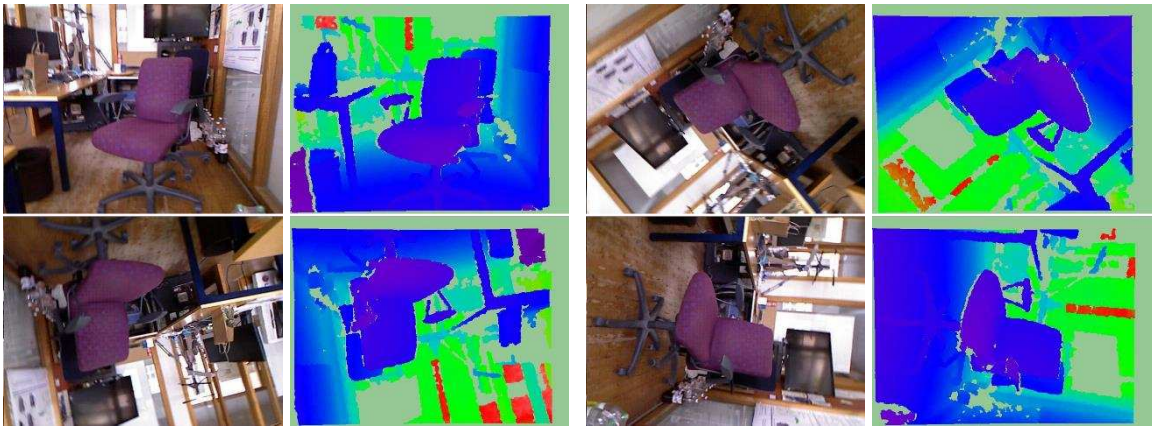


Figure 4.10: Example images from a dataset of a chair, captured with a rotating depth camera, both as color images and range images.

4.6.2 Rotational Invariance

In a second experiment we evaluated the option for rotational invariance for the NARF de- scriptors (see Section 3.3). The NARF descriptor calculation has the option to normalize the descriptor regarding the rotation around the normal, thereby making it rotationally invariant. If this option is not used, the descriptors will not match, e.g., for a top right rectangular corner and a top left rectangular corner. This can be advantageous in situations where objects have a typi- cal rotation in which they occur. A chair, e.g, typically stands upright on its feet. When this is the case, reducing the search space simplifies the problem. To compare the two options against each other, we captured a dataset of the same chair as used in Section 4.6.1. This time we did not rotate the chair, but the camera 360° around its center axis. See Figure 4.10 for example images. We captured 235 3D scans in this situation, again using a Microsoft Kinect and again manually labeling the true positions of the chair in the scans. We then searched for the model of the chair with and without the option for rotational invariance. Figure 4.11 summarizes the result. The plots show the score of the correctly found object poses for angular differences be- tween the model and the instance in the scene. A score of 0 means that the pose was not in the set of candidate poses. The NARF version with and without rotational invariance are shown in different colors. While the rotationally invariant version was able to find a pose for all angular differences, the version without rotational invariance could only reliably find poses for small angular differences (up to 25°). Yet, the spatial verification step for the rotationally invariant version took 300 ms on average per scene, whereas the version without rotational invariance took only 80 ms. This is caused by the increased size of the search space if the rotation around the normal is also a free parameter. The decision to use the NARFs with or without rotational invariance should therefore depend on the actual application.

4.6.3 Performance for Office Environments

In the next experiment, we tested the performance of our approach in an office environment. Specifically, we applied our method to the dataset already presented in Section 3.4.2, where we used it to evaluate the performance of our NARF keypoints and descriptors. This dataset includes seven point cloud models of very different kinds

of objects. Five furniture sized objects (an armchair, a cart, an office chair, a Pioneer robot, and a teddy bear) and two tabletop objects (a cup and a stapler). See Figure 3.7 for photos of the used objects. In addition to the models, the dataset includes 10 scenes captured with a 3D laser range finder in a typical cluttered office

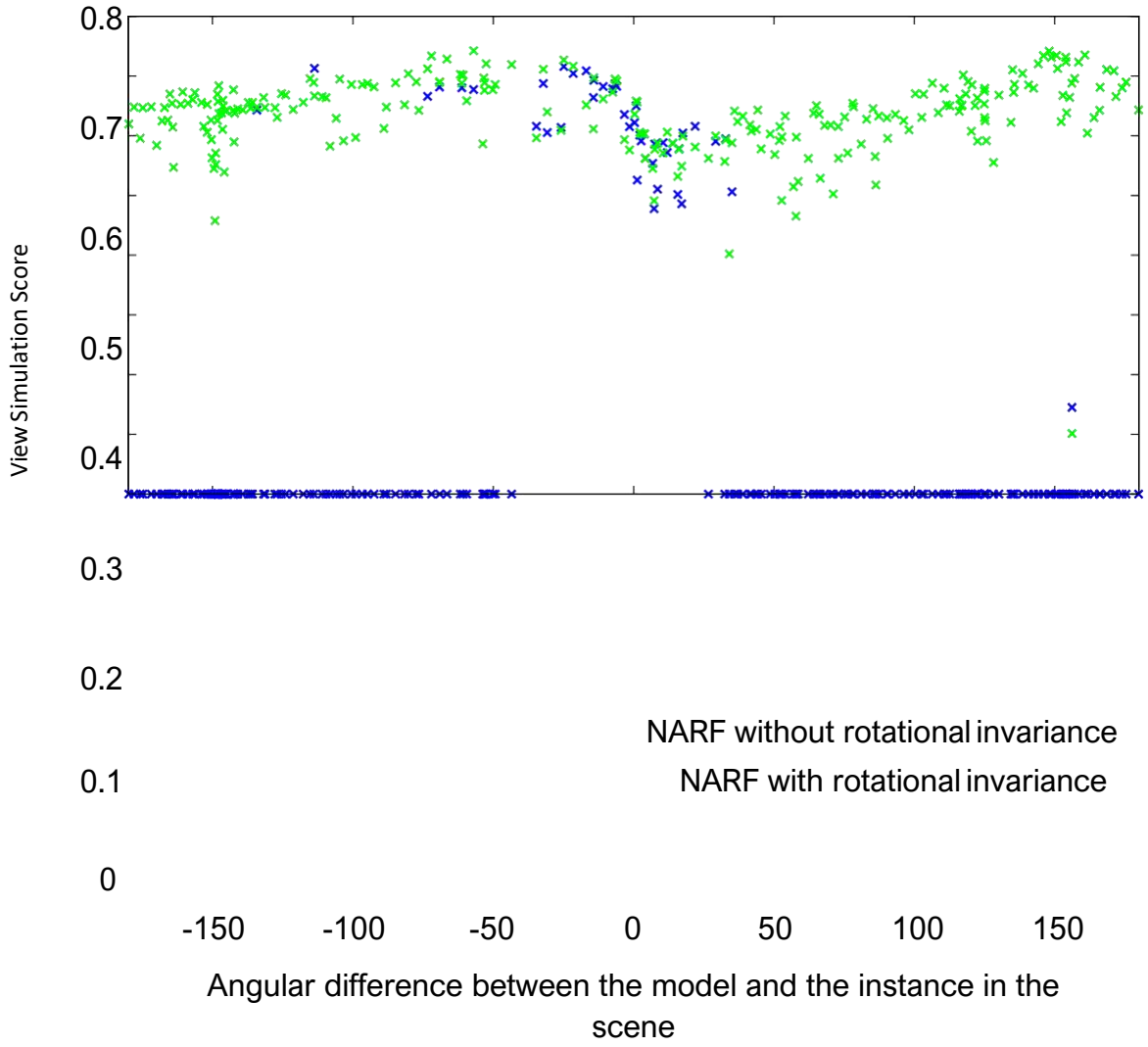


Figure 4.11: This plot shows experimental results regarding the detection of an object, in this case an office chair, with the sensor rotating around its center axis. The dots mark the scores the system determined for the correctly found object positions. A score of 0 means the object was not found. The two colors distinguish the NARF versions with and without rotational invariance. Wile the rotationally invariant version is able to find the correct pose for all rotations, the other version is only able to find poses with low angular differences between the model and the instance in the scene.

environment and 10 tabletop scenes captured with a stereo camera system. By artificially adding the models to the scenes we created 70 scenes with one known object in each. See Figure 3.9 for examples of these scenes. In Section 3.4.2 we already used this dataset in an object recognition context, yet we only analyzed the performance of the keypoints and descriptors, without the spatial verification steps that we introduced in this chapter in Section 4.4. We used our object recognition system to detect the five furniture sized objects in the 50 office scenes and the 2 tabletop objects in the 20 tabletop scenes. Since the true object positions are known, we are able to determine for each found object position if it is a true positive or a false positive. In addition, we

can also determine situations, where the position of the object was correctly determined regarding the translation, but with a wrong orientation. Figure 4.12 shows two plots that summarize the outcome. We plot the validation point score (Figure 4.12(a)) and the view simulation score (Figure 4.12(b)) for every found object position, indicating which were true positives, true positions with wrong orientations, and false positives. It can be seen that both methods distinguish well between false positives (low scores) and true positives (high scores). As expected, the view simulation gives the better results, yet the runtime requirements for this method are much higher (see below). As already discussed in Section 3.4.2, the cup is a very symmetrical object and only the handle makes it possible to clearly determine its orientation. This property is visible in the plots, since there is an overlap in the score values of the cup between the true positives and the true positions with wrong orientations. These plots indicate that simple thresholding on the score values enables us to distinguish true object positions from wrong ones. This is even better visible in Figure 4.13, where we plot the recall rates for the different objects, depending on different acceptance thresholds for the view simulation score values. We also plot the number of false positives found for each acceptance threshold. Here we see that nearly all of the objects were found in the scenes and got scores above 0.6. Only the cart was missed once, meaning that the correct pose was not in the set of potential poses

determined by the feature matches. Using an acceptance threshold of 0.65 would produce recall rates of 90% and above for most of the objects, while not returning any false positives. Please note that we did not consider the poses with correct 3D positions but wrong orientations in this plot.

The runtime requirements for this experiment were as follow. For the office scenes with furniture sized objects we used an angular resolution of 0.5° to create range images. The range image creation took 15 ms, the feature extraction 80 ms, and the feature matching 15 ms on average. Spatial verification using only validation points took 250 ms, using only view simula- tion 7,500 ms, and using a combination (first validation point scoring and then view simulation scoring for all results above 0.3) 350 ms on average. This confirms our decision to use this combination, since it combines the better scoring results of the view simulation with the faster runtime of the validation point scoring. For the tabletop scenes we used an angular resolution of 0.25°. The range image creation took 12 ms, the feature extraction 10 ms, the feature matching 1 ms, spatial verification with only validation points 60 ms, spatial verification with only view simulation 3,500 ms, and spatial verification using a combination 230 ms.

4.7 Related Work

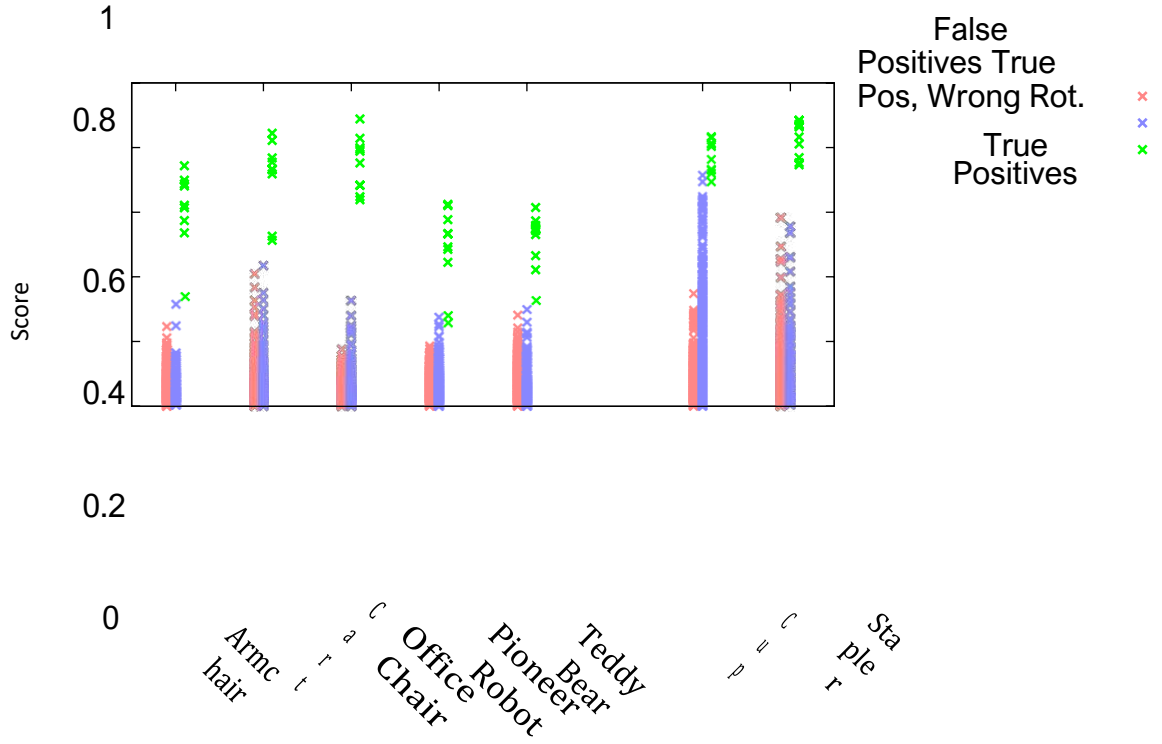
Detecting objects based on point feature matches is an established concept in computer vision. E.g., the popular SIFT algorithm proposed by Lowe [82] was originally designed for an object recognition task. In this approach, SIFT correspondences between a query image and a database of reference images are combined into clusters if they agree on a possible object pose. The probability to be a true match is computed based on how well the matched features fit and on the number of probable false matches.

Our approach operates on 3D range data. While spatial reasoning is much more convenient in this kind of data, it is typically less descriptive than visual textures. Johnson *et al.* [65] originally proposed the popular *spin-images* in an object recognition context using 3D scans. The approach computes a spin-image for every point in the model and every point in the scene and then groups correspondences together to calculate plausible transformations, which are then verified based on the overlap. Triebel *et al.* [142] used spin-images as features for an associative Markov network. The parameters of this network are learned from a manually labeled training dataset of objects and are subsequently used to find object labels for each 3D point. This means the approach does not identify actual instances of an object but determines the most likely class to which individual points belong.

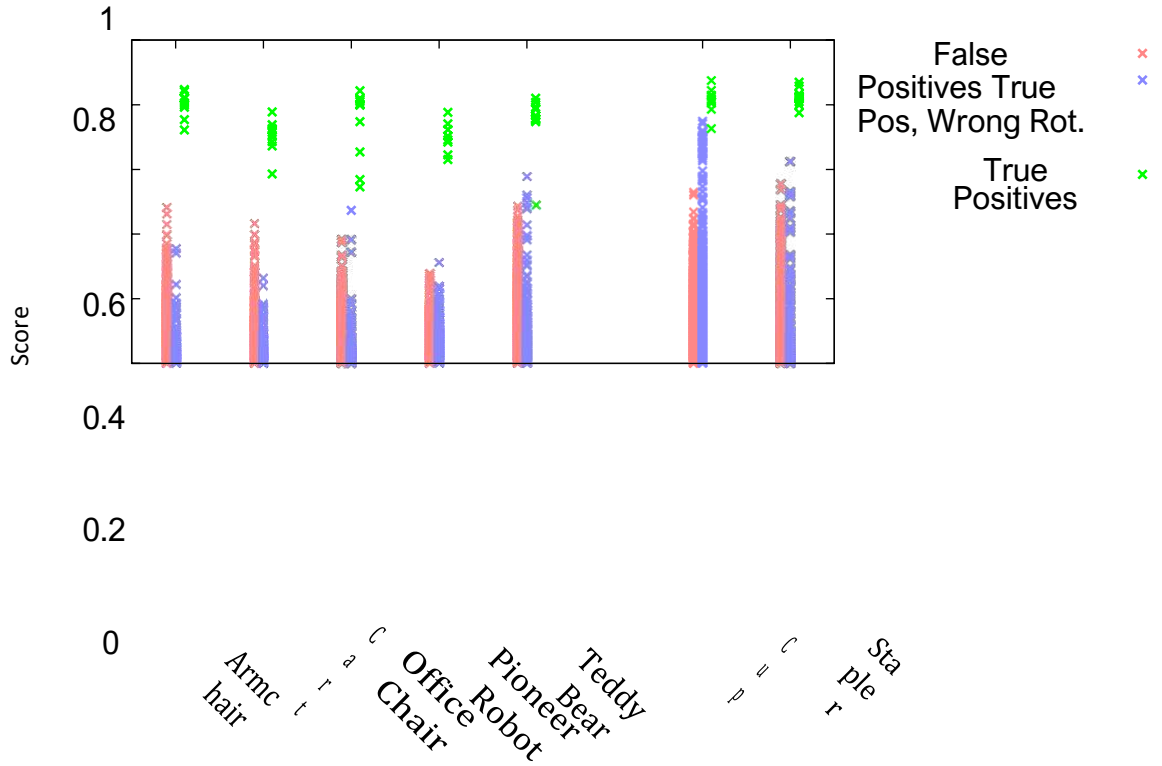
Stiene *et al.* [132] introduced an approach for object detection based on silhouettes extracted from range images. They use a feature type that is based on a fast Eigen-CSS method and a supervised learning algorithm. The approach is similar to ours in the sense that it works on range images and explicitly considers the outer shape of objects. But being restricted to the complete contour of the objects makes this approach less resistant to partial occlusion.

Blum *et al.* [12] described an object recognition approach relying on feature descriptors that are learned from RGB-D data in an unsupervised fashion, whereas they used SURF for the extraction of keypoints. The object recognition process itself is

performed using a Support Vector Machine that was trained based on the feature descriptors appearing on a labeled dataset. With this, the approach is able to determine the probable existence of an object in a certain area of the input image. The authors then propose a method to determine the 6DoF position of the object based on feature correspondences and refinement with ICP. Yet the authors do not present a method for spatial verification of the found object poses.



(a)



(b)

Figure 4.12: Experimental results for object recognition in an office environment. The plots show the scores for all evaluated object poses for the individual objects. The marked points are color coded, distinguishing between false positives, true positives and correctly found 3D positions that have wrong orientations. **(a)** shows the results for

the validation point scoring and **(b)** shows the results for the view simulation scoring. For both methods the highest scores are exclusively true positives, whereas the view simulation divides the classes better, as expected.

Rusu *et al.* [112] presented a 3D feature descriptor that encodes geometry and viewpoint information. The authors use it to simultaneously recognize objects and determine their pose. In this approach, a complete object view is described with a single feature descriptor. The system allows a very efficient retrieval of similar objects for a query from a database, but relies on a good initial segmentation of the object, which is not always possible in cluttered scenes. In addition, using a single descriptor for the whole object makes the system less robust regarding partially occluded objects.

4.8 Conclusions

In this chapter we presented an object recognition system that successfully employs the NARFs we introduced in Chapter 3. The point feature correspondences are used to calculate potential object poses, thereby reducing the size of the search space substantially compared to exhaustive

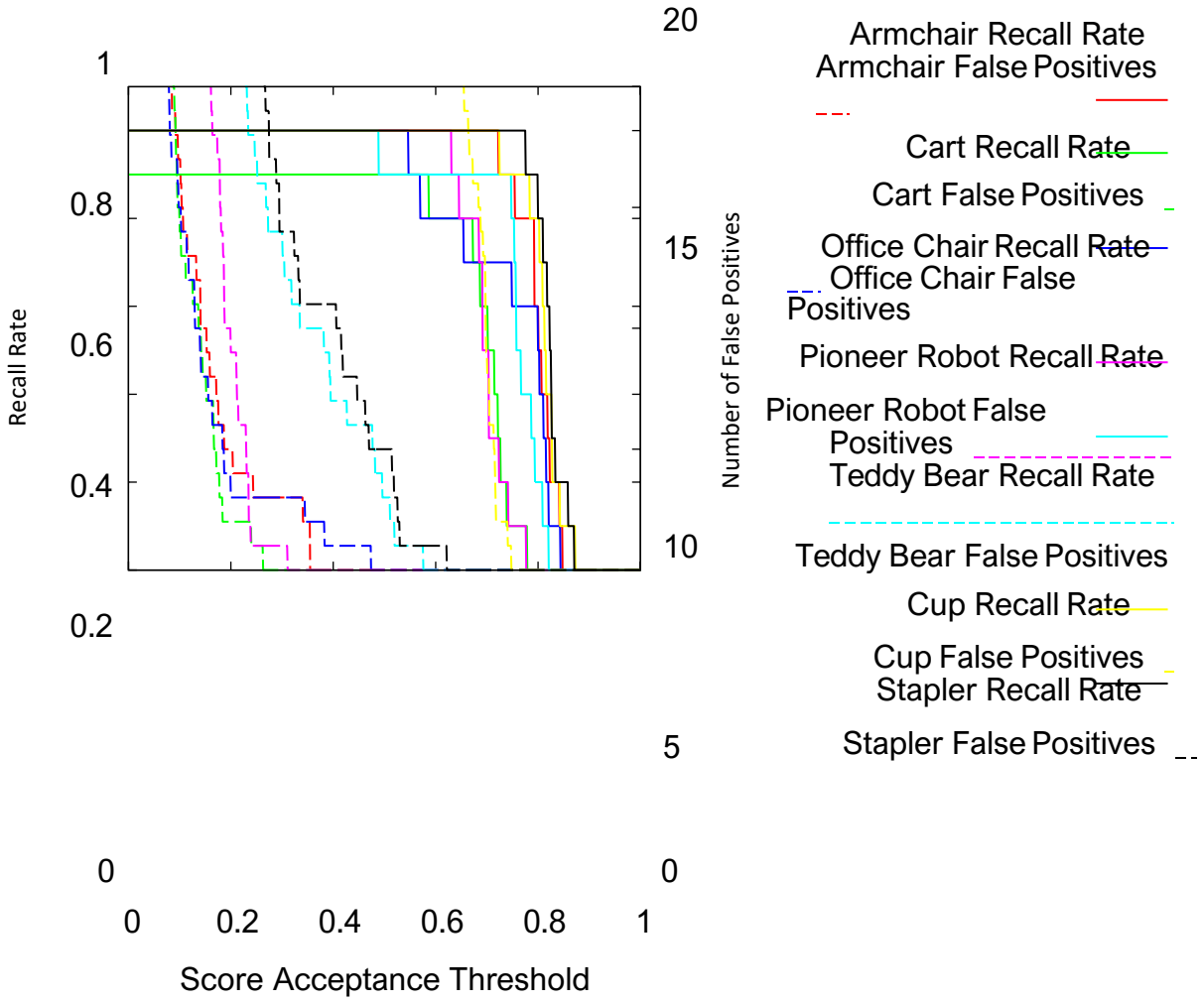


Figure 4.13: Experimental results for object recognition in an office environment. The plot shows the recall rates and the number of false positives for the individual objects, plotted against the acceptance threshold for the view simulation score. For acceptance thresholds slightly above 0.625, there are no false positives and recall rates of 80%-100%.

search. To calculate candidate object poses from the feature matches we introduced a novel algorithm for the selection of sets of correspondences that makes use of the descriptor distances. It returns clusters of correspondences that are most likely to belong to a true detection. The candidate object poses are evaluated based on two novel spatial verification methods. One very fast method based on a limited number of precomputed points, which is used to remove a large number of the false positives in advance. Second a computationally more demanding method, which simulates the expected view and compares it with the actual measurements. We presented experiments using real data from 3D laser range scanners and depth cameras. They show the capabilities of our system to recognize objects robustly, even in the presence of substantial clutter and for arbitrary 6DOF positions of the objects in the scenes. In addition, we demonstrated that even if no complete object model is available, a single view of the object can

be used to detect it reliably, for viewpoint changes of up to 30°.

Chapter 5

Unsupervised Learning of 3D Object Models from Partial Views

The creation of 3D object models is highly relevant in robotics since such models are a prerequisite for basic tasks like object recognition. Often such models are created manually in a CAD program or by scanning objects in specific setups, e.g., on a turntable. This process is often tedious regarding the involved amount of human labor. This chapter presents an approach to build object model point clouds from a set of 3D range scans in an unsupervised fashion. The system searches for partial views potentially belonging to the same object and merges them into an accurate 3D representation of the object. We use methods already applied for object recognition tasks in the previous chapter to find overlapping areas and perform the alignment. The approach performs spectral clustering on a similarity matrix of partial object views to decide which of them should be merged into one object model.

* * * *

In Chapter 4 we presented an approach for model-based object recognition. The models for this process were taken from public databases or created manually by capturing different views of an object and merging them. In this chapter we present an alternative to this model creation procedure, which learns object models in an unsupervised fashion. It applies methods from the object recognition system presented in Chapter 4 to find and align overlapping areas of partial views, belonging to the same object. These partial views in a sequence of 3D range scans do not necessarily have to belong to the same instance of an object and can be captured in different places at different times. Multiple instances of an object visible in a single scan are also utilizable. See Figure 5.1 for an example application where models from a wooden chair, a garbage bin, and an office chair are created from three, two, and one partial views visible in a single scan respectively.

Our approach first determines potential partial objects from the input point clouds. These are then iteratively merged into more complete point clouds of the objects until our system can not find enough similarity between the remaining models anymore.

The remainder of this chapter is organized as follows: In Section 5.1 we will discuss, how our system works, first as a rough overview in Section 5.1.1, then in more detail starting from Section 5.1.2. We present our experimental evaluation in Section 5.2. Afterwards we will discuss related work in Section 5.3, followed by our conclusion in Section 5.4.

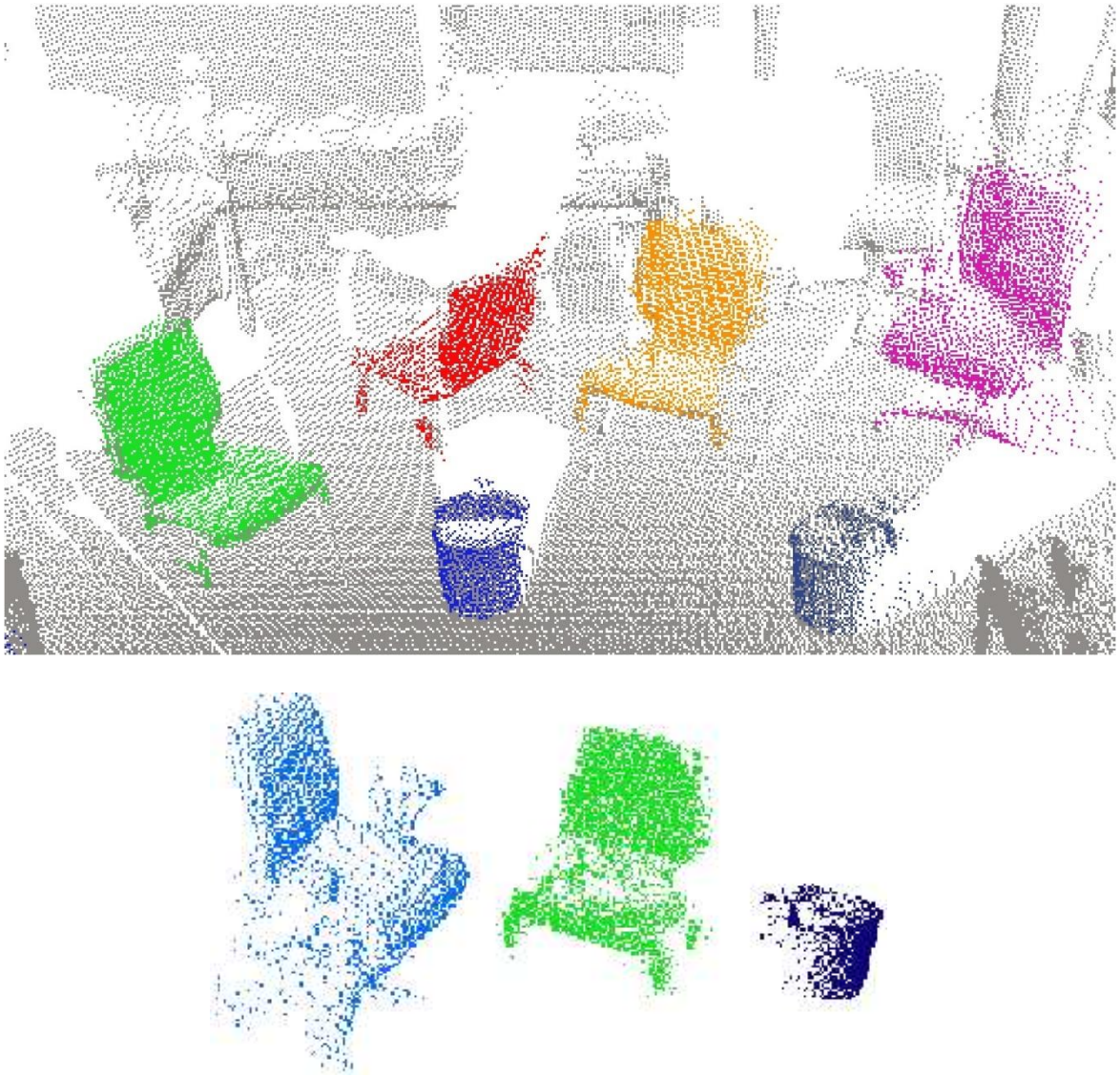


Figure 5.1: Motivational example for our model learning system. The top image shows a single 3D scan with multiple objects in the foreground: three instances of a wooden chair, two instances of a garbage bin, and one instance of an office chair. Our system merged the different views of the objects that were present multiple times and created more complete models. Those are shown in the bottom image. (Source: [104])

5.1 Unsupervised Object Model Learning

We will now describe our approach to learn 3D point cloud object models from partial views, as they are typically observed by 3D laser scanners or range cameras. We begin with an overview of the system and will then describe the individual parts in detail.

5.1.1 Overview

For a given sequence of 3D range scans our method performs the following steps to create models of the contained objects from them:

1. We first remove background elements like the floor, the walls, and the ceiling

using a plane extraction method.

2. We then perform spatial clustering on the remaining points in the scans to find sets of connected points.

3. The clusters above a certain size are then used as an initial set of partial object models.

4. These partial models are then merged in an iterative fashion:

    - First we determine a crosswise similarity measure between all of the partial models, which we specifically designed to work with partial views. We also determine a relative transformation between all model pairs, which can be used to merge the point clouds into a common coordinate frame according to the best found fit.
    - Then we use spectral clustering on the resulting similarity matrix to decide which models should be merged into a new one, which then replaces the original partial models.
    - If at least two models were merged into a new one, we repeat step 4. Else we can stop the process since no two models were similar enough anymore.

We will now discuss the individual steps in more detail.

5.1.2 Background Extraction

The first step for our system is to remove parts of the environment from the 3D scans that are not relevant for our application. For this purpose we remove points belonging to large horizontal or vertical planes from the scans. These structures typically represent the floor, the ceiling, or walls, which all are not interesting for our goal to find partial object models. They could also interfere with the clustering step we will discuss in Section 5.1.3.

The plane extraction procedure we use employs a region growing approach, which itera- tively adds points to a plane if it fits well to it and then updates the plane estimation accordingly, until no fitting point can be found anymore in a local neighborhood. In our experiments we used
0.25 m^2 as the minimum size of a plane to be removed. If this value is to small, we might risk to remove parts actually belonging to an object we are interested in. See Figure 5.2(a) and (b) for an example of this procedure.

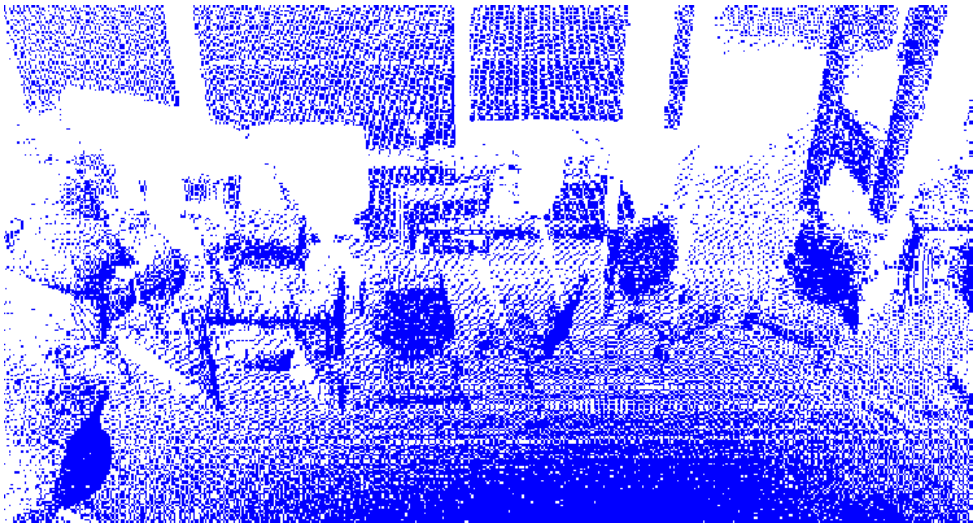
5.1.3 Spatial Clustering

After we removed the background from the 3D scans, we perform a spatial clustering step to identify individual objects in the scene. Like for the plane extraction, we do this using a region growing approach. We first select a random point from the scene and add it to the current cluster. Then we iteratively add points to this cluster that are spatially close to at least one point already in the cluster. This is done until no close by point can be found anymore. We can now add this new cluster to our set of clusters and remove the contained points from the 3D scan. All point clusters above a predefined size now form the initial set of partial objects, which we will try to merge in the next step.

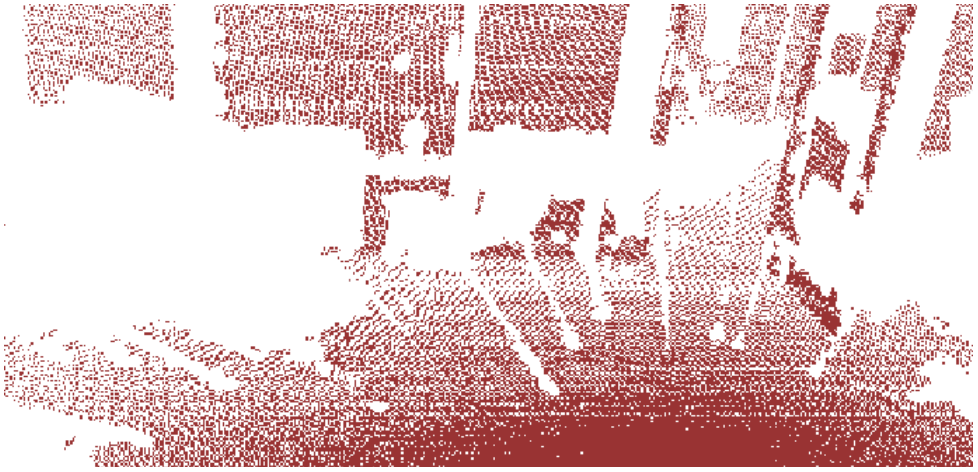
Please note that the removal of the floor, which was done in the last step, is crucial for this procedure since most of the objects are typically connected by the floor, which would then lead to one big cluster, preventing us from identifying individual objects in the scene.

See Figure 5.2(c) for an exemplary outcome of this procedure.

