Technical Report 01-2009

Autonomous Exploration and Inspection

Dirk Holz

December 2009

Publisher: Dean Prof. Dr. Kurt Ulrich Witt

University of Applied Sciences Bonn-Rhein-Sieg,
Department of Computer Science

Sankt Augustin, Germany

Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

ISSN 1869-5272

University of Applied Sciences Bonn-Rhein-Sieg
Department of Computer Science

Master Thesis

Autonomous Exploration and Inspection

Dirk Holz

Bonn
June 23, 2009

A thesis submitted to the
Bonn-Rhein-Sieg University of Applied Sciences
in partial fulfillment for the degree of
Master of Science in Autonomous Systems

Work conducted at the
Fraunhofer Institute for Intelligent
Analysis and Information Systems

1. Examiner: Prof. Dr. Gerhard K. Kraetzschmar,
Bonn-Rhein-Sieg University of Applied Sciences

2. Examiner: Dr. Erich Rome,
Fraunhofer Institute Intelligent Analysis and Information Systems

3. Examiner: Dr. Stefan May,
Institut National de Recherche en Informatique et en Automatique

Abstract

Autonomous mobile robots need internal environment representations or models of their environment in order to act in a goal-directed manner, plan actions and navigate effectively. Especially in those situations where a robot can not be provided with a manually constructed model or in environments that change over time, the robot needs to possess the ability of autonomously constructing models and maintaining these models on its own. To construct a model of an environment multiple sensor readings have to be acquired and integrated into a single representation. Where the robot has to take these sensor readings is determined by an exploration strategy. The strategy allows the robot to sense all environmental structures and to construct a complete model of its workspace. Given a complete environment model, the task of inspection is to guide the robot to all modeled environmental structures in order to detect changes and to update the model if necessary. Informally stated, exploration and inspection provide the means for acquiring as much information as possible by the robot itself.

Both exploration and inspection are highly integrated problems. In addition to the according strategies, they require for several abilities of a robotic system and comprise various problems from the field of mobile robotics including Simultaneous Localization and Mapping (SLAM), motion planning and control as well as reliable collision avoidance. The goal of this thesis is to develop and implement a complete system and a set of algorithms for robotic exploration and inspection. That is, instead of focussing on specific strategies, robotic exploration and inspection are addressed as the integrated problems that they are. Given the set of algorithms a real mobile service robot has to be able to autonomously explore its workspace, construct a model of its workspace and use this model in subsequent tasks e.g. for navigating in the workspace or inspecting the workspace itself. The algorithms need to be reliable, robust against environment dynamics and internal failures and applicable online in real-time on a real mobile robot. The resulting system should allow a mobile service robot to navigate effectively and reliably in a domestic environment and avoid all kinds of collisions. In the context of mobile robotics, domestic environments combine the characteristics of being cluttered, dynamic and populated by humans and domestic animals.

SLAM is going to be addressed in terms of incremental range image registration which provides efficient means to construct internal environment representations online while moving through the environment. Two registration algorithms are presented that can be applied on two-dimensional and three-dimensional data together with several extensions and an incremental registration procedure. The algorithms are used to construct two different types of environment representations, memory-efficient sparse points and probabilistic reflection maps. For effective navigation in the robot's workspace, different path planning algorithms are going to be presented for the two types of environment representations. Furthermore, two motion controllers will be described that allow a mobile robot to follow planned paths and to approach a target position and orientation. Finally this thesis will present different exploration and inspection strategies that use the aforementioned algorithms to move the robot to previously unexplored or uninspected terrain and update the internal environment representations accordingly. These strategies are augmented with algorithms for detecting changes in the environment and for segmenting internal models into individual rooms. The resulting system performed very successfully in the 2008 and 2009 RoboCup@Home competitions.

Acknowledgments

First of all, I would like to thank Gerhard Kraetzschmar, Erich Rome and Stefan May for being great advisors. The interesting discussions and their support had a major influence on this thesis. Furthermore, I would like to thank Christopher Lörken, Hartmut Surmann, Kai Pervölz, Thomas Wisspeintner, Peter Molitor, Sebastian Blumenthal, Thorsten Linder and all the others for the great time at IAIS and Mike Reckhaus, Stefan Christen and Edmund Milke for the daily get-together at the BRSU RoboCup lab. I really enjoyed their support and the atmosphere in these two think tanks.

I would like to thank Azamat Shakhimardanov, Sasi Kiran Gade and David Droeschel for proof-reading parts of the thesis and for giving hints on individual formulations and the structure of the thesis.

My thanks to Walter Nowak, Ronny Hartanto, Jan Paulus, Geovanny Giorgiana, Thomas Breuer, Christian Müller, Frederik Hegger, Zha Jin, Paul Ploeger and Gerhard Kraetzschmar for the nice collaboration in the b-it-bots RoboCup@Home team and the continued support during the work on this thesis.

I would like to thank Andreas Nüchter, Kai Lingemann and Joachim Hertzberg for the nice collaboration as well as for providing the floor plan of the AVZ building.

My thanks to Pedro Felzenszwalb for providing his source code for efficiently computing distance transforms of 2D images. I would like to thank Martin Magnusson for providing his implementation of 3D NDT-based scanmatching and the fruitful discussions at the beach of Nice. My thanks to Shane O'Sullivan for providing his source code for efficiently computing Voronoi diagrams for two-dimensional point sets. I would like to thank David Applegate, Robert Bixby, Vašek Chvátal and William Cook for providing the source code of their great TSP solver. My thanks to Andrew Howard, Cyrill Stachniss and Giorgio Grisetti as well as Austin Eliazar and Ronald Parr for providing their particle filter implementations.

I would like to thank Zoran Zivkovic for providing the sketch of the apartment environment used in the introductory visualizations and the corresponding data set. My thanks to Cyrill Stachniss and Giorgio Grisetti for providing the data sets of the third floor of the CSAIL building at MIT, buildings 079 and 101 as well as the campus at University of Freiburg. I would like to thank Dirk Hähnel for recording data sets at the Intel Research Lab, Belgioioso Castle and the University of Washington. My thanks to Nick Roy and Andrew Howard for setting up and maintaining the Radish data set repository and providing several log files used in this thesis. I would like to thank Mike Bosse and John Leonard for providing the Infinite Corridor data set and Patrick Beeson for providing the ACES data set. My thanks to Javier Minguez for recording the data set at the University of Zaragoza. I would like thank Andreas Nüchter for setting up a data set repository for 6D-SLAM as well as Martin Magnusson, Oliver Wulf, Dorit Borrmann, Jan Elseberg and Kai Lingemann for providing data sets that have been used throughout the work presented in this thesis.

Für Anna und meine Familie.

Contents

List of Figures

Chapter 1

Introduction

Up to now, robots have been predominantly used in automated manufacturing where they normally execute preprogrammed sequences of actions with a focus on high repeatability and accuracy. In this domain perceiving, modeling or reasoning about the environment is not necessary especially when the robot's activity is monitored and controlled by human operators. Unpredicted changes, a human entering the robot's workspace for example, simply lead to a shutdown of that robot. The robot's environment is itself controllable so to say.

However, autonomous service robots that assist in housekeeping, serve as butlers, guide visitors through exhibitions in museums and trade fairs, or provide care to elderly and disabled people could substantially ease everyday life for many people and present an enormous economic potential (Haegele et al., 2001; Pollack et al., 2002; Siegwart et al., 2003). Moreover, regarding the aging society in most industrialized countries the application of service robots in (elderly) health care might not only be helpful but necessary in the future. These service robots face the challenging task of operating in real-world indoor and domestic environments. Domestic environments tend to be cluttered, dynamic and populated by humans and domestic animals. In order to adequately react to sudden dynamic changes and avoid collisions, these robots need to be able to constantly acquire and process, in real-time, information about their environment. Furthermore, in order to act in a goal-directed manner, plan actions and navigate effectively, a robot needs an internal representation or *map* of its environment. Nature and complexity of these representations highly depend on the robot's task and application space.

Especially service robot applications, where a robot has not only to act in a human's environment but also interact with that human, pinpoint another fundamental challenge regarding environment representations – namely accounting for human spatial concepts or combining spatial and symbolic information in a single environment representation. A common concept of spatial information makes human-robot interaction more comfortable as it allows to e.g. assign tasks to the robot by means of natural speech, like for instance "Robot, bring me a cup of coffee from the machine at the end of the corridor".

Rather simple environments, like for instance the workspace of an industrial robot arm, can be modeled by hand to provide the robot with all the information necessary to cope with an assigned task. For more complex environments this might get unfeasible, regarding the involved costs, or not possible at all e.g. if the robot has to operate in a hazardous environment. Here, the robot needs to autonomously construct an appropriate environment representation on its own. Of course, the same holds true when the robot has to operate in a preliminary unknown environment. An according problem definition is given by Tovar et al. (2006): *"Autonomous robots must possess the ability to explore their environments, build representations of those environments, and then use those representations to navigate effectively in those environments."* To construct an environment representation, the robot has to *sense* its surrounding environmental structures from different positions and orientations in the environment and to integrate or aggregate the acquired information into a common representation – the robot's environment model.

Dynamic environments necessitate not only that the environment model is autonomously constructed, but also that it is autonomously maintained and kept up to date during operation. Here, an important ability is to detect all kinds of novelties, i.e. inconsistencies between an internal representation and the real environment. These inconsistencies can be caused by changes in an already explored area or when encountering previously unvisited and not yet modeled areas. From these abilities the following two tasks can be derived:

1. **Exploration:** A mobile robot has be able to autonomously explore an assigned workspace and construct an internal environment representation applicable in all kinds of subsequent tasks, e.g. moving to a designated position inside its workspace or *inspecting* and surveying the workspace itself.

2. **Inspection:** A robot that has already fully explored its workspace, has to be able to plan its actions, e.g. movements, in a way that it is able to perceive and recognize all important changes in its workspace in order to update the internal environment representation or as a task in its own right like for instance building surveillance.

In principal, exploration and inspection are quite similar. The only difference is that, in the case of inspection, the environment is known and the robot already has a complete internal environment model. In the case of exploration the robot's environment is not known or only partially known. Hence, the knowledge forming the basis for the robot's decision making process is imperfect. That is, an exploring robot does not know how many locations must be approached in order to cover the complete workspace.

A visualization of a possible outcome of an exploration or inspection task is shown in Figure 1.1. Here the robot has planned, respectively, as set of poses (position and orientation in the workspace) and a path leading through an apartment. Acquiring sensory information about surrounding objects at each of the planned locations allows the robot to observe all environmental structures in its workspace. That is, it forms the basis for constructing a complete environment model (exploration) and, in the case of inspection, detecting all relevant changes in the environment like for instance furniture that has been moved or doors that have been opened or closed since their last observation.

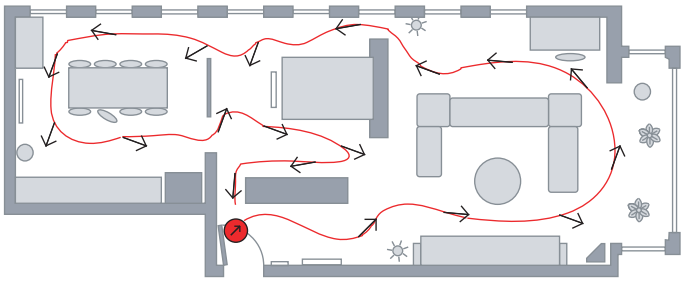


Figure 1.1: Typical exploration or inspection task. The set of vehicle poses (black arrows) and the path (red curve) allow the robot (red circle) to sense all environmental structures and objects in its workspace.

For a more concrete example, consider a domestic service robot that is given the task to serve a cold drink from the refrigerator to a guest in the living room (see Figure 1.2). Aside of the activities like interacting with the host and the guest, grasping objects like a can of soft drink, or other manipulation tasks, the robot needs to solve several problems related to navigation: If the environment is initially unknown, the robot must i) *explore the environment* and ii) *build an environment model* or *map*. Both during this exploration and map building phase and during everyday operation later on, the robot needs to iii) *localize itself* and iv) *localize task-relevant objects* (such as the refrigerator) within its environment representation. As self-localization requires a map of the environment, while mapping requires the ability to self-localize, these two problems need to be considered jointly as simultaneous localization and mapping (SLAM). SLAM has not only been a substantial research focus in the robotics community over the last decades but is also regarded as a major precondition of truly autonomous robots (Wang, 2004; Stachniss, 2006). For actually moving to certain locations in the environment, the robot needs to iv) *plan obstacle-free paths* and v) *follow planned paths*. Due to the fact that it operates in a dynamic environment, the robot must also constantly acquire information about the environment during navigation, and use this information to vi) *update the map* and vii) *avoid collisions*.

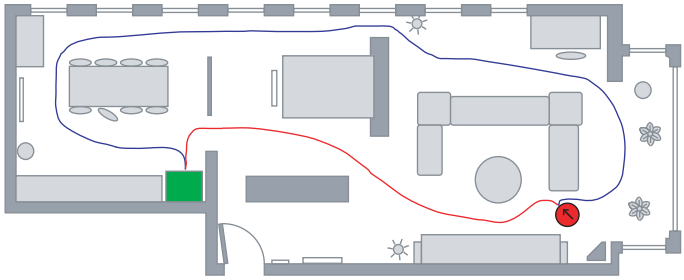


Figure 1.2: Typical navigation task. The robot (red circle) needs to navigate to the refrigerator (green box) to pick up and deliver a drink.

A scenario like the one described above is a typical task within RoboCup@Home. RoboCup @Home is a new league inside the RoboCup competitions that focuses on *real-world applications* and *human-machine interaction* with autonomous mobile robots. The aim is to foster the development of useful robotic applications that can assist humans in everyday life[1]. According to its rules and regulations by Nardi et al. (2007, 2008) it currently consists of a qualification round where the participating teams and especially their robot platforms have to master a set of tests involving basically all of the aforementioned issues particularly focussing, amongst human-robot-interaction, on mobile manipulation and acting in general as well as navigation in dynamic *and* cluttered, human-populated environments. The scenario in which these tests take place is a home-like or domestic environment that is, in the first years of the competition, specifically constructed like the arenas of other RoboCup leagues. However, it will be more and more enriched and finally completely replaced by the real world as it is the *ultimate scenario for real-world applications* (Nardi et al., 2007). With the set of tests each representing a typical task for a service robot assisting humans in their everyday life, the RoboCup @Home league provides a benchmark to evaluate the performance and applicability of ones own service robot and research results.

1.1 Autonomous Exploration and Inspection

An autonomous mobile robot performing SLAM has in principle the ability of wandering around in an unknown or partially unknown environment while modeling the thereby visited and sensed environmental structures. However, SLAM by itself is purely passive and the motion of the robot is not actively controlled. That is, a model constructed by this means does not necessarily cover the complete environment or all environmental structures since the robot might examine the same place again and again instead of analyzing previously unvisited areas. What is missing is a planning module that directs the robot to exactly these areas in order to sense all parts of its workspace. This additional requirement accomplishes what is, in the context of automated model construction, meant by the term *exploration* and what is summarized under point 3 of the following three steps that a robot has to repeat in order to construct a complete model of the environment:

1. **Acquire** new information by sensing the environmental structures,
2. **Integrate** the newly acquired information into an initially empty world model (by localizing the robot and transforming the gathered information accordingly, i.e. by performing SLAM),
3. **Decide**, based on the current model, where the robot has to move in order to acquire new information about environmental structures in the next sensing action (Step 1) or terminate if some criterion is met (e.g. the constructed map models the complete environment).

This loop of operations reflects a general model of perception and action – namely Neisser's *perceptual cycle*, that is shown in Figure 1.3. This abstract framework shows how the model (i.e. the knowledge about the environment) directs the exploration process acquiring new information about the environment that is again used to update the model.

[1] See the web pages of the RoboCup @Home league at `http://www.ai.rug.nl/robocupathome`.

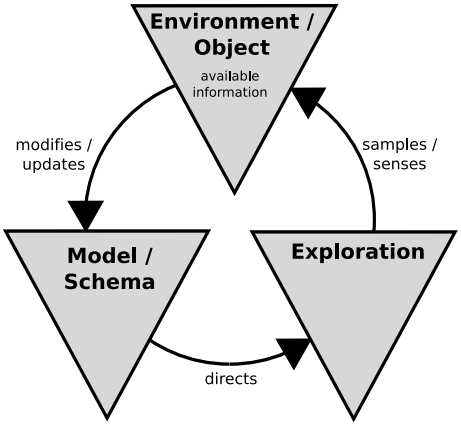


Figure 1.3: Neisser's *Perceptual Cycle*. The figure, adapted from (Neisser, 1976), depicts how the model (i.e. the knowledge about the environment acquired so far) influences or directs the exploration process acquiring new information about the environment that is again used to update the model (cf. Farris, 2003).

To construct an environment model in a completely autonomous manner, the problem of (motion) planning has to be addressed jointly with SLAM coining the commonly used abbreviation *SPLAM (Simultaneous Planning, Localization and Mapping)*. Depending on what is actually planned the following three problems can be distinguished (Makarenko et al., 2002):

- **Classic Exploration**: Only planning where to acquire sensor readings in order to obtain a complete model of the environment is referred to as *Classing Exploration* (Intersection II in Figure 1.4).
- **Active Localization**: Only planning locations where the robot is likely to localize itself, neglecting the exploration task, is referred to as *Active Localization* (Intersection III in Figure 1.4).
- **Integrated Exploration**: Fully integrating mapping and localization in planning the robot's actions is referred to as *Integrated Exploration* (Intersection IV in Figure 1.4). An example is to actively re-visit already explored terrain in order to reduce inconsistencies in the model by re-localizing the robot, i.e. *active loop-closing* (Stachniss et al., 2004).

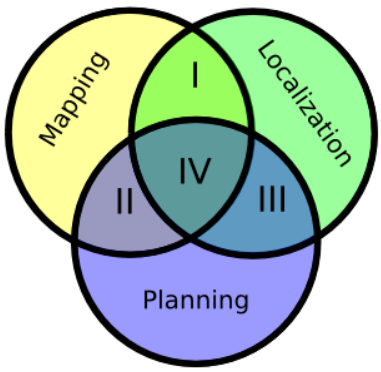


Figure 1.4: Basic and integrated tasks of an autonomous mobile robot as shown in (Makarenko et al., 2002) and (Stachniss, 2006). The diagram depicts how the the different integrated tasks, namely *Simultaneous Localization and Mapping (SLAM)* (I), *'Classic' Exploration* (II), *Active Localization* (III) and *Integrated Exploration* (IV) are formed by combining the basic tasks mapping, localization and planning.

Active localization refers to planning vehicle poses where the probability of successfully localizing the robot is high. Gaining new information about the environment, as in the case of exploration and inspection, is not considered. The other extreme, solely planning vehicle poses for visiting previously unmodeled terrain, e.g. by means of the exploration procedure described above, is referred to as *classic exploration* and neglects the probability of localizing the robot at the planned position. Fusing both, i.e. planning the robot's poses and motions so that it fully covers the environment with its observations while guaranteeing that the robot is always localized within its environment model, is referred to as *integrated exploration* (Makarenko et al., 2002). For a concrete example, consider a robot that is confronted with a set of candidate poses for the next sensing action. An integrated exploration strategy will select a pose where both the expected information gain and the probability of localizing the robot is high (Freda et al., 2006).