(a)



(b)



(c)

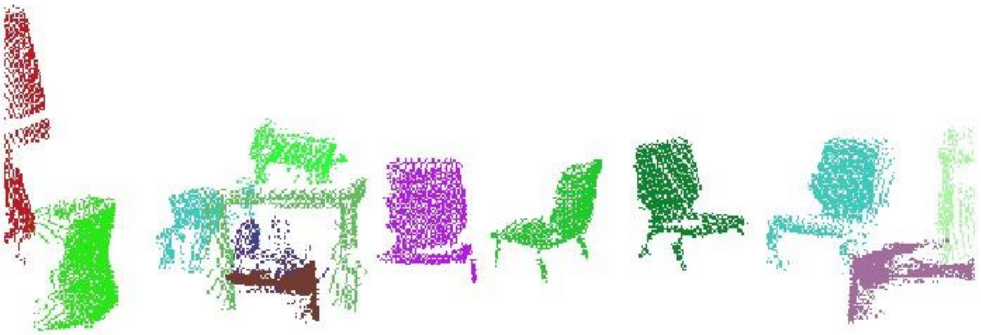


Figure 5.2: (a): Example point cloud of an office. **(b)**: Found large horizontal and vertical planes. **(c)**: The structures remaining when the planes are removed. The different colors represent the found clusters. (Source: [104])

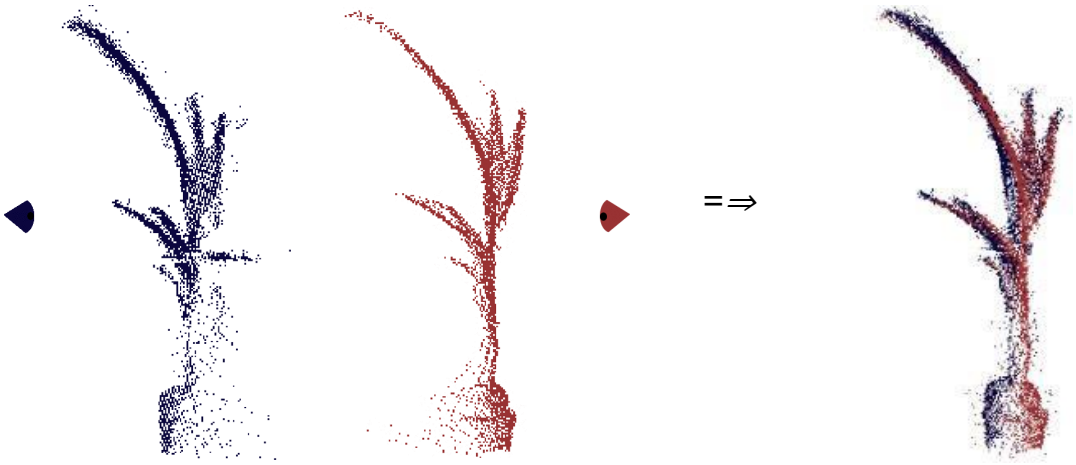


Figure 5.3: This figure shows an example how two partial models of a plant are merged into one. The triangles mark the original viewpoints from which the scans were perceived. (Source: [104])

5.1.4 Determining Partial Similarity and Relative Poses

We now have a set of partial object models $\mathbb{M} = \{M_1, \ldots, M_{|\mathbb{M}|}\}$. Every object model contains a set of point clouds $P^M = \{P_1^M, \ldots, P_{|P|}^M\}$ and a set of viewpoints $T^M = \{T_1^M, \ldots, T_{|P|}^M\}$, meaning the poses from which these point clouds were originally observed.

Let $\hat{P}^M$ be the merged point cloud incorporating all original view point clouds:

$$\hat{P}^M = \bigcup_{i \in \{1,\ldots,|P^M|\}} \{\hat{\mathbf{x}} \mid \hat{\mathbf{x}} = T_i^M \cdot \mathbf{x} \quad \forall \mathbf{x} \in P_i^M\}. \tag{5.1}$$

It is now our goal to find similar regions in pairs of models and to determine potential alignments between them. For this purpose we employ a similar method as the one we proposed for object recognition in Chapter 4. Let us consider a pair of models M_i and M_j. As we did in the object recognition system (see Section 4.1) we now sample different views of the models, meaning we calculate range images from different perspectives (see Section 4.1), making sure that the original viewpoints T^M are also used. Then we calculate point features on the different views (we used a support size of 30 cm in our experiments), match the features from M_i against those from M_j and use the found matches to calculate relative transformations between the two models. Since we only need a small number of matches, with a minimum of one, we need only a partial overlap of the two models to find a correspondence between them (see Section 4.3). Afterwards, to reduce the error of these transformations, we apply an additional ICP step. Before this step, the typical error for true relative transformations is about 10 cm in translation and about 2° in orientation. After the ICP step it typically decreases to 3 cm in translation and about 0.5° in orientation.

Figure 5.3 shows an example for our model merging procedure. It shows the point clouds of a plant that is visible from two different perspectives in two different scans, which are then merged into a new, more complete model.

We now need to calculate a score for each of the candidate transformations. We do this based on view simulations regarding the original partial views and the merged model. This process is similar to what we did in Section 4.4.2 to score candidate poses

in the object recognition system. The main difference is, that we now try to find inconsistencies in multiple observations of the same structure, whereas we only evaluated one perspective of a complete model in the object recognition system. Our scoring method for the potential relative poses between models

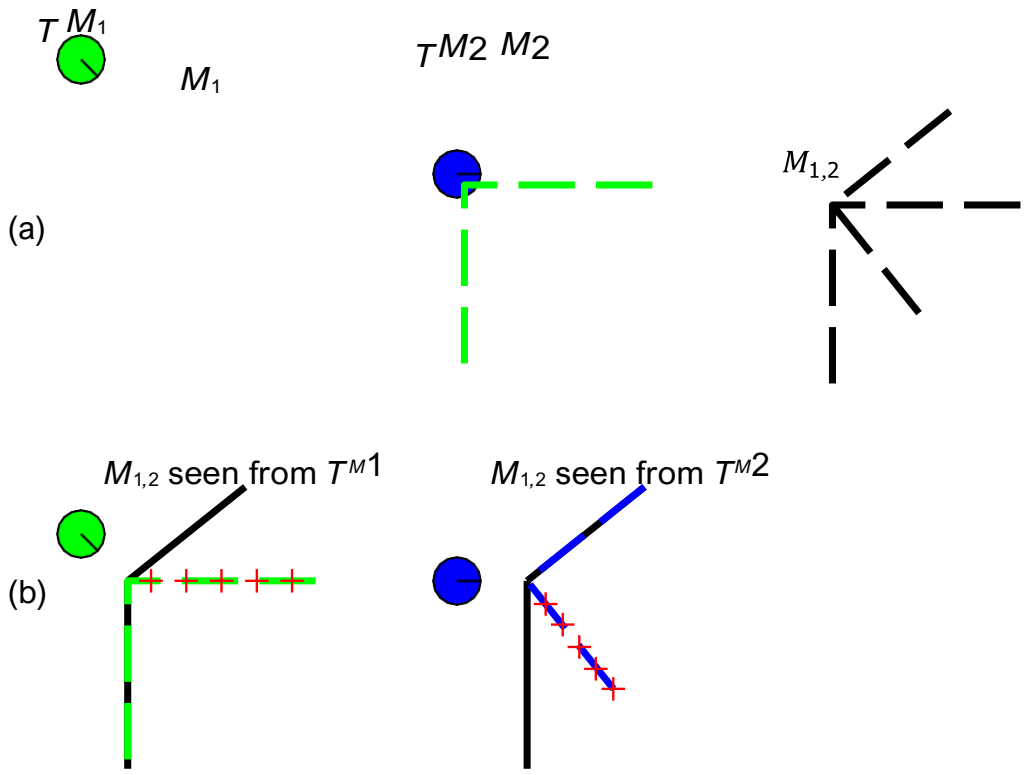


Figure 5.4: Visualization of the idea behind the scoring function. **(a)** shows the two models M_1 and M_2, which only include one point cloud from one view each. The perspective from which these were originally seen, T_M1 and T_M2 are also shown. The rightmost structure shows the combined model $M_{1,2}$ resulting from a wrong alignment. **(b)** shows view simulations of $M_{1,2}$ from the perspectives defined by T_M1 and T_M2, overlaid by the corresponding original view. Assuming that the merged model is correct, the points marked by red crosses should not have been visible in the original view. Therefore this match will get a low score.

works as follows:

Let us assume that we are evaluating the potential relative pose $T_{i,j}$ between the models M_i and M_j, which leads to the combined model $M_{i,j}$. We now calculate range images for all of the original perspectives of the source models, namely the ones defined by T_{Mi} and T_{Mj}, for the point cloud of the merged model $M_{i,j}$. We then compare these range images with their corresponding original observations. These range image views will most likely differ also for correct relative poses, but inconsistencies show themselves as areas where we get larger ranges for the original view than for the merged model. This indicates that areas were visible that should have been occluded if the merged model were correct. Figure 5.4 visualizes this situation in a two-dimensional example.

To compute the score for one of the views in a merged model, we consider all the range image pixels that were visible in the original view. The merged model will typically have more valid pixels, but since we want to evaluate how well the original observation

fits the model, we can only consider pixels that are visible in both, the original and the simulated view. For each of these we calculate the difference in range Δr. Based on a maximum error of the difference in range e_{max}, we define the score of this particular pixel as

$$s = 1 - \frac{\min(|\Delta r|, e_{max})}{e_{max}}, \tag{5.2}$$

which means that $s \in [0, 1]$. The score for the complete view is the average over all valid pixels. The score for the complete model is the average over all the scores of the individual views. To reduce the runtime requirements, we first perform this scoring step at a low resolution

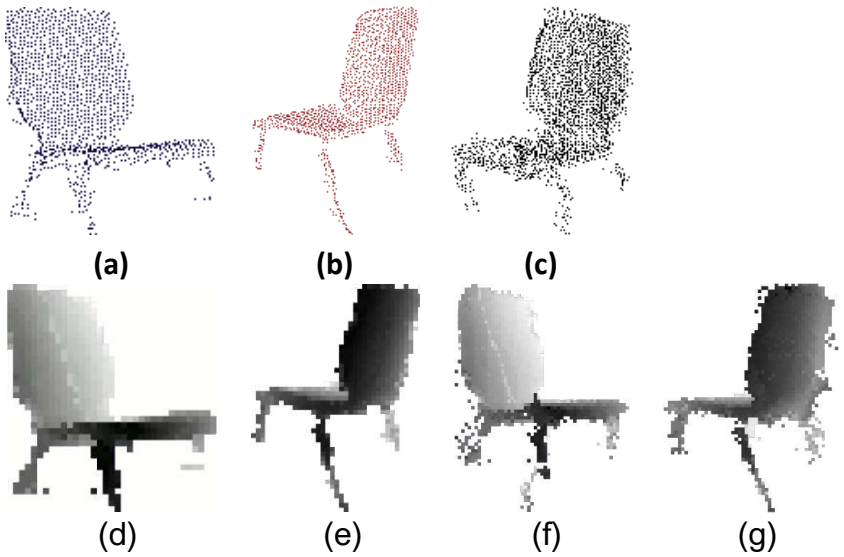


Figure 5.5: Example for the views used in the scoring process. **(a)** and **(b)** show the original views of a wooden chair. **(c)** shows the merged model according to a correctly found transformation between the two partial models. **(d)** and **(e)** show the range images corresponding to (a) and (b). **(f)** and **(g)** show the range images corresponding to the merged model from (c), seen from the same perspectives as (d) and (e). For the calculation of the score of the model shown in (c), the range image from (f) will be compared with (d) and the range image from (g) with (e). (Source: [104])

(1° per pixel) and only repeat it at a higher resolution (0.3° per pixel) if the score in the first step is above a minimum value. Figure 5.5 shows an example for the scoring step regarding the
model of a wooden chair.

The relative poses calculated from the feature matches typically return multiple solutions for the same pair of models. We calculate the score for all the solutions, but keep only the one with the highest score as the best relative transformation between the models. Based on these best relatives transformations we build a similarity matrix for all our models, where the M M elements are the scores of the best relative transformations of the corresponding model pairs. See Figure 5.6 for an example of such a matrix. Please note that we define a minimum score for each similarity value in the matrix, below which we set the entry to zero (0.7 in our current implementation).

5.1.5 Merging of Models

We now have to decide, which models are to be joined during one iteration of our algorithm. For this purpose we employ the single cluster graph partitioning algorithm [97] on the similarity matrix. This method tries to find the single best cluster of well connected nodes in a graph by maximizing the average consensus. In our context the partial models are the nodes in a fully connected graph, whereas the found similarities provide weights for the edges. The method boils down to finding the dominant eigenvector of the similarity matrix, which can then be discretized into an indicator vector where each element is either 1, if the corresponding model is part of the main cluster, or 0 if it is not. If the best found cluster according to this method includes at least two models, we merge them into one new model and replace the original models

with the new one. Then we go back to the calculation of the similarity matrix for this new set of models, thereby starting the next iteration. Please note that only the entries affected by the removed and added models need to be recomputed in the next step. We will now present



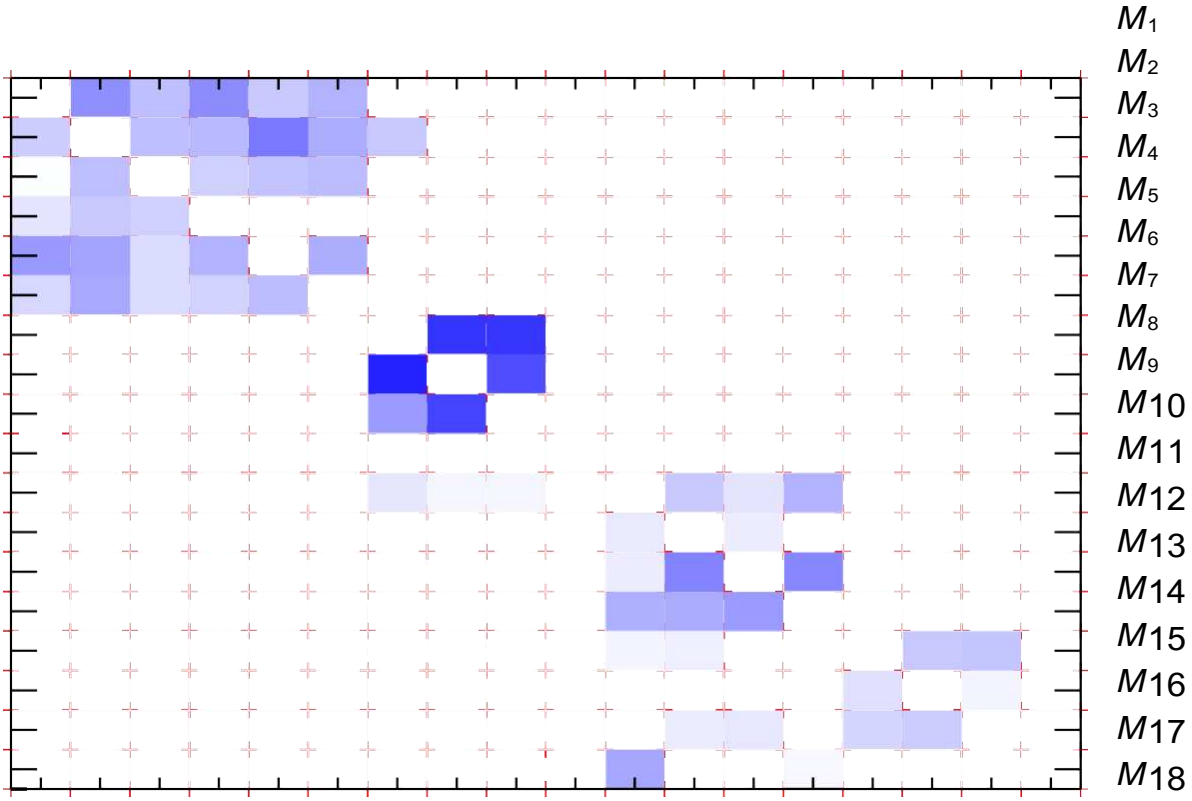
M_1 M_2 M_3 M_4 M_5 M_6



M_7 M_8 M_9 M_{10} M_{11} M_{12}



M_{13} M_{14} M_{15} M_{16} M_{17} M_{18}



M_1 M_2 M_3 M_4 M_5 M_6 M_7 M_8 M_9 M_{10} M_{11} M_{12} M_{13} M_{14} M_{15} M_{16} M_{17} M_{18}

Figure 5.6: Individual model views that were extracted from the list of scans and the similarity matrix. The darker the value in the matrix, the higher the score value for the best found relative transformation between the two partial models. (Source: [104])



Figure 5.7: The used objects for the experiment. A plant, a garbage bin, a small pioneer robot, a wooden chair, a monitor, and an office chair. (Courtesy of Michael Ruhnke)

the experimental evaluation of our approach.

5.2 Experimental Evaluation

To evaluate our approach, we captured datasets of 3D scans using a tilting SICK laser scanner, from which we can extract range images of up to 0.3° per pixel. We will now present two experiments. In the first one we will show how the set of models evolves during the iterative procedure. In the second one we will analyze how the number of available viewpoints influences the result.

5.2.1 Development of the Model Set

For the first experiment we acquired seven 3D scans in a cluttered office environment. After background subtraction and clustering, our system extracted an initial object set containing 18 partial models. Figure 5.6 shows this initial set of partial models with the corresponding similarity matrix. Please note that we do not calculate the similarity of an object to itself, which is why the diagonal in the matrix is empty. For better visibility we ordered the partial views according to the object they belong to in this visualization. $\{M_1, M_2, M_3\}$, $\{M_4, M_5, M_6\}$ are partial views of a plant, $\{M_7, M_8, M_9\}$ of a garbage bin, M_{10} of a small pioneer robot, $\{M_{11}, M_{12}, M_{13}, M_{14}\}$ of a wooden chair, $\{M_{15}, M_{16}, M_{17}\}$ of a monitor, and M_{18} of an office chair. See Figure 5.7 for photos of the used objects.

The blocks in the matrix already show that our similarity measure properly separates the different objects, although there are some pairs that falsely received high similarity values, like M_{11} and M_{18}. Yet, as we will see, this does not necessarily lead to wrong object merges.

Figure 5.8 shows how the set of models evolved after two iterations. Some of the models of the plant, M_1, M_2 and M_3, are already correctly merged. The garbage bin models M_7, M_8, and M_9 were also correctly merged. Yet, since the garbage bin is completely symmetrical, our system was only able to merge them by maximizing the surface overlap and was not able to reconstruct the complete round structure. This is a general problem of a lack of structure and could only be solved with additional data about the captured scans.

The final model set is shown in Figure 5.9. It took 18 minutes on a 1.7 GHz Pentium 4 processor until our system converged to this state. There are no falsely merged models in the final set and our system only failed to merge M_{15} with $M_{16:17}$, due to a very limited overlap. Additional views could enable our system to also merge these observations correctly. This behavior will be demonstrated in the next experiment.

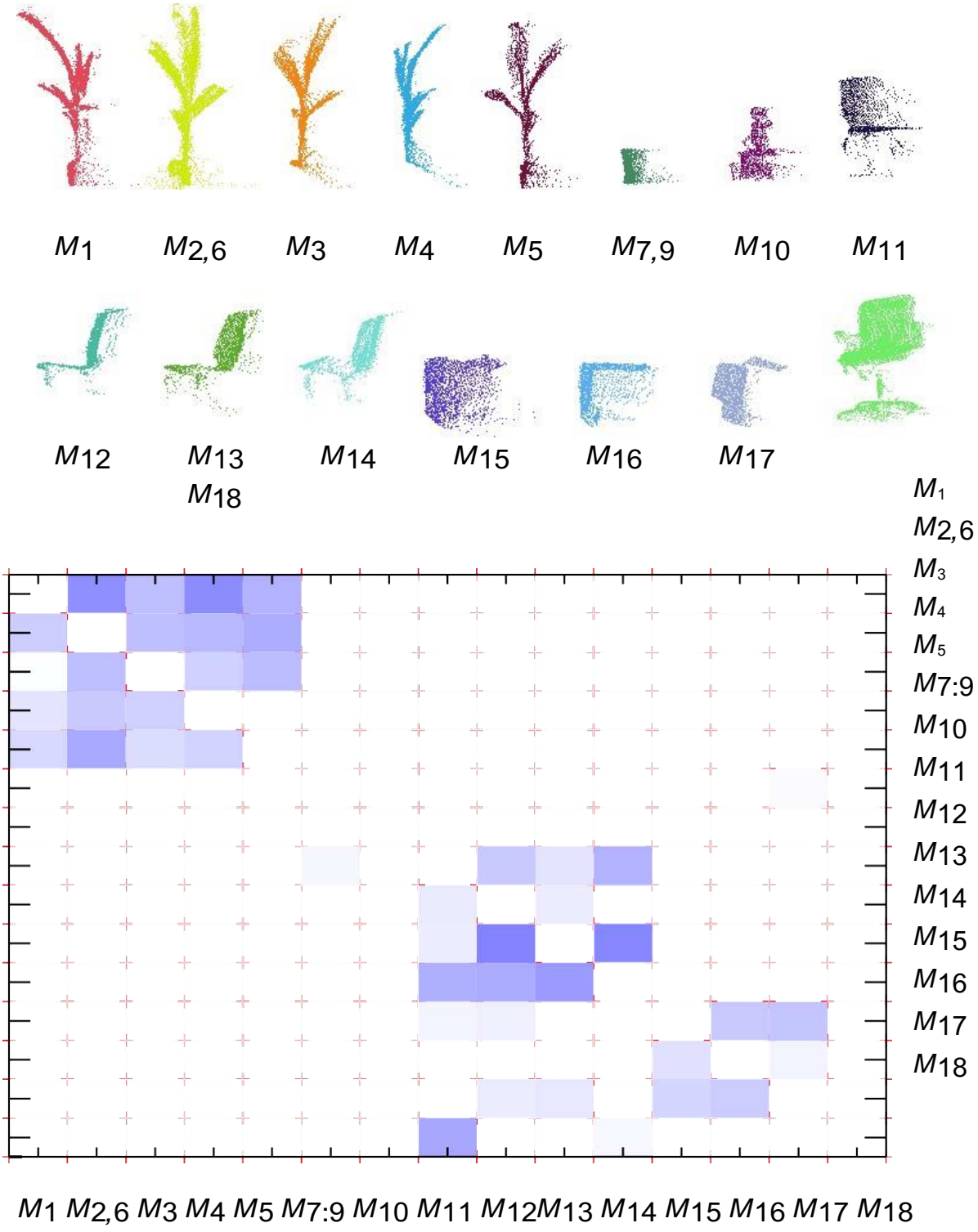


Figure 5.8: Individual model views and similarity matrix after two iterations of our model merging procedure. Two views from the plant and three from the garbage bin are already correctly merged. (Source: [104])

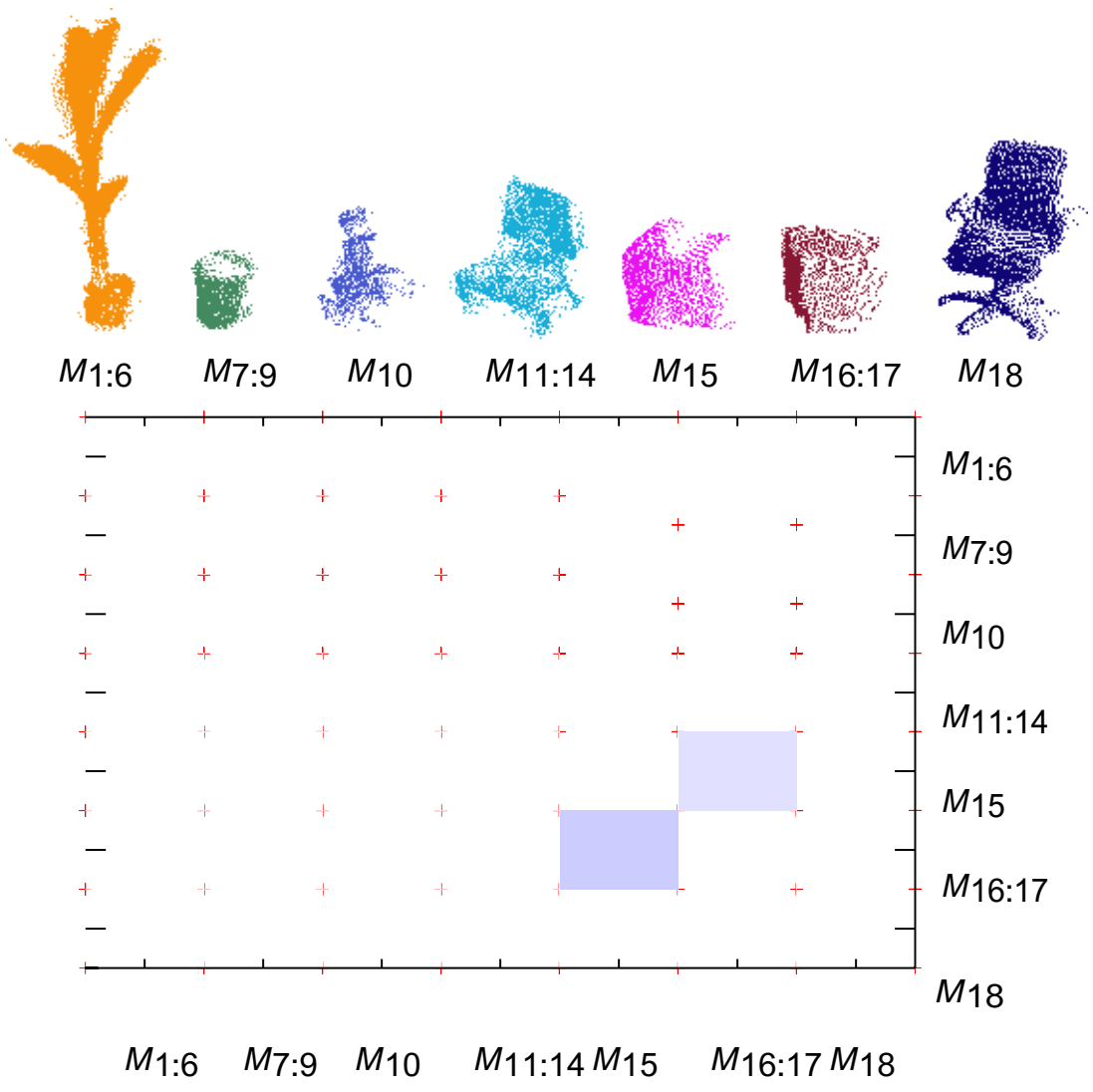


Figure 5.9: Final models and the corresponding similarity matrix. There are no falsely merged models and our system only failed to merge the remaining two models of the monitor, due to the limited overlap. (Source: [104])

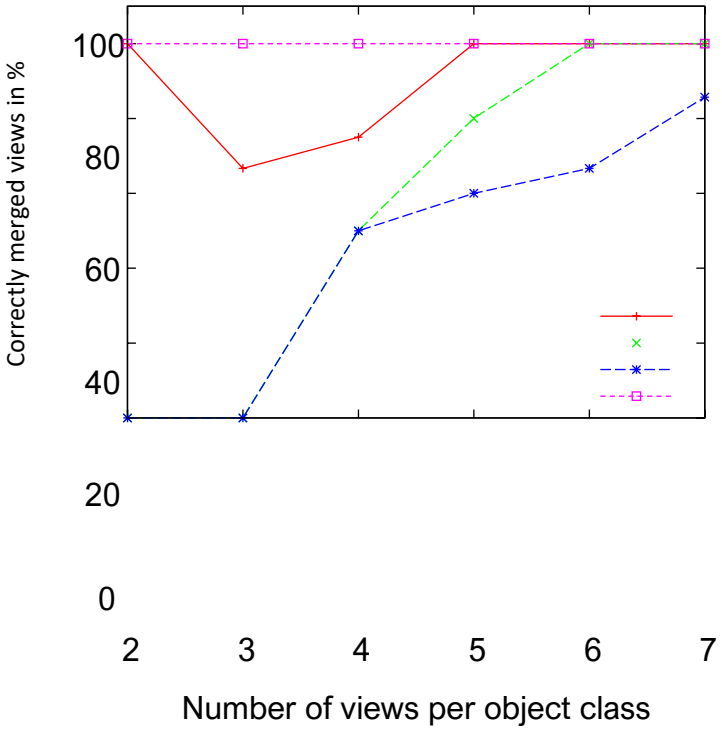


Figure 5.10: This plot shows how the number of available views for a certain model influences the merging capability. As expected, the more views are available for the system, the better the matching.

5.2.2 Influence of the Number of Views

In the second experiment we analyzed, how the number of available views for an object in- fluences the merging results. For this purpose we collected seven views of a garbage bin, a monitor, a plant, and a chair, with evenly distributed perspectives around the objects. We used these sets to test the merging capabilities for all set sizes between two and seven.

The plot in Figure 5.10 summarizes the outcome of this experiment. It shows which percent- age of the sets could be successfully matched depending on the set size. The different colors are for the different objects. It shows the expected behavior, meaning that more views simplify the merging process. The needed number of views depends strongly on the structure of the object and how much the different views are influenced by self-occlusion. E.g., the views of the garbage bin do not differ much with changed perspectives and the plant provides a lot of unique structure to ease the matching process. Therefore these two models need very few views to reach high success rates. On the other hand, the monitor and the chair are strongly influenced by self-occlusion, which is why more views are needed here to create complete models.

5.3 Related Work

The creation of object models is a highly relevant topic in perceptual tasks since they are a vital input for systems that need to find instances of these objects or try to learn

classifiers for certain types of objects. Being able to learn objects in an unsupervised fashion from unconnected data can save a lot of manual work and thereby ease the creation of large model databases.

In computer vision several methods for object discovery, meaning the search for reoccurring structures in unlabeled visual images, have been proposed in the past [149, 118, 133, 67]. A useful comparison of such methods was published by Tuytelaars *et al.* [143].

In the context of 3D range based perception most attempts to discover objects do not actually learn a complete 3D representation of objects but aim at finding classifiers to group the found structures into different types. E.g., Endres *et al.* [37] presented an approach for unsupervised discovery of object classes in 3D range scans. They use a variant of Spin Images to represent surfaces and model object classes as distributions of these features. They apply Latent Dirichlet Allocation [11] to cluster 3D objects regarding shape similarity. The learned models can later

on be used to detect these object classes in new data. This system aims at object categories rather than specific objects and does not create 3D models, but a collection of shape properties encoded in feature descriptors.

Other approaches use changes over time to find and model non-stationary objects. E.g., Anguelov *et al.* [3] learn 2D models of non-stationary objects from multiple instances of com- plete 2D maps of an environment, captured at different times, between which the objects moved. The objects are found through differencing and represented in a hierarchical fash- ion. Herbst *et al.* [58] follow a similar goal, but using RGB-D (color and depth) sensors. They build colored 3D maps of an environment at different times and detect segments that moved in between. These segments are merged into object models using spectral clustering. By using RGB-D data they can use shape and texture to distinguish different objects. Compared to these two approaches, we can also detect static objects and do not rely on a pre-built map, but on unconnected 3D scans, each captured from only one perspective.

Shin *et al.* [116] also presented a system for unsupervised object discovery and classifica- tion. Like we do, they first perform background extraction, but based on a saliency measure. Then they find potential object parts using a clustering approach that considers metric distance and similarity regarding extracted features. These object parts then form objects in a hierarchi- cal fashion. One advantage of their method is that the object classification step is able to split clusters again, thereby being robust against mistakes that happened in the clustering step. The extracted object types can then be used to classify parts of the environment. While this method works well to create and find object classes, it does not provide the means to merge different views into a single point cloud model.

Moosmann *et al.* [92] focused on object class discovery in 3D outdoor datasets (trees, cars, buildings, etc.). Their method does not rely on a perfect segmentation of the scans. Instead they propose a hierarchical segmentation, where each level in the hierarchy represents different segmentation parameters. A feature descriptor is calculated for every node, enabling similarity checks between them. This similarity measure is then used to create and assign object classes. This method is only able to detect structures that occur frequently and also aims at providing a general classifier for object classes instead of creating specific point cloud models.

5.4 Conclusions

In this chapter we presented a novel method for unsupervised object model learning from an unconnected collection of 3D range scans. The method employs point features to address this hard 3D perception problem. Our system finds potential objects using a combination of back- ground subtraction and spatial clustering on the individual scans. These partial object views are then merged into more complete models based on a similarity measure that is derived from the methods we used for object recognition in Chapter 4. We presented experiments using real data from a 3D laser range scanner that showed that our system can robustly detect and merge object instances from cluttered scenes while being able to reliably distinguish different objects that should not be merged.

Chapter 6

Using a Dictionary of Surface Primitives for 3D Environment Modeling

The way how a robot models a 3D environment can have a strong influence on its performance in different tasks. Most representations, like databases of individual scans, point clouds, or 3D grids have high memory requirements since they con- tain much redundant information. In this chapter we introduce a new approach to represent 3D environments. It is based on the insight that typical human made envi- ronments contain many repetitive structures. We therefore developed a method that describes a 3D scene based on a limited number of surface patches. Our system learns a dictionary of surface primitives from a collection of scans. This dictionary can then be used to model 3D scenes by means of indices to the primitives and positions, where they occur. Our system determines a model for a given dataset under consideration of the Bayesian information criterion, which weighs the model complexity against the accuracy of the reconstruction. We show that even a small dictionary can be used to accurately describe typical 3D scenes in a condensed form. In addition, this representation has the benefit of being semantically richer than the raw sensor data since it directly encodes similarities in the environment.

* * * *

In the previous chapters we discussed methods to find similar structures in 3D scenes. First in the context of basic feature matching, then to find known object models in 3D scenes and then to create such object models by merging several overlapping observations into one representation. Finding similar structures can also be used to simplify the description of a scene. If there are several identical objects in a 3D environment, we only have to describe the object once and then save the information, where the individual instances occur. This makes it possible that we do not have to provide a detailed description of the same structure over and over again in our representation of the environment. If we only consider large objects for the description, this may only have a small impact on the overall amount of data since identical objects only occur irregularly, like, e.g., chairs, tables or closets in an office environment. But at a smaller scale, 3D structures are very repetitive. There are many patches of flat surfaces, rectangular corners, straight edges, etc. in human-made environments. To detect those occurrences would enable us to describe them all with a small number of *surface primitive* respectively. This brings us back to the problem of finding similar structures in 3D data.

We will present a system that, given a dataset of 3D scenes, learns a *dictionary* of *words*. Each of these words contains one of the aforementioned surface primitives, describing a small

part of a 3D scene in the form of a point cloud. This dictionary can then be used to describe a 3D scene, by assigning the most similar surface primitive to each individual part. These *instances* of the words only need the information which surface primitive is to be used, typically in form of a simple index, and the 6DOF position in the scene, where it occurs.

It is a common concept in the area of data compression to find identical data blocks and represent them with a single dictionary word. This leads to a compression because the links to the dictionary typically take less space than the word itself. A popular example for a lossless compression algorithm that is based on this principle is the LZW-algorithm [93]. This method can be used for arbitrary data streams since the dictionary is created during runtime and en- coded in the compressed file. In the approach presented in this chapter, we are not focused on lossless compression of arbitrary data, but on the potentially lossy compression of 3D range data captured with noisy sensors. In vision, there exists an abundance of lossy and lossless image compression methods, e.g., the famous JPEG standard [148]. Yet in the area of 3D mea- surements such methods are not yet very common. Our approach aims at closing this gap and replaces parts of the scene by surface patches that are not identical, but similar. Finding these similarities in a naive way is computationally very demanding, which is why we will make use of the feature extraction methods presented in Chapter 3. The NARF descriptor is used to perform the necessary similarity comparisons. We also adapted the NARF keypoint extraction method to fit the problem of placing word instances in a scene.

Besides reducing the amount of data necessary to describe a 3D scene, this representation also provides more semantic context since it explicitly describes which structures in the scene are similar to each other.

The basic principle of our method is to build the dictionary and reconstruction of a scene together and optimize it in an iterative fashion by minimizing the Bayesian information cri- terion (BIC). This provides a trade-off between the complexity of the model and its accuracy while needing very few user defined parameters. This means that the size of the dictionary is determined by the algorithm, based on the actual need of the current scene.

See Figure 6.1 for a visual example of a scene that was described with a dictionary con- taining 114 words. The points in blue mark areas which can be accurately described by the words from the dictionary. Red points mark areas were the reconstruction error was too high. The top left shows an example for one word that was chosen by the algorithm to be part of the dictionary. The boxes show different perspectives for the point cloud stored in the word.

We tested this approach on large real world datasets and our experiments suggest that even a small dictionary (below 100 words) can be used for accurate reconstructions of typical indoor 3D scenes.

The remainder of this chapter is structured as follows: in Section 6.1 we will give a formal definition of the problem at hand. In Section 6.2 we discuss our method to evaluate a model together with a dictionary. Section 6.3 is about heuristic methods we employ to reduce the search space. In Section 6.4 we present our iterative method to learn a dictionary and a model for a given scene. Our experimental evaluation is contained in Section 6.5. Finally, we discuss related work in Section 6.6 and will conclude the chapter in Section 6.7.

6.1 Formal Problem Definition

The input for our system is a dataset of 3D scans with known sensor poses. The contained 3D measurements can be merged into a single point cloud $S = \{\mathbf{s}_1, \mathbf{s}_2, \ldots\}$. It is our goal to construct a dictionary $D = \{\mathbf{w}_1, \mathbf{w}_2, \ldots\}$, where each word $\mathbf{w}$ contains a small point cloud

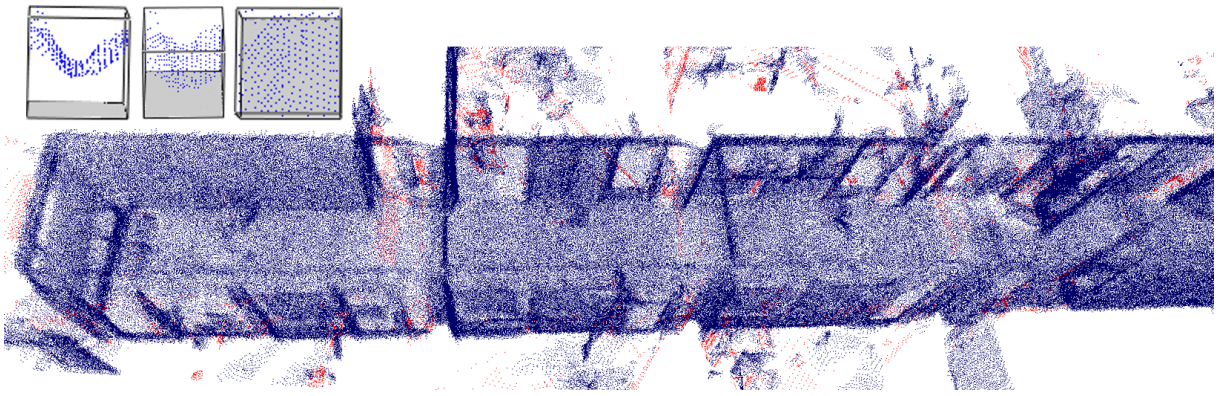


Figure 6.1: This figure shows a 3D scene of a corridor which was reconstructed from a model with a dictionary of size 114. The dark blue points show areas that were accurately described by the model. Points from the original point cloud that are not covered well by the dictionary are shown in red. The boxes in the top left visualize an exemplary word that is part of the dictionary for this scene. The point cloud from the word is shown in three different perspectives.

$\mathbf{w}^P$. In our specific implementation, these point clouds contained in the words are subsets of the original point cloud S from a fixed size neighborhood (diameter ρ) around a point $\mathbf{p} \in S$:

$$\mathbf{w}^P = \{\, \mathbf{s} - \mathbf{p} \;|\; \mathbf{s} \in S, \|\mathbf{s} - \mathbf{p}\| \leq \frac{\rho}{2} \,\}. \tag{6.1}$$

A given 3D scene can now be approximated as a list of instances, where these words occur. We will call this a model $M(D, S) = \{\mathbf{m}_1, \mathbf{m}_2, \ldots\}$ for dictionary D and scene S. Each instance
$\mathbf{m} \in M$ is a tuple $\mathbf{m} = (\mathrm{id}, T)$ where id is the index of a word in the dictionary and T describes a 3D transformation. E.g., $\mathbf{m}_j = (i, T_j)$ would mean that the points from $\mathbf{w}_i^P$, which is the point cloud contained in word $\mathbf{w}_i$, have to be transformed using T_j, to be in the right positions to form
a part of the reconstructed point cloud. Therefore, the reconstructed scene is defined as the union of all those transformed point clouds:

$$R(M, D) = \bigcup_{j=1}^{|M|} T_j\, \mathbf{w}_{\mathrm{id}_j}^P. \tag{6.2}$$

Since the point clouds from the different word instances might be overlapping, there will be areas that are unnecessarily dense. Therefore our specific implementation includes a sparsifi- cation step that reduces the resolution in those areas to a pre-defined value, meaning that we enforce a minimum distance between the points in R.

6.2 Model Evaluation

The quality of the reconstruction clearly depends strongly on the available words in the

dictio- nary. With a growing number of words in the dictionary, the chances to find a good match at a certain position in the 3D scene also increases. Yet, this also increases the memory footprint of the dictionary. An exaggerated dictionary can easily need more memory than the scene for which we want to use it. We need to find a balance between the size of the dictionary and the accuracy of the reconstruction. The Bayesian Information Criterion (BIC) is designed to cap- ture this tradeoff. The BIC is a general tool for model selection problems. It is typically defined as

$$\mathrm{BIC} = -2\ln(l) + k\ln(N). \tag{6.3}$$

Here l is the likelihood of the current model, k is the number of free parameters in the model and N is the number of data points in the original observation. For the problem at hand, this leads to

$$\mathrm{BIC}(S, M, D) = -2\ln(l(S, R(M, D))) + \left(\sum_{i=1}^{|D|} p_i\right)(7|M| + 3|\mathbf{w}|)\ln(3|S|). \tag{6.4}$$

The likelihood depends on the original point cloud and the reconstructed point cloud since it captures the error between the two. It will be defined below. The number of free parameters in the dictionary is the size of the model, multiplied by seven because every instance contains one index and one 6DOF transformation, plus the complete number of points in the word point clouds, multiplied by three since the points are three-dimensional. The last term is the number of points in the original point cloud, multiplied by three. It is our goal to find a model that minimizes the BIC, thereby being maximally consistent with the input data, while having a limited complexity.

Under the assumption that the points in S are independent of each other, i.e., we assume that the error of each point in the cloud can be evaluated individually, we can define the log- likelihood as follows:

$$\ln(l(S,R)) = \ln\left(\prod_{i=1}^{|S|} p(\mathbf{s}_i \mid R)\right) \tag{6.5}$$

$$= \sum_{i=1}^{|S|} \ln(p(\mathbf{s}_i \mid R)). \tag{6.6}$$

This means that we can analyze how well the single points are represented in the reconstructed scene and sum up their log-likelihoods. The calculation of the probability for the individual points requires the introduction of a data association between $\mathbf{s}_i$ and the points from the recon- structed scene $R = \{\mathbf{r}_1, \mathbf{r}_2, \ldots\}$:

$$p(\mathbf{s}_i \mid R) = \sum_{j=1}^{|R|} p(\mathbf{s}_i \mid \mathbf{r}_j, a_{i,j}) p(a_{i,j}). \tag{6.7}$$

Here, $a_{i,j}$ defines that there is a data association between $\mathbf{s}_i \in S$ and $\mathbf{r}_j \in R$.

Unfortunately, we lack any knowledge about the data associations. Therefore, we assume a uniform distribution of the prior $p(a_{i,j})$, leading to

$$p(\mathbf{s}_i \mid R) = \sum_{j=1}^{|R|} p(\mathbf{s}_i \mid \mathbf{r}_j, a_{\mathbf{s}_i, \mathbf{r}_j}) \cdot \frac{1}{|R|}. \tag{6.8}$$

Given a data association $a_{i,j}$, we model the probability for the point pair as a spherical Gaussian:

$$p(\mathbf{s}_i \mid \mathbf{r}_j, a_{i,j}) = \frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{1}{2\sigma^2} \|\mathbf{s}_i - \mathbf{r}_j\|^2\right). \quad (6.9)$$

The standard deviation σ should be dependent on the resolution of the source point cloud. Since we aim for scenes captured with real sensors, like laser scanners or 3D cameras, the resolution differs with the distance from the sensor. Assuming an angular resolution of α, we compute σ_{s_i} as follows:

$$\sigma_{s_i} = \omega \tan\frac{\alpha}{2} \text{range}(\mathbf{s}_i), \quad (6.10)$$

whereas range denotes the distance between the point $\mathbf{s}_i$ and the original sensor position.

In our implementation, we truncate the Gaussian distribution at 3σ. This reduces the runtime since we only have to consider the points in a local neighborhood in $\mathbb{R}^3$ instead of the whole point cloud.

Up to now we described how to evaluate the BIC of a given model M and scene D. We will now start to explain, how our system constructs a dictionary and a model from a given scene.

6.3 Heuristic Search Space Reduction

As defined in Section 6.1, a model $M = (\mathcal{D}, \mathcal{S}, \Pi)$ for dictionary $\mathcal{D}$ and scene $\mathcal{S}$ contains tuples $\mathbf{m}_1, \mathbf{m}_2, \ldots$, where every tuple $\mathbf{m} = (\text{id}, T)$ contains the index to a word and its position in the scene. Finding the optimal dictionary and model for a given scene, meaning the combination with the smallest BIC, is hard as the search space is huge. In theory, every combination of arbitrarily shaped words has to be checked for every possible combination of positions in the scene. Since we cannot check the complete search space, we have to rely on heuristics to find a good, even if not optimal, solution. The main heuristics used to reduce the search space for our system are:

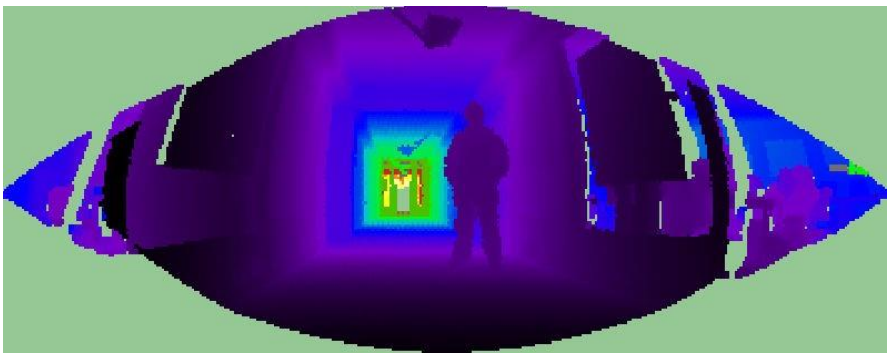
- We use words of a fixed size and shape, namely spheres with a pre-defined diameter. We will refer to this diameter as the *support size* ρ of the words. This is in accordance with what we called the size of the area covered by a 3D point feature descriptor in Chapter 3.

- We determine the position of the instances in the scene only once at the beginning and do not change them afterwards, i.e., we have fixed positions for the instances, even if the word they refer to is variable. We define the following requirements regarding the placement of the instances:

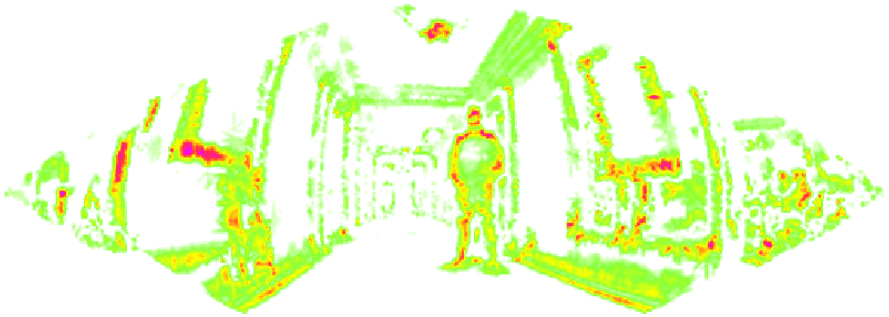
  - The scene should be completely covered according to the support size ρ of our words but without an excessive overlap.
  - The same pose should be found for every instance of a complex recurring structure in order to minimize the number of words in the dictionary and to facilitate the feature matching process.

To achieve the above requirements for the placement of the instances, we use a combination of a keypoint extraction method and uniform sampling on the scene. The keypoint extraction is supposed to find points in the scene that have stable positions like corners or points with partially stable positions, e.g., on straight edges. The uniform sampling is supposed to cover areas where the surface does not change in any significant way, like flat surfaces.

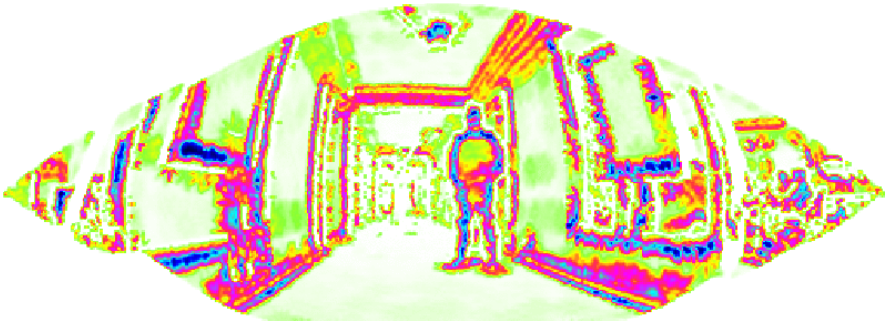
(a)



(b)



(c)



(d)

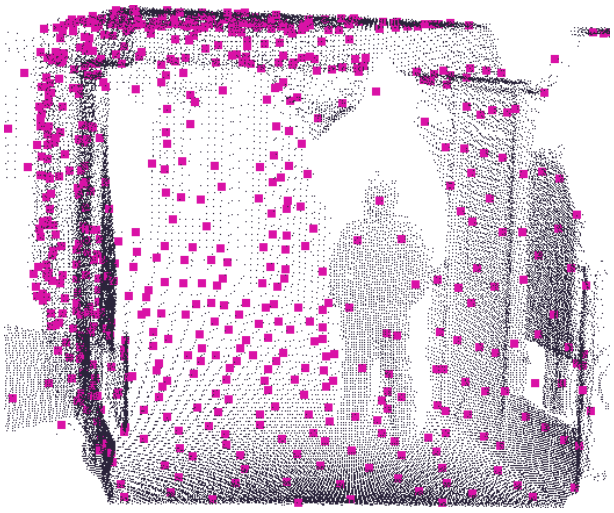


Figure 6.2: This figure visualizes the procedure used for deciding where to place the words in the scene.

(a) A range image of the original scene, showing a person standing in a corridor. **(b)** The scores for the original NARF keypoint extraction. **(c)** Scores for our adapted version, that also gives high scores to areas close to straight edges. **(d)** The points extracted from these scores marked in the original point cloud.

For this purpose we adapted the NARF keypoint extraction method presented in Section 3.2. The original NARF keypoint detection was created to return points where the surface is stable (to ensure a robust estimation of the normal), but where there are sufficient changes in the immediate vicinity. Based on this criteria a score (the interest value) is calculated for every point in a range image (see Figure 6.2(a,b) for an example) and the keypoints are found using non-maximum suppression. To adapt this method for the current application, we developed an new scoring function that, while still giving high scores for the original criteria, also favors points on straight edges. For this purpose we added a term that increased the interest value for points close to high curvature values (see Figure 6.2(c)). In addition, to ensure coverage of the complete scene, we exchanged the non-maximum suppression for the selection of keypoints with the following procedure:

1. find the point with the highest interest value
2. add this point to the list of selected points
3. set all scores in a certain radius around the point to a negative value
4. repeat until no point with a score of at least zero is left

Thereby we first add all points on complex structure, which typically have a precise position, then the points on straight edges, where the positions are well defined in one axis, and in the end the remaining points on uniform structures (see Figure 6.2(d)). The radius mentioned above is chosen according to the support size ρ of our words since a big enough overlap has to be guaranteed.

In the next step, the NARF descriptor (see Section 3.3) is calculated for all selected points. These descriptors will enable us to perform fast similarity checks between the instances and the words in the dictionary. Figure 6.3 gives a visual example for these similarity checks. The dark areas in the individual images are suited to be represented by only one word since they have a very similar surface structure. Additionally, as explained in Section 3.3) this process also provides a unique 6DOF coordinate frame that is calculated from the local structure, thereby providing the means to calculate the 6D transformations needed for every instance in our model.

With this done, we already determined the size of the model and the 3D positions of the instances in the scene. Please note, that for this case the calculation of the BIC from Equation 6.4 can be simplified to

$$\mathrm{BIC}'(S, M, D) = -2\ln(l(S, R(M, D))) + \quad (6.11)$$

$$3\sum_{i=1}^{|D|}|\mathbf{w}_i^p|\ln(3|S|)$$

since the term $|M|\ln(3|S|)$ from the original equation is a constant and will only influence the absolute number of the BIC, but not the minimization process.

We have a local point cloud for every instance and a descriptor for every local point cloud. The main information missing for a complete model are the indices of the dictionary words to be used for the instances. If a dictionary is given, where every word also includes a descriptor for its contained point cloud, (,) can be calculated by assigning the most similar word to every instance. This is done by finding the word with the minimum descriptor distance between the NARF descriptor associated with the word and the NARF descriptor associated with the

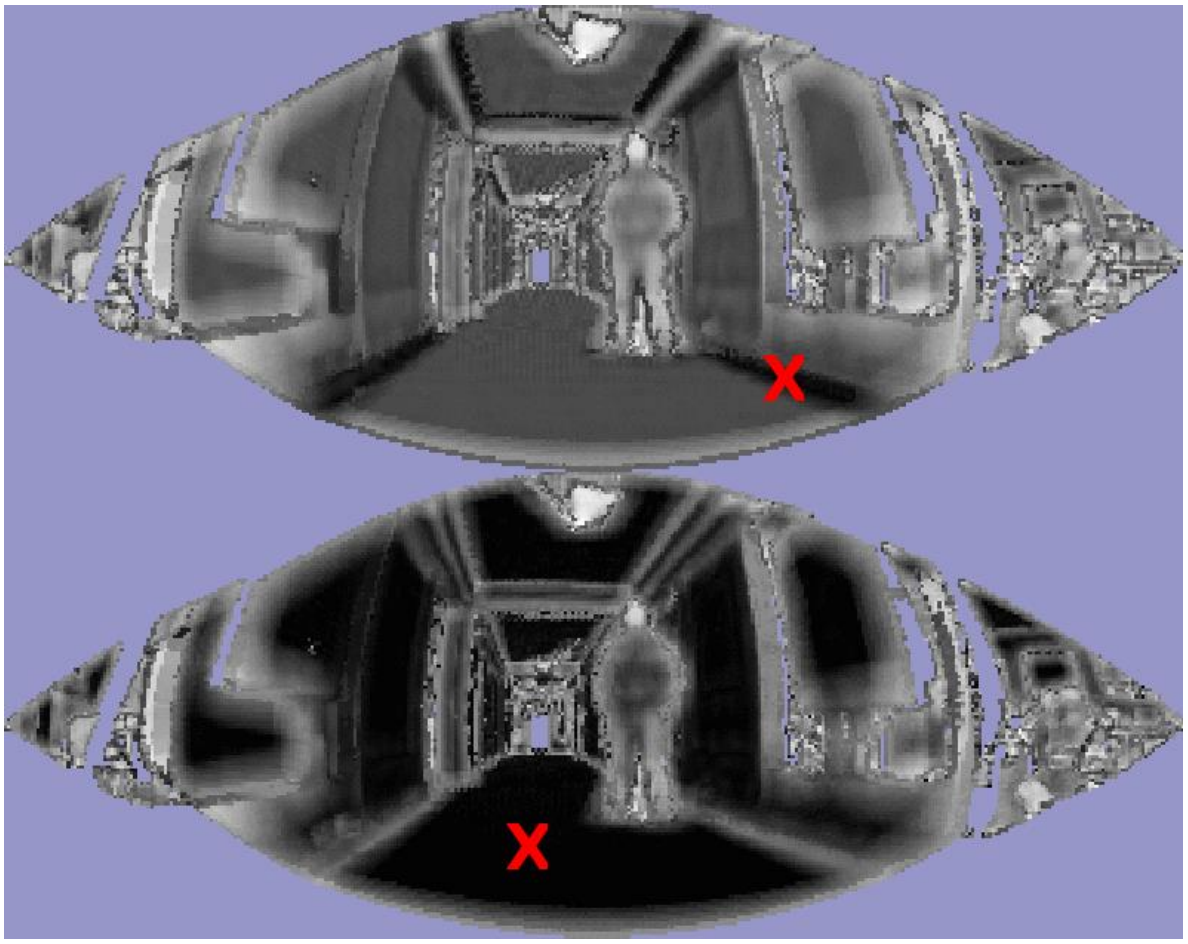


Figure 6.3: This figure visualizes how the features can be used to determine the similarity between two regions in the 3D data. It shows the same scene as in Figure 6.2(a). The feature at the position marked with a cross is compared with every other position in the scene and the descriptor distances are visualized. Dark means high similarity and bright means low similarity. In these examples, a feature describing a

90° angle between two flat surfaces (top) and a completely flat surface (bottom) are visualized.

instance. The transformation T for an instance $\mathbf{m} = (\mathrm{id}, T)$ can be calculated based on the transformations associated with the NARFs as $T = T_a^{-1} T_s$, where T_a is the transformation associated with the NARF from the dictionary word and T_s is the transformation associated
with the NARF from the current instance.

We will now explain the procedure to iteratively find a good dictionary and model regarding the corresponding BIC value.

6.4 Dictionary Refinement

Based on what we discussed in Section 6.3, we can construct a temporary dictionary $\mathcal{D}_S$, that contains all words occurring in the scene S at the keypoint positions, whereas each word is
associated with the local point cloud, the extracted transformation and the NARF descriptor. This dictionary could be used to represent the original scene perfectly since

every partial surface from the scene is available in it. Yet, the dictionary is very large, which would lead to a very high BIC value.

Instead, we try to build a good dictionary and model by iteratively changing the dictionary

and keeping the new version if it has a smaller BIC value than the last version. The concrete implementation works as follows:

1. Start with an empty dictionary D.
2. Randomly perform one of the following actions:
   - Insert action: Choose a random word from $D_S \setminus D$ and add it to D.
   - Delete action (if $|D| \geq 2$): Remove a random word from D.
   - Replace action (if $|D| \geq 1$): Remove a random word from D and add a random word from $D_S \setminus D$ in its place.
3. Create the model $M_D(S, \epsilon)$ as explained in Section 6.3. Please note that the assignment of words to the instances can be calculated rather efficiently by caching the results of the last iteration and only performing updates regarding the changes caused by the last action taken in step 2.
4. Calculate the reconstructed scene from the current model and dictionary D (M_D, ϵ) as described by Equation 6.2.
5. Calculate BIC(S, M, D) as described in Section 6.2.
6. If the BIC did improve, we accept the new dictionary. Otherwise we backtrack to the former dictionary. If a maximum number of iterations is reached we stop the procedure. Else we go back to step 2.

6.5 Experimental Evaluation

We will now present some experiments to evaluate how our approach performs in different settings. We will analyze the size of the dictionary $|D|$, the size of the model $|M_D|$, the number of inliers (points from the original scene that are correctly represented in the reconstructed point cloud), the number of outliers (points contained in the reconstructed point cloud that do not exists in the original data), and how much memory is saved compared to the original point cloud representation. For the latter we use a standard binary encoding. Every 3D point uses 3 floating point values (12 Bytes), every 6DOF transformation uses 6 floating point values (24 Bytes), and a word index uses one integer (12 Bytes). The inliers and outliers are determined regarding a maximum distance to the next correct point of σ_{s_i} (see Section 6.2).

Our algorithm ends the iterative process, when the BIC did not decrease for 100 iterations.

6.5.1 Influence of the Standard Deviation Parameter

Our method has mainly two adjustable parameters: the support size ρ and ω, which is used to calculate the standard deviation σ (see Equation 6.10) for a certain measurement range. The standard deviation σ determines how accurate the model has to be because it restricts the max- imum error between point positions in the likelihood computation. To analyze the concrete influence of different values of σ on the model creation process, we performed a small scale experiment with the point cloud of a chair

visualized in Figure 6.4(a). We performed the model creation process for σ [0.1 mm, 20 mm] and ρ = 20 cm. Figure 6.4(b) shows how the different values influence the dictionary size. As expected, the dictionary size decreases with increasing

values of σ since errors in the approximation receive a lower weight in the likelihood evaluation and therefore less words are needed to describe the point cloud. Figure 6.4(d) shows the recon- structed point cloud for a dictionary of size 34, whereas Figure 6.4(e) shows the reconstructed point cloud for a dictionary of size 12. As expected, the cloud in (e) does not represent the origi- nal point cloud as well as the one shown in (d). The backrest is flat, while the original is slightly curved and the fine structure of the feet of the chair is also not well represented. Figure 6.4(c) shows the evolution of the BIC and the dictionary size during the iterative refinement for the model from (d). While the BIC stabilizes after about 300 iterations, the dictionary size needs slightly above 400 to get to a nearly constant level. In the beginning every inserted word has a high chance to be a better approximation of the structure somewhere in the point cloud and therefore justifies the increase in the dictionary size. Later on, mainly replacements of words still have a chance to lower the BIC, leading to a much slower progress.

For the remaining experiments we will use $\omega = 0.5$ to calculate the range and resolution dependent values for σ, which is a good tradeoff between accuracy and size.

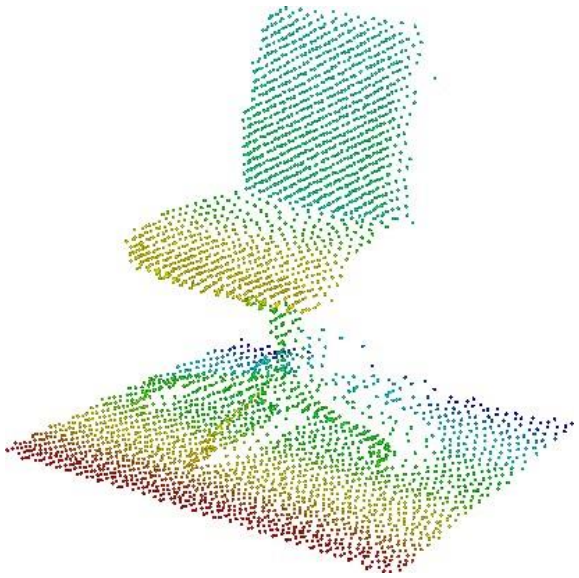
6.5.2 Small Indoor Scene

We tested our approach on a small scale indoor scene of a person standing in a corridor (see Figure 6.5(a)). The final dictionary after 3,048 iterations of our algorithm contained only 68 words with a support size of $\rho = 20$ cm. The evolution of the BIC for an increasing dictionary size is depicted in Figure 6.5(b). Figure 6.5(c) shows the reconstructed point cloud for a dic- tionary with only 10 words, whereas Figure 6.5(d) shows the reconstructed point cloud for the final dictionary with 68 words. While flat structures are already well represented with 10 words, round structures are much better approximated with the larger dictionary. Please note that the visual difference between the original scene and the reconstructed scenes mainly results from the different resolution since the reconstructed point cloud contains more points in low density regions of the original cloud. The final reconstructed point cloud covers 88.3% of the scene with 4.2% of outliers, whereas the largest errors occur in the low density regions of the scan far away from the sensor. The resulting size of the binary model file is 199.2 kB. Compared to the original point cloud of size 1,208.9 kB, this means a compression down to 16.5%. Additionally, we tested the influence of a standard compression algorithm, namely gzip [33]. Using gzip on the original point cloud led to a file of size 1,089.9 kB, which is 90.1% of the original file size. Using gzip on the binary files of our model led to a file of size 160.2 kB, which is 80.0% of the original file size. In general, the compression that can be achieved with gzip is roughly between 80% and 90% of the original file size for both, the original point clouds and our models. If we compress both representations with gzip, the relative size difference does only change slightly. For the current dataset it is 14.7% instead of the original 16.5%. Table 6.1 summarizes the results of the experiments.

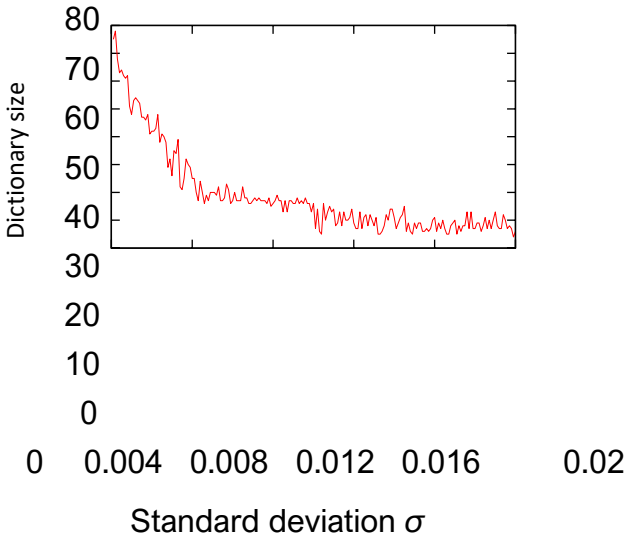
6.5.3 Dictionary Transfer

Subsequently we wanted to evaluate, if the dictionary of one scene can be used to reconstruct another similar scene. We used the dictionary from the last experiment (Section 6.5.2) to create a model for the scene shown in Figure 6.6(a), using the method explained in Section 6.4. The resulting model includes 5,263 word instances and the

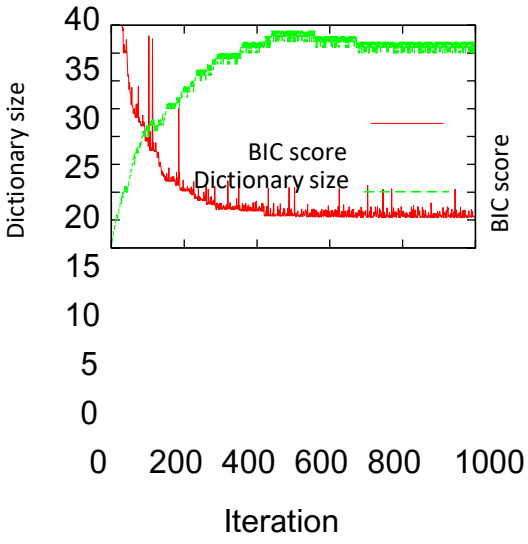
reconstructed point cloud has 89% inliers and 12% outliers (see Figure 6.6(b)). This result is comparable to the experiment above, with mainly an increased outlier rate. Yet the scene is still well represented, leading to the conviction that our approach provides an accurate reconstructions even when the dictionary was learned for another scene.



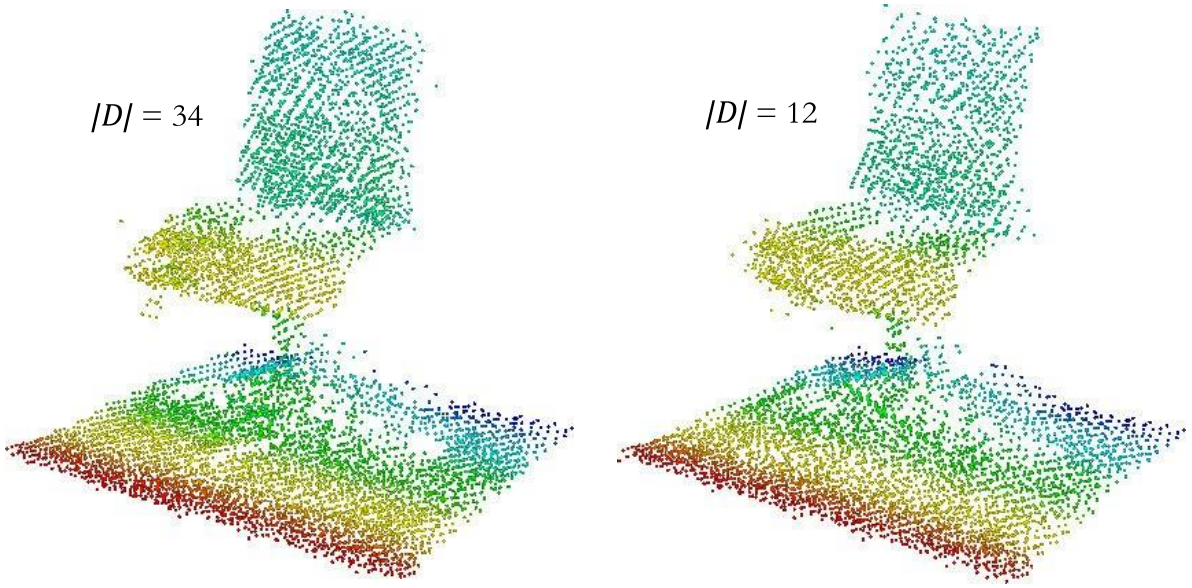
(a)


Dictionary size
80
70
60
50
40
30
20
10
0
0
0.004
0.008
0.012
0.016
0.02
Standard deviation σ

(b)


Dictionary size
BIC score
40
35
30
25
20
15
10
5
0
BIC score
Dictionary size
0
200
400
600
800
1000
Iteration

(c)



(d) (e)

Figure 6.4: (a): a point cloud of a chair. **(b)**: the evolution of the dictionary size for increasing values of σ. **(c)**: the development of the BIC and dictionary size during the iterative dictionary refinement for σ = 3.2 mm and $|D|$ = 34 (see (d)). **(d)** Reconstructed point cloud for σ = 3.2 mm, leading to $|D|$ = 34.

(e) Reconstructed point cloud for σ = 14.4 mm, leading to $|D|$ = 12. (Source: [106])

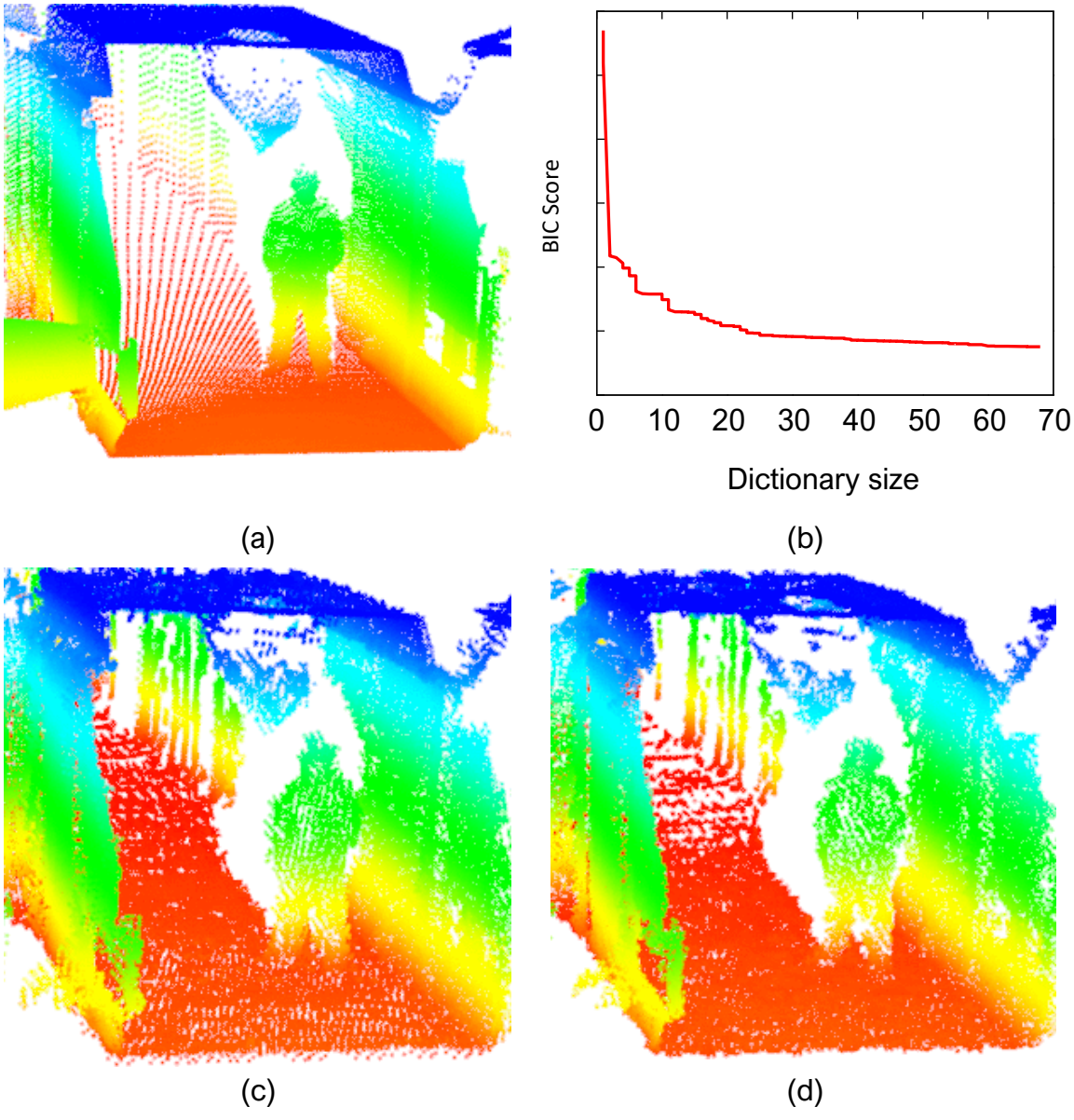
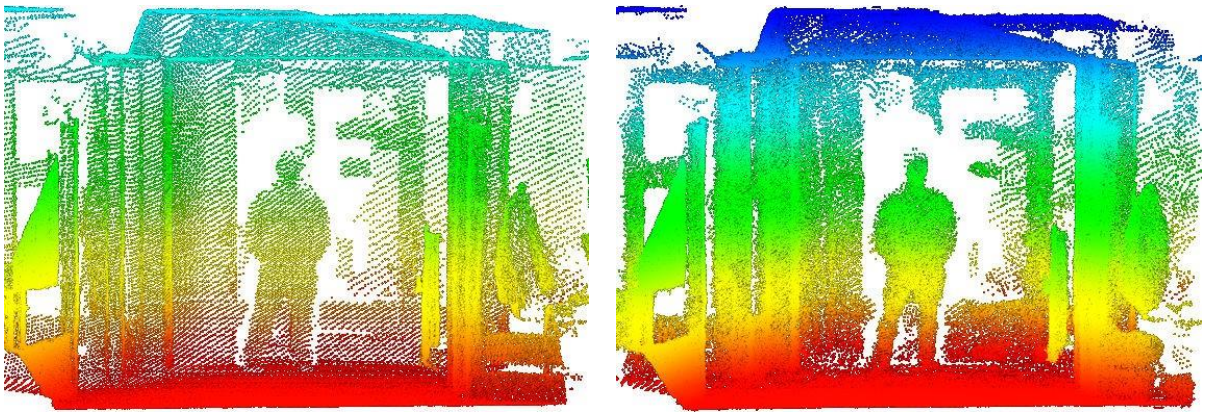


Figure 6.5: This figure shows the results of an experiment for a 3D scene of a corridor with a person standing in it. **(a)** shows the original point cloud, color coded according to the z-axis for better visibility.