What is, however, not explicitly shown in Figure 1.4 is another essential capability of the robot to solve exploration and inspection tasks, namely *Motion Planning* and *Motion Control*. Coming back to the example in the beginning of this chapter, integrated exploration and inspection address problem i, ii, iii and iv. A complete system for performing such an integrated task, however, also needs to address problems v, vi and vii. An exploration or inspection strategy only determines a set of vehicle poses where the robot needs to acquire environmental information for a complete coverage. How to move to these locations, preferably using shortest paths and avoiding all kinds of collisions, is the problem of path planning and path following.

As can clearly be seen, robotic exploration and inspection incorporate a wide variety of problems in the field of mobile robotics. Furthermore, these problems are strongly interwoven as e.g. the choice of a particular environment representation affects the choice of algorithms for mapping and subsequent tasks like for instance motion planning. It might also constraint the choice of sensors and necessitate certain feature extraction mechanisms.

1.2 Contribution

All of the above problems have been well researched in robotics, at least in isolation. For each of these problems a large variety of sophisticated algorithms have been proposed. They coexist legitimately, since they are designed or especially appropriate for a specific purpose. However, despite the huge body of literature available, the problem of robust and computationally efficient navigation in domestic environments as well as their exploration and inspection cannot be considered solved yet. The first issue is *robustness*. In real-world environments, robotic systems need to act reliably and the involved components and algorithms need to be robust against unpredictable changes in the environment and internal failures. Especially in RoboCup@Home, there is only a short preparation time and only five to ten minutes to solve a complex task. Hence, algorithms need to be robust and the overall system has to act reliably as there might be no time left to re-start the work on an assigned task in case of failures or other unexpected events. Advancing robustness, however, often comes with increasing complexity that affects the *real-time applicability* of the algorithm and the overall system which is the second issue. *Scalability* is another issue since the computational complexity of many sophisticated approaches e.g. in SLAM either directly results in prohibitive memory and runtime requirements if applied to realistically-sized or large real-world environments, or at least cannot be used online in a reasonable fast cycle time. The fourth issue is *integration*. The aforementioned problems are strongly interwoven as, for example, the choice of the environment representation affects the choice of localization and path-planning algorithms. Identified best-in-class solutions may have different underlying assumptions hindering integration or necessitating possibly complex transformations from one representation into another. Efficiency problems may occur especially if such transformations cannot be done once and offline, but need to be done constantly or in regular intervals due to environmental dynamics. Furthermore, if published implementations are available at all, they are often not modular and easily re-usable as they depend on a specific architecture, development framework or inter-module communication.

Instead of proposing yet another toolkit for navigational purposes or exploration and inspection tasks, the goal of the work presented here is to design and implement a (complete) set of algorithms for autonomously performing SLAM, planning paths and controlling the motion of a mobile ser-

vice robot as well as exploration and inspection strategies making use of these algorithms, i.e. an approach addressing the aforementioned problems i) to vii) which is robust, efficient and scalable. The algorithms are implemented in a modular and reusable way. That is, implemented algorithms do not necessitate a certain control architecture, intermodule-communication or operating system. They form building bricks, so to say, that can be easily integrated in existent robot control architectures. Dependencies on external libraries are, furthermore, kept at a minimum. Primary design goals are robustness, simplicity and real-time applicability of the algorithms and the overall system.

SLAM is addressed in terms of range image registration, i.e. scan matching, providing a highly efficient yet robust and reliable component for localization and mapping. Constructed and used are two types of environment representations, memory efficient sparse point maps and probabilistic reflection maps. By means of an incremental registration procedure, the proposed algorithms are able to construct and update accurate and consistent maps online, i.e. during operation. The inherent difference to other SLAM approaches like for instance Rao-Blackwellized particle filters is being computationally very efficient (e.g. an average processing time per scan of approximately 5 ms to 10 ms compared to > 250 ms). For navigational purposes, different algorithms are presented for planning shortest obstacle-free paths on both types of environment representations together with procedures that are specifically designed for navigation in dynamic environments. Furthermore, the system consists of two non-linear motion controllers for, respectively, following planned paths and approaching target positions under a given orientation. The above algorithms are used to develop different exploration and inspection strategies to autonomously acquire and update all information being necessary to cope with typical service robot tasks. Exploration and inspection are addressed as the integrated problems that they are, so to say.

1.3 Outline

The remainder of this thesis is organized as follows:

Chapter 2 - Sensing, Perception and Actuation
Perceiving objects surrounding the robot and acquiring information about environmental structures is the focus of this chapter. It, furthermore, provides information about basic techniques, e.g. for estimating the robot's movement, sensors for perceiving the robot's surroundings and about the mobile service robot that will be used throughout the thesis. Previous work partially used in this thesis is also briefly presented.

Chapter 3 - Simultaneous Localization and Mapping
This chapter covers the concept of sparse point maps and an ICP-based matching algorithm to construct two-dimensional and three-dimensional sparse point maps on the basis of 2D and 3D range images. Furthermore, it describes NDT-based scan matching as an alternative to ICP-based matching and the construction of probabilistic reflection maps.

Chapter 4 - Path Planning and Motion Control
This chapter addresses the problems of path planning and path following on sparse point maps and probabilistic reflection maps as well as the involved motion controllers for following planned paths and approaching target poses, e.g. determined by an exploration or inspection strategy.

Chapter 5 - Exploration and Inspection
This section finally presents different strategies for exploring and inspecting a robot's workspace including algorithms for a human-guided exploration as well as fully autonomous exploration and inspection.

Chapter 6 - Conclusion and Future Work
This chapter contains some concluding words in addition to the results presented in Chapters 3, 4 and 5 and gives an overview on how the overall approach and resulting system can be improved and extended.

Each chapter contains an individual introduction to the problems and approaches addressed therein as well as a section providing experimental results and concluding remarks.

1.4 Publications

Parts of this thesis have been or will be published in the following book chapters, conference and workshop proceedings:

- D. Holz, D. Droeschel, S. May, H. Surmann. Fast 3D Perception for Collision Avoidance and SLAM. In *Mobile Robots Navigation*. IN-TECH Education and Publishing, Vienna, Austria. ISBN = 978-953-7619-X-X. 2009. Accepted. (Titles of book and chapter are subject to change)

- D. Holz, G. K. Kraetzschmar and E. Rome. Robust and Computationally Efficient Navigation in Domestic Environments. In *Proceedings of the 13th RoboCup International Symposium*, 2009. To appear.

- D. Holz. Effiziente 2D-Navigation für Mobile Service Roboter. In *Proceedings of the 11. Fachwissenschaftlicher Informatik-Kongress, Lecture Notes in Informatics (LNI)*, pages 31–34, 2009. Best Paper Award.

- D. Holz, C. Lörken and H. Surmann. Continuous 3D Sensing for Navigation and SLAM in Cluttered and Dynamic Environments. In *Proceedings of the International Conference on Information Fusion (FUSION)*, 2008.

Fractions of this thesis and intermediate results have also been published in the following RoboCup @Home team description papers:

- D. Holz, J. Paulus, T. Breuer, G. Giorgana, M. Reckhaus, F. Hegger, C. Müller, Z. Jin, R. Hartanto, P. G. Plöger and G. K. Kraetzschmar. The b-it-bots RoboCup@Home 2009 Team Description Paper. In *Proceedings CD of the 13th RoboCup International Symposium*. Graz, Austria, 2009. To appear.

- D. Holz, J. Paulus, T. Breuer, G. Giorgana, R. Hartanto, W. Nowak, J. Jackanapes, P. G. Plöger and G. K. Kraetzschmar. The b-it-bots Robocup@Home 2008 Team Description Paper. In *Proceedings CD of the 12th RoboCup International Symposium*. Suzhou, China, 2008.

Furthermore, algorithms and methods originating from the work presented here have been applied in other projects and published in the following journal articles, conference and workshop proceedings:

- S. May, D. Droeschel, D. Holz, S. Fuchs and A. Nüchter. Robust 3D-Mapping with Time-of-Flight Cameras. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2009. To appear.

- D. Droeschel, S. May, D. Holz, P. Ploeger and Sven Behnke. Robust Ego-Motion Estimation with ToF Cameras. In *Proceedings of the European Conference on Mobile Robots (ECMR)*, 2009. To appear.

- H. Surmann, D. Holz, S. Blumenthal, T. Linder, P. Molitor and V. Tretyakov. Teleoperated Visual Inspection and Surveillance with Unmanned Ground and Aerial Vehicles. *International Journal of Online Engineering (iJOE)*, 4(4):26–38, 2008.

- S. Blumenthal, D. Droeschel, D. Holz, T. Linder, P. Molitor and H. Surmann. Omnidirectional sensors for mobile robots. In *Workshop proceedings of the International Conference on Simulation, Modeling and Programming for Autonomous Robots (SIMPAR)*, pages 415–425, 2008.

- S. Blumenthal, D. Holz, T. Linder, P. Molitor, H. Surmann and V. Tretyakov. Teleoperated Visual Inspection and Surveillance with Unmanned Ground and Aerial Vehicles. In *Proceedings CD of the International Conference on Remote Engineering and Virtual Instrumentation (REV)*, 2008.

The work presented in this thesis is an ongoing research and development project. Intermediate results have been presented in the following internal technical reports:

- D. Holz. Towards Autonomous Exploration and Inspection. Technical Report R&D2, Bonn-Rhein-Sieg University of Applied Sciences, 2008.
- D. Holz. Competing in RoboCup@Home: Prerequisites for Acting in the real Real-World. Technical Report R&D1, Bonn-Rhein-Sieg University of Applied Sciences, 2007.

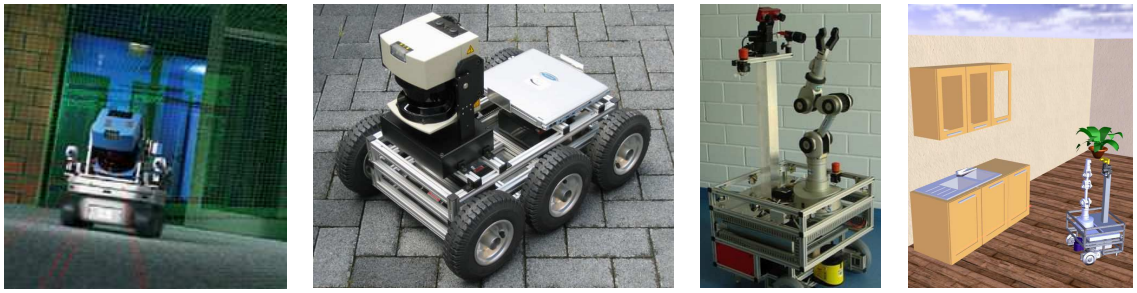
Chapter 2

Sensing, Perception and Actuation

Autonomous robots need sensors to acquire information about their surroundings and their internal state. This chapter will briefly present the sensors that have been used in the context of the work presented in this thesis. It will, furthermore, give an overview on the primary used robot platform for conducting real-world experiments and participating in the RoboCup@Home league. Algorithms and results from previous work used in this thesis are briefly described at the end of this chapter.

2.1 Mobile Robot Platforms

Throughout the work on this thesis, different robot platforms have been used, like for instance the *Kurt3D* platform (Surmann et al., 2003; Nüchter et al., 2005a) and different mobile robot platforms based on the *VolksBot*-concept (Wisspeintner and Bose, 2005). One of the latter is the mobile service robot "Johnny Jackanapes".



(a) (b) (c) (d)

Figure 2.1: Mobile robot platforms. Shown here are the KURT3D robot platform (a), the six-wheeled version of the VolksBot RT platform (b) and the service robot "Johnny Jackanapes" in a real and a simulated environment (c+d).

Kurt3D is a non-holonomic six-wheeled differential drive robot platform that has been developed at the Fraunhofer Institute for Autonomous Intelligent Systems. Originally designed for the autonomous inspection of sewerage pipes, it is now used as a platform for general research in robotics (Worst, 2003). It has a size of 45 cm (length) × 33 cm (width) × 26 cm (height). The robot's maximum velocity is 5.2 m/s. Two 90 W motors (200 W short-term) are used to power the 6 wheels, whereas the front and rear wheels have no tread pattern to enhance rotating. The robot has a weight of 15.6 kg. The particular 3D version is equipped with the IAIS 3D laser scanner presented later in this chapter. With this sensor on top the height increases to 47 cm and the weight to 22.6 kg. The outdoor version of KURT with four larger rubber wheels and the same laser scanner has been used several times in the RoboCup Rescue league during the past years and became vice world champion in Lisbon 2004. An image of KURT3D is shown in Figure 2.1.a. A Kurt3D platform with a 3DOF crane manipulator to lift magnetic objects has been used in the EU-FP6 project MACS.

VolksBot is a modular mobile robot construction kit allowing for application-based rapid prototyping (Wisspeintner and Bose, 2005). Figure 2.1.b shows a VolksBot RT6, a six-wheeled version

of the VolksBot Rough Terrain (RT) robot platforms. On top of the robot is the outdoor version of the same 3D laser scanner. It has a size of 70 cm (length) × 48 cm (width) × 26 cm (height) and a weight of 17.5 kg. The six tube tires are powered by two 150 W motors. The robot's maximum velocity is 1.1 m/s. With the 3D laser scanner on top its height increases to 47 cm and the weight to 24.5 kg. The VolksBot RT3 – a three-wheeled robot platform – has been used for the RoboCup @Home League in Atlanta 2007. Two tube tires are driven by the two 150 W motors, the third wheel is a caster wheel. Its maximum velocity is 1.4 m/s. The platform has a size of 58 cm (length) × 52 cm (width) × 31 cm 26 cm (height) and a weight of 13 kg. With the same 3D laser scanner on top its height increases to 47 cm and the weight to 20 kg.

The service robot platform "Johnny Jackanapes" (see Figure 2.1.c and 2.1.d) is a customized variant with an integrated manipulator mounted in a way to provide good reachability and maneuverability. The overall platform size is (51×51×120)cm (W×L×H) and its weight is 60 kg. The drive unit used for locomotion uses a differential drive with two actively driven wheels, powered by two 150 W motors, and two caster wheels to enhance rotating and stability under load. The robot's maximum velocity is 2 m/s. The manipulator is a Neuronics Katana 6M180 robot arm. It is equipped with six motors providing five degrees of freedom w.r.t. the gripper's position and orientation in its reachable workspace. It allows for precise movements with high repeatability. The sixth motor is used to open and close the two-fingered gripper, which is equipped with infrared reflectance as well as force sensors. The arm's weight is approx. 4 kg and it can handle a maximum payload of 500 g. It's operation radius is 60 cm. For recognizing and localizing objects, the robot possesses a stereovision camera mounted on a pan-tilt unit allowing for a nearly complete panoramic view of the scene. The primary sensor for perceiving environmental structures is a SICK LMS 200 2D laser range finder. Furthermore, an array of ultra-sonic sensors is mounted for approximately measuring distances to surrounding obstacles. These are, however, not used in the context of this thesis. Figure 2.2 shows the robot in detail and all of its sensors and actuators.

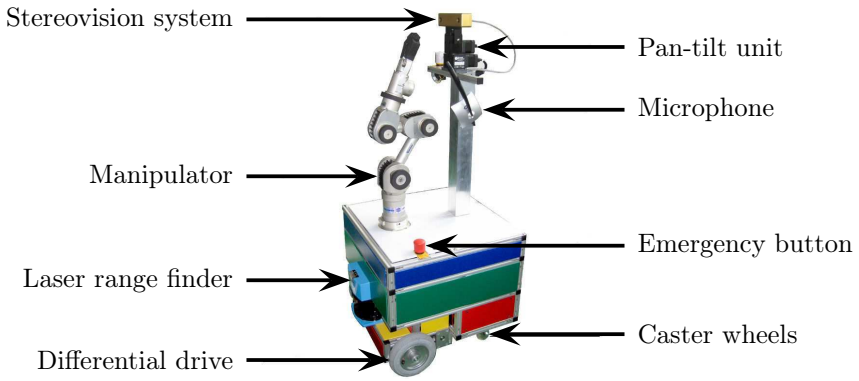


Figure 2.2: Mobile service robot platform "Johnny Jackanapes". Shown is a photo of the platform with drive unit, manipulator and sensors.

The remainder of this chapter will further describe the sensors used throughout this thesis as well as the means for controlling and estimating the robot's movement.

2.2 Self-Perception – Interoceptive Sensors

Interoceptive sensors are those sensors that measure internal states like for instance the battery level. Other important measurements are those of speed, accelerations as well as the resulting shifts in position and orientation. This is normally carried out using odometry. Odometry refers to counting or measuring wheel rotations. Based on the measured rotations and a kinematic model of the robot, overall platform velocities can be computed.

Mobile robot platforms can be distinguished, amongst other characteristics, by their type of locomotion. Most commercially available wheeled robots have differential drives, Ackermann drives or synchro-drives, i.e. vehicles with non-holonomic constraints. A *non-holonomic system* is a system in which a return to the original configuration $\mathbf{q}_0$ does not guarantee returning to the original system state $\mathbf{x}_0$, i.e. $(\mathbf{q}_0 = \mathbf{q}_t) \rightarrow (\mathbf{x}_0 = \mathbf{x}_t)$ might not hold. The output of a non-holonomic system is path-dependent and the number of coordinates required to represent a system's configuration $\mathbf{q}$ completely is more than its (*controllable* or *differential*) degrees of freedom (DDOF).

The aforementioned platforms are differential drive robots. That is, one or multiple wheels per side are driven differentially. When the wheels on both sides turn into the same direction with the same rotational velocities, the robot is moving forward. When turning the wheels in opposite directions, again with the same rotational velocities, the robot turns on the spot just like a tank. In this context, it should be mentioned that the service robot platform "Johnny Jackanapes" is not exactly a differential drive robot. The two actively driven wheels form a differential drive unit. The complete robot, however, does not exactly behave like a differential drive robot, as the caster wheels at the back side form a wiggling tail that is not modeled in the kinematic model of differential drive robots.

The wheel rotations are measured using optical or electro-mechanical encoders that are either mounted at the motors or directly at the driven wheels. These encoders emit a certain number of tics per wheel rotation. Counting these tics allows to calculate the number of wheel rotations in a particular time interval and, thus, the velocities of the wheels as well as the distances that would have been travelled by an individual wheel. Referring to the number of measured encoder tics for the left and right wheel(s) as, respectively, N_{left} and N_{right}, the incremental travel distance of a wheel Δd can be calculated as

$$\Delta d_{\text{left}} = \frac{\pi D N_{\text{left}}}{n \Delta e} \quad \text{and} \quad \Delta d_{\text{right}} = \frac{\pi D N_{\text{right}}}{n \Delta e} \tag{2.1}$$

where D is the wheel diameter, n the gear ratio and Δe the encoder resolution, i.e. the number of emitted tics per wheel rotation. Note that all three values can differ for left and right wheel and should be measured individually.