(b) shows the evolution of the BIC during the iterative refinement. **(c)** shows the point cloud recon- structed with an early dictionary containing only 10 words. **(d)** shows the reconstructed point cloud using the final representation with 68 words in the dictionary. The planar corridor is already well ex- plained with a dictionary of size 10. Yet, round regions (e.g., on the person) or sparse structures (like the lamp above the head of the person) are much better represented with an increasing number of words.



(a) (b)

Figure 6.6: (a): Another point cloud of a scene of a corridor with a person. **(b)**: A reconstructed point cloud using the dictionary originally created for the scene in Figure 6.5(a). (Source: [106])

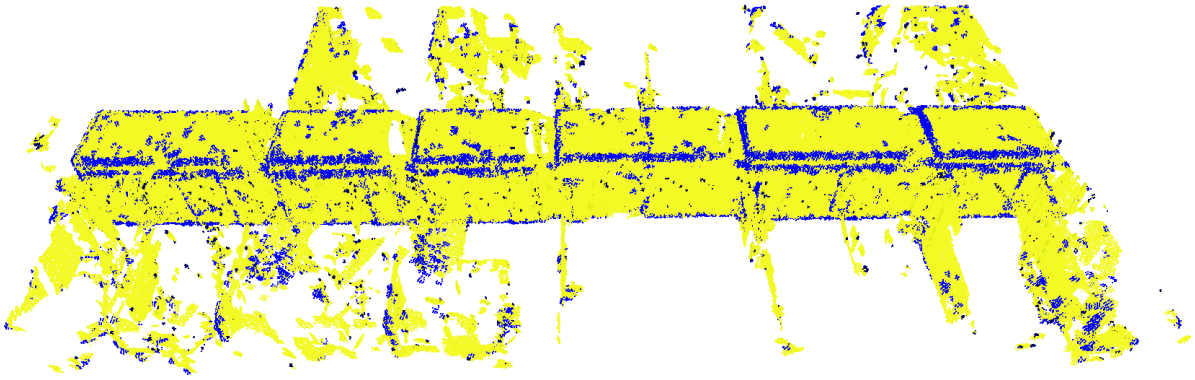
6.5.4 Large Scale Indoor Dataset

Up to now we showed that our approach can be used to accurately describe small scale 3D scenes with a substantially lower memory footprint than the original scene. It is reasonable to assume that the performance will improve for larger scenes, where the repetitiveness of the environment can be exploited even better. To test this assumption, we captured a dataset of 31 3D range scans in the corridor of an office building and used our approach to build a model of the resulting combined point cloud. The resulting reconstructed 3D map is visualized in Figure 6.7(a). The final dictionary included 339 words. The most frequently used word in the model is a mostly planar patch, which is not surprising because there are many flat surfaces in this environment. The occurrences of this word are colored yellow. The remaining instances are colored blue. The plot in Figure 6.7(b) shows how often the different words from the dictionary occur in the model. The number of occurrences decreases quickly, which confirms the assump- tion that a small dictionary can already lead to an accurate description of the environment. The model and dictionary together require only 5% of the memory, the original point cloud needed, with 98.7% inliers and only 0.3% outliers. This confirms our assumption that the approach re- turns even better results for a larger dataset. Figure 6.7(c) and (d) show a close-up view from inside of the corridor, for the original point cloud and the reconstructed version respectively. The structures in the corridor are well represented in the reconstruction. The main visual differ- ence is that the individual scans are visible in the original point cloud, because of the varying density, whereas the density in the reconstructed version is more uniform. Additional results for this experiment can be found in Table 6.1, which summarizes our experiments.

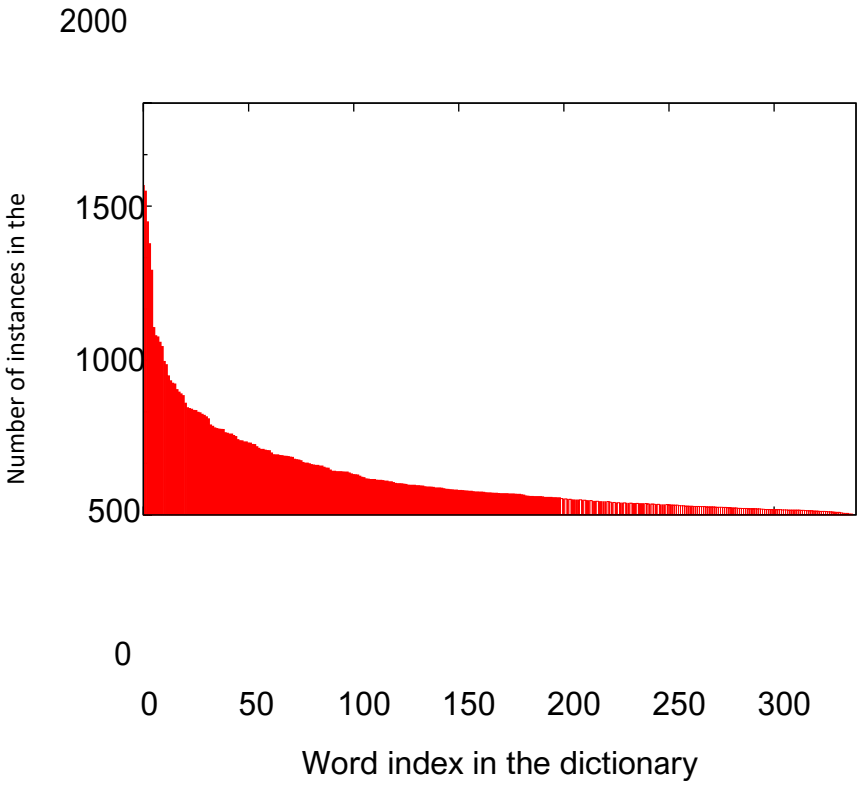
6.5.5 Outdoor Dataset

In another experiment we investigated the ability of our system to model a large scale outdoor dataset. For this purpose we collected 3D scans on our campus of the Technical Faculty of the University of Freiburg, containing roads, buildings, trees, and

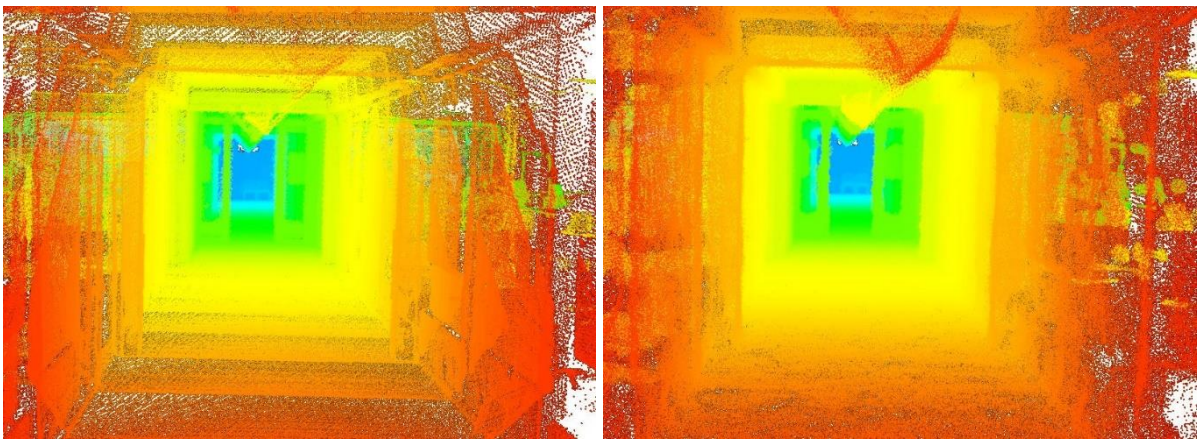
vegetation. We made this dataset publicly available [27]. The reconstructed point cloud is shown in Figure 6.8. For a support size of ρ = 0.5 m our system learned a dictionary of 159. The reconstruction quality is very high with 99.5% inliers and only 1.2% outliers. The compression for this case is down to 17.8%. This is higher than what we achieved on the large indoor dataset, since the outdoor data includes more complex structures like trees and natural vegetation. Yet, also in this case



(a)


2000
1500
1000
500
0
Number of instances in the
0
50
100
150
200
250
300
Word index in the dictionary

(b)



(c) **(d)**

Figure 6.7: Experimental results for a dataset of a corridor. **(a)** shows the reconstructed point cloud of the map generated from a corridor dataset. The most common word, which is mostly planar, is colored in yellow, the rest in blue. **(b)** shows how often the different words from the dictionary occur in the model. **(c)** shows a close up view of the inside of the corridor from the original point cloud. **(d)** shows the corresponding reconstructed point cloud. (Source: [106])

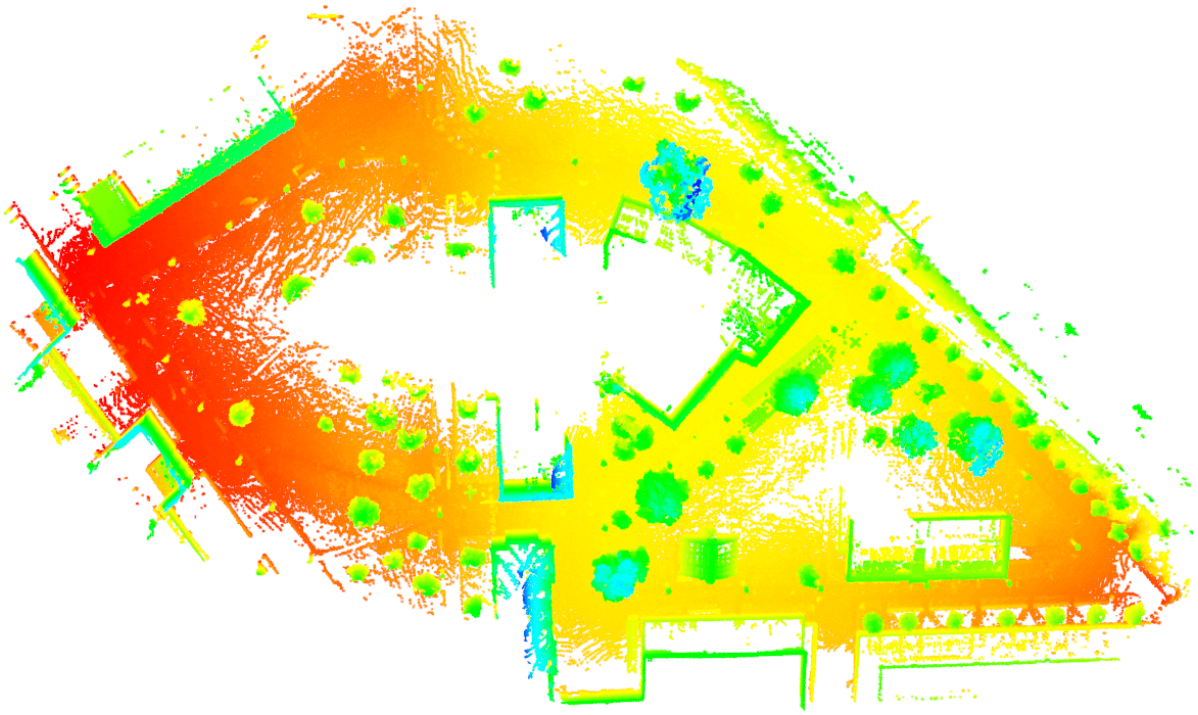


Figure 6.8: Top view of a point cloud reconstructed from a 3D outdoor dataset. The color differs with the height of the points (the z-axis)

our approach was able to accurately reconstruct a complex environment with a small number of surface primitives. Our system took 22.3h to learn the model for this dataset. The high runtime is caused by the large scale of this dataset, containing a large number of instances. As for the other experiments, Table 6.1 summarizes the results of this experiment.

Our results suggest that typical real world scenes can be accurately described by a small number of reoccurring words. The runtime requirements of our dictionary creation method are currently too high to consider them for online applications. Yet, a previously learned dictionary could be used for such applications.

6.6 Related Work

The compression of data is an important to save costs for storage and bandwidth. There are general lossless data compression methods like the LZW-algorithm [93] or methods specifically designed for a certain kind of data, like the JPEG [148] standard for visual images, both for lossless and lossy compression. There is also a lot of activity in the field of compression of mesh-based 3D models [1].

Spinello *et al.* [121] presented an unsupervised approach to discover repetitive object pat- terns in single camera images, which is based on a voting scheme regarding image descriptors. These patterns can then be used for model prediction and completion or for model compression. This is very similar to our concept but relies on camera images, not on 3D range data.

There are many different ways to represent 3D data with different advantages and

disad- vantages. Range Images and point clouds (see Section 2.3) are frequently used in this work. They are representations that are close to the original sensor data, but have high memory re- quirements. *Voxel grids* are another possible structure to represent 3D data. Voxel stands for volumetric pixel and those grids basically divide the 3D space in equidistant cells where each

Dataset	Single scene (see Figure 6.5)	Corridor (see Figure 6.7)	360° outdoor (see Figure 6.8)
dictionary size $\|D\|$	68	339	159
support size ρ	0.2 m	0.3 m	0.5 m
word occurrences $\|M\|$	6,599	62,792	873,464
memory req. original point cloud without and with gzip	1.21 MB 1.09 MB	37.77 MB 33.99 MB	137.47 MB 122.11 MB
memory requirement our model without and with gzip	0.20 MB 0.16 MB	1.85 MB 1.67 MB	24.47 MB 21.25 MB
relative memory requirement without and with gzip	16.5% 14.7%	4.9% 4.9%	17.8% 17.4%
inliers	88.3%	98.7%	99.6%
outliers	4.2%	0.3%	1.2%
runtime	11 min	20 min	22.3 h

Table 6.1: Summary of experimental results. The experiments were performed on an Intel I7 quad core machine.

cell stores information about the covered space, e.g., if it is empty or occupied, either in a prob- abilistic or a boolean fashion. While queries regarding the values of cells are very efficient, namely $O(1)$, the memory requirements are cubic regarding the size of the environment one wants to represent.

Octree-based structures [152] represent the world as hierarchical three-dimensional grids and are popular for 3D mapping approaches because they efficiently represent filled and empty space and can also be used in a probabilistic fashion. Yet they are actually better suited to represent continuous volumes than surfaces, as they are measured by typical range sensors.

Another popular method to represent 3D environments are *surface maps* [7, 57, 75, 98], also called or *height maps* or *elevation maps*. These maps are two-dimensional grids, where every cell stores a height value. These maps are efficient regarding their memory footprint, as long as the resolution is low. Yet they cannot represent arbitrary 3D structures, mainly regarding multiple layers in one cell. To overcome this restriction, Triebel *et al.* developed the *multi- level surface (MLS) maps* [141] where every cell can save information about multiple layers. Unfortunately, the memory requirements still depend on the size of the environment and the resolution of the grid, independent of the actually contained 3D structures.

Puente *et al.* [31] uses planar patches to build approximate 3D maps. While plane extraction can be performed efficiently and flat structures are often dominant in man-made environments, this approach is very restricted in what it can represent.

6.7 Conclusions

In this chapter, we presented a novel method to model 3D environments by means of

surface primitives, which are used to describe reoccurring structures. We thereby minimize redundant information storage, which leads to a substantial reduction of the memory requirement. We use the Bayesian Information Criterion (BIC) as a decision criterion in the creation of the dictionary and the model to weight the memory requirements against the accuracy of the reconstruction. We presented experiments that suggest that even a small dictionary learned from the original data can be used to accurately describe complex scenes. The use of our NARF descriptors

and an adapted version of the NARF keypoint extraction made an efficient implementation of similarity checks between dictionary words and areas in the scene possible. Our method to represent 3D data does not only have lower memory requirements, it is also very convenient for further processing in approaches that require frequent similarity tests. Since information about similar structures in the environment is explicitly encoded in our representation, the need for preprocessing would be substantially reduced.

Chapter 7

Place Recognition based on 3D Range Scans

Place recognition is one of the fundamental tasks in mobile robotics. It is the ability to recognize previously seen parts of the environment based on the current obser- vation and a database of previous observations. The wide range of applications of place recognition includes localization (determine the initial pose), SLAM (de- tect loop closures), and change detection in dynamic environments. In the past, relatively little work has been carried out to address this problem using 3D range data and the majority of approaches focuses on detecting similar structures without estimating relative poses. In this chapter, we present an algorithm based on 3D range data that is able to reliably detect previously seen parts of the environment and at the same time calculates an accurate transformation between the correspond- ing scan pairs. We present an extensive set of experiments using publicly available datasets in which we compare our system to other state-of-the-art approaches.

* * * *

Place recognition describes the ability to detect that a robot revisited an already known area. It is a crucial part in key navigation tasks including localization and SLAM. The majority of state-of-the-art place recognition techniques have been developed for vision or two dimensional range data. Relatively few approaches work on three-dimensional laser range scans and can efficiently calculate the similarity and the relative transformation between two scans.

In the previous chapters we presented different systems, where finding correspondences between small parts of 3D scenes played an important role. The scale of the structures our ap- proaches were matching decreased steadily, from complete objects to partial objects, to surface patches. In this chapter we want to match structures on a larger scale, basically determining if two complete scans show the same scene.

The contribution of this chapter is a novel approach to place recognition using 3D range data. Our approach transforms a given 3D range scan into a range image and uses a combina- tion of a Bag-of-Words (BoW) approach and a point-feature-based estimation of relative poses that are then individually scored. Our approach provides high recall rates, outperforming other state-of-the-art approaches, and calculates accurate relative poses between corresponding scans. Figure 7.1 shows an example application. It visualizes how the calculated relative transforma- tions between scans can be used as edges (loop closures) in a pose graph. This enables us to apply our

approach as a front-end for a graph-based SLAM system (see Section 2.4.1). We tested our approach on different kinds of platforms: ground robots and flying vehicles. We used

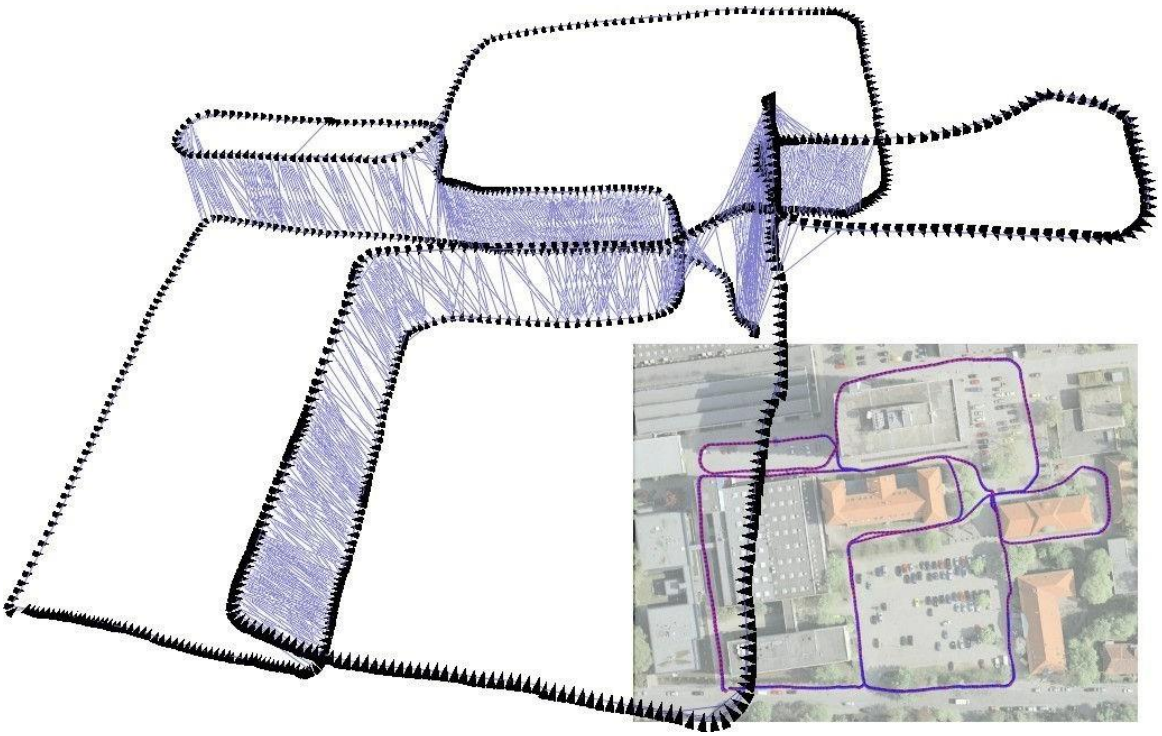


Figure 7.1: Scan correspondences found by our place recognition system for the Hanover2 dataset (see Section 7.2.1 for details on the dataset). The image shows the graph of the trajectory (black nodes) and the found loop closures between the scans (blue/gray lines). The z-axis of the trajectory represents the scan index to make the loop closures better visible. The image in the bottom right shows an aerial image from Google Earth (© Google), overlain with the trajectory of the dataset.

a combination of publicly available datasets and datasets we captured with our own robots. For the sake of repeatability we made all those datasets publicly available.

The remainder of this chapter is organized as follows: We will present our system in Sec- tion 7.1, whereas we first give an overview over the algorithm in Section 7.1.1, followed by more details on the individual components. Our experimental evaluation will be discussed in Section 7.2, followed by related work in Section 7.3. Section 7.4 concludes this chapter.

7.1 Our Approach

We will now present our place recognition system. First in a rough overview, which presents the idea of our algorithm, followed by a more detailed explanation.

7.1.1 Overview

Given a database of 3D scans and a scan as input query, our algorithm returns a set of scans which are potential matches with the input. Additionally, it calculates a transformation and a score for every returned scan, reflecting how certain the system is that the scans actually match. More formally, let $\mathcal{Z}$ denote the database of 3D range

measurements and $\mathbf{z}_{query}$ a query scan. The goal of our approach is to calculate $B(\mathbf{z}_{query}) = (\mathbf{z}_1, T_1, s_1, \ldots, \mathbf{z}_n, T_n, s_n)$, which is a set of candidate matches. Here, $\mathbf{z}_i \in Z, i \in \{1, \ldots, n\}$ are the potential measurement candi- dates from the database which are similar to the current query $\mathbf{z}_{query}$, T_i denotes the estimated

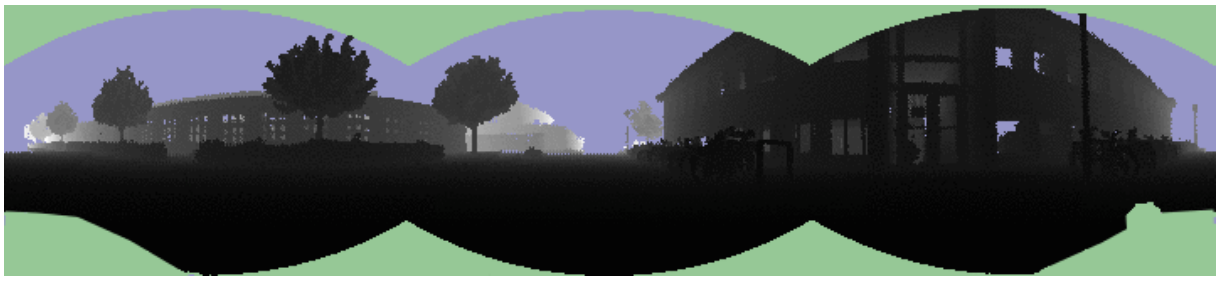


Figure 7.2: Example range image from the FreiburgCampus360_3D dataset (see Section 7.2.1 for details on the dataset). The image pixel positions represent the spherical coordinates of the points. The gray values represent the measured ranges. Blue points are maximum range readings and green points are unknown space.

transformation from $\mathbf{z}_{\text{query}}$ to $\mathbf{z}_i$, and s_i is a score reflecting the confidence about the match. Our algorithm for calculating $B(\mathbf{z}_{\text{query}})$ mainly consists of the following four steps.

1. Given a database of 3D range measurements Z (training set), calculate a set of features from the 3D scans and build a dictionary for a Bag-of-Words approach.

2. Use the Bag-of-Words approach to get an initial similarity measure for all scans in the database Z with respect to the query scan $\mathbf{z}_{\text{query}}$. This measure is then used to order the database scans from most similar to least similar. Let the resulting ordered set be
   $\bar{Z}(\mathbf{z}_{\text{query}}) = (\bar{\mathbf{z}}_1, \ldots, \bar{\mathbf{z}}_{|Z|})$.

3. For each pair $(\mathbf{z}_{\text{query}}, \bar{\mathbf{z}}_k)$, $\bar{\mathbf{z}}_k \in \bar{Z}(\mathbf{z}_{\text{query}})$, starting with $k = 1$, calculate a set of possible transformations between $\mathbf{z}_{\text{query}}$ and $\bar{\mathbf{z}}_k$ by matching point features of the corresponding scans. Note that this set of features is not the same as the one used for the Bag-of-words approach since the parameters for the feature extraction differ.

4. Score each of the possible transformations and get the transformation T_k with the highest score s_k. If this score is above an acceptance threshold then $(\bar{\mathbf{z}}_k, T_k, s_k)$ is a candidate for a recognized place, i.e., it is added to $B(\mathbf{z}_{\text{query}})$.

The last two steps are repeated until a timeout occurs or $k = |Z|$. Note that if there are no time constraints, the first two steps can be skipped so that all scans in Z are checked.

Although we work with a database of 3D range scans, we do not use this data directly. We rather represent each three-dimensional range scan by its dual, namely a range image (see Fig- ure 7.2). If the 3D scan is captured from one point in space, i.e., the sensor does not move while the 3D points are generated, the range image contains the same information as the scan. The advantage of the range image is that it allows us to model unknown areas as well as maximum range readings more efficiently. See Section 2.3.2 for more details on range images. In the following sections, we will describe the individual components of our approach in more detail.

7.1.2 Feature Extraction

Our approach uses the NARFs (Normal Aligned Radial Features) that we presented in Chap- ter 3. These point features are used to build a dictionary for the Bag-of-Words approach and also to find corresponding regions between two 3D measurements. Since each NARF encodes a full 3D transformation, a single feature correspondence between two scans is already sufficient to calculate a relative transformation.

There are three parameters needed for the extraction of NARFs. First, the size of the feature descriptor, second the maximum number of calculated features, and finally the support size, which is the size of the area around the feature point that is used to calculate the descriptor. We chose 36 as the descriptor size. For the Bag-of-Words approach, a high number of features describing small parts of the environment is most useful. Therefore we extract 2000 features with a support size of 1/10 of the average range in the database . However, when matching a new query $\mathbf{z}_{query}$ against , a smaller number of more distinctive features is needed. Here, we extract 200 features with a support size of 1/5 of the average range in . Intuitively, a small support size makes the features susceptible to noise and less distinctive, whereas a large support size makes them more expensive to compute and less robust to partial occlusion and missing data. However, we found the values above to provide reasonable trade-offs between those properties. The descriptors can be compared using standard norms like the Manhattan distance. The resulting measure (the *descriptor distance*) describes the similarity between the described regions. Here, a high value reflects a low similarity. Furthermore, NARFs can either be used in a rotationally invariant version or without invariance regarding the rotation around the normal. For example, in the rotationally variant case the features distinguish between a top right corner and a top left corner of a square, whereas they do not in the rotationally invariant case. This is a useful distinction for place recognition purposes because wheeled robots capturing 3D scans often move with very little change in their roll and pitch angle. Accordingly, they do not need the rotational invariance around the normal vector for the features. The same is the case if the robot is equipped with an IMU. This can reduce the computational complexity of the problem since the feature matching is more robust with one degree of freedom less.

7.1.3 The Bag of Words Approach

We employ a Bag-of-Words approach as a fast initial method to pre-order the scans according to their similarity to the given query scan $\mathbf{z}_{query}$. Bag-of-Words approaches are based on the idea that similar structures in an environment will create similar distributions of features. The goal is to obtain a general representation for those feature distributions. We want to encode each scan in terms of a small set of words (the *dictionary*). To learn this set, we use a training database of 3D scans and calculate 2000 NARFs for each scan. This leads to 2000 feature descriptors (each of size 36). We then apply k-means clustering to obtain a total of 200 clusters. Our dictionary is now made of 200 words, each being the averaged descriptor of its cluster. We found that this size provides a reasonable trade-off between being able to generalize and being descriptive enough. Given this dictionary, we can now express each scan $\mathbf{z}_i$ in terms of the words of the dictionary by selecting the closest word for every feature descriptor (regarding the Euclidean distance). For each $\mathbf{z}_i$, we obtain a histogram H_i having 200 bins. The number in each bin reflects how often the corresponding word is present in $\mathbf{z}_i$. Given the histogram of the query scan H_{query} (obtained in the same way), we calculate $H_{query} \; H_i \; _2$ as the distance between the histograms. This distance is then used as an initial similarity measure to create the ordered set $(\mathbf{z}_{query})$, as described in Section 7.1.1.

7.1.4 Determining Candidate Transformations

Each NARF encodes a full 3D transformation. Therefore, the knowledge about a single

fea- ture correspondence between two scans enables us to retrieve all six degrees of freedom of the relative transformation between them by calculating the difference between the two encoded transformations. To obtain the candidate transformations for each scan pair, we order the fea-

ture pairs according to increasing descriptor distance (see Section 7.1.2) and evaluate the trans- formations in this order. In other words, we calculate a score for each of these transformations (see Section 7.1.5). In our experiments we stop after a maximum number of 2000 evaluated transformations when using the rotationally variant version of the NARFs. In the rotationally invariant case we evaluate up to 5000 transformations due to the increased search space size introduced by the additional degree of freedom.

7.1.5 Scoring of Candidate Transformations

The result of the feature matching is a list of relative poses $\bar{T}_k = \{\bar{T}_{k1}, .., \bar{T}_{kn}\}$ for the can- didate pair $(\mathbf{z}_{query}, \bar{\mathbf{z}}_k)$, $\bar{\mathbf{z}}_k = \bar{\mathbf{z}}_k(\mathbf{z}_{query})$. Our goal is to evaluate those candidate transformations and calculate a score/likelihood for each $\bar{T} \in \bar{T}_k$ reflecting the confidence of the transformation given a model of our sensor. Recall that we use 3D range data, i.e., each measurement $\mathbf{z}$ is a set of 3D points. This enables us to evaluate the candidate transformation $\bar{T}$ on a point-by-point basis by assuming that the points are mutually independent.

The method for the evaluation of candidate poses we describe below is based on the same basic principles as the method we presented for the same purpose in the context of object recog- nition in Section 4.4.1. Yet, we had to adapt the scoring method to the matching of two indi- vidual range scans instead of the matching of a known point cloud object model against a range scan.

Let V be the set of *validation points* from the query scan $\mathbf{z}_{query}$. This set could contain all points from $\mathbf{z}_{query}$ but we will only use a representative subset of $\mathbf{z}_{query}$ as described in Section 7.1.6. Given a candidate transformation $\bar{T} \in \bar{T}_k$, we first transform each $\mathbf{p} \in V$ in the reference frame of $\mathbf{z}_{query}$ into a point $\mathbf{p}'$ in the reference frame of $\bar{\mathbf{z}}_k$. Since we represent our scans as range images, we can calculate the pixel position $\mathbf{p}'(x, y)$ in the range image of $\bar{\mathbf{z}}_k$ in which the point $\mathbf{p}'$ would fall into as well as the range value r' the point should have. Let $\mathbf{p}_k(x, y) \in \bar{\mathbf{z}}_k$ be the point that is already at this pixel position (in the range image of $\bar{\mathbf{z}}_k$) hav- ing the range value $r_k(x, y)$. For each $\mathbf{p} \in V$ we will now calculate a score $s_{\bar{T}}(\mathbf{p}) \in [0, 1]$ and a weight $w_{\bar{T}}(\mathbf{p}) > 0$ reflecting how good the prediction r' is explained by the observation $r_k(x, y)$. The point scores will then be used to calculate the overall likelihood $s(T)$ for the transformation T as

$$s(T) = \frac{\sum_{\mathbf{p} \in V} w_{\bar{T}}(\mathbf{p})\, s_{\bar{T}}(\mathbf{p})}{\sum_{\mathbf{p} \in V} w_{\bar{T}}(\mathbf{p})}. \tag{7.1}$$

Let $\Delta r = r_k(x, y) - r'$ be the difference between the observed and the predicted range. To evaluate Δr, we have to consider the model of the sensor. In the case of a laser scanner, a pulse of light moves from the origin of the sensor along a line to the measured point. Each range image pixel represents one such beam. There are several different cases that need to be considered regarding the interpretation of Δr:

1. The observation is within a confidence interval $\Delta r_{max} > 0$ of the prediction, i.e., $\Delta r < \Delta r_{max}$. In other words, what we expected to see mostly fits with what we measured. In

this case, we calculate the score as $s_1^-(\mathbf{p}) = 1 - \frac{|\Delta r|}{\Delta r_{\max}}$ and weight it by $w_1^-(\mathbf{p}) = 1$, which represents a triangular distribution. While a Gaussian would be a more realistic representation, we chose a triangular distribution because it is less expensive to compute. All the other cases will receive a score $s_T^-(\mathbf{p}) = 0$. Thus, the associated weight $w_T^-(\mathbf{p})$
reflects the confidence about how *wrong* the transformation T^- is.

2. The observed range is larger than the predicted one, i.e., $\Delta r > \Delta r_{max}$. This means that we actually observed something behind $\mathbf{p}'$ and basically looked through it. This could be the evidence for a dynamic or partially transparent obstacle, but in general it is a strong
indicator for a wrong transformation. We therefore penalize the overall likelihood by a high weight $w_T(\mathbf{p}) = w_{seeThrough} \geq 1$. We used $w_{seeThrough} = 25$ in our experiments.

3. The observed range is smaller than the predicted range, i.e., $\Delta r < \Delta r_{max}$. In this case, there are two more situations to distinguish:

   (a) $\bar{T}^{-1} \cdot \mathbf{p}_k(x, y)$ exists in $\mathbf{z}_{query}$. This means that we could not see $\mathbf{p}'$ in $\bar{\mathbf{z}}_k$ because of an already known obstacle. In this case we give a low weight $w_T(\mathbf{p}) = w_{knownObstacle} \leq 1$ in order to enable us to receive relatively high scores even if the overlap
   (b) between scans is low. We used $w_{knownObstacle} = 0.5$ in our experiments.

   $\bar{T}^{-1} \cdot \mathbf{p}_k(x, y)$ does not exist in $\mathbf{z}_{query}$. This could be evidence for a formerly unseen or dynamic obstacle, but in general it is a strong indicator for a wrong transformation. Similar to case 2, we penalize this by a high weight $w_T(\mathbf{p}) = w_{unknownObstacle} \geq 1$. We used $w_{unknownObstacle} = 15$ in our experiments.

4. $\mathbf{p}_k(x, y)$ is an unobserved point in the range image of $\bar{\mathbf{z}}_k$. This means that $\mathbf{p}'$ could not be observed because it is outside of the scan. We treat this the same as 3a.