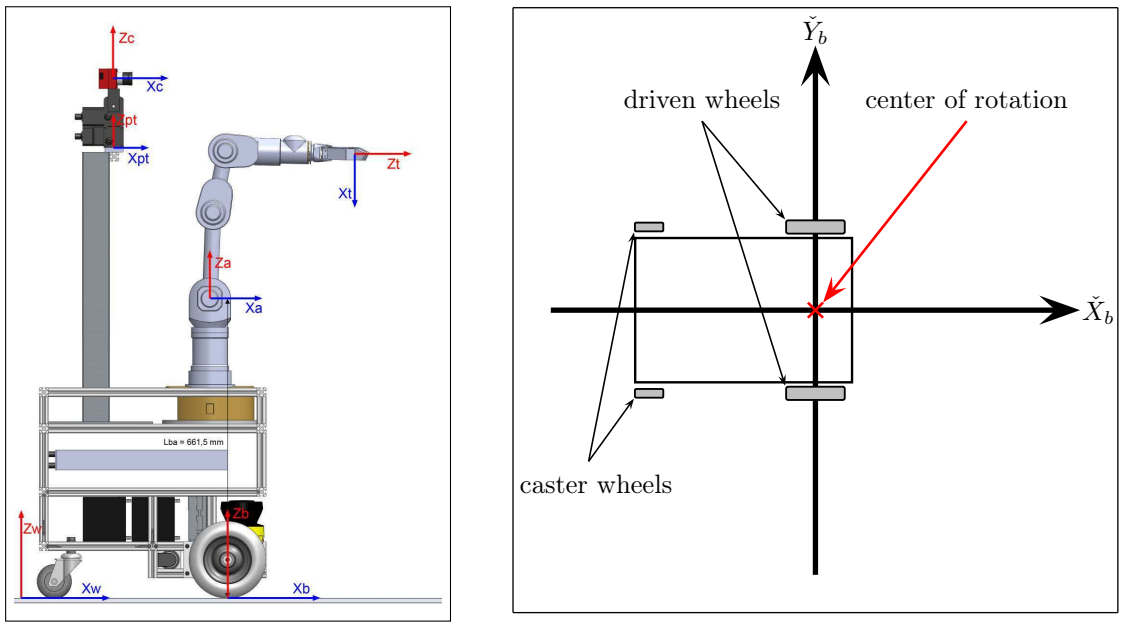
With the travelled distances of the individual wheels, the distance travelled by the robot can be calculated as well as the displacement of its center of rotation and the change in orientation.

$$\Delta d = \frac{\Delta d_{\text{right}} + \Delta d_{\text{left}}}{2} \tag{2.2}$$

$$\Delta \theta_z = \frac{\Delta d_{\text{right}} - \Delta d_{\text{left}}}{b} \tag{2.3}$$

where b is the distance between left and right wheel. Translational and rotational velocities of the robot naturally result from dividing position and orientation displacement by the time in which the encoder tics have been measured. Note that Eq. (2.3) is only an approximation assuming that, respectively, the change in orientation $\Delta\theta_z$ and the difference between Δd_{right} and Δd_{left} is small.

The pose (position and orientation) of a rigid mobile robot in a planar environment is described by $(x\ y\ \theta_z)^T$. A vector $(x\ y)^T$ without a specified orientation θ_z will be referred to as a *location* e.g. used to describe the positions of robot(s) and objects in the environment. Here it is important to not only know the pose of an object but also the coordinate system in which the pose is described. In the robot's coordinate frame $\{R\}$, the origin is formed by the robot's center of rotation. That is, all poses specified w.r.t. $\{R\}$ specify the position and orientation of objects relative to the base of the robot. Figure 2.3 shows the different coordinate frames attached to "Johnny Jackanapes". Every sensor has its own coordinate frame to which the corresponding sensor readings are referenced to. Furthermore, the manipulator forms a chain of coordinate frames from the gripper to the robot base where every joint is represented by an own coordinate frame. When processing sensory information it is, first, transformed into the coordinate frame of the robot base $\{B\}$ which also forms the coordinate frame which is meant when referring to the robot frame $\{R\}$. Throughout this thesis we will assume right-handed coordinate frames. For $\{R\}$ it means that the robot is driving



(a) Sideview (b) Topview

Figure 2.3: Coordinate systems. Shown are (a) the different coordinate frames of the sensors, the manipulator and (b) the robot base which is the robot frame $\{R\}$.

along the x-axis with the positive y axis extending to the robot's left and the z-axis corresponding to the height of objects (see Figure 2.3).

Based on the estimates of the change in orientation $\Delta\theta_z$ and position Δd as well as the above coordinate frame definition, the relative pose of the robot can be determined. That is the robot's position and orientation at time t is expressed in coordinate frame $\{R_{t-1}\}$ and with respect to the pose at time $t-1$ respectively. It the remainder of the thesis this relative pose shift will be referred to as an *odometric pose shift estimate* $\Delta\mathcal{P}$. Both $\Delta\theta_z$ as well as Δx and Δy can be directly derived from the individually travelled distances Δd_{left} and Δd_{right}

$$\Delta\mathcal{P} = \begin{pmatrix}\Delta x & \Delta y & \Delta\theta_z\end{pmatrix}^T \tag{2.4}$$

$$\Delta x = \Delta d \cos\Delta\theta_z = \frac{\Delta d_{\text{right}} + \Delta d_{\text{left}}}{2}\cos\Delta\theta_z \tag{2.5}$$

$$\Delta y = \Delta d \sin\Delta\theta_z = \frac{\Delta d_{\text{right}} + \Delta d_{\text{left}}}{2}\sin\Delta\theta_z \tag{2.6}$$

$$\Delta\theta_z = \Delta\theta_z = \frac{\Delta d_{\text{right}} - \Delta d_{\text{left}}}{b} \tag{2.7}$$

For incrementally updating a (global) pose estimate $\mathcal{P}_t$, i.e. a representation of the robot's pose in an external coordinate frame, the relative pose shift needs to be 1) transformed from $\{R_{t-1}\}$ into the external frame and 2) added to the last pose estimate $\mathcal{P}_{t-1}$:

$$\mathcal{P}_t = \begin{pmatrix}x_t & y_t & \theta_{z,t}\end{pmatrix}^T \tag{2.8}$$

$$x_t = x_{t-1} + \Delta x\cos\theta_{z,t-1} - \Delta y\sin\theta_{z,t-1} \tag{2.9}$$

$$y_t = y_{t-1} + \Delta x\sin\theta_{z,t-1} + \Delta y\cos\theta_{z,t-1} \tag{2.10}$$

$$\theta_{z,t} = \theta_{z,t-1} + \Delta\theta_z, \quad \theta_{z,t} \in [-\pi, \pi] \tag{2.11}$$

Such an incremental estimation of the robot's pose based on a previous pose estimate and an estimate of the pose shift or the robot's velocities is also referred to as *dead reckoning*. The inherent problem of this kind of pose estimation is that small errors and inaccuracies in the estimation of the

pose shift accumulate. Hence, odometric estimates can be highly inaccurate and erroneous. That is the difference between estimated and real position and orientation increases during the robot's movement. With an increasing number of estimations, the estimated trajectory seems to drift away from the actually driven one. Especially for differential drive robots, a measured rotation angle when turning on the spot or driving fast often considerably deviates from the actual angle. Primary reasons for these errors are wheel slippage and bumps on the ground. Using, for example, gyroscopes or inertial measurement units (IMUs) can improve pose shift estimates leading to less significant errors in the global pose estimate (Borenstein and Feng, 1996). Correcting global pose estimates based on information about environmental structures sensed during the robot's movement will be described, in detail, in Chapter 3. Controlling the robot's motion so that a certain vehicle pose is reached is going to be addressed in Chapter 4. The remainder of this chapter focuses on acquiring information about the environment and obstacles in the robot's vicinity.

2.3 Perceiving the Environment – Exteroceptive Sensors

For acquiring information about surrounding environmental structures, a large variety of sensors is used in mobile robotics. These range from infrared and ultrasonic sensors over visual sensors to commercially available 3D laser scanners used e.g. by surveying technicians. Passive triangulation systems, like for instance stereo cameras, acquire multiple images of the same scene. Based on disparities between these images and the known relative position and orientation offsets between the cameras, depth information is computed. Active triangulation systems consist of a light emitting and a light receiving component. A common technique is to project a particular pattern onto environmental structures using a laser. A camera captures an image of the scene and derives distance information based on the deformations of the projected pattern. Another possibility for contact-free distance measurements is the time-of-flight principle used by most commercially available laser scanners. These sensors emit a light signal and derive distance information from the received signal after being reflected by objects in the surrounding environment. Deriving distance information is based either on the time between emission and reception of the signal or on the phase shift between emitted ad received signal. Size, weight, energy consumption and prize of a sensor normally determine whether it can be applied on a particular mobile robot platform. This thesis focuses on time-of-flight sensors. For an overview on the large variety of sensors available it is referred to (Everett, 1995).

2.3.1 2D Laser Range Finders

2D laser range finders (LRFs) that measure, with high frequency and accuracy, the distances to environmental structures surrounding the robot became the de facto standard in mobile robotics. They measure distances by means of the time of flight of an emitted laser impulse. By using a rotating mirror, the laser impulse is deflected and multiple measurements are taken during the mirror's rotation to cover a two-dimensional plane. One 2D laser scan then consists of multiple subsequently acquired range measurements. The measured distances correspond to the distance to surrounding objects that intersect the sensed two-dimensional plane.

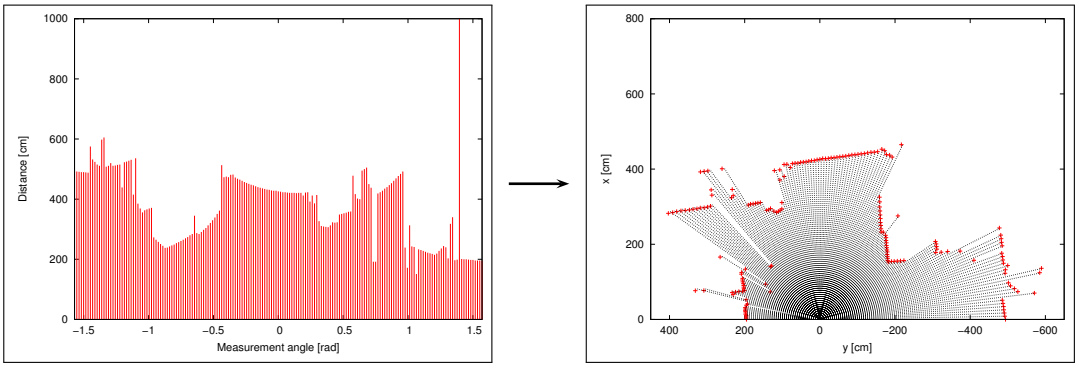
A laser scan S is a set of 2-tuples (d, θ) where d is a distance measurement and θ the angle under which the measurement has been taken, i.e.

$$S = \{(d_i, \theta_i) \mid i \in [1, N_s]\}.$$

Here, N_s denotes the cardinality of S, i.e. $N_s = |S|$, and the number of distance measurements respectively. Each tuple (d_i, θ_i) in S forms the polar coordinates of a point $\mathbf{p}_i$ measured on the surface of an object in the surrounding environment, i.e.

$$\forall i \in [1, N_s] \; : \qquad \mathbf{p}_i = \begin{pmatrix} d_i \cos\theta_i \\ d_i \sin\theta_i \\ 0 \end{pmatrix} = \begin{pmatrix} x_i. \\ y_i \\ z_i \end{pmatrix}$$

Depending on the measurement principle, S can be a totally ordered set with respect to, respectively, i and θ_i in either clockwise or anti-clockwise direction, i.e. for $i < j : \theta_i < \theta_j$ or $\theta_i > \theta_j$. A 2D laser range scan is exemplarily depicted in Figure 2.4.



(a) Measurements (polar) (b) Scan points (Cartesian)

Figure 2.4: Example of a 2D laser range scan. Shown are (a) the actual distance measurements in polar coordinates and (b) the resulting Cartesian coordinates of measured points.

The laser scanners used in this thesis are the SICK S300, the SICK LMS 200 and the outdoor version of the LMS 200 (the SICK LMS 291). The S300 has a size of (102×105×152)mm (W×L×H) and a weight of 1.2 kg. The size of the apex angle limiting the scan plane is 270° with an angular resolution of 0.5°. It delivers 2D laser scans in intervals of 80 ms (≈ 12.5 Hz). The maximum measurable distance is 30 m. The LMS 200 and the LMS 291 have a size of (156×155×120)mm (W×L×H) and a weight of 4.5 kg. Their apex angle has a size of 180°. The angular resolution can be adapted. Possible values are 0.25°, 0.5° and 1°. The latter allows for retrieving 2D laser scans in intervals of approx. 13.32 ms (≈ 75 Hz). In theory the maximum measurable distance of both is 80 m on objects with high reflectivity. All three scanners do not only measure the time until an emitted laser beams is received but also the intensity of the reflected signal. These remission values provide information about the reflectivity of objects. For more information about the laser range finders it is referred to the corresponding product specifications available at `http://www.sick.com`.

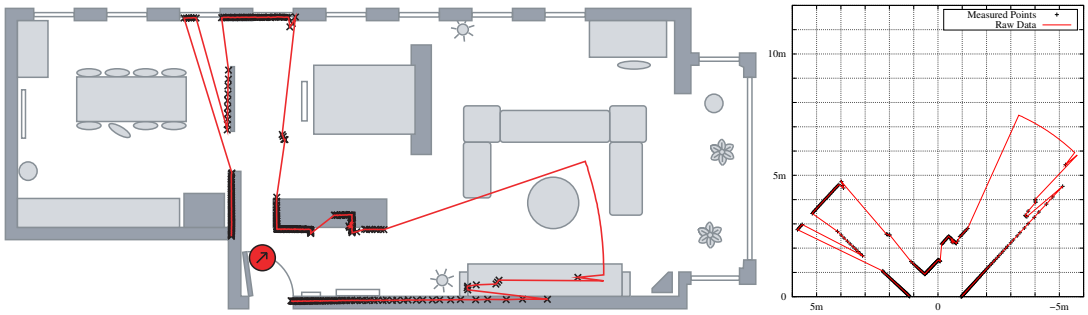
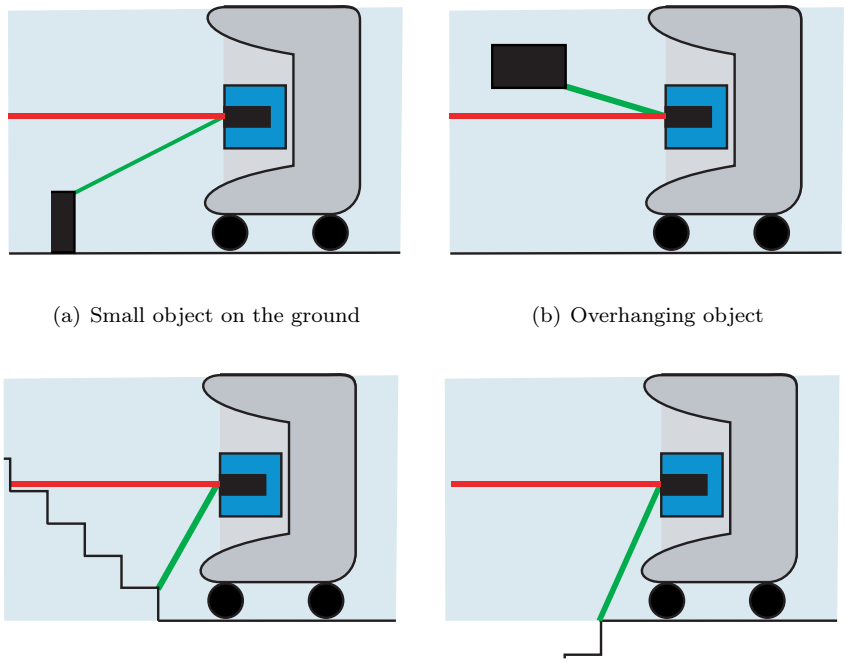


Figure 2.5: 2D laser range scan in an example scenario. Depicted is a range scan from a data set recorded by Zivkovic et al. (2007) together with an approximate floor plan of the scenario. Note, that the couchtable has not been sensed at all since it did not intersect the scanner's measurement plane.

The inherent drawback of 2D laser range finders, in the context of simultaneous localization and mapping (SLAM) as well as collision avoidance, is that objects not intersecting the scanner's measurement plane are not perceived. Consider for example the couch table in Figure 2.5. The 2D laser range scan taken in this example scenario does adequately model surrounding environmental structures whereas not a single measurement has been taken on the surface of the couch table. That is, the couch table is not at all perceived. Hence, it cannot be modeled in the robot's internal

environment representation. As there is no obstacle in the corresponding model region, the robot might plan a path that directly leads through the couch table. Furthermore, a collision with the couch table cannot be avoided as it does not intersect the scanner's measurement plane. Even when standing directly in front the of the table not a single measurement would be reflected. In this example, the scanner is mounted too high so that even the legs of the table do not intersect the measurement plane. However, even when intersecting the scanner's measurement plane, especially table and chair legs are not always adequately perceivable. Depending on material and shape of table legs, e.g. round metal rods, only a portion of emitted laser scans are reflected in a way so that they are received by the scanner. The same holds for true for objects whose surface is less reflective.

Tables and chairs are not the only objects in domestic environments that are hard to perceive with 2D laser range finders. Objects like for instance table tops, open drawers, small objects lying on the ground or stairs might not be appropriately perceivable by the robot and modeled in its internal environment representation (see Figure 2.6). When mounting the scanner in another height to perceive a specific class of obstacles, other types of obstacles are still not perceivable. Even the usage of several 2D range scanners in different heights does not appropriately solve this problem. For adequately handling all kinds of obstacles in a cluttered and dynamic environment 3D information becomes crucial.



(a) Small object on the ground

(b) Overhanging object

(c) Ascending Stair

(d) Descending Stair

Figure 2.6: Different types of obstacles. The figure shows four different examples of obstacles in a robot's workspace. They have in common that the robot is not able to reliably avoid them by means of simple 2D perception. The measurement plane of a standard 2D laser range finder is depicted in red only intersecting the ascending stair. By means of 3D perception a single distance measurement (green line) allows the robot to perceive the obstacles.

2.3.2 3D Laser Range Finders

One possibility to acquire three-dimensional data is to mount a 2D laser range finder on an actuator to gain an additional degree of freedom. Different setups have been proposed. For an overview it is referred to (Wulf and Wagner, 2003). Figure 2.7.a shows the IAIS 3D laser scanner (IAIS 3DLS-K). It consists of two SICK LMS 291 2D laser range finders mounted on a rotatable carrier. This carrier is continuously rotated around the vertical axis. Depending on the current orientation,

the 2D laser range scans of the scanners are transformed into a sensor-centric coordinate frame. The transformed scans are aggregated to form a local 3D point cloud (see Figure 2.7.b). Wulf and Wagner (2003) refer to this kind of sensor as a *yawing scanner* since the rotation is carried out around the z-axis.

(a) (b) (c)

Figure 2.7: Continuously rotating 3D laser scanner (a) and example data (b) taken in front of the Robotics Pavilion at Fraunhofer IAIS. A photo of the scene is shown in (c). The color of the 3D points corresponds to the received remission values.

The inherent drawback of this setup, in the context of collision avoidance, is that is takes quite long (up to 4 s depending on the wanted angular resolution) to acquire a complete 3D scan. Even when not waiting for a complete point cloud, but processing every single 2D scan as it arrives has the drawback of having larger areas in the robot's vicinity not in sight until the other scanner arrives at the corresponding rotation angle. This is, in fact, the reason why the two scanners are not mounted vertically as for the normal acquisition of 3D laser scans, but almost diagonally. This decreases the size of the unseen region when processing the acquired information scanwise.

A 3D scanner setup that is more adequate for the purpose of collision avoidance is the so-called *pitching scanner* (Wulf and Wagner, 2003). Here the 2D laser range finder is mounted horizontally and rotated around the y-axis. Figure 2.8 shows the IAIS 3DLS. This pitching laser scanner has been used in the early stage of the work presented here (Holz et al., 2008). For the RoboCup@Home world championship in Atlanta 2007, it was mounted on a three-wheeled VolksBot RT3 platform allowing to acquire 3D scans of the arena. With the additional rotation axis, driven by a standard servo motor, the scanner has a vertical aperture angle of up to $\Theta_{\text{pitch}} = 120°$ with a maximum angular resolution of $\Delta\theta_{\text{pitch}} = 0.25°$. Taking a 3D scan by rotating the scanner over the complete vertical range and using a horizontal angular resolution of $\Delta\theta_{\text{yaw}} = 0.25°$ results in 3D point clouds containing 346 080 points. However, as a relatively low angular resolution is sufficient for robust collision avoidance and has benefits in terms of speed concerns while still providing a sufficient detail for mapping purposes, an angular resolution of $\Delta\theta_{\text{yaw}} = 1°$ is preferable. As already mentioned, a single 2D laser scan of 181 distance measurements is read in approximately 13.32 ms ($\approx$ 75 Hz) in this operating mode. Reliable navigation in domestic environments requires for a fast and continuous 3D perception of surrounding environmental structures and obstacles. Therefore, the scanner is continuously pitched around its horizontal axis in a nodding-like fashion allowing the robot to perceive surrounding environmental structures in 3D while moving through its workspace. This methodology has been introduced in (Holz, 2006; Holz et al., 2008). Since a rotation over the complete aperture angle Θ_{pitch} might yield a couple of single 2D laser scans primarily containing useless information, e.g. only floor points if the scanner is directed downwards, an *area of interest (AOI)* has been defined in (Holz et al., 2008). This area restricts the range of used rotation angles $\theta_{\text{pitch}} \in [\theta_{\text{pitch, min}} : \theta_{\text{pitch, max}}]$ so that it contains primarily relevant information.

To be able to react to suddenly appearing obstacles in front of the robot even this restricted area might be too large to timely perceive especially small objects intersecting only a single measurement plane during one rotation. Therefore, the boundaries of the AOI ($\theta_{\text{pitch, min}}$ and $\theta_{\text{pitch, max}}$) as well as the scanner's pitch rate ($\Delta\theta_{\text{pitch}}$/13.32 ms corresponding to the number of taken consecutive 2D laser scans during one pitch movement, can be adjusted e.g. to depend on the robot's current

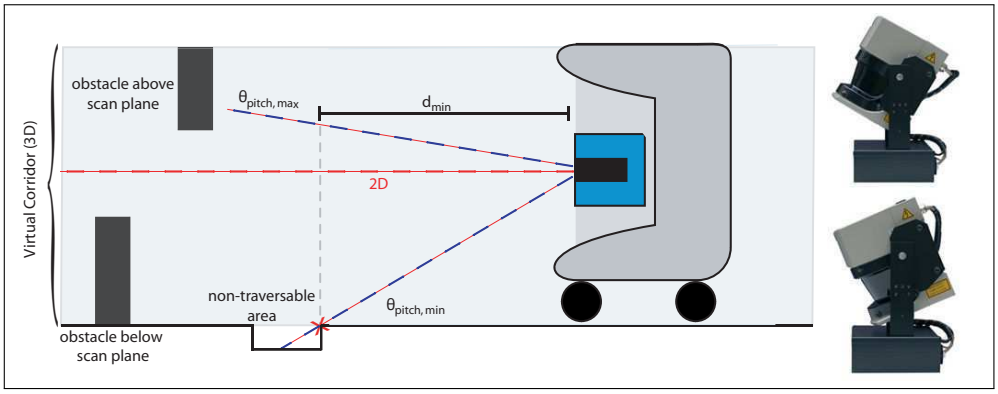


Figure 2.8: Continuously pitching scanner and virtual corridor. By continuously rotating the scanner in a nodding-like fashion over the area of interest (AOI), all obstacles in the virtual corridor are perceived.

velocity or special characteristics of the environment. Increasing the upper bound $\theta_{\text{pitch, max}}$ when moving slow allows to perceive more information about the environmental boundaries such as walls due to the height of the measured points. For moving faster it can be decreased so that only the volume corresponding to the robot's height is sensed. This minimal area is since (Holz et al., 2008) referred to as the *virtual corridor*. It extends the idea of *virtual roadways* (Lingemann et al., 2005a) to the third dimension, i.e. with respect to the robot's boundaries (in 3D) and thus possible areas of collision. The concept of the virtual corridor is depicted in Figure 2.8. Here lower bound $\theta_{\text{pitch, min}}$ and upper bound $\theta_{\text{pitch, max}}$ of the AOI correspond to the size of the virtual corridor and thus to the robot's boundaries (marked with the slightly colored background). d_{min} corresponds to the distance from which on the full virtual corridor can be perceived during the pitch movement. It has to be chosen appropriately, e.g. for a dense environment it has to be rather small whereas $d_{\text{min}} = 1\,\text{m}$ is absolutely sufficient when driving fast along an uncluttered corridor. The minimum size of the AOI for driving fast covers exactly the virtual corridor while the maximum size corresponds to a complete 3D scan over the full 120° of Θ_{pitch}. This allows to construct *complete* 3D models of the environment containing all perceivable information as in (Surmann et al., 2005).

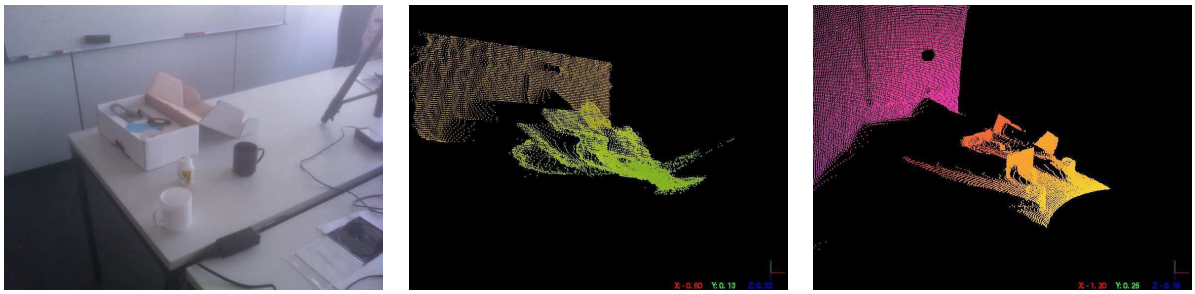
By continuously sensing and monitoring the virtual corridor the different types of obstacles (see Figure 2.6) can be perceived. Small obstacles lying on the ground and overhanging objects do not intersect the measurement plane when using simple 2D perception, i.e. when holding the scanner in a fixed horizontal position (red line). With the continuously pitching scanner they are perceived and can thus be avoided. The ascending stair is perceivable with 2D perception but depending on the scanner's height, the measured distance is larger than the distance to the first step. With 3D perception, the first step is perceived just like a small object lying on the ground. For descending stairs, however, a little trick needs to be applied that is presented in Chapter 2.3.4.

2.3.3 3D Time-of-Flight Cameras

The inherent drawback of laser range finders is that they measure only one distance at a time. Furthermore, to acquire 3D information multiple 2D scans need to be taken while rotating the scanner. 3D time-of-flight cameras are sensors that directly acquire 3D information. Three of these cameras have been used during the work presented here, namely the SwissRanger SR2-B, the SwissRanger SR3000 and SwissRanger SR4000 all from Mesa Imaging. With an array of infrared-LEDs they emit an almost sinusoidal amplitude-modulated light signal. A CCD/CMOS chip receives the light signal after being reflected by environmental structures. Each pixel thereby corresponds to an individual distance measurement. That is, the camera measures 176×144 distances within a single frame. Depending on the used integration time, frame rates of up to 54 frames per second can be reached. The measurement principal is phase-based. That is, the

measured distance is derived from the phase shift between the emitted and the received signal. This effects a limitation of the unambiguousness interval. Distances larger than the wavelength appear closer and need to be carefully filtered out. Consider for example that a phase shift of almost 2π corresponds to a distance of 7.5 m. An object being 8 m away from the sensor is, if at all, perceived at a distance of 0.5 m.

Compared to laser range finders, 3D time-of-flight cameras are rather small and lightweight. The SwissRanger SR4000, for example, has a size of (65×65×68)mm (W×L×H) and a weight of approx. 1.3 kg. This compactness and the fast measuring rates come with several drawbacks, namely erroneous measurements and noise. Besides the aforementioned distance ambiguity, erroneous measurements occur in the vicinity of transitions from one surface to another. The reflected light from the two surfaces interfere and cause measurements between them. These transitions are called *jump edges*. *Multiple ways reflections* further lead to rounded corners e.g. in the transition from floor to walls. Furthermore, measurements are less accurate compared to laser scanners and more noisy. Noise is caused by several systematic and non-systematic. Whereas systematic errors can be addressed by means of calibration, non-systematic errors need adequate filtering techniques. Several filtering techniques and calibration mechanisms have been presented by May et al. (2006), Fuchs and May (2007), Fuchs and Hirzinger (2008) and Pathak et al. (2008). 3D time-of-flight cameras form a rather new family of sensors and an evolving technology. Figure 2.9 shows a simple comparison on the quality of data acquired with an early SwissRanger SR2-B and a recent SwissRanger SR4000. Future cameras may significantly improve the quality of acquired information. For more detailled information about the used 3D time-of-flight cameras it is referred to the product specifications available at `http://www.mesa-imaging.ch`.



(a) (b) (c)

Figure 2.9: 3D data perceived by a time-of-flight camera. Shown is a scene (a) with a highly reflective table top and several small objects. The 3D data acquired with a SwissRanger SR2-B is noisy and inaccurate. The cups are not adequately perceived. The data acquired with a SwissRanger SR4000 is less noisy and the cups are accurately sensed.

2.3.4 Extracting Relevant Information from 3D Data

Real-time applicability does not only necessitate a fast acquisition of information but also to efficiently process the acquired information. Due to the larger amount of data and the higher dimensionality of information, directly processing raw 3D data is not feasible in many applications. Especially in the context of navigation, existing state-of-the-art approaches that show the capability of being applicable online normally perform on less complex and less information bearing 2D laser data (cf. e.g. Lingemann et al., 2005a). In order to combine these well-studied and well-performing algorithms with the rich continuously gathered 3D data it is suggestive to break down the three-dimensionality of the data into a slim two-dimensional representation that still holds all necessary 3D information but is nevertheless efficient enough to apply such efficient algorithms. For this purpose *virtual maps* have been presented in (Holz, 2006) and (Holz et al., 2008). These maps, in which the robot and the sensor respectively form the origin of the coordinate frame, only relevant information is stored that has been extracted from 3D data.

Two types of virtual 2D maps are distinguished – *2D obstacle maps* and *2D structure maps*. Both are generated from consecutive single 2D laser range scans acquired during the continuous pitching movement of the laser scanner presented above or extracted from one depth image of a SwissRanger camera. Note that the following descriptions will focus on continuously pitching lasers scanners. The actual implementation, however, is the same for all kinds of 3D sensors.

To be able to apply the same algorithms for collision avoidance, mapping and localization purposes to both standard 2D laser scanners and virtual 2D maps constructed by means of 3D perception, the representation of the virtual maps is chosen to extend the representation of standard laser scans. That is, they are organized as a vector of distance measurements d_i ordered by the discretized measurement angle ($\theta_{\text{yaw},i}$). This extended representation has, compared to a 2D laser scanner, a variable aperture angle $\Theta \in [0°, \ldots, 360°]$ and a variable angular resolution $\Delta\theta_{\text{yaw}}$. It is implemented as a vector of $N = \Theta/\Delta\theta_{\text{yaw}}$ points indexed by the accordingly discretized angle in which the measured point is lying from the robot's perspective. Furthermore, each point is represented by means of Cartesian and polar coordinates to avoid algorithm-dependent transformations.

2D Obstacle Maps

In the case of the obstacle maps the *minimum* distance in each scan direction ($\theta_{\text{yaw},i}$), projected in the xy-plane in which the robot is moving, is extracted and inserted into the obstacle map. These measurements correspond to the closest objects or obstacles in that particular direction regardless of the actual pitch angle θ_{pitch} of the scanner. Of course, only those points whose height above ground would intersect with the robot's bounds and the virtual corridor respectively are inserted into the map. This explicitly includes obstacles like small objects lying on the ground or overhanging objects like open drawers as shown in Figure 2.8. Objects not intersecting the virtual corridor pose no treat to the robot and can thus, in the case of the obstacles maps, be ignored. In the update procedure of *2D structure maps* they are, of course, used since a lot of information would be neglected otherwise.

In order to represent non-traversable areas and especially areas that correspond to holes in the ground, like for instance descending stairs in the examples of Figure 2.6, artificial obstacles are inserted into the map. Such non-traversable areas are characterized by those distance measurements that correspond to points in the real environment that are located below floor level. Note that the robot is assumed to be only able to traverse flat floor areas what is, after all, feasible for domestic indoor environments. Once such a measurement occurs in a perceived laser scan its intersection with the floor plane is computed and an artificial measurement at exactly this points it added to the obstacle map. Thereby the robot is able to perceive descending stairs and stop before the first step is reached. Such an intersection point and artificial distance measurement representing the stair as an obstacle is depicted with a red cross in Figure 2.8.

To take into account that other robot platforms are be able to climb stairs and move on ascending or descending ramps a more sophisticated approach has to be applied, that not only perceives distances measurements on floor level as traversable areas but that extracts those areas corresponding to stairs and ramps. This can be accomplished, for example, by applying the segmentation algorithm in (Nüchter et al., 2005b). It classifies points regarding their correspondence to floor, wall, object or ceiling structures. This, however, has to be seen as future work, and is not further considered in this thesis.

By the aforementioned means, the robot obtains an egocentric map containing all obstacles and non-traversable areas close to the robot. An obstacle map that exemplarily shows how small objects are perceived is depicted in Figure 2.10.

2D Structure Maps

In the case of the structure maps the *maximum* distance in each scan direction ($\theta_{\text{yaw},i}$), projected in the xy-plane, is extracted and inserted into the structure map. Extracting maximum distances automatically filters out all objects that do not extend over the full height of the AOI since the scanner will eventually look above or beneath these objects. The robot thereby replaces a previously measured smaller distance value with the newly obtained larger distance reading

(a) Obstacle map (b) Example scenario (c) Structure map

Figure 2.10: Demonstration of the virtual 2D map types in an example scenario. The *obstacle map* is generated by extracting minimum distances (projected into 2D) in the continuously acquired 3D data. Extracting maximum distances results in the *structure map* (Holz et al., 2008).

in that direction. The resulting map will thus only contain those points that most probably correspond to the environmental bounds while all points that belong to smaller or overhanging obstacles are filtered out as are those that belong to dynamic obstacles. Such a structure map is exemplarily depicted in Figure 2.10. Whereas the obstacle map shows the red and blue cans representing the obstacle type of small objects lying on the ground, the structure map only contains the environmental boundaries.

Update Procedures of Obstacle and Structure Maps

The steps to keep both representations egocentric and to update them according to newly acquired range scans are:

1.) **Transformation** of the map to keep it sensor-centric (e.g. according to odometry or a known pose).

2.) **Removal** of obsolete points to handle dynamics and inaccurate pose shift estimates.

3.) **Replacement** of already saved points using more relevant points from the current laser scan.

If the robot stands still and no pose shift has been estimated respectively, steps 1.) and 2.) are skipped. The same holds true if the the virtual maps are used as efficient representations of single 3D sensor readings, e.g. as obtained from 3D cameras. In its initial state, the map is filled with *dummy points* that are chosen in a way that they are replaced during the first update, i.e. points corresponding to the maximal measurable distance for obstacle maps and distances of 0 m for structure maps.

Transformation of the map to keep it egocentric: According to the robot's movement the pose shift between the current and the last map update (i.e. current and last reception of a laser scan) consists of a rotation $R_{\Delta\theta}$ around the z-axis by an angle $\Delta\theta$ and a translation $(\Delta x, \Delta y)^T$. The egocentric maps need to be transformed according to:

$$\begin{pmatrix} x_{i,t+1} \\ y_{i,t+1} \end{pmatrix} = \begin{pmatrix} \cos\Delta\theta & -\sin\Delta\theta \\ \sin\Delta\theta & \cos\Delta\theta \end{pmatrix} \begin{pmatrix} x_{i,t} \\ y_{i,t} \end{pmatrix} + \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} \tag{2.12}$$

where t and $(t+1)$ represent discrete points in time.

As Equation (2.12) transforms the map based on Cartesian coordinates, the values of the polar coordinates have to be adjusted accordingly. Due to the discretization of the N valid angles two points could fall into the same vector index. In this specific case the point being more relevant with

respect to the map type has priority. Vector indices being unassigned after the transformation are filled with dummy points.

Removal of obsolete Points to handle Dynamics: If the maps are not intended to only represent the gathered 3D information of one rotation over the AOI or a single range image as acquired by a 3D camera, but to be used endlessly, i.e. updated with every sensor reading, it is suggestive to remove points after a certain while. Therefore, the number of transformations applied during step 1.) is stored for every single point. To deal with dynamic obstacles a saved point is removed and replaced by a dummy point after its count of transformations exceeds a threshold (e.g. 500 transformations, approx. 5 s in the case of the continuously pitching IAIS 3DLS). This is because an obstacle passing by or crossing the robot's path leaves a trace of non-existent points in the obstacle map. This is not a drawback of the approach but a simple accounting for the uncertainty in the obstacle's movement. Points being removed by this means that correspond to static obstacles will immediately be measured again if the obstacle is still in the virtual corridor and thus still originates a possible source for a collision.