5. $r_k(x, y)$ is a far range reading (i.e., exceeding the max range of the sensor) in the range image of $\bar{\mathbf{z}}_k$. There are two more situations to distinguish, for which we need to consider the original range r of $\mathbf{p}$ in $\mathbf{z}_{query}$:

   (a) The point should actually be closer to the sensor in $\bar{\mathbf{z}}_k$, i.e., $r' \leq r$. In this case it is improbable that $\mathbf{p}$ is out of range and therefore we treat this the same as case 2.
   (b) The point moved further away from the sensor in $\bar{\mathbf{z}}_k$, i.e., $r' > r$. In this case it is possible that $\mathbf{p}$ moved out of range and we give a medium high weight $w_T(\mathbf{p}) = w_{farRange} \geq 1$. We used $w_{farRange} = 5$ in our experiments.

To avoid that slight errors in the estimate of a correct transformation lead to a very small score, e.g., if the point lies on an obstacle boundary and we hit the much further away neighbor instead, we actually consider not only $\mathbf{p}_k(x, y)$ as a correspondence for $\mathbf{p}'$, but also its neighbors in a small pixel radius $e \in \mathbb{N}$ (3 in our experiments) around it and select the point with the least negative influence on the complete score.

Please note that while most of the above rules are valid for all major range sensing devices (3D laser scanners, stereo cameras, ToF-cameras, cameras with texture projectors, . . .), the used parameter values are tuned for laser scanners and especially the handling of unknown points (case 4) and far ranges (case 5) can differ significantly for other devices.

The values we chose are strict regarding any indication that our transformation might be false. This is because in general, false negatives are far less hurtful (considering, e.g., a SLAM system) than false positives. To allow more robustness, e.g.,

to dynamic obstacles, one can reduce the weights $w_{\text{seeThrough}}$, $w_{\text{unknownObstacle}}$, and $w_T(\mathbf{p}) = w_{\text{farRange}}$.

7.1.6 The Set of Validation Points

Up to now we did not say, how the set of validation points $\mathcal{V}$, from which we select $\mathbf{p}$, is obtained. In principle it could contain all the points from $\mathbf{z}_{\text{query}}$. However, this would lead to a

high number of points to be tested and thus would be computationally expensive. We therefore use only a subset of $\mathbf{z}_{\text{query}}$. A random subset of a fixed size could be used, but it is better to select points that have some significance in the scene, or two scans could get a high score, just because the floor or a big wall is well aligned. Furthermore, the points should be evenly distributed over the scan in 3D space to be invariant regarding the non-uniform resolution of
3D scans. To achieve this, we use the set of keypoints $\bar{V}$ (i.e., the points where the NARFs were extracted) that we calculated in the feature extraction process as a base to create the set of validation points. We add a random point from to and then iteratively add the point $\bar{\mathbf{p}}_i$ that has the highest 3D distance to all points already in , until a maximum size is reached (200 points in our current implementation). This has the interesting property that each ordered subset $\mathbf{p}_0, \ldots, \mathbf{p}_j$ of the ordered set = $\mathbf{p}_0, \ldots, \mathbf{p}$ is a subsampled version of with mostly equidistant points for every j. This also means that one can stop the calculation of $s(\bar{T})$ (see Eq. 7.1) before handling each point in if the score is already to low after a certain minimum of handled points (30 points in our experiments) since this subset already represents the whole set quite well. The score $s(\bar{T})$ for the transformation $\bar{T}$ is not necessarily the same as for $\bar{T}^{-1}$ (by switching the role of $\mathbf{z}_{\text{query}}$ with $\bar{\mathbf{z}}_k$). We therefore adapt the scoring to

$$s'(\bar{T}) = \min(s(\bar{T}), s(\bar{T}^{-1})), \tag{7.2}$$

as the score for the pair $(\mathbf{z}_{\text{query}}, \bar{\mathbf{z}}_k)$ with transformation $\bar{T}$.

7.1.7 Self-Similarity

There are scans that qualify only poorly for the pose estimation process because of a high self- similarity, e.g., corridors with very few distinctive structure. For areas like these it is hard to determine a correct relative transformation since different positions in the vicinity (e.g., along the corridor) lead to very similar observations. To prevent false positives (false transformations getting a high score) in those areas, we calculate a self-similarity value for every scan. We do this by matching the scan $\mathbf{z}$ against itself, using the procedure described above and consider only transformations that are not close to the identity matrix. We call the highest score in this set *self* ($\mathbf{z}$) and consider it as a measure for self similarity because it represents the definition of self similarity above: the position in the scan itself cannot be clearly distinguished because close by positions lead to similar measurements. We now use the self similarity value *self* ($\mathbf{z}$) to adapt the scoring and obtain the final score for a transformation between $\mathbf{z}_{\text{query}}$ and $\bar{\mathbf{z}}_k$ as follows:

$$s^*(\bar{T}) = (1 - (self(\mathbf{z}_{\text{query}}) + self(\bar{\mathbf{z}}_k))/2)\, s'(\bar{T}). \tag{7.3}$$

Recall that we perform the steps described so far for each candidate transformation $\bar{T}$ $\bar{T}_k$. If the best score out of all candidates is above a threshold, $\bar{\mathbf{z}}_k$ represents a potential loop closure, i.e.,

$$B(\mathbf{z}_{\text{query}}) := B(\mathbf{z}_{\text{query}}) \cup (\bar{\mathbf{z}}_k, T_k, s_k), \tag{7.4}$$

with $T_k = \text{argmax}_{\bar{T} \in \bar{T}_k} s^*(\bar{T})$ and $s_k = s^*(T_k)$.

7.1.8 Implementation Details

We perform some additional steps to improve the results. After an initial scoring of the candi- date transformations for a scan pair $(\mathbf{z}_{\text{query}}, \bar{\mathbf{z}}_k)$ we first remove transformations with a very low

score. We then cluster the transformations and identify those describing very similar relative poses, keeping only the best ones in the candidate list. Next, we perform ICP to improve the transformation estimate, using only the set of validation points to speed up this step. Finally, we update the scores given the corrected transformations and return the transformation associated with the highest score as the result.

7.2 Experimental Evaluation

We will now present the real world experiments carried out to evaluate our approach.

7.2.1 Datasets

We used four publicly available datasets of 3D scans for our experiments, namely two outdoor datasets and two indoor datasets. In the following we will give an overview over these datasets and their specific challenges.

The following datasets were used in our experiments:

- For the first indoor dataset we chose **AASS-loop**[1] [26]. This dataset was also used in the related work [48, 85], which makes a comparison easier. Its main challenge is that it contains some highly ambiguous areas in long corridors.

- For the second indoor dataset we captured 3D scans with a flying **Quadrotor** robot, equipped with a 2D laser scanner [54, 53]. This dataset [29] is challenging because of a higher noise level and the existence of highly similar scans from different poses in a corridor environment.

- For the first outdoor dataset we chose **FreiburgCampus360_3D** [27]. It contains high resolution 360° scans and its main challenge is the large distance between consecutive scans, stressing the system's ability for translational invariance.

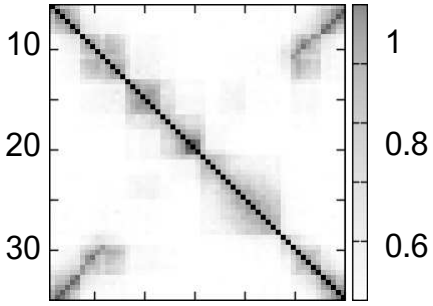
- For the second outdoor dataset we chose **Hannover2** [28]. This dataset was also used in related work [48, 85], which makes a comparison easier. This is a challenging dataset because it contains a high number of very sparse scans and the robot traverses different areas with very similar structure.

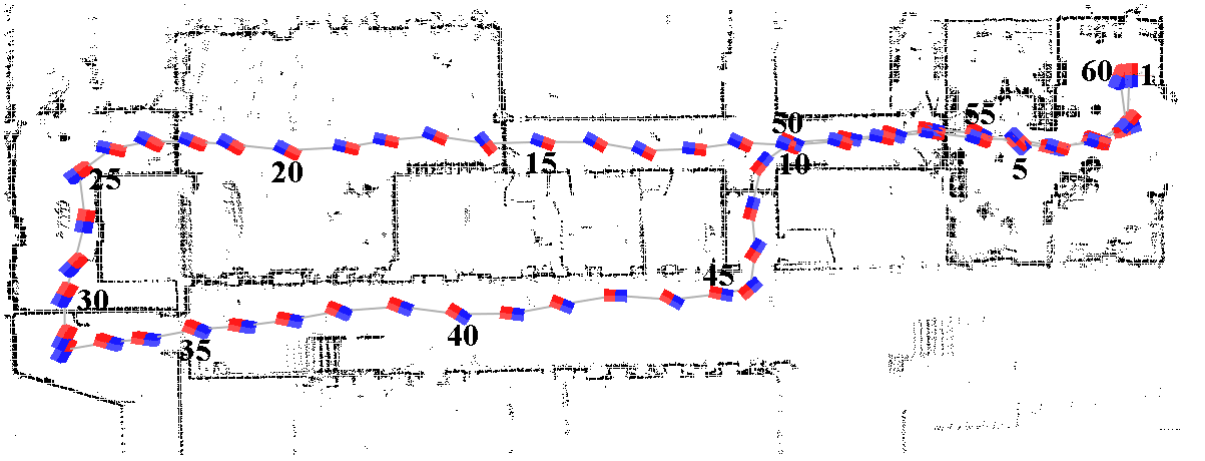
All datasets apart from the Quadrotor dataset were recorded with 2D laser scanners mounted on pan/tilt units. We acquired SLAM trajectories using the provided odometry and manually verified scan matching as edges in a graph-based SLAM system (see Section 2.4.1), which we solved using g^{2}o [70]. These trajectories were used to evaluate false/true positives and false/true negatives in our system. In the Quadrotor dataset the helicopter occasionally captured a 3D scan by flying downwards and upwards again while hovering around the same spot. The trajectory was estimated using the navigation system of the quadrotor as described by Grzonka *et al.* [54, 53]. Please refer to Figure 7.3 and Figure 7.4 for more information about the datasets.

[1]Courtesy of Martin Magnusson, AASS, Örebro University, Sweden

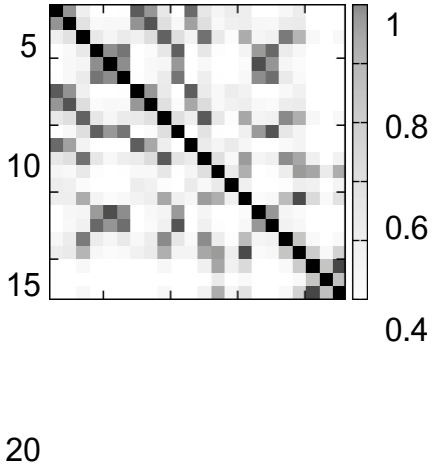
[2]Courtesy of Oliver Wulf, Leibniz University, Germany

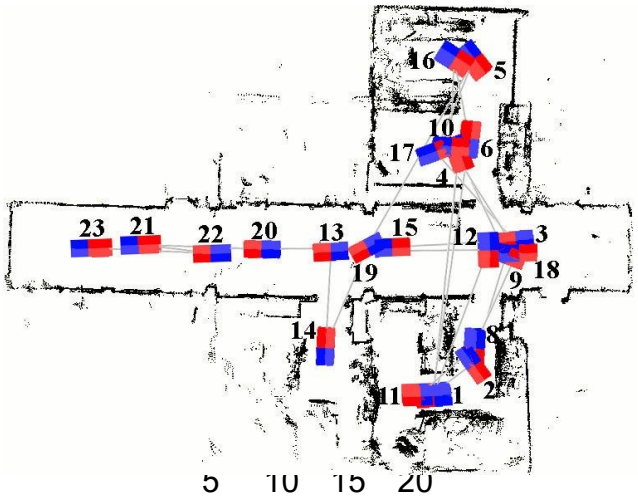
AASS-loop





Quadrotor



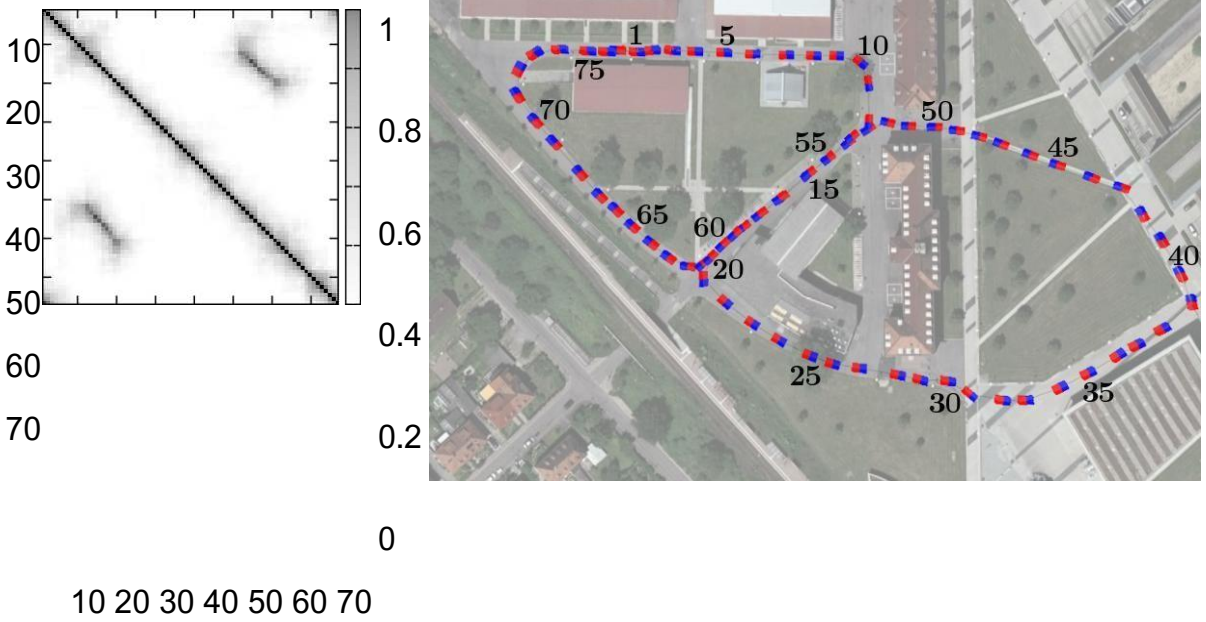


0.2 0

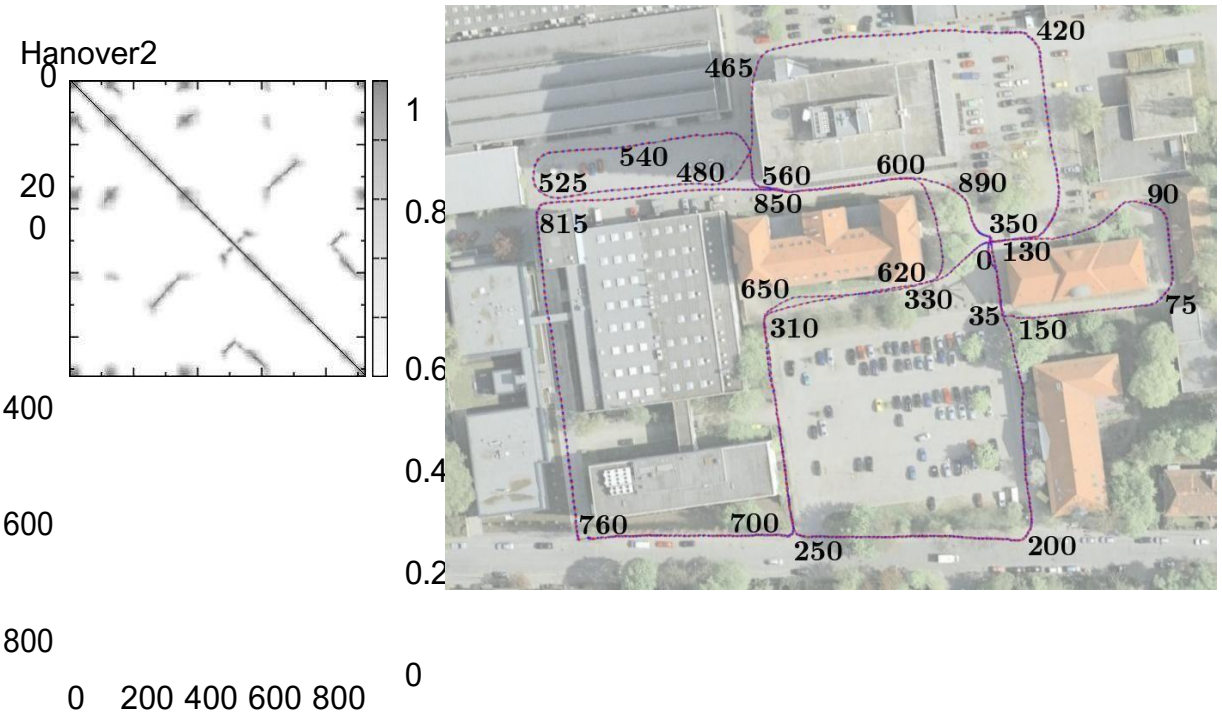
	stop and go	# scans	# points	#far ranges	res. α	traj. length	dist.	mean range	max range
AASS-loop	yes	6023	8087317160 8	n/a 95515	0.7° 1.0°	111.4 m 79.1 m	1.89 m 3.6 m	2.81 m 1.9 m	67.1 m 6.1 m
Quadrotor	no								

Figure 7.3: Top: SLAM trajectories and ground truth confusion matrices for the used indoor datasets. The trajectory is plotted on a 2D projected laser map. The gray values in the confusion matrices repre- sent the amount of overlap between the scans given the true relative poses. **Bottom:** Overview over the properties of the indoor datasets: **stop and go**=scans captured in stop and go fashion, **#scans**=number of scans, **#points**=average number of points per scan, **#far ranges**=average number of far range read- ings per scan, **res.** α=usable angular resolution for range images, **traj. length**=trajectory length, **dist.**=average distance between consecutive scans, **mean range**=average measured range value, **max range**=maximum range value

FreiburgCampus360



Hanover2



	stop and go	# scans	# points	#far ranges	res. α	traj. length	dist.	mean range	max range
Freiburg	yes	77	155562	564513622	0.45°	723 m	9.51 m	8.9 m	50 m
Hanover	no	923	12959		1.3°	1247.8 m	1.35 m	7.18 m	29 m

Figure 7.4: Top: SLAM trajectories and ground truth confusion matrices for the used outdoor datasets. The trajectories are overlaid on Google Earth aerial images. The gray values in the confusion matrices represent the amount of overlap between the scans given the true relative poses. **Bottom:** Overview over the properties of the outdoor datasets: **stop and go**=scans captured in stop and go fashion, **#scans**=number of scans, **#points**=average number of points per scan, **#far ranges**=average number of far range readings per scan, **res.** α=usable angular resolution for range images, **traj. length**=trajectory length, **dist.**=average distance between consecutive scans, **mean range**=average measured range value, **max range**=maximum range value

7.2.2 *Performance Analysis*

We calculated the confusion matrices for the datasets by matching each scan with every scan in the database and returning the score of the best found transformation. See Figures 7.5, 7.6, 7.7, and 7.8 for the different datasets. In these figures, (a) shows the respective ground truth confusion matrix and (b) the output of our system. The dark areas that are not close to the main diagonal mark loop closures. Here the system was able to match scans from different points in time where the robot visited a previously visited area (see also Figure 7.3 and Figure 7.4). Visual comparison of the two matrices for each dataset already gives the impression that our result is close to the ground truth version in most areas. To perform a more objective evaluation, we evaluate each match and check if it is a false positive. We do this by comparing the ground truth transformations between the scans with our found transformations and check if the difference exceeds an error value.
Figures 7.5, 7.6, 7.7, and 7.8 (c) give an overview over the number of true positives and false negatives and the resulting recall rate as a function of the distance between scans, using the minimum acceptance threshold for which no false positive was found. These plots show that our approach is highly invariant to translation because recall rates of above 70% could be achieved for all datasets for distances between the scans that were below the average measured range in the dataset.

Figures 7.5, 7.6, 7.7, and 7.8 (d) plot the number of false positives as a function of the acceptance threshold. The recall rate for a manually set maximum distance between scans is also shown. For the AASS-loop dataset we used 1.0 m as the distance to consider two scans a match. This is the same as was used in related work [85, 49]. The minimum acceptance threshold for which we received no false positive is 0.09. Above this value we have a recall rate of 0.938. For the Quadrotor dataset we used 2.0 m as the distance to consider two scans a match. We chose this value as the upwards rounded average range measurement in the dataset. The minimum acceptance threshold for which we received no false positive is 0.25. Above this value we have a recall rate of 0.75. For the FreiburgCampus360_3D dataset we used 10.0 m as the distance to consider two scans a match. We chose this value as the upwards rounded average range measurement in the dataset. The minimum acceptance threshold for which we received no false positive is 0.05. Above this value we have a recall rate of 0.958. For the Hanover2 dataset we used 3.0 m as the distance to consider two scans a match. This is the same as was used in related work [85, 49]. The minimum acceptance threshold for which we received no false positive is 0.19. Above this value we have a recall rate of 0.925. Please note that our evaluations do not include the diagonal elements of the confusion matrices, meaning we did not match scans against themselves.

Granström and Schön [49] used a machine learning algorithm based on boosting. Therefore they had to split their dataset into learning and test sets for the cross validation and could not evaluate the complete confusion matrix at once. They reported rates of 0.53 0.14 (min 0, max 0.88) for the AASS-loop dataset, where we achieved 0.938, and 0.63 0.6 (min 0.28, max 0.76) for the Hanover2 dataset, where we achieved 0.925. Magnusson *et al.* [85] evaluated their system in a SLAM scenario, where only scans that are at least 30 scans apart are evaluated. In this scenario they got 0.7 as the recall rate for AASS and 0.47 for Hanover2, respectively at 100% precision. With the same setting we got 1 (0.08 acceptance threshold) for AASS and 0.911 (0.19 acceptance threshold) for Hanover2. For an acceptance threshold of above 0.25, our

system does not return any false positives for all the tested datasets. Considering this as the acceptance threshold for all datasets, we achieve an overall recall rate of at least 75%.

Figure 7.9 summarizes the results for the different datasets in one plot. It shows the ROC (Relative Operating Characteristic) curves for the datasets, which plot the true positives rate

against the false positives rate. In our context that means that the bottom left represents an acceptance threshold of 1.0 for the minimum score to consider two scans a match and 0.0 for the top right. The general idea of these curves is that an approach is better, the higher the curve is above the diagonal. Below the diagonal would be worse than random assigments. Please note that the data used to create this plot does not include the values for a threshold of 0.0 (it actually starts at 0.01), which is why the curves do not reach the top right. The performance on the Quadrotor dataset is the lowest in this plot. The reason for this is that this is the dataset with the highest noise in the 3D scans.

7.2.3 Timings and Influence of the BoW approach

The values given so far are the results we receive, when we do not restrict the time requirements of our system.

For the AASS-loop dataset it takes us 881 ms to extract interest points, features and valida- tion points per scan and 585 ms to match a scan against the database, meaning 10 ms for each scan pair. The equivalent values for the Quadrotor dataset are 305 ms, 102 ms, and 4 ms, for the FreiburgCampus360_3D dataset 1107 ms, 838 ms, and 11 ms, and for the Hanover2 dataset 316 ms, 4132 ms, and 4 ms respectively. All experiments were performed using an Intel I7 quad-core PC.

When using the Bag-of-Words approach, there is an additional overhead for the creation of the histograms (including feature extraction), which is 894 ms for AASS-loop, 276 ms for Quadrotor, 730 ms for FreiburgCampus360_360, and 246 ms for Hanover2.

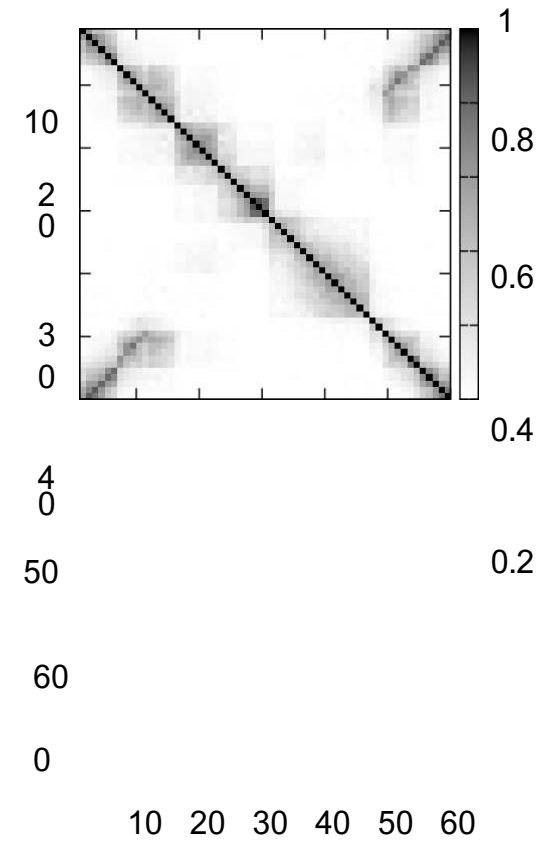
Using the Bag-of-Words pre-ordering of the potential corresponding scans, we can define a timeout for the database query. Please refer to Figures 7.5, 7.6, 7.7, and 7.8 (e) for an overview, how the recall rates (for the respective minimum acceptance threshold and maximum scan dis- tance) evolve for increasing timeout values. It can be seen that the additional overhead for the histogram calculation is only justifiable for the biggest dataset, namely Hanover2. Here, a recall rate of close to 80% can already be reached after one second per database query. In the same plots there is also a comparison between the rotationally invariant and non-invariant version of the NARFs. It can be seen that the additional degree of freedom introduced by the rotational invariance increases the typical runtime to achieve a certain recall rate and that the maximum achievable recall rate is lowered. But overall, the recall rates are still above the values of the other state-of-the-art systems in the related work.

Please note that we used the Freiburg dataset to learn the dictionary for the Bag-of-Words approach. Therefore the result from Figure 7.7(e) might be overconfident.

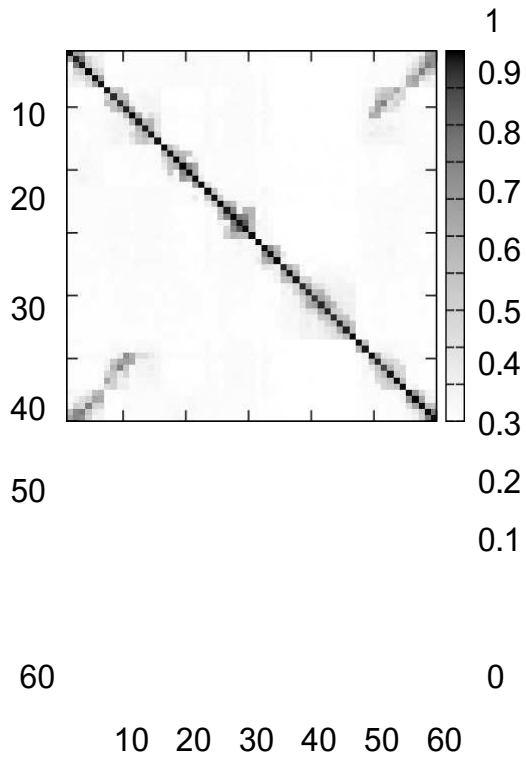
7.3 Related Work

In the past, the problem of place recognition has been addressed by several researchers and ap- proaches for different types of sensors have been developed. Cameras are often the first choice. Compared to 3D data, vision features are typically very descriptive and unique. However, spa- cial verification is naturally easier in 3D data. One very successful approach using vision is the Feature Appearance Based MAPping algorithm (FABMAP) proposed by Cummins and New- man [25]. This algorithm uses a

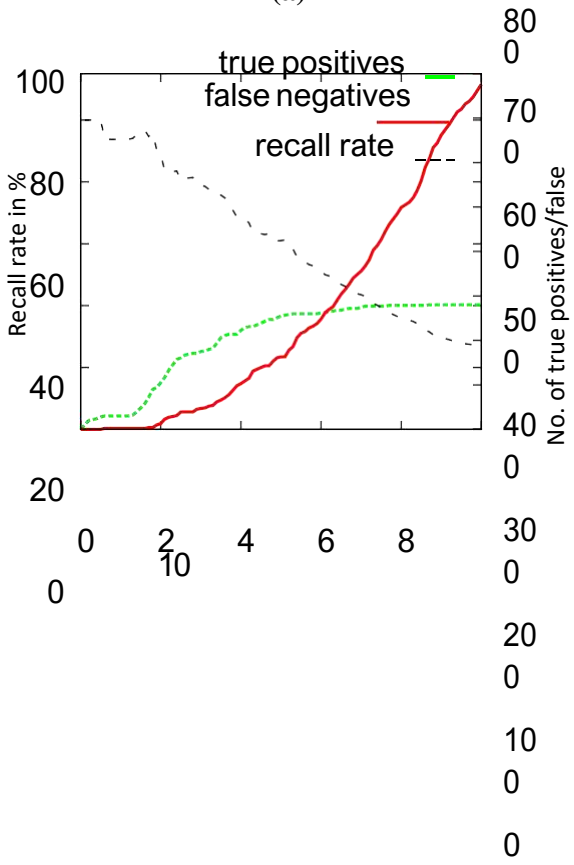
Bag-of-Words approach based on SURFs [9] extracted from omni-directional camera images and was shown to work reliably even on extremely large-scale datasets. Paul and Newman [100] presented an extended version called FAB-MAP 3D, which also incorporates 3D information into the framework. They report a substantial improvement

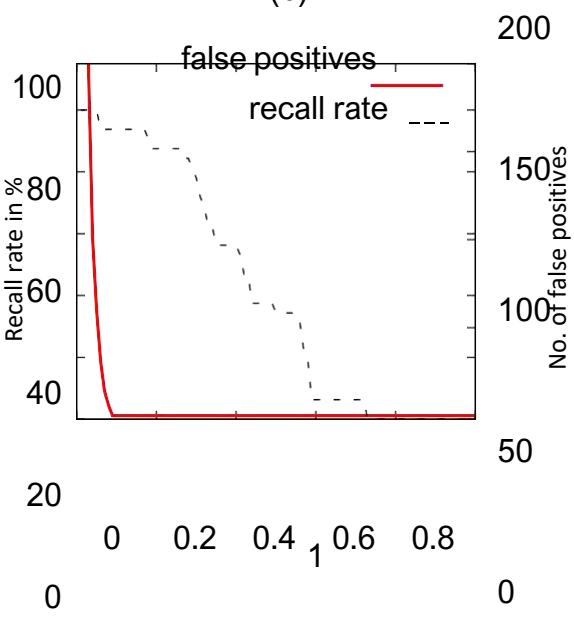

10
20
30
40
50
60
10 20 30 40 50 60
1
0.8
0.6
0.4
0.2

(a)


10
20
30
40
50
60
10 20 30 40 50 60
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0

(b)


true positives
false negatives
recall rate
Recall rate in %
No. of true positives/false
0 2 4 6 8
10
0 20 40 60 80 100
0 10 20 30 40 50 60 70 80


false positives
recall rate
Recall rate in %
No. of false positives
0 0.2 0.4 0.6 0.8 1
0 20 40 60 80 100
0 50 100 150 200

Maximum distance to neighbor scans in m

Minimal score to consider two scans a match

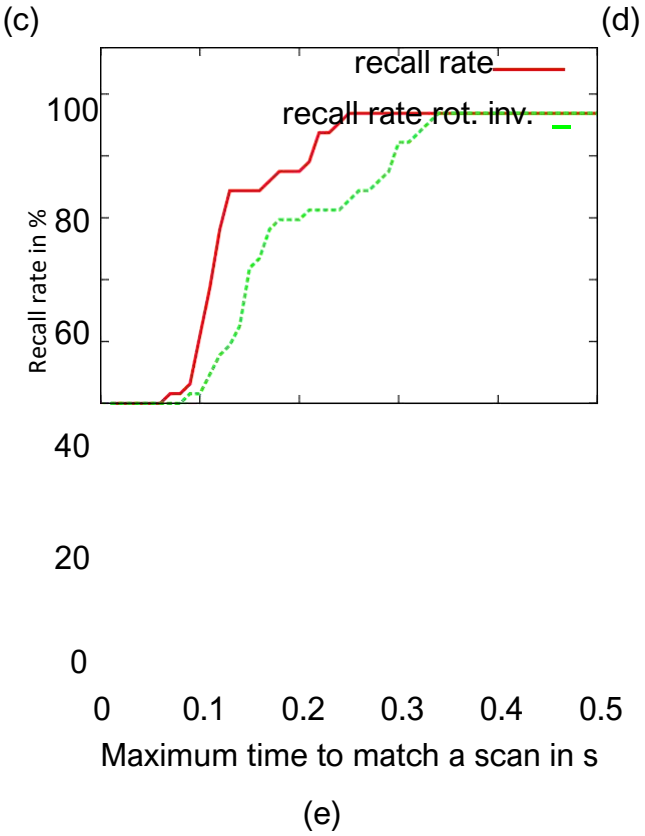
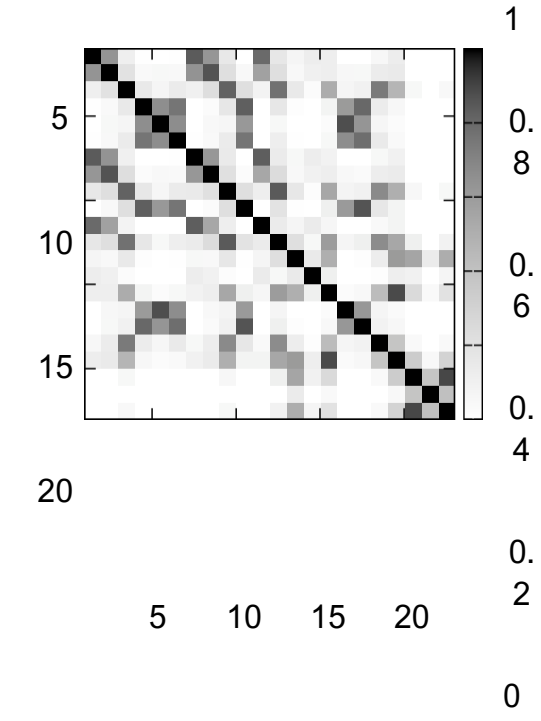
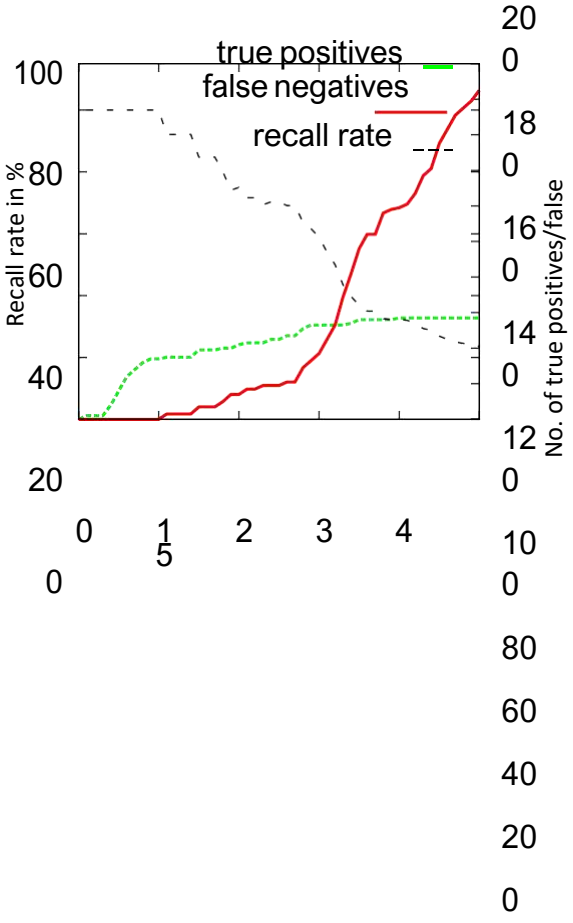


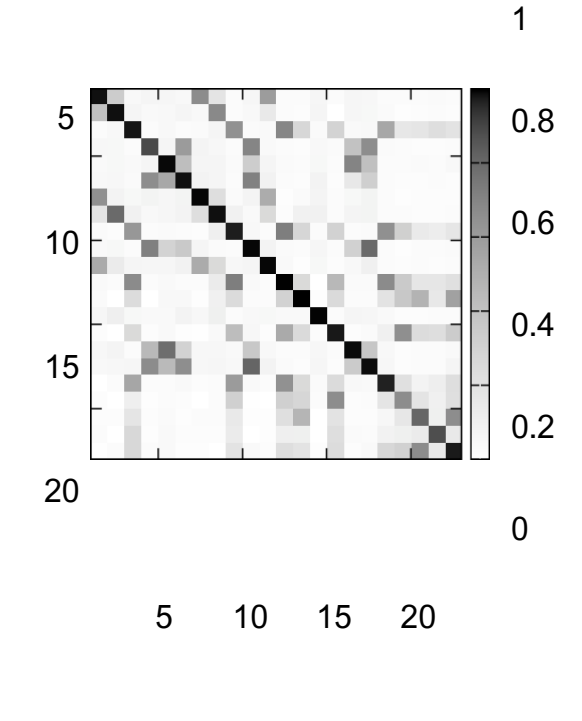
Figure 7.5: Results for the AASS-loop dataset **(a)**: Ground truth confusion matrix of the dataset. The gray values represent the overlap of the scans given the true relative pose. **(b)**: Confusion matrix created by our system. **(c)**: The number of true positives, false negatives, and the resulting recall rate for dif- ferent maximum distances between scans to consider them overlapping. Respectively for the minimum acceptance threshold that did not return any false positives. **(d)**: Number of false positives and the recall rate for different minimum scores. The recall rate is determined regarding a maximum distance of 1.0 m between the scans. **(e)**: The recall rate dependent on the maximum time the system has to match a scan against the database, using the BoW approach. The two plots represent the recall rate with and without the rotational invariance in the NARFs.