Furthermore, if pure odometry is used to estimate the robot's pose shift for the transformation in step 1.) instead of pose tracking to accurately determine the pose shift, errors and inconsistencies in both types of maps may arise from imprecise odometry. These are, in the same way, removed from the map. As a side note, it is to remark that even the rotated single 2D laser scans during the nodding-like movement of the sensor can be used for fast pose tracking algorithms like the one presented in (Lingemann et al., 2005b) if floor points are filtered out and the number of residual points is still sufficient for matching a newly acquired scan against the last one.

Replacement of already saved Points: The final update procedure highly depends on the map type as described above. In a nutshell, a point $\mathbf{p}_i$ stored in an obstacle map is replaced with a point $\mathbf{s}_i$ in the current laser scan S if the angle of acquisition $\mathbf{s}_i^\theta$ equals the discretized angle $\mathbf{p}_i^\theta$ and the measured distance $\mathbf{s}_i^d$ is less than or equal to $\mathbf{p}_i^d$; just as a point $\mathbf{p}_i$ stored in a structure map is overwritten with $\mathbf{s}_i$ if $\mathbf{s}_i^\theta = \mathbf{p}_i^\theta$ and $\mathbf{s}_i^d \geq \mathbf{p}_i^d$. The height $\mathbf{s}_i^z$ of an acquired point in a perceived environmental structure is used as an additional information in both types of maps resulting in a 2.5D representation. This will be extended in future work to not only store the particular height of the most recent points but to store minimum and maximum height of all points measured within a range of approximately 10 cm around that most recent point. By this simple extension a complete egocentric 3D model of the surrounding environmental structures can be reconstructed on the basis of the virtual 2D maps. In the case of obstacle maps the height information $\mathbf{p}_i^z$ of an acquired point $\mathbf{p}_i$ is also used to neglect those points that do not lie within the virtual corridor and are hence not relevant for representing nearby obstacles.

2.3.5 Extracting Simple Features

Extracting certain features from sensor data is an important part in the perception for mobile robots. In indoor environments, especially the extraction of lines (and planes) from raw point data is of interest, as these normally correspond to walls and other static environmental structures. A linear time algorithm for extracting line segments from raw 2D laser range scans, later referred to as *LENCOMP*, has been presented by Pauly et al. (1998) and Surmann et al. (2001b). In a first pre-processing step, points that lie close to each other are joint and replaced by their centroid. By this means, the number of points in the range scan and the maximum point density are reduced. In the second step, the algorithm runs over the residual points, comparing each measurement to its neighbors in order to determine whether or not they belong to the same line segment. A point $\mathbf{p}_{j+1}$ is added to a line segment starting at $\mathbf{p}_i$ and ending at $\mathbf{p}_j$ if the following three conditions are met:

$$\frac{||\mathbf{p}_i - \mathbf{p}_{j+1}||}{\sum_{k=i}^{j} ||\mathbf{p}_{k-1} - \mathbf{p}_k||} > 1 - \frac{0.3}{(1 + 1.5 \cdot (j+1))} \tag{2.13}$$

$$\frac{||\mathbf{p}_{j-1} - \mathbf{p}_{j+1}||}{||\mathbf{p}_{j-1} - \mathbf{p}_j|| + ||\mathbf{p}_j - \mathbf{p}_{j+1}||} > 0.8 \tag{2.14}$$

$$||\mathbf{p}_j - \mathbf{p}_{j+1}|| < 3 \cdot d_{\text{Step}} \tag{2.15}$$

The numerical constants have been empirically determined by Surmann and Liang (Surmann et al., 2001a). The conditions in the inequalities 2.13 and 2.14 are met, when $\mathbf{p}_{j+1}$ approximately lies on the same line as $\mathbf{p}_i$ and $\mathbf{p}_j$. Whether or not the line segment can be elongated when adding $\mathbf{p}_{j+1}$ is determined by inequality 2.15. The threshold distance d_{Step}, called *step distance*, determines how far a line segment can be elongated. A typical result of applying this simple algorithm is shown in Figure 2.11.

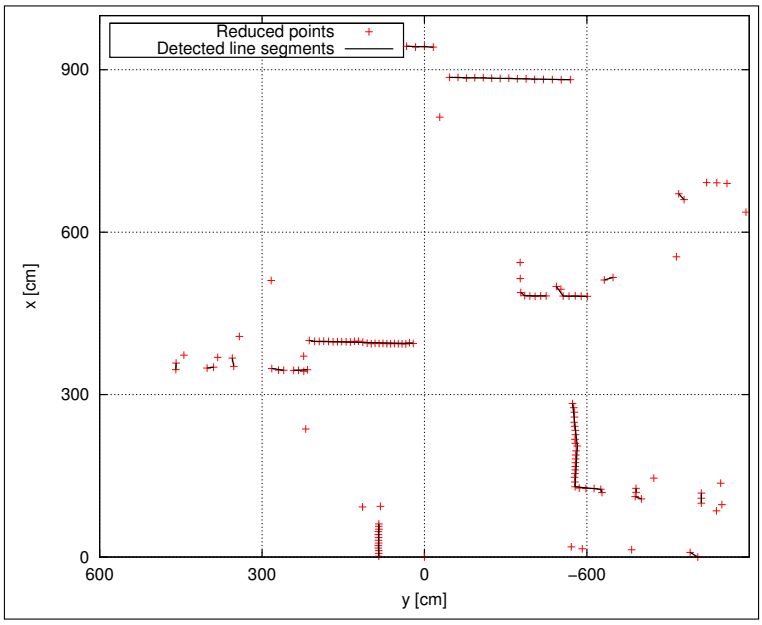


Figure 2.11: Simple line detection in 2D scans. With a threshold of $d_{\text{Step}} = 6\,\text{cm}$, 25 line segments have been extracted from the 133 reduced points.

Compared to other more sophisticated line detection algorithms, this simple procedure is highly efficient. It can be applied to both raw 2D laser range scans as well virtual obstacle and structure maps. Surmann et al. (2001b) use the algorithm to extract line segments in every 2D scan making up the 3D point cloud acquired by a pitching 3D laser scanner. The detected lines are then merged to polygons corresponding to planes in the sensed environmental structures.

Xavier et al. (2005) proposed different algorithms for detecting lines as well as circles and legs in a fashion being similar to the aforementioned. An algorithm for detecting straight lines in 2D range scans based on principal component analysis has been proposed by Lee et al. (2006b). A comparison of different line extraction algorithms for laser range scans can be found in (Nguyen et al., 2005).

2.4 Reactive Collision Avoidance

In addition to the goal-directed motion control of a mobile robot e.g. to reach a certain position as described in Chapter 4, reactive collision avoidance is important in dynamic and human-populated

environments. That is, the motion of the robot needs to be adapted in the presence of obstacles suddenly appearing in the robot's vicinity. In this thesis, three simple reactive behaviors are used that originate from (Holz, 2006) and (Lörken, 2007). The application of these behaviors to the aforementioned virtual obstacle and structure maps as well as raw 2D laser range scans has been presented in (Holz et al., 2008). One behavior slows down the robot if obstacles appear in the virtual corridor and in front of the robot in the virtual obstacle map respectively. If the distance to the nearest obstacle in the virtual corridor falls below some threshold, the robot is completely stopped. Another behavior turns the robot on the spot once it has been completely stopped. This avoids that the robot gets caught in dead ends or corners. Alternatively, the robot can be moved backwards so that it is positioned in free space again. Then an alternative path can be planned to reach the position that is to be approached.

The third behavior is more complex compared to the aforementioned ones. It slightly adapts the rotational velocity of the robot so that it prefers moving along free space. Consider for example, the robot has planned a path along a longer corridor. As this path results from searching for the shortest path between two positions, it can run directly along one of the walls. When exactly following such a path, this third behavior causes that the robot is not directly moving along the wall, but instead along the middle of the corridor. The concept of the behavior is to steer the robot towards a *freespace* orientation. The origin of determining the freespace orientation lies in the early work of Surmann and Peters (2001) for fuzzy-based control of autonomous mobile robots. A comparable behavior was obtained by applying a fuzzy controller with fuzzy rules like the following:

```
IF COMMAND is straight-ahead
AND IF FRONT-SENSOR is very-near AND FRONT-LEFT-SENSOR is very-near AND
FRONT-RIGHT-SENSOR is near
THEN SPEED is positive-small, ANGLE is negative-small
```

An adaption to 2D laser range scans has been presented in (Lingemann et al., 2005a). Here the following fuzzy rule is applied to every single distance measurement:

```
IF (angle_i is in driving direction) AND (distance_i is large)
THEN drive in this direction.
```

The actual driving direction of the robot, the wanted freespace orientation α_{free}, further adapted to meet our requirements, results as follows. Note that the fuzzy AND is implemented as a multiplication.

$$\alpha_{\text{free}} = \operatorname{atan2}\left(\sum_{i=1}^{N} \sin \mathbf{s}_i^{\theta} \cdot f_{\theta}\left(\mathbf{s}_i^{\theta}\right) \cdot f_d\left(\mathbf{s}_i^{d}\right),\ \sum_{i=1}^{N} \cos \mathbf{s}_i^{\theta} \cdot f_{\theta}\left(\mathbf{s}_i^{\theta}\right) \cdot f_d\left(\mathbf{s}_i^{d}\right)\right) \tag{2.16}$$

The functions $f_{\theta}\left(\mathbf{s}_i^{\theta}\right)$ and $f_d\left(\mathbf{s}_i^{d}\right)$ relate the i-th range reading in the form of the polar coordinates $\left(\mathbf{s}_i^{\theta}, \mathbf{s}_i^{d}\right)_{i=1\ldots N}$ as obtained from a 2D laser scanner or an obstacle map to the fuzzy sets *"angle is in driving direction"* and *"distance is large"*. N is the number of points in the map and the laser range scan respectively.

$$f_{\theta}(\theta) = \cos\left(\frac{\theta}{1.2}\right) \tag{2.17}$$

$$f_d(d) = \frac{1}{1 + \exp\left(-\left(\frac{d - dto_{\max}}{dto_{\min}}\right)\right)} \tag{2.18}$$

Here $dto_{\min}$ and $dto_{\max}$ determining the slope and the inflection point of the exponential, correspond to the thresholds of the behavior that slows down the robot. That is, if an object appears in front of the robot within a range $dto_{\max}$, the behavior starts slowing down the robot according to the distance to that object. If the distance to the object falls below $dto_{\min}$, the robot is completely stopped or moved backwards. Plots of the weighting functions $f_{\theta}(\theta)$ and $f_d(d)$ as well as the resulting application of the fuzzy AND are shown in Figure 2.12.

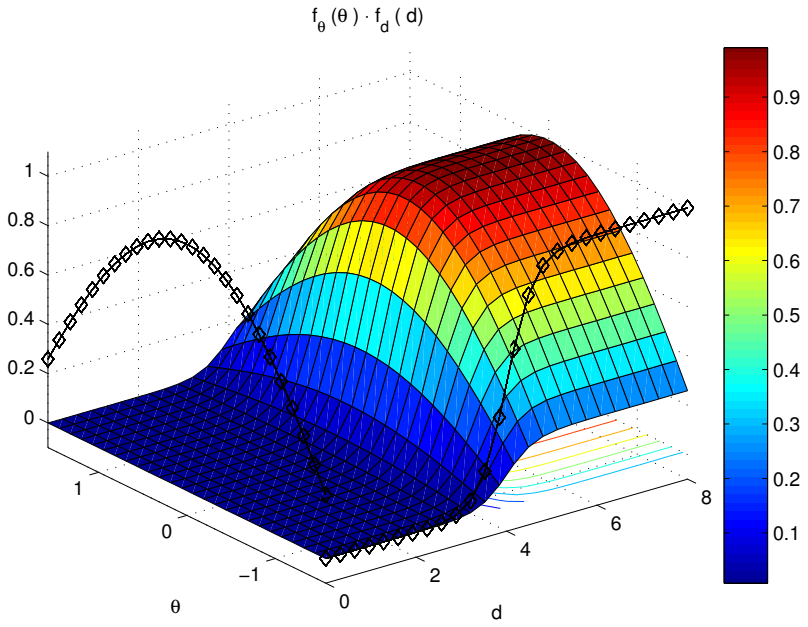


Figure 2.12: Composed weighting function $f_\theta\left(\mathbf{s}_i^\theta\right) \cdot f_d\left(\mathbf{s}_i^d\right)$ to determine the freespace orientation in a laser scan or obstacle map S.

The behavior simply adapts the rotational velocity, set e.g. by a motion controller for following a planned path, so that the robot slightly moves towards free space thereby swerving to avoid collisions. The influence of the behavior can be adapted and is kept rather small so that the robot can enter narrow passages and follow paths that lead away from the maximally free space in the robot's workspace. Referring to the resulting weighting function $f_\theta(\theta) \cdot f_d(d)$, the robot prefers moving straight and not adjusting its translational velocity.

(a) 2D Perception (b) Example scenario (c) 3D Perception

Figure 2.13: Behavior based collision avoidance an an example scenario by means of 2D perception (2D laser range finder in a fixed horizontal position) and continuous 3D perception together with the concept of obstacles maps. The small test objects are successfully avoided by the robot when using 3D perception (Holz et al., 2008). A video showing the robot perform in a similar experiment is available under `http://www.b-it-bots.de/media`.

A typical result of applying the three behaviors during navigation is shown in Figure 2.13. The robot was put into an example scenario bounded by movable walls with an exit in the opposite corner. The experiment was repeated two times. In the first run the scanner was held in a horizontal position (2D perception) comparable to a standard 2D laser range finder. In the second run, the scanner was continuously rotated over the aforementioned area of interest in a nodding-like fashion (3D perception). In both experiments, a constant translational velocity of $0.3\,\mathrm{m\,s^{-1}}$ was set with

no rotational velocity. That is, the robot was commanded to move forward. The application of the behaviors successfully moved the robot through the exit and outside the scenario. With 2D perception, small test objects lying on the ground were not perceived. The robot crushed into these objects and pushed them through the exit. With 3D perception, the objects have been perceived and the robot successfully avoided them. As can be seen in the figure, simply commanding the robot to move forward while enabling the three collision avoidance behaviors leads to an emergent behavior of wandering around. This strategy will be used to perform a random exploration in Chapter 5.3.1.

Chapter 3

Simultaneous Localization and Mapping

This thesis and the presented set of algorithms are organized in three parts. With simultaneous localization and mapping (SLAM) this chapter addresses the first part and the first problem in an integrated approach to robotic exploration and inspection. The next section gives an introduction to SLAM and an overview on related work. Constructing environment maps (mapping) based on laser range scans and given that we know the poses of the robot where it has acquired the range scans is described in Chapter 3.2. Using range scans and a partially built map to determine where the robot has taken the scans is then addressed in the remainder of this chapter. Chapters 3.3 and 3.4 present two scanmatching algorithms. Given laser range scans, these algorithms determine the change in position and orientation between the robot poses where the scans have been taken. Chapters 3.5 to 3.7 present different approaches to SLAM based on these algorithms. Experimental results verify that, using the proposed approaches, a mobile robot has the ability of constructing a map and localizing itself in this map online and while moving through its workspace.

3.1 Introduction to SLAM

Durrant-Whyte and Bailey (2006) describe SLAM "as a process by which a mobile robot can build a map of an environment and at the same time use this map to deduce its location". A large variety of approaches has been presented over the last decades and SLAM is often referred to as a solved problem. However, this does not necessarily hold true for practical implementations. Proposed algorithms are, for example, not applicable online due to their complexity, not scalable, e.g. do not work or get ridiculously time-consuming if the environment exceeds a certain size or the amount of information to process is too large, or simply do not work outside of "perfect" simulation environments.

Proposed approaches differ, amongst others, in formulating the problem, the means to cope with the addressed problem and in representing the environment.

3.1.1 Types of Environment Representations

An internal environment representation or *map* represents the robot's knowledge about its *world*, i.e. its workspace and surrounding environmental structures as well as objects contained therein, at a certain level of abstraction. If several representations are concurrently used in a single robot control architecture this level normally corresponds to the level of abstraction of the architectural layer where that very representation is being used. In general, two types of environment representions can be distinguished:

I. ***Egocentric*** **Environment Representations:** The robot itself forms the origin of egocentric representations. All information being stored in such a representation is given relative to robots position and orientation. If the robot moves, the position and orientation of features stored in the model have to be transformed to keep the representation egocentric.

II. ***Allocentric*** **Environment Representations:** The information contained in these types of representations is given with respect to some global reference frame. While the robot moves, its position and orientation inside this frame change while position and orientation of stored features remain static.

A comprehensive survey on egocentric and allocentric spatial representations, definitions and empirical studies from the perspective of human spatial cognition can be found in (Klatzky, 1998).

Another classification of environment representations can be made regarding their nature – namely *geometric environment representations*, *topological environment representations* and the combination of both in *hybrid environment representations*. What all types of representations have in common is the fundamental demand that the stored information corresponds to the real environment, i.e. they have to be correct and consistent.

Metric Environment Representations

The most self-evident way of representing environments is to describe their geometrical structure. Such a knowledge representation is the metric map. This type of environment representation can be further classified into *continuous* and *discrete* metric representations. Note that also rather exotic types of environment representations can be found in recent robotics literature; these two classes, however, represent the most common approaches.

- **Continuous Metric Maps or *Feature Maps*** maintain a list of *features* whose coordinates are stored in the map using a continuous range of values. These features can be any kind of extracted geometric objects like for instance lines corresponding to walls in the real environment or points representing a single distance measurement as obtained by a laser scanner. Especially the latter allows to process raw sensor data, i.e. points measured in the surrounding environment, to form a geometric model of the sensed environmental structures (Lu and Milios, 1997a). Other possible primitives are, for example, corners or visual features like for instance *SIFT* (Lowe, 1999, 2004), *SURF* (Bay et al., 2006, 2008) or *KLT* (Shi and Tomasi, 1994). However, if the environment scales up, the number of stored points can exceed dimensions, where the processing of the map gets unfeasible. Exactly this issue will be addressed in Chapter 3.5.

- **Discrete Metric Maps or *Grid Maps*** are based on fine-grained grids (cf. Thrun, 2002) and thus use, contrary to feature maps, a discrete range of values to represent space. Perceived information is encoded in attributes of the grid cells – in the case of occupancy grids (Moravec and Elfes, 1985) either binary, i.e. $free$ or *occupied*, or a probabilistic representation. In such probabilistic occupancy grids (Elfes, 1989; Moravec, 1989) an object sensed in the robot's surrounding environmental structures increases the probability of being occupied in the according grid cells. The cells are initialized with a prior probability of 0.5, i.e. unknown occupancy. A value larger than 0.5 confirms the robot's belief in the proposition that the cell is occupied whereas a probability smaller than 0.5 suggests free space.

  Grid Maps form a fundamental basis of probabilistic mapping algorithms (Thrun, 2002, 2003), especially when using rather inaccurate sensors like sonars (Pandey et al., 2007). The main drawback of grid maps is the trade-off between resolution and space efficiency – small grid cells allow for a detailled representation of the environment, whereas larger grid cells decrease the total number of necessary cells and reduce memory requirements. Besides the memory complexity, the main difference between occupancy grid maps and continuous geometric representations, is that the continuous representations do not distinguish between free space and unknown terrain. They only model the surface of objects but not the space occupied by them.