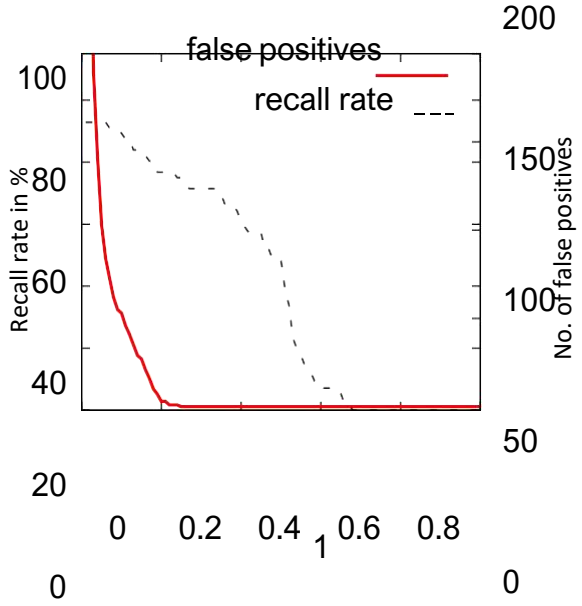
(a)



Maximum distance to neighbor scans in m



(b)



Minimal score to consider two scans a match

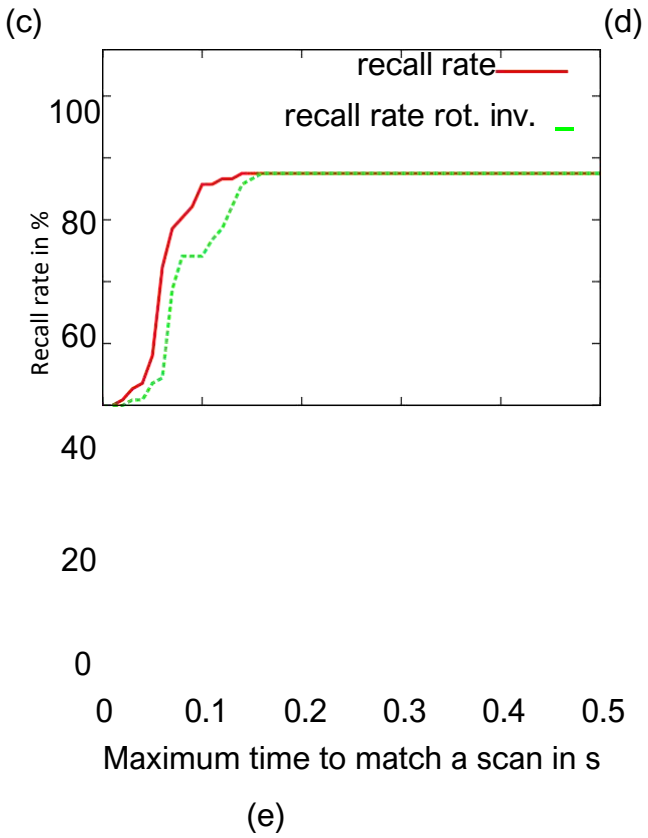
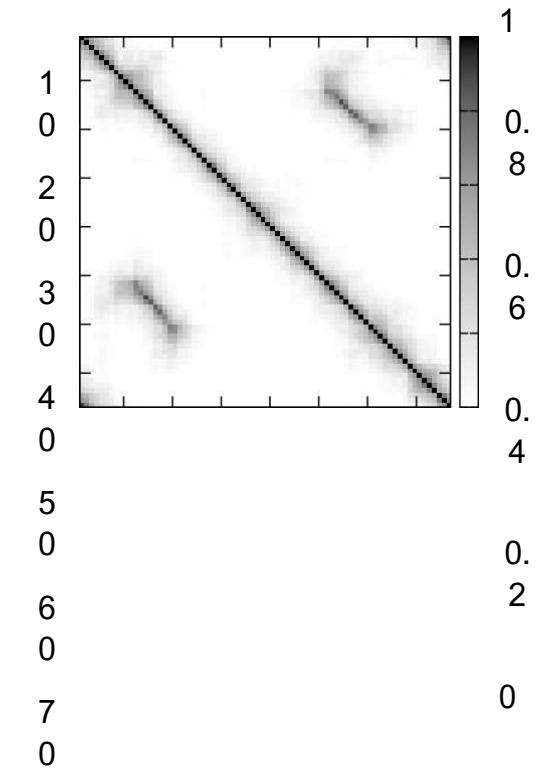
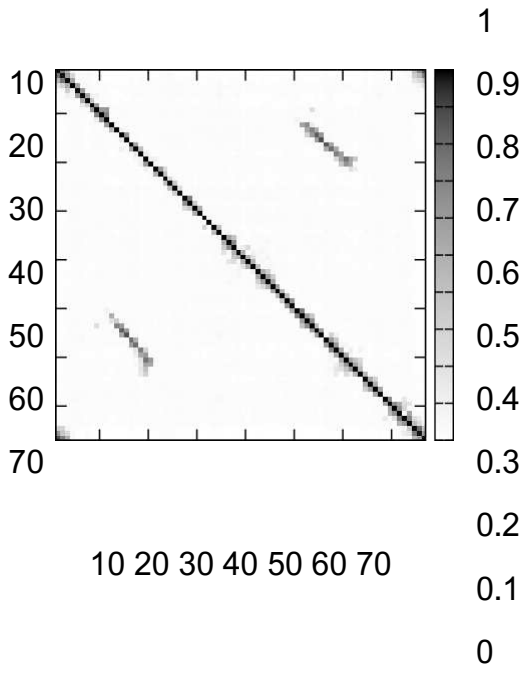
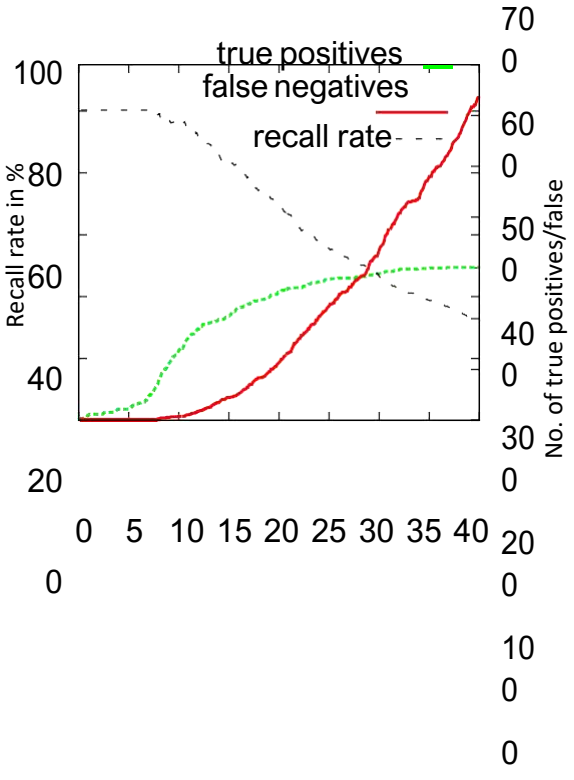


Figure 7.6: Results for the Quadrotor dataset. **(a)**: Ground truth confusion matrix of the dataset. The gray values represent the overlap of the scans given the true relative pose. **(b)**: Confusion matrix created by our system. **(c)**: The number of true positives, false negatives, and the resulting recall rate for dif- ferent maximum distances between scans to consider them overlapping. Respectively for the minimum acceptance threshold that did not return any false positives. **(d)**: Number of false positives and the recall rate for different minimum scores. The recall rate is determined regarding a maximum distance of 2.0 m between the scans. **(e)**: The recall rate dependent on the maximum time the system has to match a scan against the database, using the Bag-of-Words approach. The two graphs represent the recall rate with and without the rotational invariance in the NARFs.



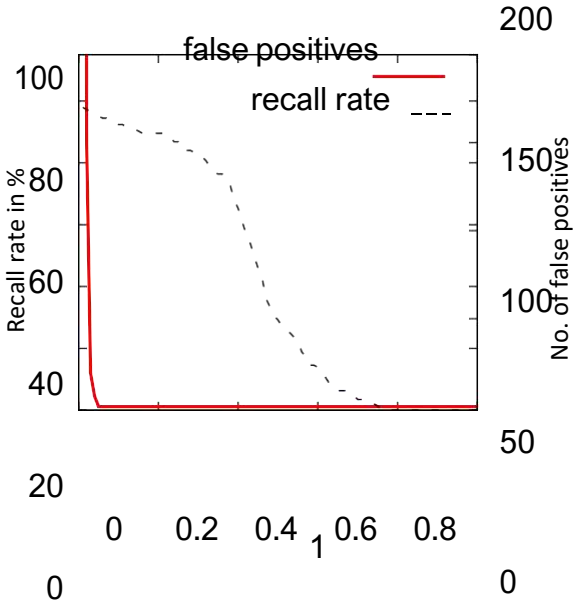


(a)



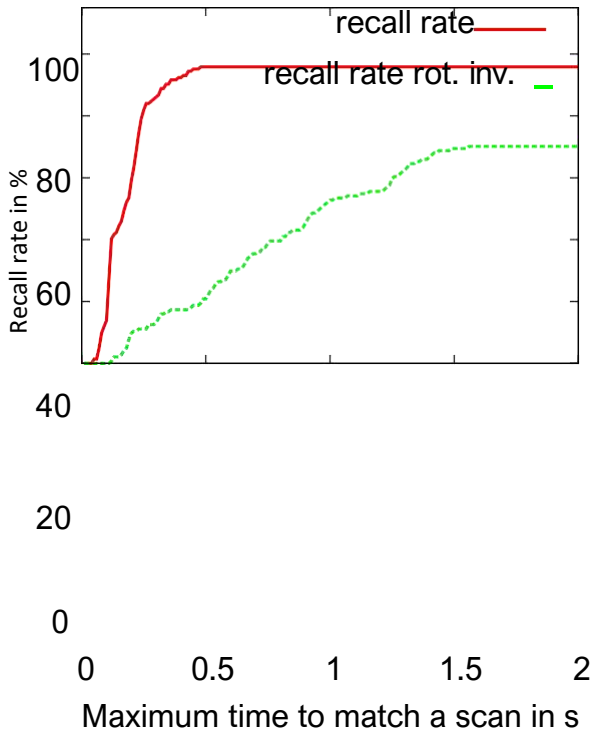
Maximum distance to neighbor scans in m

(b)



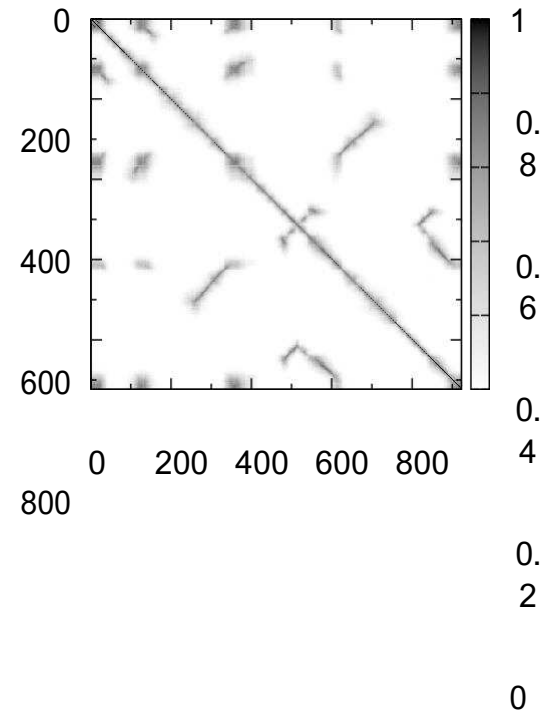
Minimal score to consider two scans a match

(c) (d)

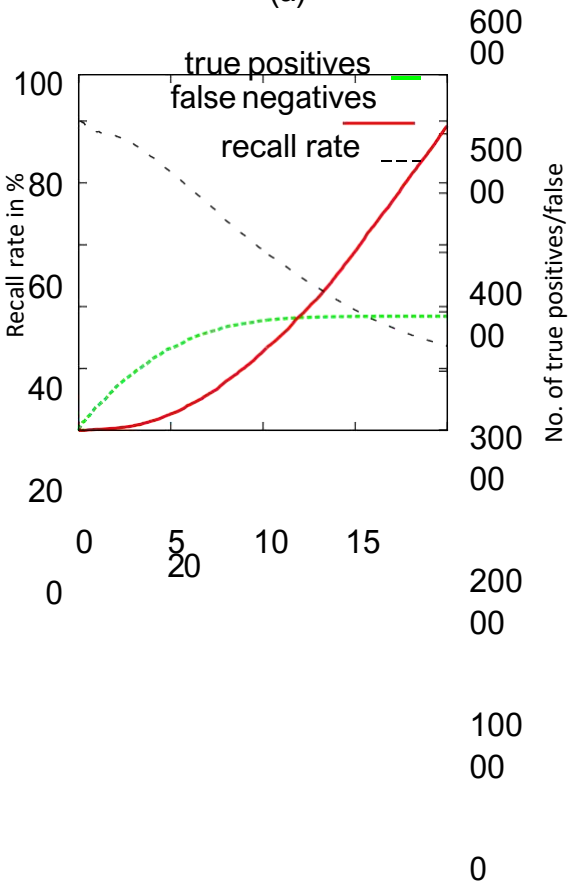


(e)

Figure 7.7: Results for the FreiburgCampus360_3D dataset. **(a)**: Ground truth confusion matrix of the dataset. The gray values represent the overlap of the scans given the true relative pose. **(b)**: Confusion matrix created by our system. **(c)**: The number of true positives, false negatives, and the resulting recall rate for different maximum distances between scans to consider them overlapping. Respectively for the minimum acceptance threshold that did not return any false positives. **(d)**: Number of false positives and the recall rate for different minimum scores. The recall rate is determined regarding a maximum distance of 10.0 m between the scans. **(e)**: The recall rate dependent on the maximum time the system has to match a scan against the database, using the Bag-of-Words approach. The two graphs represent the recall rate with and without the rotational invariance in the NARFs.

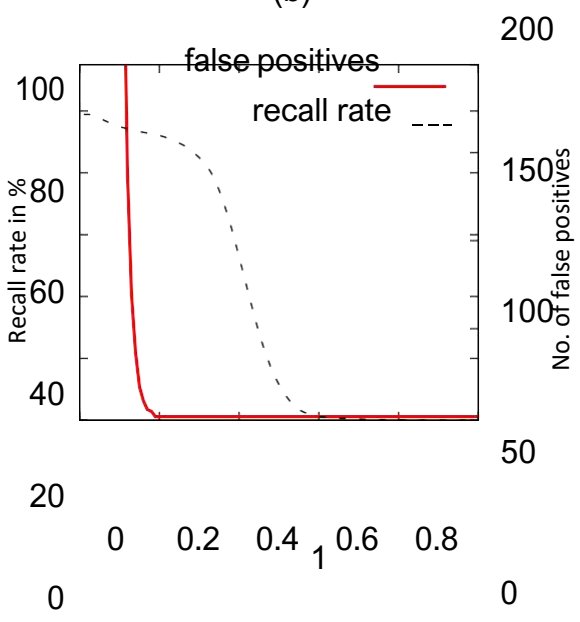


(a)



Maximum distance to neighbor scans in m

(b)



Minimal score to consider two scans a match

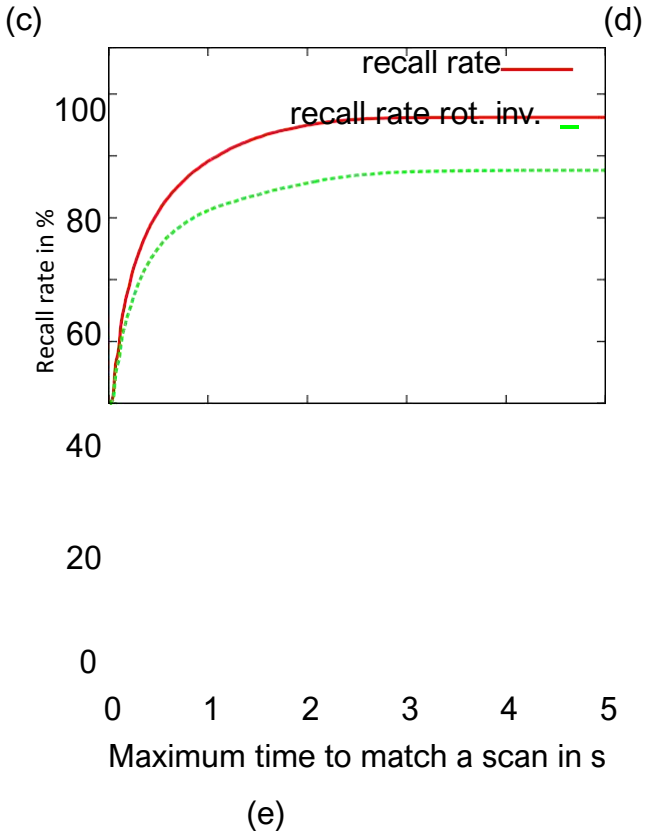
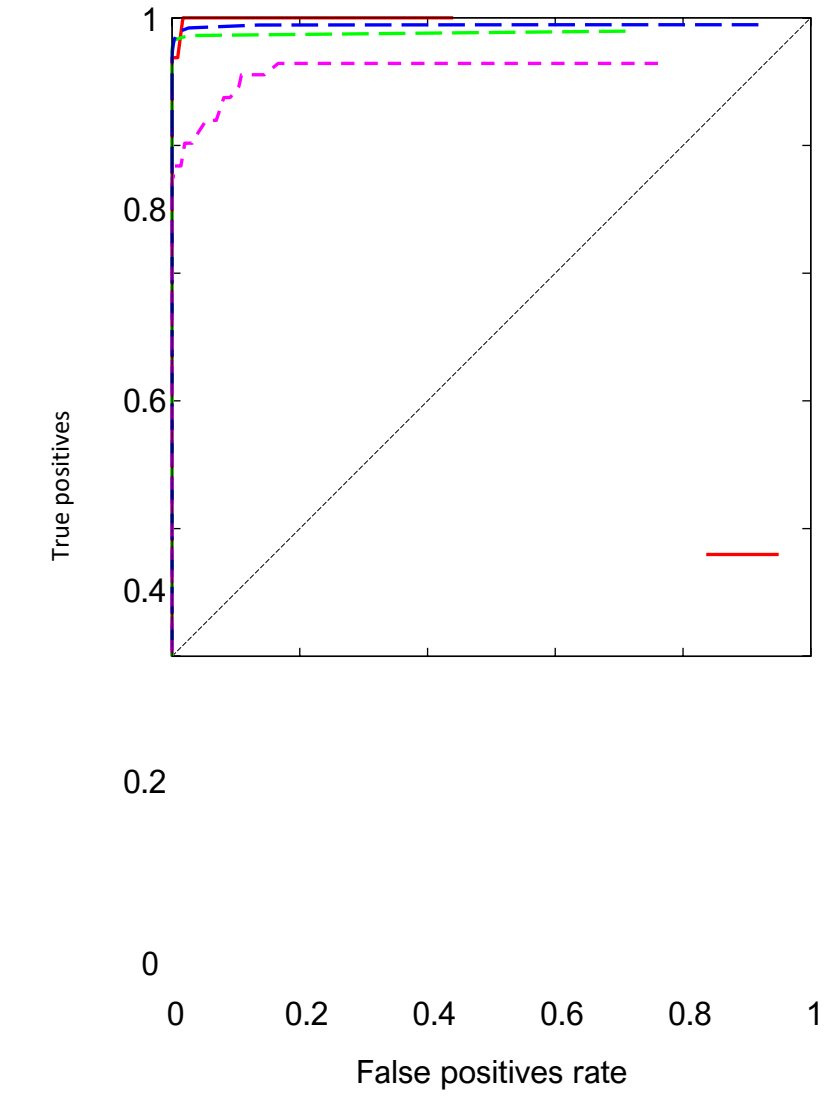
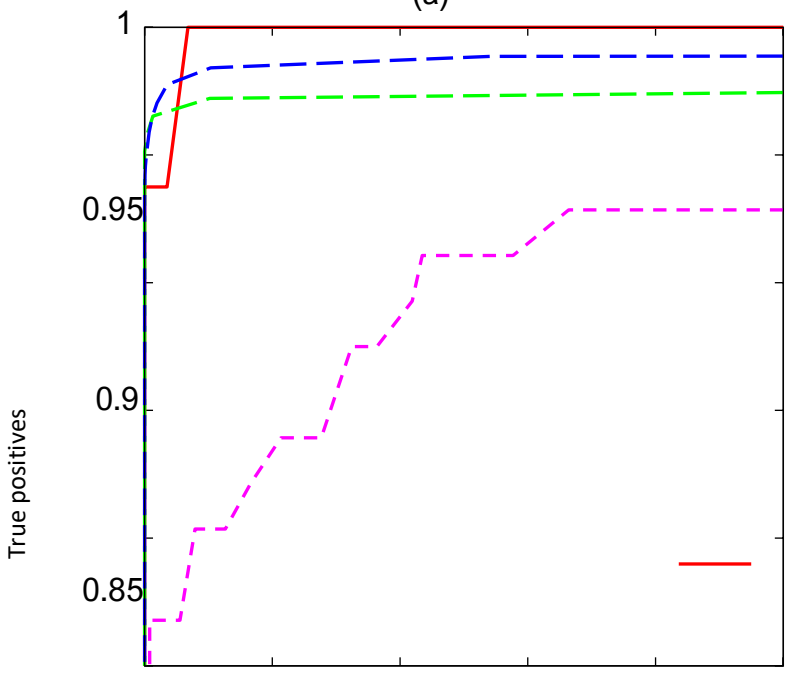


Figure 7.8: Results for the Hanover2 dataset. **(a)**: Ground truth confusion matrix of the dataset. The gray values represent the overlap of the scans given the true relative pose. **(b)**: Confusion matrix created by our system. **(c)**: The number of true positives, false negatives, and the resulting recall rate for dif- ferent maximum distances between scans to consider them overlapping. Respectively for the minimum acceptance threshold that did not return any false positives. **(d)**: Number of false positives and the recall rate for different minimum scores. The recall rate is determined regarding a maximum distance of 3.0 m between the scans. **(e)**: The recall rate dependent on the maximum time the system has to match a scan against the database, using the Bag-of-Words approach. The two graphs represent the recall rate with and without the rotational invariance in the NARFs.


1
0.8
0.6
0.4
0.2
0
True positives
0
0.2
0.4
0.6
0.8
1
False positives rate

(a)


1
0.95
0.9
0.85
True positives

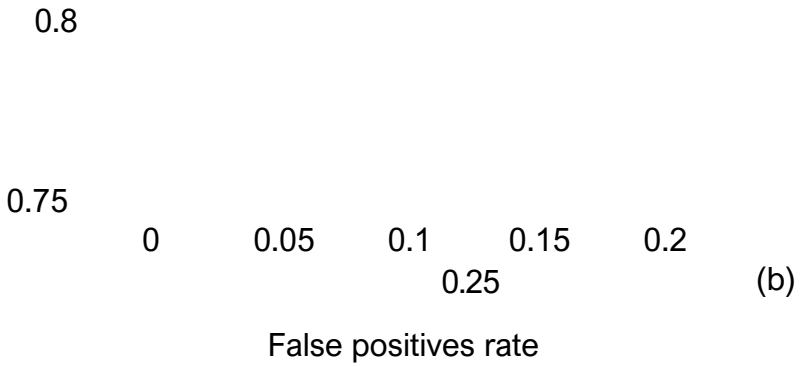


Figure 7.9: (a) shows the ROC (Relative Operating Characteristic) curves for the individual datasets.

(b) is a zoomed in version of the relevant area from the top plot. These curves plot the true positives rate against the false positives rate. The general interpretation is that a result is better, the closer the plot is to the top left corner (close to 1 on the y-axis and close to 0 on the x-axis). The data to create this specific plot does not include the values for an acceptance threshold of 0.0, which is why the curves do not reach the top right. True positives are considered for maximum distances between scan of 1.0 m for the AASS- loop dataset, 2.0 m for the Quadrotor dataset, 10.0 m for the Freiburg360_3D dataset, and 3.0 m for the Hanover2 dataset. This is equivalent to the plots in Figures 7.5(d), 7.6(d), 7.7(d), and 7.8(d).

regarding precision-recall, compared to the vision-only approach. The idea to use a BoW ap- proach with visual words was first introduced by Sivic and Zisserman [119] in an approach called Video Google, which implements a system to query for a specific object in a video stream. Next to FABMAP there are a number of successful approaches that later-on employed this method for vision-based place recognition [22, 45, 101].

An approach that is similar to ours regarding the utilization of point features to create can- didate transformations is described in the PhD thesis of Huber [64]. His approach extracts Spin Images [65] from 3D scans and uses them to match each scan against a database. Huber re- ported 1.5 s as the time requirement to match one scan against another. Even considering the advances in computer hardware since 2002, our approach is substantially faster.

Li and Olson [81] create visual images from LIDAR data, which enables them to use feature extraction methods from the vision sector to create a more universally usable point feature extractor. This feature extraction method is usable in 2D and 3D, although a 2D projection of the points is performed in the 3D case. Therefore relative poses computed from feature correspondences will also just be 2D.

Several approaches have been designed especially for 2D range data. For, example, Bosse and Zlot [15] presented a loop closure solution that builds local maps from consecutive 2D scans for which they compute histogram-based features. The correlation between these features is than used to match the local maps against each other. Tipaldi *et al.* [138] perform place recognition on 2D range scans using the so-called FLIRT-features (Fast Laser Interest Region Transform). The features are used to find correspondences between points in the scans and transformations are extracted using RANSAC. Granström *et al.* [48] proposed a machine learn- ing approach to detect loops in 2D range scans. They extract a combination of rotation-invariant features from the scans and use a binary classifier based on boosting to recognize previously visited locations.

Granström *et al.* [49] also extended their system to 3D range scans. Their system only de- tects the existence of a loop closure and does not determine the relative transformation between scans. Magnusson *et al.* [85] also proposed a system for place recognition based on 3D data. They utilize the Normal Distribution Transform as features to match scans. These features are global appearance descriptors, which describe the whole 3D scan instead of just small areas as it is the case for our features. While being very fast, their system does also not estimate relative poses. In Section 7.2.2, we compared our algorithm to these two methods. The results indicate that, while being slower, our approach yields substantially higher recall rates.

7.4 Conclusions

In this chapter we presented a robust approach to 3D place recognition. Our approach provides very high recall rates and also computes accurate relative pose estimates between the involved 3D range scans. We presented a novel sensor model that enables us to efficiently evaluate rel- ative poses between scans and introduced a method to evaluate self similarity to further reduce the risk for false positives. We evaluated our approach on freely available real world datasets and compared the performance of our

system with that of other state-of-the-art approaches. While our computational requirements are higher than those, our recall rates on the shared datasets are substantially better. We have shown that our proposed point feature type, the NARF, is also suitable for this kind of problem by providing an efficient way to find candidate transforma- tions between the scans, thereby freeing us from the need to search through a high dimensional parameter space.

7. Conclusion and Future Work

This work addresses a number of highly relevant techniques for advanced 3D perception in robotics. We presented innovative approaches for point feature extraction, object recogni- tion, object and environment modeling, place recognition, localization in aerial images, and traversability analysis. All these techniques have in common that they help a mobile robot to understand its environment. This includes information like where it is, where it can go, what is around it, and how its surrounding is structured. We extended the state of the art in these areas and created useful tools to approach the above mentioned 3D perception problems.

First, we introduced the Normal Aligned Radial Feature (NARF), providing a novel keypoint extraction method together with a new type of descriptor for 3D range scans. We then presented several approaches that build upon this point feature type to solve hard perception problems. We began with an object recognition system, which uses NARFs to find similar structures between object models and 3D scenes to determine potential object positions. These positions are then evaluated based on spatial verification, meaning we described methods to evaluate how well an assumed object pose fits the actual sensor measurements.

The point cloud object models needed for our object recognition system were initially cre- ated manually by driving around an object with a mobile platform and merging several scans. To make the modeling process more convenient, we developed an approach for unsupervised object modeling. This system takes an unordered set of 3D range scans and identifies potential partial objects. It then searches for similar structures that might belong to the same object type, but seen from slightly different perspectives. The process to find potential overlapping object parts in the data is similar to our object recognition method, but adapted for incomplete data. If enough views of an object are available in the data, our system is able to create complete models without any manual user input.

The insight that typical human-made environments contain many repetitive structures, e.g., from identical objects in the environment, led us to the idea to detect such reoccuring elements. We then only need to describe each unique structure once and store where it appears. This enables us to reduce the complexity of 3D scenes substantially. We implemented a system that builds a dictionary of surface patches from a 3D environment and can express a scene by means of these patches. We were able to show that even a small dictionary can be used to accurately describe typical 3D scenes in a condensed form, with the additional benefit that this form of representation is semantically richer since it directly encodes similarities in the environment.

Up to this point, the scale of the structures our approaches were matching decreased steadily, from complete objects to partial objects, to surface patches. Yet, there are are also areas in robotics, where the detection of similarities between larges structures is relevant. In place recognition approaches, whole scans are matched against each other to determine if a robot

already visited a certain area. This is useful for global localization purposes but also for loop closure detection in SLAM approaches. We presented such a system that can match individual 3D scans against a database of scans and reliably detects if the current observation fits one of the known places. In addition, our system calculates highly accurate relative poses between the corresponding scans.

The found links between 3D scans can be used in a SLAM system to create high quality 3D maps. However, if loop closures are infrequent, there will still be accumulated errors in the trajectory, which lead to global inconsistencies in the map. To get around this problem, we considered a source of information, which is nowadays freely available for most urban areas, namely aerial imagery. We developed a method to localize a robot in an aerial image, using 3D scans or stereo camera data. This information is then used as a prior in a mapping approach, leading to maps with increased local and global consistency.

All these approaches are mostly independent of the actual robotic platform used, apart from basic requirements regarding the sensor setup. But in practice, there are certain perceptual problems that depend strongly on the employed platform and approaches to solve them have to be adapted to the specific robot. This is especially true for the question where a robot can go in an environment. We introduced Obelix, a robotic pedestrian assistant, able to navigate in crowded urban environments. We described the complete navigation system, whereas our focus was on the traversability analysis. Such a complete system has high demands on the robustness of the individual components since every one of them can cause the robot to fail at its tasks. We showed in extensive experiments that Obelix is able to travel autonomously over large distances, avoiding positive and negative obstacles, lawn, and dynamic obstacles.

To summarize, in this work we provided the means to enable a mobile robot to accurately model its environment regarding spatial structure and reoccuring elements, determine its posi- tion by different means, recognize objects relevant for its tasks, and evaluate which areas it can safely traverse.

All the presented systems address hard perception problems and we evaluated them all in realistic scenarios and on real world data, receiving high quality results. Despite the encourag- ing outcomes, we of course plan to continue our work and further improve the capabilities of our systems, as discussed below.

Future Work

We focused mostly on 3D range data in this work. A promising next step is the integration of visual information, meaning either intensity and color information from cameras, or the remis- sion information that laser scanners provide. While we used the latter for vegetation detection, this information could also be used to match textures on otherwise featureless (regarding the 3D composition) surfaces to match or distinguish structures. Especially given the recent advent of cheap RGB-D sensors, like the Microsoft Kinect or the Asus Xtion Pro Live, combining visual information and 3D structure to solve perception problems will be a worthwhile research topic. Another useful extension to our current research would be a multi-scale extension of the NARFs. At the moment, the support size of the features, meaning the size of the area used for the keypoint detection and the descriptor calculation, is a parameter defined by the user. Yet, there are situations, where a fixed support size is suboptimal. For the object recognition, e.g., the

selection of this size should depend on the size of the object, which of course complicates scenarios where we want to search for several objects of very different sizes. Simply performing the search at different scales would be computationally expensive. Therefore, finding a method to determine the support size based on the structure or encoding information for multiple scales

into each feature would be worth investigation. This would also be advantageous for our en- vironment modeling approach based on a dictionary of surface primitives. Currently, these primitives have a fixed size. Differently sized or scalable patches might enable us to produce even more compact representations. In the same context, the positions of the word instances in the scene are currently at fixed positions. It would be interesting to investigate if one can change instance positions in the process to acquire better models, without a substantial increase in runtime.

There is, of course, always room for improvements. Robotic perception is a complex and versatile area of research without clear limits regarding what is possible and what is not. Yet, with the content of this work, we hope to have provided some useful insights and tools to facilitate the way to more intelligent and more capable robots.

11. References

[1] Adams, M.D., *Sensor Modelling, Design and Data Processing for Autonomous Navigation*. World Scientific Series in Robotics and Intelligent Systems. Singapore, World Scientific Publishing Co. Ltd., 1999.

[2] Arkin, R.C., *Behavior-Based Robotics*. Cambridge MIT Press, MA, 1998.

[3] Bar-Shalom, Y., Li, X.-R., *Estimation and Tracking: Principles, Techniques, and Software.* Norwood, MA, Artech House, 1993.

[4] Borenstein, J., Everet,t H.R., Feng, L., *Navigating Mobile Robots, Systems and Techniques*. Natick, MA, A.K. Peters, Ltd., 1996.

[5] Borenstein, J., Everett, H.R., Feng, L., *Where Am I? Sensors and Methods for Mobile Robot Positioning.* Ann Arbor, University of Michigan. Available at http:// www-

personal.engin.umich.edu/~johannb/position.htm.

[6] Breipohl, A.M., *Probabilistic Systems Analysis: An Introduction to Probabilistic Models, Decisions, and Applications of Random Processes.* New York, John Wiley & Sons, 1970.

[7] Bundy, A. (editor), *Artificial Intelligence Techniques, a Comprehensive Catalogue*. New York, Springer-Verlag, 1997.

[8] Canudas de Wit, C., Siciliano, B., and Bastin G. (editors), *Theory of Robot Control*. New York, Spinger-Verlag, 1996.

[9] Carroll, R.J., Ruppert, D., *Transformation and Weighting in Regression*. New York, Chapman and Hall, 1988.

[10] Cox, I.J., Wilfong, G.T. (editors), *Autonomous Robot Vehicles*. New York, Spinger- Verlag, 1990.

[11] Craig, J.J., *Introduction to Robotics: Mechanics and Control*. 2nd edition. Boston, Addison-Wesley, 1989.

[12] de Silva, C.W., *Control Sensors and Actuators.* Upper Saddle River, NJ, Prentice Hall, 1989.

[13] Dietrich, C.F., *Uncertainty, Calibration and Probability*. Bristol, UK, Adam Hilger, 1991.

[14] Draper, N.R., Smith, H., *Applied Regression Analysis*. 3rd edition. New York, John Wiley & Sons, 1988.

[15] Everett, H.R., *Sensors for Mobile Robots, Theory and Applications*. New York, Nat- ick, MA, A.K. Peters, Ltd., 1995.

[16] Faugeras, O., *Three-Dimensional Computer Vision, a Geometric Viewpoint*. Cam- pridge MIT Press, MA, 1993.

[17] Genesereth, M.R., Nilsson, N.J., *Logical Foundations of Artificial Intelligence.* Palo Alto, CA, Morgan Kaufmann, 1987.

[18] Haralick, R.M., Shapiro, L.G., *Computer and Robot Vision, 1+2*. Boston, Addison- Wesley, 1993.

[19] Jones, J., Flynn, A., *Mobile Robots, Inspiration to Implementation*. Natick, MA,
A.K. Peters, Ltd., 1993. [also available in German and French].

[20] Kortenkamp, D., Bonasso, R.P., Murphy, R.R. (editors), *Artificial Intelligence and Mobile Robots; Case Studies of Successful Robot Systems*. Cambridge, MA, AAAI Press / MIT Press, 1998.

[21] Latombe, J.-C., *Robot Motion Planning*. Norwood, MA, Kluwer Academic Publish- ers, 1991.

[22] Lee, D., *The Map-Building and Exploration Strategies of a Simple Sonar-Equipped Mobile Robot*. Cambridge, UK, Cambridge University Press, 1996.

[23] Leonard, J.E., Durrant-Whyte, H.F., *Directed Sonar Sensing for Mobile Robot Nav- igation*. Norwood, MA, Kluwer Academic Publishers, 1992.

[24] Manyika, J., Durrant-Whyte, H.F., *Data Fusion and Sensor Management: A Decen- tralized Information-Theoretic Approach*. Ellis Horwood, 1994.

[25] Mason, M., *Mechanics of Robotics Manipulation.* Cambridge, MA, MIT Press, 2001.

[26] Murphy, R.R., *Introduction to AI Robotics.* Cambridge, MA, MIT Press, 2000.

[27] Nourbakhsh, I., *Interleaving Planning and Execution for Autonomous Robots, Nor- wood, MA,* Kluwer Academic Publishers, 1997.

[28] Raibert, M.H., *Legged Robots That Balance*, Cambridge, MA, MIT Press, 1986.

[29] Ritter, G.X., Wilson, J.N., *Handbook of Computer Vision Algorithms in Image Alge- bra*. Boca Raton, FL, CRC Press, 1996.

[30] Russell, S., Norvig, P., *Artificial Intelligence, a Modern Approach*. Prentice Hall International, 1995.

[31] Schraft, R.-D., Schmierer, G., *Service Roboter. Natick, MA,* A.K. Peters, Ltd, 2000. [also available in German from Springer-Verlag].

[32] Sciavicco, L., Siciliano, B., *Modeling and Control of Robot Manipulators*. New York, McGraw-Hill, 1996.

[33] Todd, D.J, *Walking Machines, an Introduction to Legged Robots*. Kogan Page Ltd, 1985.

Papers

[34] Aho, A.V., "Algorithms for Finding Patterns in Strings," in J. van Leeuwen (editor), *Handbook of Theoretical Computer Science*, Cambridge, MA, MIT Press, 1990, Volume A, chapter 5, 255–300.

[35] Arras, K.O., Castellanos, J.A., Siegwart, R., "Feature-Based Multi-Hypothesis Localization and Tracking for Mobile Robots Using Geometric Constraints," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'2002)*, Washington, DC, May 11–15, 2002.

[36] Arras,K.O., Persson, J., Tomatis, N., Siegwart, R., "Real-Time Obstacle Avoidance for Polygonal Robots with a Reduced Dynamic Window," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2002),* Wash- ington, DC, May 11–15, 2002.

[37] Arras, K.O., Siegwart, R.Y., "Feature Extraction and Scene Interpretation for Map- Based Navigation and Map Building," in *Proceedings of SPIE, Mobile Robotics XII,* Vol. 3210, 1997, 42–53,.

[38] Arras, K.O., Tomatis, N., "Improving Robustness and Precision in Mobile Robot Localization by Using Laser Range Finding and Monocular Vision," in *Proceedings of the Third European Workshop on Advanced Mobile Robots (Eurobot 99)*, Zurich, September 6–9, 1999.

[39] Astolfi, A., "Exponential Stabilization of a Mobile Robot," *in Proceedings of 3rd European Control Conference,* Rome, September 1995.

[40] Barnard, K., Cardei V., Funt, B., "A Comparison of Computational Color Constancy Algorithms." *IEEE Transactions in Image Processing*. 11: 972–984, 2002.

[41] Barron, J.L., Fleet, D.J., Beauchemin, S.S., "Performance of Optical Flow Tech- niques." *International Journal of Computer Vision*, 12:43–77, 1994.

[42] Batavia, P., Nourbakhsh, I., "Path Planning for the Cye Robot," *in Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'00)*, Takamatsu, Japan, November, 2000.

[43] Borenstein, J., Koren, Y., "The Vector Field Histogram – Fast Obstacle Avoidance for Mobile Robots." *IEEE Journal of Robotics and Automation*, 7, 278–288, 1991.

[44] Brock, O., Khatib, O., "High-Speed Navigation Using the Global Dynamic Window Approach," *in Proceeding of the IEEE International Conference on Robotics and Automation*, Detroit, May 1999.

[45] Brooks, R., "A Robust Layered Control System for a Mobile Robot," *IEEE Trans- actions of Robotics and Automation*, RA-2:14–23, March 1986.

[46] Brown, H.B., Zeglin, G.Z., “The Bow Leg Hopping Robot", *in Proceedings of the IEEE International Conference on Robotics and Automation*, Leuwen, Belgium, May 1998.

[47] Bruce, J., Balch,T., and Veloso, M., “Fast and Inexpensive Color Image Segmenta- tion for Interactive Robots,” *in Proceedings of the IEEE/RSJ International Confer- ence on Intelligent Robots and Systems (IROS’00)*, Takamatsu, Japan, October 31– November 5, 2000.

[48] Burgard,W., Cremers, A., Fox, D., Hahnel, D., Lakemeyer, G., Schulz, D., Steiner, W., Thrun, S., “Experiences with an Interactive Museum Tour-Guide Robot,” *Arti- ficial Intelligence*, 114, 1–53, 2000.

[49] Burgard, W., Derr, A., Fox, D., Cremers, A., “Integrating Global Position Estima- tion and Position Tracking for Mobile Robots: The Dynamic Markov Localization Approach,” *in Proceedings of the 1998 IEEE/RSJ International Conference of Intel- ligent Robots and Systems (IROS’98)*, Victoria, B.C., Canada, October 1998.

[50] Burgard, W., Fox, D., Henning, D., “Fast Grid-Based Position Tracking for Mobile Robots,” *in Proceedings of the 21th German Conference on Artificial Intelligence (KI97)*, Freiburg, Germany, Springer-Verlag, 1997.

[51] Burgard, W., Fox, D., Jans, H., Matenar, C., Thrun, S., “Sonar-Based Mapping of Large-Scale Mobile Robot Environments using EM,” *in Proceedings of the Interna- tional Conference on Machine Learning*, Bled, Slovenia, 1999.

[52] Campion, G., Bastin, G., D'Andréa-Novel, B., "Structural Properties and Classifi- cation of Kinematic and Dynamic Models of Wheeled Mobile Robots." *IEEE Trans- actions on Robotics and Automation*, 12, No. 1, 47–62, 1996.

[53] Canudas de Wit, C., Sordalen, O.J., "Exponential Stabilization of Mobile Robots with Nonholonomic Constraints." *IEEE Transactions on Robotics and Automation*, 37, 1791–1797, 1993.

[54] Caprari, G., Estier, T., Siegwart, R., "Fascination of Down Scaling–Alice the Sugar Cube Robot." *Journal of Micro-Mechatronics*, 1, 177–189, 2002.

[55] Castellanos, J.A., Tardos, J.D., Schmidt, G., "Building a Global Map of the Envi- ronment of a Mobile Robot: The Importance of Correlations," *in Proceedings of the* 1997 *IEEE Conference on Robotics and Automation*, Albuquerque, NM, April 1997.

[56] Chen, C.T., Quinn, R.D., "A Crash Avoidance System Based upon the Cockroach Escape Response Circuit," *in Proceedings of the IEEE International Conference on Robotics and Automation*, Albuquerque, NM, April 1997.

[57] Chenavier, F., Crowley, J.L., "Position Estimation for a Mobile Robot Using Vision and Odometry," *in Proceedings of the IEEE International Conference on Robotics and Automation*, Nice, France, May 1992.

[58] Chong, K.S., Kleeman, L., "Accurate Odometry and Error Modelling for a Mobile Robot," *in Proceedings of the IEEE International Conference on Robotics and Auto- mation*, Albuquerque, NM, April 1997.

[59] Choset, H., Walker, S., Eiamsa-Ard, K., Burdick, J., "Sensor-Based Exploration: Incremental Construction of the Hierarchical Generalized Voronoi Graph." *The International Journal of Robotics Research,* 19, 126–148, 2000.

[60] Cox, I.J., Leonard, J.J., "Modeling a Dynamic Environment Using a Bayesian Mul- tiple Hypothesis Approach." *Artificial Intelligence*, 66, 311–44, 1994.

[61] Dowlingn, K., Guzikowski, R., Ladd, J., Pangels, H., Singh, S., Whittaker, W.L., "NAVLAB: An Autonomous Navigation

Testbed." *Technical report CMU-RI-TR- 87-24, Robotics Institute*, Pittsburgh, Carnegie Mellon University, November 1987.

[62] Dugan, B., "Vagabond: A Demonstration of Autonomous, Robust Outdoor Naviga- tion," *in Video Proceedings of the IEEE International Conference on Robotics and Automation*, Atlanta, GA, May 1993.

[63] Elfes, A., "Sonar-Based Real World Mapping and Navigation," *in* [10].

[64] Ens, J., Lawrence, P., "An Investigation of Methods for Determining Depth from Focus." *IEEE Transactions. on Pattern Analysis and Machine Intelligence*, 15: 97– 108, 1993.

[65] Espenschied, K. S., Quinn, R. D., "Biologically-Inspired Hexapod Robot Design and Simulation," *in AIAA Conference on Intelligent Robots in Field, Factory, Ser- vice and Space*, Houston, Texas, March 20–24, 1994.

[66] Falcone, E., Gockley, R., Porter, E., Nourbakhsh, I., "The Personal Rover Project: The Comprehensive Design of a Domestic Personal Robot," *Robotics and Autono- mous Systems, Special Issue on Socially Interactive Robots,* 42, 245–258, 2003.

[67] Feder, H.J.S., Slotin, J-J.E., "Real-Time Path Planning Using Harmonic Potentials in Dynamic Environments," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Albuquerque, NM, April 1997.

[68] Fox, D., "KLD-Sampling: Adaptive Particle Filters and Mobile Robot Localiza- tion." *Advances in Neural Information Processing Systems 14*. MIT Press, 2001.

[69] Fox, D., Burgard,W., Thrun, S., "The Dynamic Window Approach to Collision Avoidance." *IEEE Robotics and Automation Magazine*, 4:23–33, 1997.

[70] Gander,W., Golub, G.H., Strebel, R., "Least-Squares Fitting of Circles and Ellipses." *BIT Numerical Mathematics*, vol. 34, no. 4, pp. 558–578, December 1994.

[71] Genesereth, M.R. "Deliberate Agents." *Technical Report Logic-87-2.* Stanford, CA, Stanford University, Logic Group, 1987.

[72] Golub, G., Kahan,W., "Calculating the Singular Values and Pseudo-Inverse of a Matrix." *Journal SIAM Numerical Analysis*, 2:205–223, 1965.

[73] Gutmann, J.S., Burgard, W., Fox, D., Konolige, K., "An Experimental Comparison of Localization Methods," *in Proceedings of the 1998 IEEE/RSJ International. Con- ference of Intelligent Robots and Systems* (IROS'98), Victoria, B.C., Canada, Octo- ber 1998.

[74] Guttman, J.S., Konolige, K., "Incremental Mapping of Large Cyclic Environments," *in Proceedings of the IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA)*, Monterey, November 1999.

[75] Hashimoto, S., "Humanoid Robots in Waseda University - Hadaly-2 and WABIAN," *in IARP First International Workshop on Humanoid and Human Friendly Robotics*, Tsukuba, Japan, October 1998.

[76] Heale, A., Kleeman, L.: "A Real Time DSP Sonar Echo Processor," *in Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'00)*, Takamatsu, Japan, October 31–November 5, 2000.

[77] Horn, B.K.P., Schunck, B.G., "Determining Optical Flow," *Artificial Intelligence*, 17:185–203, 1981.

[78] Horswill, I., "Visual Collision Avoidance by Segmentation," *in Proceedings of IEEE International Conference on*

Robotics and Automation, 902–909, 1995, IEEE Press, Munich, November 1994.

[79] Jacobs, R. and Canny, J., "Planning Smooth Paths for Mobile Robots," *in Proceed- ing. of the IEEE Conference on Robotics and Automation*, IEEE Press,1989, pp. 2–7.

[80] Jennings, J., Kirkwood-Watts, C., Tanis, C., "Distributed Map-making and Naviga- tion in Dynamic Environments," *in Proceedings of the 1998 IEEE/RSJ Intl. Confer- ence of Intelligent Robots and Systems (IROS'98)*, Victoria, B.C., Canada, October 1998.

[81] Jensfelt, P., Austin, D., Wijk, O., Andersson, M., "Feature Based Condensation for Mobile Robot Localization," *in Proceedings of the IEEE International Conference on Robotics and Automation*, San Francisco, May 24–28, 2000.

[82] Kamon, I., Rivlin, E., Rimon, E., "A New Range-Sensor Based Globally Conver- gent Navigation Algorithm for Mobile Robots," *in Proceedings of the IEEE Inter- national Conference on Robotics and Automation*, Minneapolis, April 1996.

[83] Kelly, A., "Pose Determination and Tracking in Image Mosaic Based Vehicle Posi- tion Estimation," *in Proceeding of the IEEE/RSJ International Conference on Intel- ligent Robots and Systems (IROS'00)*, Takamatsu, Japan, October 31–November 5, 2000.

[84] Khatib, M., Chatila, R., "An Extended Potential Field Approach for Mobile Robot Sensor-Based Motions," *in Proceedings of the Intelligent Autonomous Systems IAS- 4*, IOS Press, Karlsruhe, Germany, March 1995, pp. 490–496.

[85] Khatib, M., Jaouni, H., Chatila, R., Laumod, J.P., "Dynamic Path Modification for Car-Like Nonholonomic Mobile Robots," *in Proceedings of IEEE International Conference on Robotics and Automation*, Albuquerque, NM, April 1997.

[86] Khatib, O., Quinlan, S., "Elastic Bands: Connecting, Path Planning and Control," *in Proceedings of IEEE International Conference on Robotics and Automation*, Atlanta, GA, May 1993.

[87] Ko, N.Y., Simmons, R., "The Lane-Curvature Method for Local Obstacle Avoid- ance," *in Proceedings of the 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'98)*, Victoria, B.C., Canada, October 1998.

[88] Koenig, S., Simmons, R., "Xavier: A Robot Navigation Architecture Based on Par- tially Observable Markov Decision Process Models," *in* [20]

[89] Konolige, K.,. "A Gradient Method for Realtime Robot Control," *in Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems*, Takamatsu, Japan, 2000.

[90] Konolige, K., "Small Vision Systems: Hardware and Implementation," in P*roceed- ings of Eighth International Symposium on Robotics Research*, Hayama, Japan, October 1997.

[91] Koperski, K., Adhikary, J., Han, J., "Spatial Data Mining: Progress and Challenges Survey Paper,". *in Proceedings of the ACM SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery*, Montreal, June 1996.

[92] Koren, Y., Borenstein, J., "High-Speed Obstacle Avoidance for Mobile Robotics," *in Proceedings of the IEEE Symposium on Intelligent Control*, Arlington, VA, August 1988, pp. 382-384.

[93] Koren, Y., Borenstein, J., "Real-Time Obstacle Avoidance for Fast Mobile Robots in Cluttered Environments," *in Proceedings of the IEEE International Conference on*

Robotics and Automation, Los Alamitos, CA, May 1990.

[94] Kuipers, B., Byun, Y.-T., "A Robot Exploration and Mapping Strategy Based on a Semantic Hierarchy of Spatial Representations." *Journal of Robotics and Autono- mous Systems*, 8:47–63, 1991.

[95] Lamon, P., Nourbakhsh, I., Jensen, B., Siegwar,t R., "Deriving and Matching Image Fingerprint Sequences for Mobile Robot Localization," *in Proceedings of the 2001 IEEE International Conference on Robotics and Automation*, Seoul, Korea, May 21–26, 2001.

[96] Latombe, J-C., Barraquand, J., "Robot Motion Planning: A Distributed Presentation Approach." *International Journal of Robotics Research*, 10: 628–649, 1991.

[97] Lauria, M., Estier, T., Siegwart, R.: "An Innovative Space Rover with Extended Climbing Abilities." *in Video Proceedings of the 2000 IEEE International Confer- ence on Robotics and Automation*, San Francisco, May 24–28, 2000.

[98] Lazanas, A., Latombe, J.-C., "Landmark-Based Robot Navigation," *in Proceedings of the Tenth National Conference on AI*. San Jose, CA, July 1992.

[99] Lazanas, A. Latombe, J.C., "Motion Planning with Uncertainty: A Landmark Approach." *Artificial Intelligence*, 76:285–317, 1995.

[100] Lee, S.-O., Cho, Y.-J., Hwang-Bo, M., You, B.-J., Oh, S.-R.: "A Stabile Target- Tracking Control for Unicycle Mobile Robots," *in Proceedings of the 2000 IEEE/ RSJ International Conference on Intelligent Robots and Systems*, Takamatsu, Japan, October 31–November 5, 2000.

[101] Lumelsky, V., Skewis, T., "Incorporating Range Sensing in the Robot Navigation Function." *IEEE Transactions on Systems, Man, and Cybernetics*, 20:1990, pp. 1058–1068.

[102] Lumelsky, V., Stepanov, A., "Path-Planning Strategies for a Point Mobile Automa- ton Moving Amidst Unknown Obstacles of Arbitrary Shape," in [10].

[103] Maes, P., "The Dynamics of Action Selection," *in Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, 1989, pp. 991– 997.

[104] Maes, P., "Situated Agents Can Have Goals," *Robotics and Autonomous Systems*, 6: 49–70. 1990.

[105] Martinelli, A., Siegwart, R., "Estimating the Odometry Error of a Mobile Robot during Navigation," *in Proceedings of the European Conference on Mobile Robots (ECMR 2003)*, Warsaw, September 4–6, 2003.

[106] Maybeck,P.S., "The Kalman Filter: An Introduction to Concepts," *in* [10].

[107] Minguez, J., Montano, L., "Nearness Diagram Navigation (ND): A New Real Time Collision Avoidance Approach," *in Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Takamatsu, Japan, October 2000.

[108] Minguez, J., Montano, L., "Robot Navigation in Very Complex, Dense, and Clut- tered Indoor / Outdoor Environments," *in Proceeding of International Federation of Automatic Control (IFAC2002)*, Barcelona, April 2002.

[109] Minguez, J., Montano, L., Khatib, O., "Reactive Collision Avoidance for Naviga- tion with Dynamic Constraints," *in Proceedings of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2002.

[110] Minguez, J., Montano, L., Simeon, T., Alami, R., "Global Nearness Diagram Navi- gation (GND)," *in Proceedings of*

the 2001 IEEE International Conference on Robotics and Automation, 2001.

[111] Montano, L., Asensio, J.R., "Real-Time Robot Navigation in Unstructured Environ- ments Using a 3D Laser Range Finder," *in Proceedings of the IEEE/RSJ Interna- tional Conference on Intelligent Robot and Systems*, IROS 97, IEEE Press, Vol. 2. Grenoble, France, September 1997, pp. 526–532.

[112] Moravec, H. and Elfes, A.E., "High Resolution Maps from Wide Angle Sonar," *in Proceedings of the 1985 IEEE International Conference on Robotics and Automa- tion*, IEEE Press, At.-Louis, MO, March 1985, pp. 116–121.

[113] Moutarlier, P., Chatila, R., "Stochastic Multisensory Data Fusion for Mobile Robot Location and Environment Modelling," *in Proceedings of the 5th International Symposium of Robotics Research*, Tokyo, 1989, pp. 207–216.

[114] Nayar, S.K., "Catadioptric Omnidirectional Camera." *IEEE CVPR*, pp. 482–488, 1997.

[115] Nilsson, N.J., "Shakey the Robot." *SRI, International, Technical Note*, Menlo Park, CA, 1984, No. 325.

[116] Nourbakhsh, I.R., "Dervish: An Office-Navigation Robot," *in* [20].

[117] Nourbakhsh, I.R., Andre. D., Tomasi, C., Genesereth, M.R., “Mobile Robot Obsta- cle Avoidance via Depth from Focus,” *Robotics and Autonomous Systems*, 22:151– 158, 1997.

[118] Nourbakhsh, I.R., Bobenage, J., Grange, S., Lutz, R., Meyer, R, Soto, A., “An Affective Mobile Educator with a Full-Time Job.” *Artificial Intelligence*, 114:95– 124, 1999.

[119] Nourbakhsh, I.R., Powers, R., Birchfield, S., “DERVISH, an Office-Navigation Robot.” *AI Magazine*, 16:39–51, summer 1995.

[120] Pell, B., Bernard, D., Chien, S., Gat, E., Muscettola, N., Nayak, P., Wagner, M., Williams, B., “An Autonomous Spacecraft Agent Prototype.” *Autonomous Robots*, No. 5, 1–27, 1998.

[121] Pentland, A.P., “A New Sense for Depth of Field.” *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI),* 9:523–531, 1987.

[122] Philippsen, R., Siegwart, R., “Smooth and Efficient Obstacle Avoidance for a Tour Guide Robot,” *in Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2003)*, Taipei, Taiwan, 2003.

[123] Pratt, J., Pratt, G., “Intuitive Control of a Planar Bipedal Walking Robot,” *in Pro- ceedings of the IEEE International Conference on Robotics and Automation (ICRA '98)*, Leuven, Belgium, May 16–21, 1998

[124] Raibert, M. H., Brown, H. B., Jr., Chepponis, M., “Experiments in balance with a 3D One-Legged Hopping Machine.” *International Journal of Robotics Research*, 3:75–92, 1984.

[125] Ringrose, R., “Self-Stabilizing Running,” *in Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '97)*, Albuquerque, NM, April 1997.

[126] Rowe, A., Rosenberg, C., Nourbakhsh, I., “A Simple Low Cost Color Vision Sys- tem,” *in Proceedings of Tech Sketches for CVPR 2001*. Kuaii, Hawaii, December 2001.

[127] Rubner, Y., Tomasi, C., Guibas, L., “The Earth Mover’s Distance as a Metric for Image Retrieval,” *STAN-CS-TN-98-86, Stanford University*, 1998.

[128] Schlegel, C., “Fast Local Obstacle under Kinematic and Dynamic Constraints,” *in Proceedings of the IEEE International Conference on Intelligent Robot and Systems (IROS 98)*, Victoria, B.C. Canada 1998, pp. 594–599.

[129] Schultz, A., Adams, W., “Continuous Localization Using Evidence Grids,” *in Pro- ceedings of the IEEE International Conference on Robotics and Automation (ICRA’98)*, May 16–21, 1998, Leuven, Belgium, pp.2833–2839 [also available as NCARAI Report AIC-96-007].

[130] Schweitzer, G., Werder, M., “ROBOTRAC – a Mobile Manipulator Platform for Rough Terrain,” *in Proceedings of the International Symposium on Advanced Robot Technology (ISART),* Tokyo, Japan, March, 1991.

[131] Shi, J., Malik, J., “Normalized Cuts and Image Segmentation.” *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 82:888–905, 2000.

[132] Siegwart R., et al., (2003) “Robox at Expo.02: A Large Scale Installation of Personal Robots.” *Journal of Robotics and Autonomous Systems*, 42:203–222, 2003.

[133] Siegwart, R., Lamon, P., Estier, T., Lauria, M, Piguet, R., “Innovative Design for Wheeled Locomotion in Rough Terrain,” *Journal of Robotics and Autonomous Sys- tems*, 40:151–162, 2002.

[134] Simhon, S., Dudek, G., "A Global Topological Map Formed by Local Metric Maps," *in Proceedings of the 1998 IEEE/RSJ International Conference on Intelli- gent Robots and Systems (IROS'98)*, Victoria, B.C., Canada, October 1998.

[135] Simmons, R., "The Curvature Velocity Method for Local Obstacle Avoidance," *in Proceedings of the IEEE International Conference on Robotics and Automation*, Minneapolis, April 1996.

[136] Smith, R., Self, M., Cheeseman, P., "Estimating Uncertain Spatial Relationships in Robotics," *Autonomous Robot Vehicles,* I. J. Cox and G. T. Wilfong (editors), Springer-Verlag, 1990, pp. 167–193.

[137] Sordalen, O.J., Canudas de Wi,t C., "Exponential Control Law for a Mobile Robot: Extension to Path Following," *IEEE Transactions on Robotics and Automation*, 9:837–842, 1993.

[138] Steinmetz, B.M., Arbter, K., Brunner, B., Landzettel, K., "Autonomous Vision- Based Navigation of the Nanokhod Rover," *in Proceedings of i-SAIRAS 6th Inter- national Symposium on Artificial Intelligence, Robotics and Automation in Space*, Montrea1, June 18–22, 2001.

[139] Stentz, A., "The Focussed D* Algorithm for Real-Time Replanning," in *Proceed- ings of IJCAI-95*, August 1995.

[140] Stevens, B.S., Clavel, R., Rey, L., "The DELTA Parallel Structured Robot, Yet More Performant through Direct Drive," *in Proceedings of the 23rd International Symposium on Industrial Robots*, Barcelona, October 1992, pp. 485–493.

[141] Takeda, H., Facchinetti, C., Latombe, J.C., "Planning the Motions of a Mobile Robot in a Sensory Uncertainty Field." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16:1002–1017, 1994.

[142] Thrun, S., Burgard, W., Fox, D., "A Probabilistic Approach to Concurrent Mapping and Localization for Mobile Robots." *Autonomous Robots*, 31:1–25. 1998.

[143] Thrun, S., et al., "Monrovia: A Second Generation Museum

Tour-Guide Robot," *in Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'99)*, Detroit, May 1999.

[144] Thrun, S., Fox, D., Burgard, W. Dellaert, F., "Robust Monte Carlo Localization for Mobile Robots," *Artificial Intelligence*, 128:99–141, 2001.

[145] Thrun, S., Gutmann, J.-S., Fox, D., Burgard, W., Kuipers, B., "Integrating Topolog- ical and Metric Maps for Mobile Robot Navigation: A Statistical Approach," *in Pro- ceedings of the National Conference on Artificial Intelligence (AAAI)*,1998.

[146] Tomasi, C., Shi, J., "Image Deformations Are Better Than Optical Flow." *Mathe- matical and Computer Modelling*, 24:165–175, 1996.

[147] Tomatis, N., Nourbakhsh, I., Siegwart, R., "Hybrid Simultaneous Localization and Map Building: A Natural Integration of Topological and Metric." *Robotics and Autonomous Systems* 44, 3–14, 2003.

[148] Tzafestas, C.S., Tzafestas, S.G., "Recent Algorithms for Fuzzy and Neurofuzzy Path Planning and Navigation of Autonomous Mobile Robots," *Systems-Science*, 25:25– 39, 1999.

[149] Ulrich, I., Borenstein, J., "VFH*: Local Obstacle Avoidance with Look-Ahead Ver- ification," *in Proceedings of the IEEE International Conference on Robotics and Automation*, San Francisco, May 24–28, 2000.

[150] Ulrich, I., Borenstein, J., "VFH+: Reliable Obstacle Avoidance for Fast Mobile Robots," *in Proceedings of the International Conference on Robotics and Automa- tion (ICRA'98)*, Leuven, Belgium, May 1998.

[151] Ulrich, I., Nourbakhsh, I., "Appearance-Based Obstacle Detection with Monocular Color Vision," *in the Proceedings of the AAAI National Conference on Artificial Intelligence*. Austin, TX. August 2000.

[152] Ulrich, I., Nourbakhsh, I., "Appearance-Based Place Recognition for Topological Localization," *in Proceedings of t he IEEE International Conference on Robotics and Automation*, San Francisco, pp. 1023–1029, April 2000.

[153] Vanualailai, J., Nakagiri, S., Ha, J-H., "Collision Avoidance in a Two-Point System via Liapunov's Second Method." *Mathematics and Simulation,* 39:125–141, 1995.

[154] Van Winnendael, M., Visenti G., Bertrand, R., Rieder, R., "Nanokhod Microrover Heading towards Mars," *in Proceedings of the Fifth International Symposium on Artificial Intelligence, Robotics and Automation in Space* (ESA SP-440), Noord- wijk, Netherlands, pp. 69–76, 1999.

[155] Weiss, G., Wetzler, C., Puttkamer, E., "Keeping Track of Position and Orientation of Moving Indoor Systems by Correlation of Range-Finder Scans," *in Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'94)*, Munich, pp.595–601, September 12-16, 1994.

[156] Wullschleger, F.H., Arra,s K.O., Vestli, S.J., "A Flexible Exploration Framework for Map Building," *in Proceedings of the Third European Workshop on Advanced Mobile Robots (Eurobot 99)*, Zurich, September 6-9, 1999.

[157] Yamauchi, B., Schultz, A., Adams, W., "Mobile Robot Exploration and Map-Build- ing with Continuous Localization," *in Proceedings of the IEEE International Con- ference on Robotics and Automation (ICRA'98)*, Leuven, Belgium, May 1998.

[158] Zhang, Z., "A Flexible New Technique for Camera Calibration." *Microsoft Research Technical Report 98-71*., December 1998
[see also http://research.microsoft.com/~zhang].

[1] P. Alliez and C. Gotsman. Recent advances in compression of 3D meshes. In *In Advances in Multiresolution for Geometric Modelling*, pages 3–26. Springer-Verlag, 2003.

[2] F. Amigoni, S. Gasparini, and M. Gini. Good experimental methodologies for robotic mapping: A proposal. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2007.

[3] D. Anguelov, R. Biswas, D. Koller, B. Limketkai, S. Sanner, and S. Thrun. Learning hierarchical object maps of non-stationary environments with mobile robots. In *Proc. of the Conf. on Uncertainty in Artificial Intelligence (UAI)*, pages 10–17, 2002.

[4] A. Ansar, A. Castano, and L. Matthies. Enhanced Real-time Stereo Using Bilateral Fil- tering. In *Int. Symp. on 3D Data Processing, Visualization and Transmission (3DPVT)*. IEEE Computer Society, 2004.

[5] Y. Asahara, T. Yamamoto, M. Van Loock, B. Steder, G. Grisetti, and W. Burgard. Object recognition method, object recognition apparatus, and autonomous mobile robot, 2012. (PATENT).

[6] B. Balaguer, S. Carpin, and S. Balakirsky. Towards quantitative comparisons of robot algorithms: Experiences with SLAM in simulation and real world systems. In *IROS Workshop*

on Performance Evaluation and Benchmarking for Intelligent Robots, 2007.

[7] J. Bares, M. Hebert, T. Kanade, E. Krotkov, T. Mitchell, R. Simmons, and W. R. L. Whittaker. Ambler: An autonomous rover for planetary exploration. *IEEE Computer Society Press*, 22(6):18–22, 1989.

[8] A. Bauer, K. Klasing, G. Lidoris, Q. Mühlbauer, F. Rohrmüller, S. Sosnowski, T. Xu,
[9] Kühnlenz, D. Wollherr, and M. Buss. The autonomous city explorer: Towards natural human-robot interaction in urban environments. *Int. Journal of Social Robotics*, 1:127– 140, 2009.

[10] H. Bay, T. Tuytelaars, and L. Van Gool. SURF: Speeded Up Robust Features. In *Proc. of the European Conf. on Computer Vision (ECCV)*, 2006.

[11] J. Behley, V. Steinhage, and A. Cremers. Performance of histogram descriptors for the classification of 3D laser range data in urban environments. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2012.

[12] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

[13] M. Blum, J. T. Springenberg, J. Wülfing, and M. Riedmiller. A learned feature descriptor for object recognition in RGB-D data. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2012.

[14] L. Bo, X. Ren, and D. Fox. Unsupervised Feature Learning for RGB-D Based Object Recognition. In *Proc. of the Int. Symp. on Experimental Robotics (ISER)*, 2012.

[15] A. Bonarini, W. Burgard, G. Fontana, M. Matteucci, D. G. Sorrenti, and J. D. Tardos. Rawseeds a project on SLAM benchmarking. In *Proc. of the Workshop on Benchmarks in Robotics Research (IROS)*, 2006.

[16] M. Bosse and R. Zlot. Map matching and data association for large-scale two-dimensional laser scan-based slam. *Int. Journal of Robotics Research (IJRR)*, 27(6):667– 691, 2008.

[17] W. Burgard, A. B. Cremers, D. Fox, D. Hähnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun. The interactive museum tour-guide robot. In *Proc. of the National Conf. on Artificial Intelligence (AAAI)*, 1998.

[18] W. Burgard, C. Stachniss, G. Grisetti, B. Steder, R. Kümmerle, C. Dornhege, M. Ruhnke,
a. Kleiner, and J. D. Tardós. A comparison of SLAM algorithms based on a graph of relations. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2009.

[19] M. Calonder, V. Lepetit, M. Ozuysal, T. Trzcinski, C. Strecha, and P. Fua. BRIEF: Computing a Local Binary Descriptor Very Fast. *IEEE Transactions on Pattern Analysis and Machine Intelligence (T-PAMI)*, 34(7):1281–1298, 2012.

[20] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence (T-PAMI)*, 8(6):679–698, 1986.

[21] A. Censi. Scan matching in a probabilistic framework. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, pages 2291–2296, 2006.

[22] C. Chen and H. Wang. Large-scale loop-closing by fusing range data and aerial image.
i. *Int. Journal of Robotics and Automation (IJRA)*, 22(2):160–169, 2007.

[23] O. Chum, A. Mikulik, M. Perdoch, and J. Matas. Total recall ii: Query expansion revisited. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2011.

[24] L. Cremean, T. Foote, J. Gillula, G. Hines, D. Kogan, K. Kriechbaum, J. Lamb, J. Leibs,
[25] Lindzey, C. Rasmussen, A. Stewart, J. Burdick, and R. Murray. Alice: An information- rich autonomous vehicle for high-speed desert navigation. *Journal of Field Robotics (JFR)*, 2006.

[26] G. Csurka, C. R. Dance, L. Fan, J. Willamowski, and C. Bray. Visual categorization with bags of keypoints. In *In Workshop on Statistical Learning in Computer Vision, ECCV*, pages 1–22, 2004.

[27] M. Cummins and P. Newman. Appearance-only SLAM at large scale with FAB-MAP
i. 2.0. *Int. Journal of Robotics Research (IJRR)*, 2010.

[28] Dataset: AASS-loop. http://kos.informatik.uni-osnabrueck.de/3Dscans.

[29] Dataset: Freiburg360 - 360° 3D scans of the Faculty of Engineering, University of Freiburg, Germany. http://ais.informatik.uni-freiburg.de/projects/datasets/fr360.

[30] Dataset: Hanover2. http://kos.informatik.uni-osnabrueck.de/3Dscans.

[31] Dataset: QuadrotorFreiburg3D - 3D scans of building 79 of the Faculty of En- gineering, University of Freiburg, Germany, captured by a flying quadrotor robot. http://ais.informatik.uni-freiburg.de/projects/datasets/quadrotor079.

[32] A. Davison, I. Reid, , N. Molton, and O. Stasse. MonoSLAM: Real time single camera SLAM. *IEEE Transactions on Pattern Analysis and Machine Intelligence (T-PAMI)*, 29(6), 2007.

[33] P. de la Puente, D. Rodriguez-Losada, A. Valero, and F. Matia. 3D Feature Based Map- ping Towards Mobile Robots' Enhanced Performance in Rescue Missions. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2009.

[34] F. Dellaert, D. Fox, W. Burgard, and S. Thrun. Monte carlo localization for mobile robots. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 1998.

[35] P. Deutsch. GZIP file format specification version 4.3. RFC 1952 (Informational), 1996.

[36] M. Ding, K. Lyngbaek, and A. Zakhor. Automatic registration of aerial imagery with untextured 3D lidar models. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2008.

[37] C. U. Dogruer, K. A. Bugra, and M. Dolen. Global urban localization of outdoor mobile robots using satellite images. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2007.

[38] A. Doucet, N. de Freitas, and N. Gordan, editors. *Sequential Monte-Carlo Methods in Practice*. Springer Verlag, 2001.