Topological and Hybrid Environment Representations

Topological maps are graph-based knowledge representations. *Distinctive places* form the nodes of the graph, its arcs or *edges* represent their connectivity. These representations are highly space efficient compared to e.g. occupancy grid maps and allow for the application of graph-based path-planning algorithms (Beeson et al., 2003). The edges of the graph do often provide additional information like for instance distance and orientation (Moratz and Wallgrün, 2003) or a sequence of actions (Kuipers et al., 2004) to encode how the robot can travel from one node to another.

Generally, most approaches to topological maps incorporate metric information and can thus be referred to as hybrid representations (cf. Thrun, 2002).

If not built online, topological maps are constructed by either dividing complete allocentric metric maps into distinct areas forming new nodes (Thrun, 1998) or by simply connecting allocentric metric maps (Thrun and Bücken, 1996; Simhon and Dudek, 1998).

3.1.2 Formulations of the SLAM Problem

Durrant-Whyte and Bailey (2006) provide a general formulation of the SLAM problem: Given a sequence $\mathbf{U}_{0:k}$ of control inputs $\mathbf{u}_k$ and a sequence $\mathbf{Z}_{0:k}$ of observations $\mathbf{z}_k$, the robot needs to determine a map $\mathbf{m}$ of the sensed environmental structures and the sequence $\mathbf{X}_{0:k}$ of state vectors $\mathbf{x}_k$ containing the robot's position and orientation at each timestep k. The control inputs $\mathbf{u}_k$ provide information about the robot's movement between timestep $k-1$ and k. That is, they provide an estimate of state $\mathbf{x}_k$ relative to $\mathbf{x}_{k-1}$. This information can be velocities set at time $k-1$ together with the time period $k-(k-1)$ or the relative pose shift measured by means of odometry. The idea behind most SLAM approaches is to estimate the sequence of state vectors $\hat{\mathbf{X}}_{0:k}$ using the sequence of control inputs $\mathbf{U}_{0:k}$. The sequence of observations $\mathbf{Z}_{0:k}$ is then used to correct the estimate of $\mathbf{X}_{0:k}$ and to construct the environment model $\mathbf{m}$. Some approaches even neglect the correction step, i.e. they use the estimate $\hat{\mathbf{x}}_k$ to store the information contained from $\mathbf{z}_k$ in the model. In particle-filter based approaches, for example, the correction of $\hat{\mathbf{x}}_k$ by means of $\mathbf{z}_k$ is replaced by weighting the particles carrying a particular sequence of poses (comparable to $\hat{\mathbf{X}}_{0:k}$). Alternatively, some algorithms take all the information available (not only until time k) into account to determine the complete sequence of state vectors.

Probabilistic Formulation of SLAM

A majority of SLAM algorithms is probabilistic. Here the SLAM problem becomes the problem of determining the probability distribution $P(\mathbf{x}_k, \mathbf{m}|\mathbf{Z}_{0:k}, \mathbf{U}_{0:k}, \mathbf{x}_0)$ for all times k, i.e. *smoothing*. Bayes Rule and conditional independence by applying the Markov assumption allow for a recursive formulation, i.e. determining $P(\mathbf{x}_k, \mathbf{m}|\mathbf{Z}_{0:k}, \mathbf{U}_{0:k}, \mathbf{x}_0)$ from $P(\mathbf{x}_{k-1}, \mathbf{m}|\mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k-1})$ by means of a *time-update* and a *measurement-update*. The time-update corresponds to estimating $\mathbf{x}_k$ from the last "known" state $\mathbf{x}_{k-1}$ and the control input $\mathbf{u}_k$ leading to that change. Here, a so-called *motion model* is used that determines the probability distribution $P(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{u}_k)$. This model can be deduced from experiments by estimating systematic and non-systematic errors in the robot's motion and odometry measurements. The measurement-update, corresponding to the correction of the estimate, uses an observation model determining the probability distribution $P(\mathbf{z}_k|\mathbf{x}_k, \mathbf{m})$, i.e. the probability of making an observation $\mathbf{z}_k$ given the robot's position and orientation in $\mathbf{x}_k$ and information about the environment in model m. In large part, probabilistic SLAM algorithms are based on formulations using *Extended Kalman Filters* (EKFs, Leonard and Feder, 1999), *Unscented Kalman Filters* (UKFs, Chekhlov et al., 2006), Sparse Extended Information Filters (SEIFs, Thrun et al., 2004) or *Rao-Blackwellized Particle Filters* (RBPFs, Montemerlo et al., 2003; Grisetti et al., 2005). For a more comprehensive overview on probabilistic SLAM algorithms it is referred to (Thrun, 2002) and the two-part tutorial on SLAM in (Durrant-Whyte and Bailey, 2006; Bailey and Durrant-Whyte, 2006).

Graph-based SLAM

Another family of SLAM algorithms is referred to as graph-based. Here, the problem of SLAM is represented as a graph in which nodes are formed by perceived features in the environment or poses of a mobile robot and edges correspond to rigid-body constraints between the nodes. These algorithms are primarily used for generating a globally consistent environment model based on an initial estimate of the the robot's trajectory. That is, using e.g. odometry and pairwise scan-matching (see Chapter 3.1.3) an initial estimate of the robot's trajectory is computed that then forms the graph. Environmental features do not necessarily need to be considered, as they

become conditionally independent given the robot's trajectory as demonstrated e.g. in (Montemerlo et al., 2002a). The trajectory results from determining the graph's adjacency matrix that minimizes a global cost function while meeting the constraints connecting nodes. Loop closures detected in the initial alignment (when forming the graph) form additional constraints connecting the corresponding poses.

Here the problem of SLAM becomes a nonlinear optimization problem with the goal of calculating the correct adjacency matrix of the graph. Approaches to this problem have been proposed e.g. in (Frese, 2004, 2007), (Olson et al., 2006, 2007) and (Grisetti et al., 2007b). These algorithms can for example be used for global relaxation and error minimization when loop closures have been detected.

SLAM in Terms of Range Image Registration

Another way of formulating SLAM is that by interpreting it as a problem of *multi-view range image registration*, i.e. integrating and aggregating range measurements taken at different positions and orientations in the environment into a common coordinate frame forming the environment model.

Measurements in range images are referenced to a local coordinate frame with the sensor forming the frame's origin. That is, information from multiple range images taken at different positions and orientations can, without knowledge about these poses, not directly be combined. The problem of aligning two or more range images to one another and to merge the aligned images into a common coordinate frame in order to construct a single, consistent model is referred to as *registration*. This problem can be formulated as follows: Given two sets of geometrical features, a data set D or *scene* and a data set M or *model*, find a transformation $\mathbf{T}$ that minimizes the alignment error between the two sets and correctly maps D onto M. The goal is to transform D in a way that it "fits best" with M. Central questions are how to represent the error function and how to minimize it, i.e. how to search in the possibly infinite and continuous space of transformations $\mathcal{T}$ as the optimal transformation $\mathbf{T} \in \mathcal{T}$ mapping D onto M should form a global minimum in the error function.

In principal, the transformation $\mathbf{T}$ can include translation, rotation, reflection, scaling or any other deformation. Normally, range images are taken of non-deformable, static objects with the same sensor but from diverse viewpoints, i.e. from different positions and under different orientations. Here $\mathbf{T}$ is a rigid transformation that includes only translation and rotation. Thus the registration problem becomes the problem of determining the view under which the range image corresponding to data set D was taken (relative to the view under which M was obtained). The data sets can be point sets, sets of line segments (so-called *polylines*), implicit or parametric curves and surfaces etc. The range images can fully overlap, partially overlap or not overlap at all, i.e. $D \subseteq M$, $D \cap M \neq \emptyset$ or $D \cap M = \emptyset$. The registration problem consists of two parts: *data-association*, i.e. determining pairs of corresponding entities in the data sets (e.g. a point $\mathbf{d}_i \in D$ and a point $\mathbf{m}_i \in M$ that correspond to one and the same point in the real world) and a method to calculate the optimal transformation mapping D onto M based on these correspondences. However, it should be noted, that there is a large family of algorithms that not directly address the registration problem in this way, but by other means, e.g. calculating or approximating probability distributions or *beliefs* over the space of possible transformations in probabilistic approaches (see e.g. Thrun, 2002). A widely used solution to the registration problem and the alignment of two (three-dimensional) point sets is the Iterative Closest Point algorithm by Besl and McKay (see Chapter 3.3.1). *Mutli-view range image registration* refers to the problem of finding a set of transformations $\{\mathbf{T}_i \mid i = 1 \dots k\}$ that correctly aligns k range images, i.e. determining all k viewpoints in the case of rigid transformations.

3.1.3 Procedures for Mutliview Range Image Registration

Basically, three methods for registering multiple range images into a common coordinate frame can be distinguished:

1. **(Sequential) Pairwise registration**
   A set or sequence $\mathcal{D} = \{D_i \mid i = 1 \dots k\}$ of k range images D_i is registered in a pairwise-manner where the origin of the local coordinate frame $\{1\}$ of the first range image D_1 forms

the origin of the common coordinate frame for the registered images. All succeeding range images D_j with $j > 1$ are aligned with their predecessor D_{j-1}. For each alignment the transformation ${}^{\{j-1\}}\mathbf{T}_j$ mapping D_j onto D_{j-1} in the local coordinate frame $\{j-1\}$ of D_{j-1} has to be determined. Due to the sequential nature of this method, the resulting chain of transformations represents the geometric relation between D_j and D_1 and can be used to construct a single model.

The main drawback of this method is that errors in the individual registrations of range image pairs accumulate and, thus, even very small alignment errors can lead to large errors in the geometric relation between D_j and D_1 when j gets large. These errors show up especially in situations where physical loops are closed, i.e. two range images D_j and D_k with $k \gg j$ are, again, taken from almost equal viewpoints after traversing a loop with the sensor, e.g. around an object. Here, the transformations ${}^{\{j\}}\mathbf{T}_k$ emerging from registering the range images between D_j and D_k pairwise can heavily deviate from the transformation ${}^{\{j\}}\mathbf{T}_k^{\star}$ resulting from directly registering D_j and D_k yielding an incorrect location of D_k in $\{1\}$ as represented by the chain of transformations. Hence, this method is not robust enough to match multiple views into a single model (Pulli, 1999). Due to this fact, pairwise registration is usually only applied to determine initial estimates of the transformations followed by a global relaxation algorithm.

2. *Metaview* or Incremental Registration
The idea of incremental matching is, again, to sequentially register one range image after the other, but to merge the information into a single data set, thereby incrementally building a model or *metaview* (Chen and Medioni, 1992). That is, the data from D_1 is completely or partially added to the model M. Succeeding range images D_j with $j > 1$ are mapped onto and merged into M. By this means the model M is extended with every registration of a range image D_j. Hence, all computations are carried out in the common coordinate frame and the transformation ${}^{\{1\}}\mathbf{T}_j$ resulting from the registration of D_j directly gives its geometric relation to the model M.

As in the case of pairwise matching, registration errors can accumulate but there is a higher chance for an correct alignment of D_j due to the fact that M carries information from all predecessors of D_j and only a subset of M might be affected by the accumulated error from previous registrations. By this means, algorithms applying incremental registration are able to close smaller loops, i.e. do not show a large deviation between ${}^{\{1\}}\mathbf{T}_j^{-1}{}^{\{1\}}\mathbf{T}_k = {}^{\{j\}}\mathbf{T}_k$ matching D_k onto M described in the coordinate frame of D_j and the transformation ${}^{\{j\}}\mathbf{T}_k^{\star}$ if the distance traversed between the acquisitions of D_j and D_k is not too large. Smaller loops occur for example when the sensor is moved along circular path in one and the same room. Larger loops, however, can not be closed and show larger deviations between ${}^{\{j\}}\mathbf{T}_k$ and ${}^{\{j\}}\mathbf{T}_k^{\star}$, respectively, making it again necessary to additionally apply loop detection and global relaxation methods.

As a side note, it is to remark that *particle filter*-based methods (see e.g. Grisetti et al., 2005) where every particle carries an own map and sensor trajectory are able to close larger loops, without the additional application of relaxation methods, when using a large number of particles.

3. Simultaneous Registration
Both pairwise and incremental matching have, as already mentioned, the inherent shortcoming of not being able to consistently register a larger loop of range images. This is caused by the fact, that registrations of a sequence of range image $D_1 \dots D_j$ are kept unchanged when registering a subsequent range image D_{j+1}. However, information from registering D_{j+1} might improve the registration of $D_1 \dots D_j$ what is neglected or not taken into account for in both procedures (Pulli, 1999). The idea of simultaneous registration is to consider multiple or even all range images and their registration at the same time and to register them to one another until their individual location changes converge to a static equilibrium.

3.2 Mapping with Known Poses

Before going into the details of different SLAM approaches, this section addresses the problem of constructing different environment representations under the assumption that the pose of the robot is exactly known. That is, given a sequence of observations and the trajectory of the robot an initially empty environmental model needs to be constructed. Expressed as a simple question, this problem can be described as "Given all available information, how does the environment look like?". Central questions in robotic mapping are how to extract relevant information from raw sensor readings and how to represent and integrate this information over time (Grisetti et al., 2007a). Throughout this thesis, two types of environment representations will be used, namely *point maps* (i.e. continuous metric maps) and grid maps (i.e. discrete metric maps). Both are briefly described in the following.

3.2.1 Point Maps and Other Geometric Feature Maps

Metric maps form the most intuitive way of representing an environment. Amongst the possible representations, point maps are the easiest to construct since points in two-dimensional or three-dimensional space are already provided as raw measurements from range sensors. A nice characteristic of point maps in the context of range image registration is that sensor measurements contain the same type of information as that contained in a point map. That is, range image registration as described in the following chapters, can be carried out without applying additional feature extraction mechanisms of conversions.

The construction of point maps, e.g. given a sequence of two-dimensional or three-dimensional points clouds and a sequence of vehicle poses where these point clouds have been acquired, is straightforward. The coordinates of the points in each point cloud are referenced to a local coordinate frame, the *sensor frame* $\{S\}$. For processing the data from the robot's perspective and to possibly merge point clouds acquired with different sensors, the first step is to transform the coordinates of points from the sensor frame into the robot frame $\{R\}$. The origin of the robot frame lies in the center of rotation, e.g. in the middle of the axis for a differential drive robot. The x-axis of the frame is pointing along the robot's movement direction, the y-axis expands to the left of the robot and the z-axis points upwards (right-handed coordinate frame, see Craig, 1989, Ch. 2) The transformation ${}^{\{R\}}_{\{S\}}\mathbf{T}$, mapping points from the sensor frame $\{S\}$ into the robot frame $\{R\}$, includes a rotation according to the sensor's orientation on the robot w.r.t. the coordinate axes of $\{R\}$, a translation according to the position of the sensor in $\{R\}$ and scaling with some constant factor α according to the measurement units used by the sensor and the robot control software. The transformation ${}^{\{R\}}_{\{S\}}\mathbf{T}$ is normally known as the individual components are known. Applying ${}^{\{R\}}_{\{S\}}\mathbf{T}$ to a point ${}^{\{S\}}\mathbf{p}$ in frame $\{S\}$ yields the same point measured in the physical environment but represented in the robot's coordinate frame $\{R\}$. For simplicity we, first, only consider the two-dimensional case, i.e. ${}^{\{S\}}\mathbf{p} \in \mathbb{R}^2$ and the transformation reduces to a rotation $\mathbf{R}_{\theta_z}$ about the z-axis by an angle θ_z and a translation $\mathbf{t} = (t_x, t_y)^T$ along the x- and y-axes:

$$
{}^{\{R\}}\mathbf{p} = \underbrace{\begin{bmatrix} \cos\theta_z & -\sin\theta_z \\ \sin\theta_z & \cos\theta_z \end{bmatrix}}_{\text{rotation}} \underbrace{\left(\underbrace{\begin{bmatrix} \alpha \\ \alpha \end{bmatrix}}_{\text{scaling}} \underbrace{\begin{bmatrix} {}^{\{S\}}p^x \\ {}^{\{S\}}p^y \end{bmatrix}}_{{}^{\{S\}}\mathbf{p}} \right)}_{} + \underbrace{\begin{bmatrix} t_x \\ t_y \end{bmatrix}}_{\text{translation}} \tag{3.1}
$$

Expressed in terms of a homogeneous transformation matrix, this transformation can be carried out using a single matrix-vector multiplication and a 4×4 transformation matrix stored for every

range sensor:

$$\begin{bmatrix} {}^{\{R\}}p^x \\ {}^{\{R\}}p^y \\ {}^{\{R\}}p^z \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\text{translation}} \underbrace{\begin{bmatrix} \cos\theta_z & -\sin\theta_z & 0 & 0 \\ \sin\theta_z & \cos\theta_z & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\text{rotation}} \underbrace{\begin{bmatrix} \alpha & 0 & 0 & 0 \\ 0 & \alpha & 0 & 0 \\ 0 & 0 & \alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\text{scaling}} \underbrace{\begin{bmatrix} {}^{\{S\}}p^x \\ {}^{\{S\}}p^y \\ {}^{\{S\}}p^z \\ 1 \end{bmatrix}}_{{}^{\{S\}}\mathbf{p}} \tag{3.2}$$

$$= \underbrace{\begin{bmatrix} \alpha\cos\theta_z & -\alpha\sin\theta_z & 0 & t_x \\ \alpha\sin\theta_z & \alpha\cos\theta_z & 0 & t_y \\ 0 & 0 & \alpha & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\text{transformation } {}^{\{R\}}_{\{S\}}T} \underbrace{\begin{bmatrix} {}^{\{S\}}p^x \\ {}^{\{S\}}p^y \\ {}^{\{S\}}p^z \\ 1 \end{bmatrix}}_{{}^{\{S\}}\mathbf{p}} \tag{3.3}$$

Here, the transformation is already expressed as a 4×4 homogeneous transformation matrix, i.e. ${}^{\{R\}}_{\{S\}}\mathbf{T}$ can be applied to two-dimensional and three-dimensional point clouds. The height of the scanner is expressed in terms of the translation t_z along the z-axis. In addition, in the three-dimensional case, ${}^{\{R\}}_{\{S\}}\mathbf{T}$ would include rotations around the x-, y- and z-axes (see Craig, 1989, Ch. 2).

The problem of (continuous) metric mapping is now to integrate the complete sequence of point clouds into a common representation, i.e. the coordinates of all points that need to be contained in the map, have to be referenced to a common coordinate frame. The necessary transformations ${}^{\{W\}}_{\{R_i\}}\mathbf{T}$ from the individual robot frames $\{R_i\}$ into the world frame $\{W\}$ – the coordinate frame for the point map – are given with the sequence of robot poses. It should be noted, that $\{W\}$ can be an arbitrary coordinate frame, as long as the robot poses are given in or can be transformed into this frame. The transformation ${}^{\{W\}}_{\{R_i\}}\mathbf{T}$ mapping from the i-th robot's frame into the world frame is composed exactly as ${}^{\{R\}}_{\{S\}}\mathbf{T}$ in Eq. (3.3), whereas here $(t_x\ t_y\ t_z)^T$ corresponds to the robot's position in the world frame $\{W\}$ and the rotation matrix is composed of the robot's orientation about the coordinate axes of $\{W\}$. Successively applying ${}^{\{R\}}_{\{S\}}\mathbf{T}$ and ${}^{\{W\}}_{\{R_i\}}\mathbf{T}$ to a point ${}^{\{S\}}\mathbf{p}$ contained in the i-th point cloud yields the transformed point ${}^{\{W\}}\mathbf{p}$ expressed in the world coordinate frame $\{W\}$.

$$\begin{bmatrix} {}^{\{W\}}\mathbf{p} \\ 1 \end{bmatrix} = {}^{\{W\}}_{\{R_i\}}\mathbf{T}\ {}^{\{R\}}_{\{S\}}\mathbf{T}\ \begin{bmatrix} {}^{\{S\}}\mathbf{p} \\ 1 \end{bmatrix} \tag{3.4}$$

Note that both point vectors are augmented with an additional row needed for the application of homogeneous transformation matrices (Craig, 1989, Ch. 2).

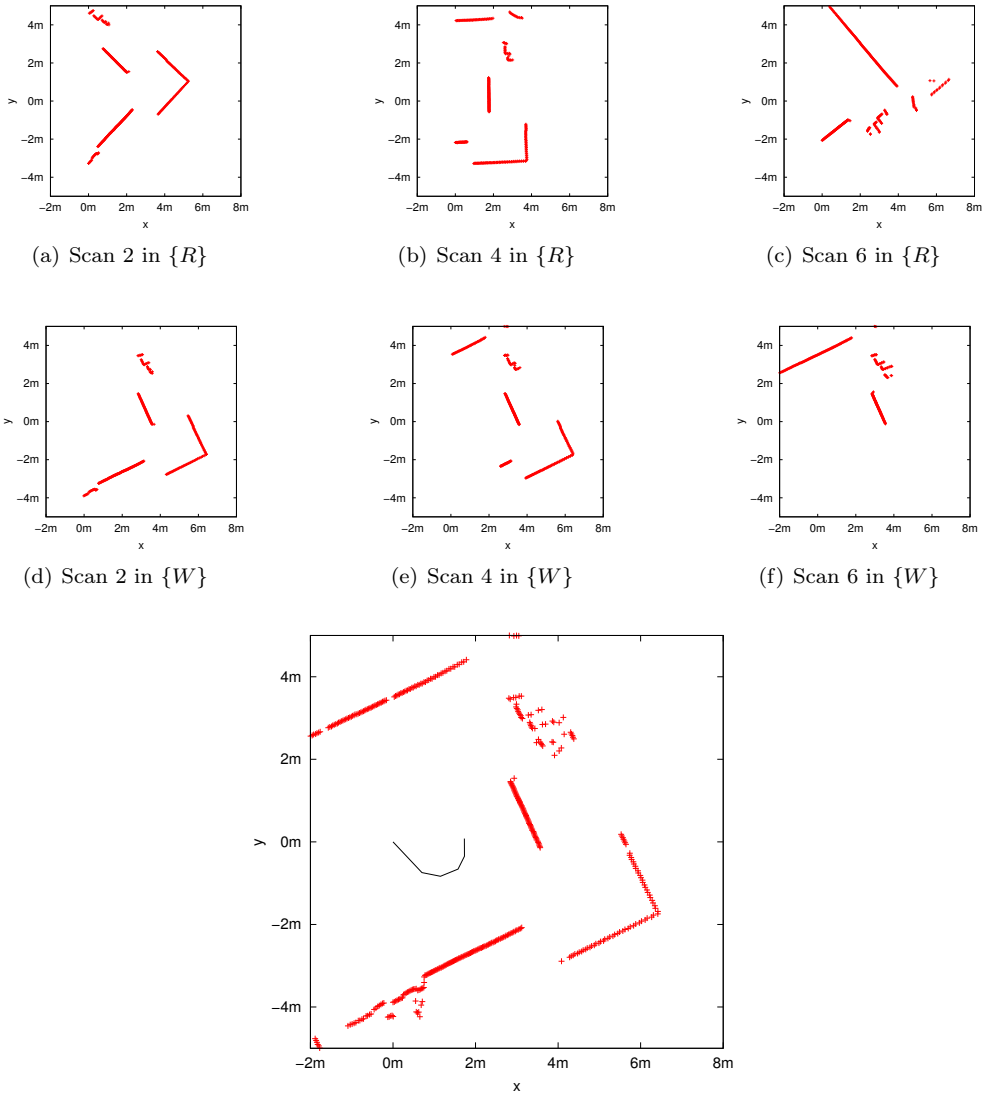
After all points from all point clouds have been transformed into the world frame, the point map M results from the union over all transformed points:

$$M = \bigcup_i \bigcup_j {}^{\{W\}}_{\{R_i\}}\mathbf{T}\ {}^{\{R\}}_{\{S\}}\mathbf{T}\ \begin{bmatrix} {}^{\{S\}}\mathbf{p}_j \\ 1 \end{bmatrix} \tag{3.5}$$

where i runs over the sequence of point clouds and vehicle poses and j over the points from one point cloud. A memory-efficient variant of a point map that avoids the duplicate storage of one and the same point is going to be presented in Chapter 3.5.

Figure 3.1 exemplary shows a sequence of 2D laser scans in the different coordinate frames and the resulting point map. In this example, the sensor frame $\{S\}$ and the robot frame $\{R\}$ coincide, as the laser scanner is mounted at the robot's center of rotation. As units equal in all frames, both transformations do not include scaling. Furthermore, the height of the scanner is neglected here.

For constructing other geometric maps, like for instance line models, polygonal representations or triangular meshes, the same transformations can be applied to the points representing the geometric primitives to be modeled. Of course, the same holds true for feature maps by solely transforming the position of a feature. It should also be noted, that mapping backwards from one frame to another, e.g. from $\{W\}$ to $\{R\}$, can be accomplished by taking the inverse of the corresponding transformation matrix.



(a) Scan 2 in $\{R\}$ (b) Scan 4 in $\{R\}$ (c) Scan 6 in $\{R\}$

(d) Scan 2 in $\{W\}$ (e) Scan 4 in $\{W\}$ (f) Scan 6 in $\{W\}$

(g) Map: Scans 1-6 in $\{W\}$

Figure 3.1: Example point map. Shown are three 2D laser scans in the robot's coordinate frame $\{R\}$ (a-c), transformed into the world frame $\{W\}$ (d-f) and the point map containing all points represented in $\{W\}$ (g).

3.2.2 Occupancy and Reflection Grid Maps

The central idea of grid maps is to partition the environment into equally sized regions, the *grid cells*. Each cell represents a particular characteristic, like for instance *traversability* (Shneier et al., 2008), of the area it covers. The advantage of grid maps is that they allow for constant time access to grid cells and the information contained therein. This fact is excessively used in Chapter 4.4 to determine whether or not a certain region in the environment is traversable by the robot.

The most prominent grid map, in the context of mapping and navigation, is the occupancy grid map introduced by Moravec and Elfes (1985). In occupancy grids it is assumed that every cell is either *occupied* or *free*. Each cell contains a single value representing the probability that the corresponding region in the robot's workspace is occupied by an object. A common assumption is that the prior probability to be occupied is 0.5 for each cell. Cells whose probability is 0.5 are also referred to as *unknown*. This also makes the main difference between continuous metric maps and grid maps. In the aforementioned point maps only the surface of environmental structures is modeled and there is no difference between free space and regions where no information is no available. That occupancy grids distinguish between unknown and free space is of particular interest in the context of robotic exploration as its goal is, in principal, the minimization of the size of unknown terrain. The primary disadvantage of occupancy grids is, however, their memory consumption. Geometric maps are quite compact as they only model geometric primitives representing environmental structures, i.e. the memory required by a geometric map only depends on the amount of information about the environmental structures itself. The size of the grid map, however, is independent of the space that is actually occupied by objects. Instead, the grid's size depends on the size of the robot's workspace and the size of a grid cell determining the resolution of the map. A larger cell size might cause discretization errors, e.g. when localizing the robot in a grid map, whereas using small cell sizes quadratically increases the number of cells and the memory requirement of the grid map respectively.

The key idea of probabilistic grid mapping algorithms is to compute the posterior over maps $\mathbf{m}$ given a sequence of vehicle poses or states $\mathbf{X}_{0:t}$ and a sequence of observations $\mathbf{Z}_{0:t}$ up to time t as

$$P\left(\mathbf{m}_t \mid \mathbf{X}_{0:t}, \mathbf{Z}_{0:t}\right)$$

or the robot's *belief* $Bel(\mathbf{m}_t)$. The difference to probabilistic SLAM formulations is that, here, the sequence of states $\mathbf{X}_{0:t}$ is given and the sequence of control inputs $\mathbf{U}_{0:t}$ does not need to be considered. As already mentioned, the map $\mathbf{m}_t$ partitions the space into a finite set of grid cells $m_t^{[xy]}$, i.e.

$$\mathbf{m}_t = \{m_t^{[xy]}\}.$$

Depending on what is actually modeled in a cell, different grid maps are distinguished of which some are briefly described in the following.

Probabilistic Occupancy Grids

In probabilistic occupancy grid maps each cell $m^{[xy]}$ has a binary occupancy value attached to it, commonly written as "1" for being occupied by an object and "0" for being free (cf. Thrun et al., 2005, Ch. 9). What is then modeled is the probability $p(m^{[xy]})$ that cell $m^{[xy]}$ is occupied. Instead of computing the posterior over all possible maps, the standard occupancy grid approach assumes that the cells are independent of each other. By this means, estimating $P\left(\mathbf{m} \mid \mathbf{X}_{0:k}, \mathbf{Z}_{0:k}\right)$ reduces to computing $P\left(m_i \mid \mathbf{X}_{0:k}, \mathbf{Z}_{0:k}\right)$ for all grid cells m_i. That is, by means of the independence assumption, the posterior over maps and the robot's belief results from the product of the individual probabilities:

$$Bel(\mathbf{m}_t) = P(\mathbf{m}_t | x_1, z_1, \ldots, x_n, z_n) = \prod_{x,y} Bel(m_t^{[xy]}) \tag{3.6}$$

The drawback of this factorization is that a relation between neighboring cells cannot be modeled. However, it is a commonly applied technique in grid mapping approaches as it drastically increases the efficiency of the mapping algorithm. When updating an occupancy grid map, the independence

assumption is again used and the central idea is to update each cell individually using a binary *Bayes Filter*.

$$Bel(m^{[xy]}) = \eta p(z_t|m_t^{[xy]}) \int p(m_t^{[xy]}|m_{t-1}^{[xy]}, x_t) Bel(m_{t-1}^{[xy]}) dm_{t-1}^{[xy]} \tag{3.7}$$

Commonly applied techniques for implementing this filter for map updates is the usage of *inverse sensor models* and the *log-odds representation* (Thrun et al., 2005, Ch. 9). Actual implementations of grid mapping algorithms use ray casting for determining which cells need to be updated. That is, one follows a range beam from its origin, i.e. the scanner's position in the world coordinate frame $\{W\}$, to the beam's end point and updates the cells lying along the beam. How to update the intersected cells is determined by the used sensor model. These sensor models often assume independence between individual beams. For a derivation of the complete occupancy grid mapping algorithm and more information on sensor models it is referred to (Stachniss, 2006, Ch. 2). Several extensions as well as a 3D grid representation can be found in (Martin and Moravec, 1996). Extensions to model the relation between neighboring cells and to learn shape models of moveable objects in dynamic environments have been proposed in (Biswas et al., 2002). Whether or not a cell is occupied by an object, i.e. the aforementioned binary value, results from simple thresholding: all cells whose occupancy probability is larger than 0.5 are marked as being occupied, the residual cells are marked as being free.

$$m^{[xy]} = \begin{cases} 1, & \text{if } p\left(m^{[xy]}\right) > 0.5 \\ 0, & \text{otherwise} \end{cases} \tag{3.8}$$

Probabilistic Reflection Maps

An alternative grid representation is the probabilistic reflection map or *counting model*. Instead of modeling the probability that a cell is occupied by an object, reflection maps represent the probability that something in a particular cell reflects a laser beam and causes a range measurement respectively. The idea is to count the number of cases $hits(x, y)$ when a range beam ends up in a cell $m^{[xy]}$ and the number of cases $misses(x, y)$ when the beam passes through $m^{[xy]}$ without being reflected. The probability that the cell $m^{[xy]}$ reflects a laser beam than simply results from the ratio:

$$Bel(m^{[xy]}) = \frac{hits(x, y)}{hits(x, y) + misses(x, y)} \tag{3.9}$$

Consider, for example, that 30 % out of 1000 range beams end up in a certain cell and 70 % pass through pass through that cell. Such a situation might be caused by taking range readings in front of semi-transparent objects like glass panes or less reflective table legs. The reflection probability for that cell is than $p(reflects(x, y)) = 0.3$ whereas the probability of being occupied in an occupancy grid will converge to zero. Using, for example a probability $p(occ \mid z) = 0.55$ that a cell is occupied when a range measurement z ends up in that cell and a probability $p(occ \mid z) = 0.45$ if the beam passes through the cell without being reflected, the value in the occupancy grid becomes (using *odds*-representation):

$$\left(\frac{0.55}{0.45}\right)^{n*0.3} * \left(\frac{0.45}{0.55}\right)^{n*0.7} = \left(\frac{0.45}{0.55}\right)^{-n*0.3} * \left(\frac{0.45}{0.55}\right)^{n*0.7} = \left(\frac{0.45}{0.55}\right)^{n*0.4}$$

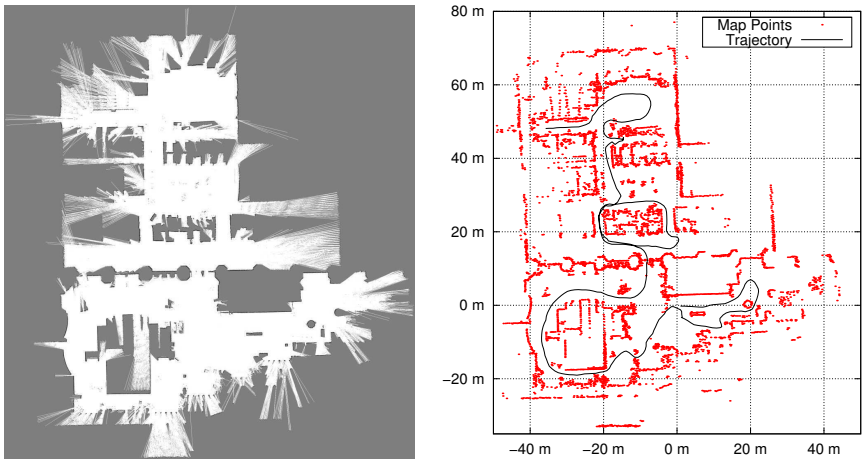
where n is the number of measurements. Especially in the context of path and motion planning on probabilistic occupancy grid maps this might yield the unwanted effect that robot wants to move through less reflective objects. On the other extreme, consider that 70 % of the beams end up in a cell and only 30 % pass through it. This might happen for doors that are normally closed. Again, the corresponding occupancy value will converge to one

$$\left(\frac{0.55}{0.45}\right)^{n*0.7} * \left(\frac{0.45}{0.55}\right)^{n*0.3} = \left(\frac{0.55}{0.45}\right)^{n*0.7} * \left(\frac{0.55}{0.45}\right)^{-n*0.3} = \left(\frac{0.55}{0.45}\right)^{n*0.4}$$

whereas the reflection probability will be $p(reflects(x, y)) = 0.3$. In general, one can say that the values in occupancy grids converge to either 0 or 1, whereas probabilities in reflection maps converge to values between 0 and 1 (Stachniss, 2006).

An example of a probabilistic reflection map for a larger indoor environment is shown in Figure 3.2.a. The trajectory of the robot has been determined using the SLAM approach proposed in Chapter 3.5. The known sequence of vehicle poses was then used to construct a probabilistic reflection map. A point map constructed using the same poses is shown in Figure 3.2.b. A reflection map has several advantages over other types of environment representations. Compared to occupancy grid maps, they seem to be more appropriate in dynamic environments and in the presence of semi-transparent objects as the contained probabilities do not converge to either 0 or 1. For this reason, all grid maps used in this thesis will be probabilistic reflection maps. Compared to point maps that only model environmental structures, reflection maps distinguish between free space and unvisited regions of the environment. This characteristic is of special interest in the context of robotic exploration since transitions between free and unvisited regions suggest where the robot has to move in order to construct a complete model of its workspace.

Bennewitz et al. (2009) proposed techniques for improving mobile robot localization in grid maps by means of probabilistic reflection maps. The same measure of reflection probability from Eq. (3.9) is also used by Frank et al. (2009). Here laser beams are manually labeled according to whether or not they hit a deformable object. The above statistics is then used to learn a probabilistic sensor model for deformable objects. Stachniss and Burgard (2003c) proposed an alternative grid map representation called *coverage map* that is quite similar to reflection maps. Here, a value between 0 and 1 determines how much of a cell is occupied by an object. That is, a cell that is completely covered by an object has a coverage value of 1, whereas a cell that is completely free has a coverage value of 0. After aligning a laser scan with the so far built model, Stachniss and Burgard extract the exact geometric primitives of environmental structures out of the range scan and compute the portion of the cell being occupied by the object that caused the reflection of the individual laser beams. In (Stachniss and Burgard, 2003a) and (Stachniss and Burgard, 2003b) they propose different exploration strategies explicitly using coverage maps.



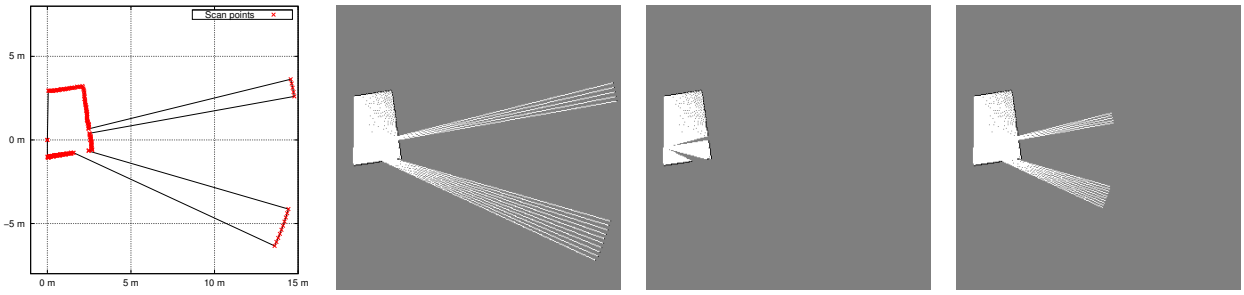
(a) Reflection Map (b) Point Map

Figure 3.2: Example of a probabilistic reflection map. Shown are a probabilistic reflection map (a) and a point map (b) constructed from a data set recorded by Nick Roy at the Edmonton Convention Centre. Grey and white regions show that the grid map distinguishes between free space and unvisited regions. The reflection map has a size of 2371 × 2440 cells with a side length of 5 cm. The point map contains 4544 points.