[39] F. Endres, C. Plagemann, C. Stachniss, and W. Burgard. Unsupervised discovery of object classes from range data using latent dirichlet allocation. In *Proc. of Robotics: Science and Systems (RSS)*, 2009.

[40] The European robotic pedestrian assistant. http://europa.informatik.uni-freiburg.de, 2009.

[41] R. Eustice, H. Singh, and J. Leonard. Exactly Sparse Delayed-State Filters. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, pages 2428–2435, 2005.

[42] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM (CACM)*, 24(6):381–395, 1981.

[43] J. D. Foley, A. Van Dam, K. Feiner, J. Hughes, and P. R.L. *Introduction to Computer Graphics*.

Addison-Wesley, 1993.

[44] U. Frese, P. Larsson, and T. Duckett. A Multilevel Relaxation Algorithm for Simultaneous Localisation and Mapping. In *IEEE Transactions on Robotics (T-RO)*, volume 21, pages 1–12, 2005.

[45] A. Frome, D. Huber, R. Kolluri, T. Bulow, and J. Malik. Recognizing objects in range data using regional point descriptors. In *Proc. of the European Conf. on Computer Vision (ECCV)*, 2004.

[46] C. Früh and A. Zakhor. An automated method for large-scale, ground-based city model acquisition. *Int. Journal on Computer Vision (ICCV)*, 60:5–24, 2004.

[47] D. Galvez-López and J. Tardos. Bags of binary words for fast place recognition in image sequences. *IEEE Transactions on Robotics (T-RO)*, 28(5):1188 –1197, 2012.

[48] N. Gelfand, N. J. Mitra, L. J. Guibas, and H. Pottmann. Robust global registration. In i. *Proc. of the third Eurographics Symp. on Geometry processing*, page 197, 2005.

[49] Google self-driving car project. http://googleblog.blogspot.de/2012/08/the-self-driving- car-logs-more-miles-on.html, 2012. Accessed: 13.02.2013.

[50] K. Granström, J. Callmer, F. Ramos, and J. Nieto. Learning to detect loop closure from range data. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2009.

[51] K. Granström and T. Schön. Learning to close the loop from 3D point clouds. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2010.

[52] G. Grisetti, C. Stachniss, and W. Burgard. Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE Transactions on Robotics (T-RO)*, 23:34–46, 2007.

[53] G. Grisetti, C. Stachniss, and W. Burgard. Nonlinear constraint network optimization for efficient map learning. *IEEE Transactions on Intelligent Transportation Systems (T-ITS)*, 10(3):428–439, 2009.

[54] H.-M. Gross, H. Boehme, C. Schroeter, S. Mueller, A. Koenig, E. Einhorn, C. Mar- tin, M. Merten, and A. Bley. TOOMAS: Interactive shopping guide robots in everyday use - final implementation and experiences from long-term field trials. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2009.

[55] S. Grzonka, G. Grisetti, and W. Burgard. Towards a Navigation System for Autonomous Indoor Flying. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2009.

[56] S. Grzonka, G. Grisetti, and W. Burgard. A Fully Autonomous Indoor Quadrotor. *IEEE Transactions on Robotics (T-RO)*, 8(1):90–100, 2012.

[57] J. Gutmann and K. Konolige. Incremental Mapping of Large Cyclic Environments. In *Proc. of the IEEE Int. Symp. on Computational Intelligence in Robotics and Automation*

(CIRA), pages 318–325, 1999.

[58] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, second edition, 2004.

[59] M. Hebert, C. Caillas, E. Krotkov, I. Kweon, and T. Kanade. Terrain mapping for a roving planetary explorer. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, pages 997–1002, 1989.

[60] E. Herbst, X. Ren, and D. Fox. Rgb-d object discovery via multi-scene analysis. In 1. *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2011.

[61] A. Hoover, G. Jean-Baptiste, X. Jiang, P. J. Flynn, H. Bunke, D. B. Goldgof, K. K. Bowyer, D. W. Eggert, A. W. Fitzgibbon, and R. B. Fisher. An experimental comparison of range image segmentation algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence (T-PAMI)*, 18(7):673–689, 1996.

[62] B. K. P. Horn. Closed-form solution of absolute orientation using unit quaternions. *Jour- nal of the Optical Society of America. A*, 4(4):629–642, 1987.

[63] A. Howard. Multi-robot mapping using manifold representations. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, pages 4198–4203, 2004.

[64] A. Howard and N. Roy. Radish: The robotics data set repository, standard data sets for the robotics community, 2003. http://radish.sourceforge.net/.

[65] Q. Huang, S. Flöry, N. Gelfand, M. Hofer, and H. Pottmann. Reassembling fractured objects by geometric matching. *ACM Transactions on Graphics (TOG)*, 25(3):569–578, 2006.

[66] D. Huber. *Automatic Three-dimensional Modeling from Reality*. PhD thesis, Robotics Institute, Carnegie Mellon University, 2002.

[67] A. Johnson and M. Hebert. Using spin images for efficient object recognition in cluttered 3D scenes. *IEEE Transactions on Pattern Analysis and Machine Intelligence (T-PAMI)*, 21(5):433–449, 1999.

[68] S. Julier, J. Uhlmann, and H. Durrant-Whyte. A new Approach for Filtering Nonlinear Systems. In *Proc. of the American Control Conf. (ACC)*, pages 1628–1632, 1995.

[69] G. Kim, C. Faloutsos, and M. Hebert. Unsupervised modeling of object categories using link analysis techniques. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2008.

[70] K. Konolige, E. Marder-Eppstein, and B. Marthi. Navigation in hybrid metric-topological maps. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2011.

[71] T. Korah and C. Rasmussen. Probabilistic contour extraction with model-switching for vehicle localization. In *IEEE Intelligent Vehicles Symp. (IVS)*, pages 710–715, 2004.

[72] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. g2o: A General Framework for Graph Optimization. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2011.

[73] R. Kümmerle, M. Ruhnke, B. Steder, C. Stachniss, and W. Burgard. A navigation system for robots operating in crowded urban environments. In *Proc. of the IEEE Int. Conf. on Robotics*

& Automation (ICRA), 2013. Accepted for Publication.

[74] R. Kümmerle, B. Steder, C. Dornhege, A. Kleiner, G. Grisetti, and W. Burgard. Large scale graph-based SLAM using aerial images as prior information. In *Proc. of Robotics: Science and Systems (RSS)*, 2009.

[75] R. Kümmerle, B. Steder, C. Dornhege, A. Kleiner, G. Grisetti, and W. Burgard. Large scale graph-based SLAM using aerial images as prior information. *Journal of Autonomous Robots*, 30(1):25–39, 2011.

[76] R. Kümmerle, B. Steder, C. Dornhege, M. Ruhnke, G. Grisetti, C. Stachniss, and A. Kleiner. On measuring the accuracy of SLAM algorithms. *Journal of Autonomous Robots*, 27(4):387–407, 2009.

[77] S. Lacroix, A. Mallet, D. Bonnafous, G. Bauzil, S. Fleury, M. Herrb, and R. Chatila. Autonomous rover navigation on unknown terrains: Functions and integration. *Int. Journal of Robotics Research (IJRR)*, 21(10-11):917–942, 2002.

[78] K. W. Lee, S. Wijesoma, and J. I. n. Guzmán. A constrained slam approach to robust and accurate localisation of autonomous ground vehicles. *Robot. Auton. Syst.*, 55(7):527–540, 2007.

[79] J. Leonard and H. Durrant-Whyte. Mobile Robot Localization by Tracking Geometric Beacons. In *IEEE Transactions on Robotics and Automation (T-RA)*, volume 7, pages 376–382, 1991.

[80] K. Y. K. Leung, C. M. Clark, and J. P. Huissoon. Localization in urban environments by matching ground level video images with an aerial image. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2008.

[81] S. Leutenegger, M. Chli, and R. Siegwart. Brisk: Binary robust invariant scalable keypoints. In *Proc. of the Int. Conf. on Computer Vision (ICCV)*, pages 2548 –2555, 2011.

[82] X. Li and I. Guskov. Multi-scale features for approximate alignment of point-based surfaces. In *Proc. of the third Eurographics Symp. on Geometry processing (SGP)*, 2005.

[83] Y. Li and E. Olson. A general purpose feature extractor for light detection and ranging data. *Sensors*, 10(11):10356–10375, 2010.

[84] D. G. Lowe. Object recognition from local scale-invariant features. In *Proc. of the Int. Conf. on Computer Vision (ICCV)*, 1999.

[85] D. G. Lowe. Distinctive image features from scale invariant keypoints. *Int. Journal on Computer Vision (ICCV)*, 2004.

[86] F. Lu and E. Milios. Globally consistent range scan alignment for environment mapping. In *Journal of Autonomous Robots*, volume 4, pages 333–349, 1997.

[87] M. Magnusson, H. Andreasson, A. Nüchter, and A. J. Lilienthal. Automatic appearance-based loop detection from 3D laser data using the normal distributions transform. *Journal of Field Robotics*, 26(11–12):892–914, November 2009.

[88] B. Matei, Y. Shan, H. Sawhney, Y. Tan, R. Kumar, D. Huber, and M. Hebert. Rapid object indexing using locality sensitive hashing and joint 3D-signature space estimation. *IEEE*

Transactions on Pattern Analysis and Machine Intelligence (T-PAMI), 28(1):1111
[89] 1126, 2006.

[90] E. Michaelsen, W. von Hansen, M. Kirchhof, J. Meidow, and U. Stilla. Estimating the essential matrix: GOODSAC versus RANSAC. In *Photogrammetric Computer Vision*, 2006.

[91] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, 1. T. Kadir, and L. V. Gool. A comparison of affine region detectors. *Int. J. Comput. Vision*, 65(1-2):43–72, 2005.

[92] M. Montemerlo and S. Thrun. Large-scale robotic 3-D mapping of urban structures. In 1. *Proc. of the Int. Symp. on Experimental Robotics (ISER)*, pages 141–150, 2004.

[93] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM 2.0: An Improved Particle Filtering Algorithm for Simultaneous Localization and Mapping that Provably Converges. In *Proc. of the Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pages 1151– 1156, 2003.

[94] M. Montemerlo et al. Junior: The stanford entry in the urban challenge. *Journal of Field Robotics (JFR)*, 25(9):569–597, 2008.

[95] F. Moosmann and M. Sauerland. Unsupervised discovery of object classes in 3D outdoor scenarios. In *Proc. of the Workshop on Challenges and Opportunities in Robot Perception (ICCV)*, pages 1038 –1044, 2011.

[96] M. Nelson. LZW data compression. *Dr. Dobb's Journal*, 14(10):36, 1989.

[97] P. Newman et al. The MOOS - Cross Platform Software for Robotics Research. http://www.robots.ox.ac.uk/ mobile/MOOS/wiki/pmwiki.php.

[98] E. Olson. *Robust and Efficient Robotic Mapping*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, June 2008.

[99] E. Olson, J. Leonard, and S. Teller. Fast Iterative Optimization of Pose Graphs with Poor Initial Estimates. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, pages 2262–2269, 2006.

[100] E. Olson, M. Walter, S. Teller, and J. Leonard. Single-cluster spectral graph partitioning for robotics applications. In *Proc. of Robotics: Science and Systems (RSS)*, 2005.

[101] C. Parra, R. Murrieta-Cid, M. Devy, and M. Briot. 3-D modelling and robot localization from visual and range data in natural scenes. In *1st Int. Conf. on Computer Vision Systems (ICVS)*, number 1542 in LNCS, 1999.

[102] M. Parsley and S. J. Julier. Slam with a heterogeneous prior map. In *Proc. of the Conf. on Systems Engineering for Autonomous Systems (SEAS)*, 2009.

[103] R. Paul and P. Newman. FAB-MAP 3D: Topological mapping with spatial and visual appearance. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2010.

[104] J. Philbin, M. Isard, J. Sivic, and A. Zisserman. Lost in quantization: Improving par- ticular object retrieval in large scale image databases. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2008.

[105] E. Rosten and T. Drummond. Machine learning for high-speed corner detection. In *In European Conf. on Computer Vision*, pages 430–443, 2006.

[106] M. Rufli, D. Ferguson, and R. Siegwart. Smooth path planning in constrained environ- ments. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2009.

[107] M. Ruhnke, B. Steder, G. Grisetti, and W. Burgard. Unsupervised learning of 3D object models from partial views. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2009.

[108] M. Ruhnke, B. Steder, G. Grisetti, and W. Burgard. Unsupervised learning of compact 3D models based on the detection of recurrent structures. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2010.

[109] M. Ruhnke, B. Steder, G. Grisetti, and W. Burgard. 3D environment modeling based on surface primitives. In *Towards Service Robots for Everyday Environments*, pages 281–300. Springer Berlin/Heidelberg, 2012.

[110] S. Ruiz-Correa, L. G. Shapiro, and M. Meila. A new signature-based method for efficient 3-D object recognition. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 769–776, 2001.

[111] R. B. Rusu, N. Blodow, and M. Beetz. Fast Point Feature Histograms (FPFH) for 3D Registration. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2009.

[112] R. B. Rusu and S. Cousins. 3D is here: Point Cloud Library (PCL). In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2011.

[113] R. B. Rusu, Z. C. Marton, N. Blodow, and M. Beetz. Learning Informative Point Classes for the Acquisition of Object Model Maps. In *Proc. of the 10th Int. Conf. on Control, Automation, Robotics and Vision (ICARCV)*, 2008.

[114] R. B. Rusu, Z. C. Marton, N. Blodow, and M. Beetz. Persistent Point Feature Histograms for 3D Point Clouds. In *Proc. of the 10th Int. Conf. on Intelligent Autonomous Systems (IAS-10)*, 2008.

[115] R. B. Rusu, G. Bradski, R. Thibaux, and J. Hsu. Fast 3D recognition and pose using the viewpoint feature histogram. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2010.

[116] A. Sanfeliu. URUS project: Communication systems. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2009. Workshop on Network Robots Systems.

[117] D. Scaramuzza, A. Harati, and R. Siegwart. Extrinsic self calibration of a camera and a 3D laser range finder from natural scenes. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 4164 –4169, 2007.

[118] D. Scharstein and R. Szeliski. Middlebury stereo vision page, 2002. http://www. mid- dlebury. edu/stereo.

[119] J. Shin, R. Triebel, and R. Siegwart. Unsupervised 3D object discovery and categoriza- tion for mobile robots. In *Proc. of the 15th Int. Symp. on Robotics Research (ISRR)*, 2011.

[120] R. Siegwart, K. Arras, S. Bouabdallah, D. Burnier, G. Froidevaux, X. Greppin, B. Jensen,
a. Lorotte, L. Mayor, M. Meisser, R. Philippsen, R. Piguet, G. Ramel, G. Terrien, and
1. N. Tomatis. RoboX at Expo.02: A large-scale installation of personal robots. *Journal of Robotics & Autonomous Systems*, 42(3-4), 2003.

[121] J. Sivic, B. C. Russell, A. A. Efros, A. Zisserman, and W. T. Freeman. Discovering ob- jects and their location in images. In *Proc. of the Int. Conf. on Computer Vision (ICCV)*, volume 1, pages 370 – 377 Vol. 1, 2005.

[122] J. Sivic and A. Zisserman. Video google: A text retrieval approach to object matching in videos. In *Proc. of the Int. Conf. on Computer Vision (ICCV)*, 2003.

[123] R. Smith, M. Self, and P. Cheeseman. Estimating uncertain spatial realtionships in robotics. In *Autonomous Robot Vehicles*, pages 167–193. Springer Verlag, 1990.

[124] L. Spinello, R. Triebel, D. Vasquez, K. O. Arras, and R. Siegwart. Exploiting repetitive object patterns for model compression and completion. In *Proc. of the European Conf. on Computer Vision (ECCV)*, 2010.

[125] C. Stachniss, U. Frese, and G. Grisetti. OpenSLAM.org – give your algorithm to the community. http://www.openslam.org, 2007.

[126] B. Steder, G. Grisetti, and W. Burgard. Robust place recognition for 3D range data based on point features. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2010.

[127] B. Steder, G. Grisetti, S. Grzonka, C. Stachniss, and W. Burgard. Estimating consis- tent elevation maps using down-looking cameras and inertial sensors. In *Proc. of the Workshop on Robotic Perception on the Int. Conf. on Computer Vision Theory and Applications*, 2008.

[128] B. Steder, G. Grisetti, S. Grzonka, C. Stachniss, A. Rottmann, and W. Burgard. Learning maps in 3D using attitude and noisy vision sensors. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2007.

[129] B. Steder, G. Grisetti, C. Stachniss, and W. Burgard. Visual SLAM for flying vehicles.
1. *IEEE Transactions on Robotics*, 24(5):1088–1093, 2008.

[130] B. Steder, G. Grisetti, M. Van Loock, and W. Burgard. Robust on-line model-based object detection from range images. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2009.

[131] B. Steder, A. Rottmann, G. Grisetti, C. Stachniss, and W. Burgard. Autonomous navi- gation for small flying vehicles. In *Workshop on Micro Aerial Vehicles at the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2007.

[132] B. Steder, M. Ruhnke, S. Grzonka, and W. Burgard. Place recognition in 3D scans using a combination of bag of words and point feature based relative pose estimation. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2011.

[133] B. Steder, R. B. Rusu, K. Konolige, and W. Burgard. NARF: 3D range image features for object recognition. In *Workshop on Defining and Solving Realistic Perception Prob- lems in Personal Robotics at the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2010.

[134] B. Steder, R. B. Rusu, K. Konolige, and W. Burgard. Point feature extraction on 3D range scans taking into account object boundaries. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2011.

[135] S. Stiene, K. Lingemann, A. Nüchter, and J. Hertzberg. Contour-based object detection in range images. In *Proc. of the Third Int. Symp. on 3D Data Processing, Visualization, and Transmission*, pages 168–175, 2006.

[136] J. Tang and P. H. Lewis. Non-negative matrix factorisation for object class discovery and image auto-annotation. In *Proc. of the Int. Conf. on Content-based image and video retrieval (CIVR)*, CIVR '08, pages 105–112. ACM, 2008.

[137] S. Thrun, M. Bennewitz, W. Burgard, A. Cremers, F. Dellaert, D. Fox, D. Hähnel,
i. C. Rosenberg, N. Roy, J. Schulte, and D. Schulz. MINERVA: A second generation mo- bile tour-guide robot. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 1999.

[138] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, 2005.

[139] S. Thrun and colleagues. Winning the darpa grand challenge. *Journal of Field Robotics (JFR)*, 2006.

[140] S. Thrun, Y. Liu, D. Koller, A. Ng, Z. Ghahramani, and H. Durrant-Whyte. Simultane- ous localization and mapping with sparse extended information filters. *Int. Journal of Robotics Research (IJRR)*, 23(7/8):693–716, 2004.

[141] G. D. Tipaldi, M. Braun, and K. O. Arras. Flirt: Interest regions for 2D range data with applications to robot navigation. In *Proc. of the Int. Symp. on Experimental Robotics (ISER)*, 2010.

[142] F. Tombari, S. Salti, and L. Di Stefano. Unique signatures of histograms for local surface description. In *Proc. of the European Conf. on Computer Vision (ECCV)*, 2010.

[143] A. Torralba, K. P. Murphy, and W. T. Freeman. Labelme: the open annotation tool, 2007. http://labelme.csail.mit.edu/.

[144] R. Triebel, P. Pfaff, and W. Burgard. Multi-Level Surface Maps for Outdoor Terrain Mapping and Loop Closing. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2006.

[145] R. Triebel, R. Schmidt, O. M. Mozos, and W. Burgard. Instance-based AMN classifi- cation for improved object recognition in 2D and 3D laser raneg data. In *Proc. of the Int. Joint Conf. on Artificial Intelligence (IJCAI)*, 2007.

[146] T. Tuytelaars, C. H. Lampert, M. B. Blaschko, and W. Buntine. Unsupervised object discovery: A comparison. *Int. Journal on Computer Vision (ICCV)*, 88(2):284–302, 2010.

[147] J. Uhlmann. *Dynamic Map Building and Localization: New Theoretical Foundations*. PhD thesis, University of Oxford, 1995.

[148] R. Unnikrishnan. *Statistical Approaches to Multi-Scale Point Cloud Processing*.

PhD thesis, Robotics Institute, Carnegie Mellon University, 2008. Chapter 4.

[149] C. Urmson. *Navigation Regimes for Off-Road Autonomy*. PhD thesis, Robotics Institute, Carnegie Mellon University, 2005.

[150] C. Urmson et al. Autonomous driving in urban environments: Boss and the urban chal- lenge. *Journal of Field Robotics (JFR)*, 25(8):425–466, 2008.

[151] G. K. Wallace. The JPEG still picture compression standard. *Commun. ACM*, 34(4):30– 44, 1991.

[152] M. Weber, M. Welling, and P. Perona. Towards automatic discovery of object categories. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, vol- ume 2, pages 101–108, 2000.

[153] O. Wulf, A. Nüchter, J. Hertzberg, and B. Wagner. Benchmarking urban six-degree-of- freedom simultaneous localization and mapping. *Journal of Field Robotics*, 25(3):148– 163, 2008.

[154] K. M. Wurm, R. Kümmerle, C. Stachniss, and W. Burgard. Improving robot navigation in structured outdoor environments by identifying vegetation from laser data. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2009.

[155] K. M. Wurm, A. Hornung, M. Bennewitz, C. Stachniss, and W. Burgard. OctoMap: A probabilistic, flexible, and compact 3D map representation for robotic systems. In *Proc. of the Workshop on Best Practice in 3D Perception and Modeling for Mobile Ma- nipulation (ICRA)*, 2010.

[156] Y. Zhong. Intrinsic shape signatures: A shape descriptor for 3D object recognition. In *Computer Vision*

Workshops (ICCV Workshops), 2009 IEEE 12th Int. Conf. on, pages 689 –696, 2009.

a. **Y. Zhuang, N. Jiang, H. Hu, and F. Yan. 3-D-laser-based scene measurement and place recognition for mobile robots in dynamic indoor environments. *IEEE Trans. on Instru- mentation and Measurement*, 62(2):438–450, 2013Referenced Webpages**

Fisher, R.B. (editor), "CVonline: On-line Compendium of Computer Vision," Available at www.dai.ed.ac.uk/CVonline/.

The Intel Image Processing Library: http://developer.intel.com/software/products/ perflib/ipl/.

Source code release site: www.cs.cmu.edu/~jbruce/cmvision.

Newton Labs website: www.newtonlabs.com.

For probotics: http://www.personalrobots.com.

Interesting Internet Links to

Mobile Robots General

homepages with mainly mobile

robots

http://ranier.hq.nasa.gov/telerobotics_page/coolrobots.html

http://www-robotics.cs.umass.edu/robotics.html

http://www.ai.mit.edu/projects/mobile-robots/

http://www.ri.cmu.edu/project_lists/index.html
http://www.roboticsclub.org/links.html
http://www.activrobots.com/
http://asl.epfl.ch (at EPFL)
http://robotics.epfl.ch (at EPFL)
http://www.cs.cmu.edu/~illah/EDUTOY (at CMU)
http://www.cs.cmu.edu/~mercator/ (at CMU)

Homepages with mainly wheeled mobile robots http://www.laas.fr/RIA/RIA.html
http://www.cs.cmu.edu/~illah/lab.html
http://www.cs.umd.edu/projects/amrl/amrl.html http://www.cc.gatech.edu/ai/robot-lab/
http://www.engin.umich.edu/research/mrl/index.html

Homepages with walking and climbing robots http://www.uwe.ac.uk/clawar/
http://www.fzi.de/divisions/ipt/
http://www.automation.hut.fi/
http://www.ai.mit.edu/projects/leglab/
http://asl.epfl.ch/
http://www.cs.cmu.edu/~personalrover/

Homepages for robots on the Web
http://telerobot.mech.uwa.edu.au/secn/links.html
http://queue.IEOR.Berkeley.EDU/~goldberg/art/telerobotics-links.html http://www.cs.uni-bonn.de/~rhino/
http://www.cs.cmu.edu/~minerva/
http://telerobot.mech.uwa.edu.au
http://www.ieor.berkeley.edu/~goldberg/GC/www.html http://pumapaint.rwu.edu/
http://mars.graham.com/wits/
http://www.cs.cmu.edu/~illah/SAGE/index.html

http://www.dislocation.net/
http://rr-vs.informatik.uni-ulm.de/rr/
http://www.eventscope.org/Eventscope/main.htm

Georg Biegelbauer and Markus Vincze. Efficient 3D Object Detection by Fitting Superquadrics to Range Image Data for Robot's Object Manipulation. In *IEEE International Conference on Robotics and Automation (ICRA), Rome, Italy*, 2007.

Nico Blodow, Dominik Jain, Zoltan-Csaba Marton, and Michael Beetz. Perception and probabilistic anchoring for dynamic world state logging. In *Pro- ceedings of 2010 IEEE-RAS International Conference on Humanoid Robots*, Nashville, TN, USA, December 6-8 2010.

Nico Blodow, Radu Bogdan Rusu, Zoltan Csaba Marton, and Michael Beetz. Par- tial View Modeling and Validation in 3D Laser Scans for Grasping. In *9th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, Paris, France, December 7-10 2009.

Gary M. Bone, Andrew Lambert, and Mark Edwards. Automated Modeling and Robotic Grasping of Unknown Three-Dimensional Objects. In *Proceed- ings of the 2008 IEEE International Conference on Robotics and Automation, Pasadena, USA*, 2008.

Chih-Chung Chang and Chih-Jen Lin. *LIBSVM: a library for support vector ma- chines*, 2001. Software available at http://www.csie.ntu.edu.tw/

~cjlin/libsvm.

Young Sang Choi, Travis Deyle, and Charles C. Kemp. A List of Household Objects for Robotic Retrieval Prioritized by People with ALS (Version 092008). *CoRR*, abs/0902.2186, 2009.

Adam Coates, Paul Baumstarck, Quoc V. Le, and Andrew Y. Ng. Scalable learning for object detection with GPU hardware. In *IROS*, pages 4287–4293, 2009.

Corey Goldfeder, Matei Ciocarlie, Hao Dang, and Peter K. Allen. The columbia grasp database. In *Robotics and Automation, 2009. ICRA '09. IEEE Interna- tional Conference on*, pages 1710–1716, Kobe, Japan, May 2009.

Robert Fergus, Pietro Perona, and Andrew Zisserman. Object class recognition by unsupervised scale-invariant learning. In *In CVPR*, pages 264–271, Wisconsin, USA, 2003.

Carlos Ferrari and John Canny. Planning Optimal Grasps. In *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*, May 1992.

Mario Fritz, Trevor Darrell, Michael Black, Gary Bradski, and Sergey Karayev. An Additive Latent Feature Model for Transparent Object Recognition. In Se- lected for Oral Presentation, editor, *NIPS*, 12/2009 2009.

Georg Arbeiter, Jan Fischer, and Alexander Verl. 3D Perception and Modeling for Manipulation on Care-O-bot 3. In: *IEEE / Robotics and Automation Soci- ety: 2010 IEEE International Conference on Robotics and Automation (ICRA)* Anchorage, Alaska, USA, May 3-8, 2010.

Ulrich Hillenbrand. Pose clustering from stereo data. In *Proceedings VISAPP International Workshop on Robotic Perception – RoboPerc*, 2008.

Stefan Hinterstoisser, Vincent Lepetit, Slobodan Ilic, Pascal Fua, and Nassir Navab. Dominant orientation templates for real-time detection of texture-less objects. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2010.

Ian Horswill. Integrating vision and natural language without central models. In *In Proceedings of the AAAI Fall Symposium on Embodied Language and Action*, 1995.

Kaijen Hsiao, Sachin Chitta, Matei Ciocarlie, and E. Gil Jones. Contact-reactive grasping of objects with partial shape information. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, October 2010.

Andrew Edie Johnson. Spin-Images: A Representation for 3-D Surface Matching. Doctoral dissertation. Technical report CMU-RI-TR-97-47, Robotics Institute, Carnegie Mellon University, August 1997.

Ulrich Klank, Dejan Pangercic, Radu Bogdan Rusu, and Michael Beetz. Real- time cad model matching for mobile manipulation and grasping. In

9th IEEE- RAS International Conference on Humanoid Robots, pages 290–296, Paris, France, December 7-10 2009.

Danica Kragic and Markus Vincze. Vision for robotics. *Found. Trends Robot*, 1 (1):1–78, 2009. ISSN 1935-8253.

Kevin Lai and Dieter Fox. 3D laser scan classification using web data and domain adaptation. In *Proceedings of Robotics: Science and Systems*, Seattle, USA, June 2009.

David G. Lowe. Distinctive image features from scale-invariant keypoints. *Inter- national Journal of Computer Vision*, 60(2):91–110, 2004. ISSN 0920-5691.

Zoltan Csaba Marton, Lucian Goron, Radu Bogdan Rusu, and Michael Beetz. Reconstruction and Verification of 3D Object Models for Grasping. In *Pro- ceedings of the 14th International Symposium on Robotics Research (ISRR09), Lucerne, Switzerland, August 31 - September 3*, 2009a.

Zoltan-Csaba Marton, Dejan Pangercic, Nico Blodow, Jonathan Kleinehellefort, and Michael Beetz. General 3D Modelling of Novel Objects from a Single View. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Taipei, Taiwan, October 18-22 2010a.

Zoltan-Csaba Marton, Dejan Pangercic, Radu Bogdan Rusu, Andreas Holzbach, and Michael Beetz. Hierarchical object geometric categorization and appear- ance classification for mobile manipulation. In *Proceedings of 2010 IEEE-RAS International Conference on Humanoid Robots*, Nashville, TN, USA, Decem- ber 6-8 2010b.

Zoltan Csaba Marton, Radu Bogdan Rusu, and Michael Beetz. On Fast Surface Reconstruction Methods for Large and Noisy Datasets. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Kobe, Japan, May 12-17*, 2009b.

Andrew Miller and Peter K. Allen. Graspit!: A Versatile Simulator for Robotic Grasping. *IEEE Robotics and Automation Magazine*, 11(4):110–122, 2004.

Hideki Nakayama, Tatsuya Harada, and Yasuo Kuniyoshi. AI Goggles: Real-time Description and Retrieval in the Real World with Online Learning. *Computer and Robot Vision, Canadian Conference*, 0:184–191, 2009.

David Nister and Henrik Stewenius. Scalable recognition with a vocabulary tree. In *CVPR '06: Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2161–2168, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2597-0.

Dejan Pangercic, Moritz Tenorth, Dominik Jain, and Michael Beetz. Combining perception and knowledge processing for everyday manipulation. In *IEEE/RSJ International Conference on Intelligent RObots and Systems.*, Taipei, Taiwan, October 18-22 2010.

Patrick Pfaff, Wolfram Burgard, and Dieter Fox. Robust Monte-Carlo Localiza- tion Using Adaptive Likelihood Models. In *EUROS*, pages 181–194, 2006.

Morgan Quigley, Siddharth Batra, Stephen Gould, Ellen Klingbeil, Quoc V. Le, Ashley Wellman, and Andrew Y. Ng. High-accuracy 3D sensing for mobile manipulation: Improving object detection and door opening. In *ICRA*, Kobe, Japan, 2009.

Mario Richtsfeld and Markus Vincze. Grasping of Unknown Objects from a Table Top. In *Workshop on Vision in Action: Efficient strategies for cognitive agents in complex environments*, 2008.

Stephen Robertson. Understanding inverse document frequency: On theoretical arguments for IDF. *Journal of Documentation*, 60 no. 5, pp 503–520, 2004.

Alvaro Collet Romea, Dmitry Berenson, Siddhartha Srinivasa, and David Fer- guson. Object recognition and full pose registration from a single image for robotic manipulation. In *IEEE International Conference on Robotics and Au- tomation (ICRA '09)*, May 2009.

Michael Ruhnke, Bastian Steder, Giorgio Grisetti, and Wolfram Burgard. Unsu- pervised Learning of 3D Object Models from Partial Views. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, Kobe, Japan, 2009.

Radu Bogdan Rusu, Nico Blodow, Zoltan Csaba Marton, and Michael Beetz. Close-range Scene Segmentation and Reconstruction of 3D Point Cloud Maps for Mobile Manipulation in Human Environments. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, St. Louis, MO, USA, October 11-15 2009a.

Radu Bogdan Rusu, Andreas Holzbach, Gary Bradski, and Michael Beetz. De- tecting and segmenting objects for mobile manipulation. In *Proceedings of IEEE Workshop on Search in 3D and Video (S3DV), held in conjunction with the 12th IEEE International Conference on Computer Vision (ICCV)*, Kyoto, Japan, September 27 2009b.

Radu Bogdan Rusu, Zoltan Csaba Marton, Nico Blodow, and Michael Beetz. Learning Informative Point Classes for the Acquisition of Object Model Maps. In *Proceedings of the 10th International Conference on Control, Automation, Robotics and Vision (ICARCV), Hanoi, Vietnam, December 17-20*, 2008a.

Radu Bogdan Rusu, Zoltan Csaba Marton, Nico Blodow, Mihai Dolha, and Michael Beetz. Towards 3D Point Cloud Based Object Maps for Household Environments. *Robotics and Autonomous Systems Journal (Special Issue on Semantic Knowledge in Robotics)*, Volume 56, Issue 11, 30 November, Pages 927–941, 2008b.

Ashutosh Saxena, Justin Driemeyer, and Andrew Y. Ng. Robotic Grasping of Novel Objects using Vision. *The International Journal of Robotics Research*, 27(2):157–173, 2008.

Ruwen Schnabel, Roland Wahl, and Reinhard Klein. Efficient RANSAC for Point-Cloud Shape Detection. *Computer Graphics Forum*, 26(2):214–226, June 2007.

J. Sivic, B.C. Russell, A.A. Efros, A. Zisserman, and W.T. Freeman. Discovering Object Categories in Image Collections. In *Proceedings of the International Conference on Computer Vision*, 2005.

Bastian Steder, Giorgio Grisetti, Mark Van Loock, and Wolfram Burgard. Robust On-line Model-based Object Detection from Range Images. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, St. Louis, MO, USA, October 2009.

Klaus H. Strobl, Elmar Mair, Tim Bodenmüller, Simon Kielhöfer, Wolfgang Sepp, Michael Suppa, Darius Burschka, and Gerd Hirzinger. The Self-Referenced DLR 3D-Modeler. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 21–28, St. Louis, MO, USA, October 2009. best paper finalist.

Sebastian Thrun and Ben Wegbreit. Shape From Symmetry. In *Proceedings of the International Conference on Computer Vision (ICCV)*, Bejing, China, 2005. IEEE.

Manuel Martinez Torres, Alvaro Collet Romea, and Siddhartha Srinivasa. MOPED: A Scalable and Low Latency Object Recognition and Pose Estimation System. In *Proceedings of ICRA 2010*, May 2010.

Rudolph Triebel, Jiwon Shin, and Roland Siegwart. Segmentation and unsuper- vised part-based discovery of repetitive objects. In *Proceedings of Robotics: Science and Systems*, Zaragoza, Spain, June 2010.

Markus Ulrich, Christian Wiedemann, and Carsten Steger. Cad-based recognition of 3d objects in monocular images. In *International Conference on Robotics and Automation*, pages 1191–1198, 2009.

K.M. Wurm, A. Hornung, M. Bennewitz, C. Stachniss, and W. Burgard. Oc- toMap: A probabilistic, flexible, and compact 3D map representation for robotic systems. In *Proc. of the ICRA 2010 Workshop on Best Practice in 3D Perception and Modeling for Mobile Manipulation*, Anchorage, AK, USA, May 2010. URL http://octomap.sf.net/. Software available at http://octomap.sf.net/.

Keenan Wyrobek, Eric Berger, H.F. Machiel Van der Loos, and Ken Salisbury. Towards a Personal Robotics Development Platform: Rationale and Design of an Intrinsically Safe Personal Robot. In *Proc. International Conference on Robotics and Automation (ICRA)*, 2008.

Zhixing Xue, Alexander Kasper, Marius J. Zoellner, and Rüdiger Dillmann. An automatic grasp planning system for service robots. In *Advanced Robotics, 2009. ICAR 2009. International Conference on*, pages 1–6, Munich, Germany, June 2009.

Yan Zhang, Andreas Koschan, and Mongi A. Abidi. Superquadric Representa- tion of Automotive Parts Applying Part Decomposition. *Journal of Electronic Imaging, Special Issue on Quality Control by Artificial Vision, Vol. 13, No. 3*, pages 411–417, 2004.