Handling Maximum Range Readings

Laser range scans contain maximum range readings if a laser beam has not been reflected by an object. Especially in rooms that are larger than the scanner's maximum measureable distance, large portions of the scan might be maximum range readings. Not receiving a reflected laser beam can also be caused by diffuse reflections, hitting less reflective objects, reflections in directions other than the scanner or other measurement failures. A typical scan containing maximum range readings is visualized in Figure 3.3.a. Using maximum range readings in the update procedure of a grid map causes that cells at the erroneous end points are falsely marked as being occupied by an object (see Figure 3.3.b). When constructing point maps, measurements that correspond to maximum range readings can be simply ignored, i.e. these false measurements are not contained in the final point map. However, completely ignoring maximum range readings neglects the information that there has not been any object reflecting a laser beam, i.e. that the region covered by the range scan is free. Hence, regions covered by these beams are still unknown as shown in Figure 3.3.c.

A common technique to handle these measurements in the update procedures of grid maps is to cut all laser beams whose measurement distance is larger than some threshold. That is, all cells along the laser beam whose distance to the scanner's position is smaller than this threshold, are updated as being free or not causing reflections. Furthermore, the cell containing the beam's end point is only updated as being occupied or causing reflections if its distance to the scanner is below the threshold. By this means, the maximum range reading is not completely ignored, but only the cells along the beam which lie outside of the circular region spanned by the distance threshold. In the example of Figure 3.3 the maximum measurable distance of the laser scanner is 15 m. The result of cutting the laser beams at a distance of 8 m is shown in Figure 3.3.d. Here, erroneous measurements do not cause that the cell containing the beam's end point is marked as being occupied or containing a reflective objects. However, the region covered by the beam is still marked as being free. As different range sensors have different maximum measurable distances, it is not reasonable to use a contant threshold for cutting laser beams, but to use a threshold depending on the sensor, e.g. 90 % of the maximum measurable distance. It should also be noted that in the generic implementation of grid maps written in the context of this thesis, the different types of handling maximum range readings can be switched during operation.



(a) (b) (c) (d)

Figure 3.3: Ignoring and cutting maximum range readings. Shown is a range scan (a) that contains maximum range readings (here 15 m). Not dealing with these measurements results in having occupied cells at these erroneous points (b). Completely ignoring max. range readings (c) neglects the information that there was no obstacle in this measurement direction. Here, all range readings are cut at a distance of 8 m (d).

Artificially Widening Beams

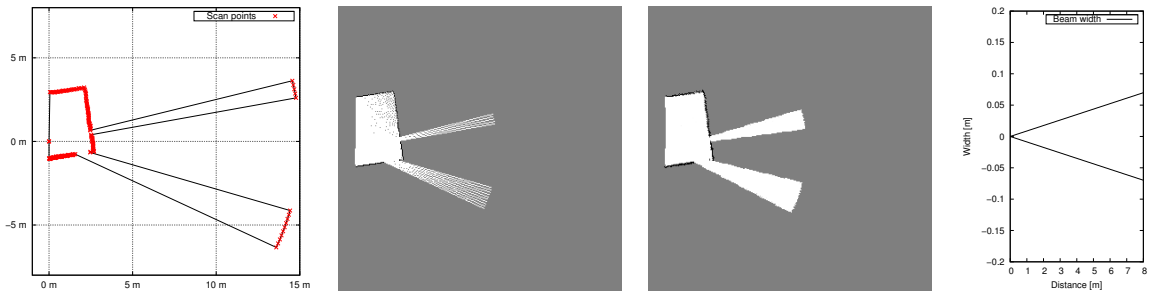
Another extension used here is the possibility of artificially widening the laser beams in the update of the grid map. Using simple ray-casting as described in the beginning of this section, the width of the laser beam is only one side length of a grid cell. Since only those cells that are covered by the thin constant-width laser beam are updated, a larger portion of cells lying between range beams are

not updated and might remain unknown. These not updated cells can cause artifacts in the form of unknown cells within regions of free space as shown in Figure 3.4.b. Removing these artifacts is quite straightforward. Instead of only updating a cell that is covered by a range beam, one can artificially widen the range beam by also updating neighboring cells along the perpendicular of the range beam. Which neighboring cells should be updated depends on their distance to the range beam, the distance d of the covered cell to the scanner's position and the angular resolution $\Delta\theta$ of the scanner. Additionally updating the neighboring cells removes the artifacts as shown in Figure 3.4.c. Here a beam width w has been used that linearly increases with the distance d. The increase thereby depends on the angular resolution $\Delta\theta$.

$$w(d) = \tan\left(\frac{\Delta\theta}{2}\right) * d \tag{3.10}$$

It should be noted that this approximation is not meant to model the beam diameter but the region covered by the angular displacement between two consecutive emissions of laser beams.

When updating a cell hit or passed by a laser beam, the artificial beam width is computed according to Eq. (3.10). Furthermore, the perpendicular of the beam at the cell center is determined. All cells being intersected by the perpendicular and whose distance to the beam does not exceed $w(d)$ are updated. Conducting this additional update for both the case when a beam passes through a cell and the case when it hits an object in that cell, completely removes the aforementioned artifacts and fully updates the complete region visible by the laser scanner. That is, by extending the regions to update along the perpendicular using $w(d)$ as a distance threshold in both directions, all cells between two range beams are updated. Again, the generic implementation of the grid map allows to enable and disable artificially widening beams during runtime as the aforementioned artifacts are of particular interest in certain applications. In the context of exploration, for example, these artifacts can attract the robot to acquire more information about the particular region.



(a) (b) (c) (d)

Figure 3.4: Widening beams. Shown is a range scan (a) and the corresponding reflection map (b). Using simple ray-casting not all cells are updated. Artificially widening the beam according to the angular resolution of the scanner (c) allows for updating all cells and the free space does not contain artifacts. The used linearly increasing beam width for an angular resolution of 0.5° is visualized in (d).

3.3 ICP-based Range Image Registration

The primary approach to SLAM presented in this thesis is based on the Iterative Closest Point (ICP) algorithm. The algorithm itself and different extensions to the original algorithm are described in the following.

3.3.1 The ICP Algorithm

The Iterative Closest Point (ICP) Algorithm was proposed by Besl and McKay and provides a "general-purpose, representation-independent method for the accurate and computationally efficient registration of 3-D shapes including free-form curves and surfaces" (Besl and McKay, 1992). The algorithm can be used for registering a variety of representations for geometric data, i.e. point sets, polylines, implicit and parametric curves and surfaces as well as triangle meshes. However, the internal handling of geometric data is solely based on points. The algorithm is then working on the points that define other geometric entities.

The search for the optimal transformation mapping some data set D onto another set M is carried out in an iterative manner, i.e. instead of performing a real search in the possibly infinite and continuous space of possible rotations and translations, the ICP algorithm iteratively refines an initial transformation. This initial transformation is either some estimate or simply the Identity matrix. In each iteration step, the algorithm determines pairs of corresponding elements in D and M that are then used to estimate a local transformation mapping the entities in D onto their corresponding entities in M. For the next iteration step, the found transformation is applied to D and to the *global* transformation. The procedure is repeated and the global transformation refined until the mapping error falls below some threshold. Being more precisely, for every point $\mathbf{d}_i \in D$ a corresponding point $\mathbf{m}_j \in M$ has to be determined. As the name already suggests, the corresponding point $\mathbf{m}_j$ is defined to be the closest point in M with respect to its Euclidean distance to $\mathbf{d}_i$, i.e.

$$\mathbf{m}_j = \underset{k=1\ldots N_M}{\arg\min} \|\mathbf{d}_i - \mathbf{m}_k\|.$$

Independently of Besl and McKay, Chen and Medioni have proposed a similar algorithm in (Chen and Medioni, 1992) where $\mathbf{m}_j$ is not the closest point, but the closest point along the local surface normal around $\mathbf{d}_i$. Zhang also proposed a similar algorithm. He uses both types of determining corresponding points, but imposes constraints on the pairs. This and other variations will be discussed later in this chapter.

What all three algorithms have in common, is the definition of, respectively, the mapping error and the error function[1] that is to be minimized by finding a rigid transformation, consisting of a rotation $\mathbf{R}$ and a translation $\Delta\mathbf{t}$, in a Least Mean Square (LMS) error sense. The data set D is iteratively rotated and translated in every iteration step, such that the points $\mathbf{d}_i \in D$ move as closest as possible to their corresponding points in M with respect to the squared Euclidean distances between the corresponding points (Besl and McKay, 1992; Chen and Medioni, 1992; Zhang, 1992, 1994):

$$E(\mathbf{R}, \Delta\mathbf{t}) = \frac{1}{N_D} \sum_{i=1}^{N_D} \sum_{j=1}^{N_M} w_{i,j} \left|\left|(\mathbf{R}\mathbf{d}_i + \Delta\mathbf{t}) - \mathbf{m}_j\right|\right|^2 \tag{3.11}$$

with $w_{i,j}$ being 1 if $\mathbf{d}_i$ and $\mathbf{m}_j$ form a pair of corresponding points and 0 otherwise. Considering this, Eq. (3.11) can be simplified to

$$E(\mathbf{R}, \Delta\mathbf{t}) = \frac{1}{N} \sum_{k=1}^{N} \left|\left|\left(\mathbf{R}\check{\mathbf{d}}_k + \Delta\mathbf{t}\right) - \check{\mathbf{m}}_k\right|\right|^2 \tag{3.12}$$

where N is the number of correspondence pairs $(\check{\mathbf{d}}_k, \check{\mathbf{m}}_k)$, i.e. those points $(\mathbf{d}_i, \mathbf{m}_j)$ for which $w_{i.j}$ is 1. This also forms a widely used error function for registration problems where the correspondences are already known.

The homogeneous transformation matrix $\mathbf{T}_{\text{LMS}}$, that is composed of the rotation $\mathbf{R}$ and the translation $\Delta\mathbf{t}$, minimizing this error function and mapping all points in D that are contained in one of the correspondence pairs onto their corresponding point in M is (in matrix notation):

$$\mathbf{T}_{\text{LMS}} = \underset{\mathbf{T}_i \in \mathcal{T}}{\arg\min} \frac{1}{N} \sum_{k=1}^{N} \left|\left| \mathbf{T} \begin{bmatrix} \check{\mathbf{d}}_k \\ 1 \end{bmatrix} - \begin{bmatrix} \check{\mathbf{m}}_k \\ 1 \end{bmatrix} \right|\right|^2 \tag{3.13}$$

[1] Note that the formulation of the error function slightly differs when using the point-to-plane metric (Chen and Medioni, 1992) instead of the original point-to-point metric (Besl and McKay, 1992).

The solution to this high-dimensional optimization problem, as proposed by Besl and McKay, is a quaternion-based method according to (Horn, 1987) and decouples the calculation of rotation $\mathbf{R}$ and translation $\Delta\mathbf{t}$. Due to the fact that the method of determining pairs of corresponding points varies in different formulations of the above optimization problem, algorithms that follow its iterative nature, are often referred to as *Iterative Corresponding Points* algorithms, instead of Iterative Closest Points. Their general procedure is as follows:

1. Search for pairs of corresponding points $(\check{\mathbf{d}}_k, \check{\mathbf{m}}_k)$ in D and M
2. Calculate the (rigid) transformation $\mathbf{T}$, composed of rotation $\mathbf{R}$ and translation $\Delta\mathbf{t}$, that maps each $\check{\mathbf{d}}_k$ to its corresponding point $\check{\mathbf{m}}_k$ by minimizing Eq. (3.13).
3. Apply transformation $\mathbf{T}$ to all points in D and to the global transformation matrix $\mathbf{T}_G$.

Repeat steps 1-3 until the termination criterion $E(\mathbf{R}, \Delta\mathbf{t}) < \epsilon$ is met.

That the ICP algorithm always converges to a local minimum with respect to the mean square error function in Eq. (3.11) is shown in (Besl and McKay, 1992).

3.3.2 Assumptions and Restrictions

Since the original ICP algorithm, as proposed by Besl and McKay and presented above, is widely used and forms the basis for many other registration algorithms, a large variety of extensions and variations have been proposed. These address generalizations of the ICP algorithm with its assumptions and restrictions, the algorithm's robustness and performance.

Restriction to Rigid Transformations

For minimizing the error function in Eq. (3.11), the ICP algorithm searches in the space of possible rotations and translations. Data sets where the optimal registration would involve e.g. scaling can not be registered. The ICP algorithm is restricted to rigid transformations – a subclass of affine transformations that, by only using rotation and translation, is range-preserving[2]. That is, after applying a rigid transformation $\mathbf{T}$ to two points $\mathbf{p}$ and $\mathbf{q}$ resulting in $\mathbf{p}'$ and $\mathbf{q}'$, the distance from $\mathbf{p}$ to $\mathbf{q}$ will be the same as that from $\mathbf{p}'$ to $\mathbf{q}'$ ($\|\mathbf{p} - \mathbf{q}\| = \|\mathbf{p}' - \mathbf{q}'\|$).
Assuming equal scaling in range images and non-deformable objects, this restriction does not affect the application of the ICP algorithm to range image registration problems. However, this changes if a sensed object is deformed in any other way. Problems that include scaling and other higher order deformations are addressed e.g. in (Feldmar and Ayache, 1994), (Feldmar and Ayache, 1996) and (Szeliski and Lavallée, 1996). An extension to the ICP algorithm taking non-rigid transformations into account can be found in (Hähnel, 2005).

Initial Transformation

What is proven by Besl and McKay is that the ICP algorithm monotonically converges to a local minimum. Due its iterative nature, convergence to the global minimum primarily depends on the initial transformation (Besl and McKay, 1992). For the registration of successive range images whose view poses only slightly differ, the Identity matrix, i.e. no transformation at all, provides a sufficient initial transformation, as it is already close the optimal transformation. If the deviations in position and orientation are larger and, consequently, the Identity matrix is no longer close enough to the optimal solution to allow the algorithm for converging to the global minimum, an appropriate initial transformation needs to be provided that is closer to the optimal solution.

[2] As a consequence, velocities, accelerations and forces are not changed when being mapped between coordinate frames. Applying a rigid transformation to a velocity vector will only affect the vector's direction, but not its magnitude (i.e. no scaling *and* no translation!).

If, however, only the deviations to the viewpose forming the origin of the model's coordinate frame are large, but the differences between consecutive range images are small, the sensor's change in position and orientation, can be tracked. That is, the (global) transformation w.r.t the model frame $\{M\}$ from pairwise or incremental registration of range image D_j can be used as the initial transformation for registering D_{j+1}. Thus, only the local transformation between D_j and D_{j+1} is initialized as being the identity matrix. A procedure like this already suffices for tracking the ego-motion of a handheld range sensor, i.e. when the pose shift between two consecutive viewposes can not be measured (May et al., 2009). When mounted on a mobile robot, the pose shift can be estimated e.g. by means of odometry, *gyrodometry* (Borenstein and Feng, 1996) or inertial sensors. For the registration in three-dimensional space, two-dimensional motion estimates, need to be extrapolated in 3D space (Borrmann et al., 2008).

If no such information is available, it is a matter of exhaustive search in the space of possible transformations to find a good initial estimate (Besl and McKay, 1992). Methods, like for instance the *data-aligned rigidity-constrained exhaustive search (DARCES)* proposed in (Chen et al., 1998), can constrain the search space and thereby speed up this pre-processing step. Furthermore, if the data sets contain distinct features, the range images can directly be registered without an initial estimation. Amongst others, Sharp et al. (2002) augment the description of data points in Cartesian space with that from a feature space.

Assumptions on Overlap, Noise and Outliers

Besl and McKay assume that the set D to be matched is a subset of M, i.e. $D \subseteq M$ and thus there is a corresponding point $\mathbf{m}_j \in M$ for every point $\mathbf{d}_i \in D$. If this assumption is not met, false correspondence pairs can negatively influence the algorithm's convergence to the global minimum, i.e. the optimal solution (see Fusiello et al., 2002). Here, two types of false correspondences can be distinguished: *initially false* correspondences and *generally false* correspondences.

A correspondence pair $(\mathbf{d}_i, \mathbf{m}_j)$ is initially false if another point $\mathbf{m}_k, k \neq j$ corresponds to the same point in space as $\mathbf{d}_i$, but $\mathbf{m}_j$ was found to be the corresponding point, e.g. due to poor initial estimates in early iteration steps. As the transformation is iteratively refined and the data set D transformed, the correct correspondence $(\mathbf{d}_i, \mathbf{m}_k)$ will be found when the transformation approaches the optimal solution. However, if the number of initially false correspondences is large, the algorithm might directly get stuck in a local minimum.

If there is not a single point $\mathbf{m}_k$ in M that corresponds to the same point as $\mathbf{d}_i$, a correspondence pair containing $\mathbf{d}_i$ is generally false, regardless of iteration step and initial transformation. Such a correspondence can be caused by noise, an erroneous measurement, i.e. an outlier, only partial overlap or occlusions. In the presence of noise the ICP algorithm can show a behavior similar to overfitting in the context of machine learning: it can degrade an optimal initial estimation by aligning the noisy data set D to the model set M. Whereas the ICP algorithm can handle Gaussian noise until a reasonable degree (Besl and McKay, 1992), it is quite sensitive to outliers. Whereas the impact of only a few outliers is negligible, occlusions in the scene can cause a significant number of (generally) false correspondences leading to incorrect alignments. Of course, the same holds true for range images, where only a small fraction of D overlaps with the model M. Several methods have been proposed to deal with partial overlap and occlusions that e.g. limit the range between corresponding points (Zhang, 1992), i.e. *reject certain correspondence pairs*, or use only a subset of $D' \subset D$ (like for instance *Reduced Points* in Nüchter et al., 2003b), i.e. *select only certain features* for the registration.

Termination Criterion

In the vanilla ICP algorithm by Besl and McKay the transformation is iteratively refined, until the *root mean square (RMS) error* falls below some threshold ϵ. The threshold ϵ thereby represents the tolerable alignment error that, however, heavily depends on the scale of the points' coordinates, the sensor's accuracy and measurement noise. If ϵ is too large, the algorithm might stop refining the transformation before the data is well aligned; if ϵ is too small the algorithm might get stuck in a (local) minimum with an error larger than ϵ causing the algorithm to not terminate at all. For

this reason, it is suggestive to prefer a smaller threshold and to use additional termination criteria in cases where an appropriate value for ϵ can not be determined in advance. A run of a typical error curve in the course of an iterative refinement by the ICP algorithm is shown in Figure 3.5 together with two examples of inadequate choices for ϵ. With $\epsilon = 200\,\text{mm}$ the algorithm would terminate before the root mean square error converges. Here, the error converges at 127.1414 mm, i.e. the algorithm would never converge with a threshold below that value, e.g. $\epsilon = 75\,\text{mm}$.

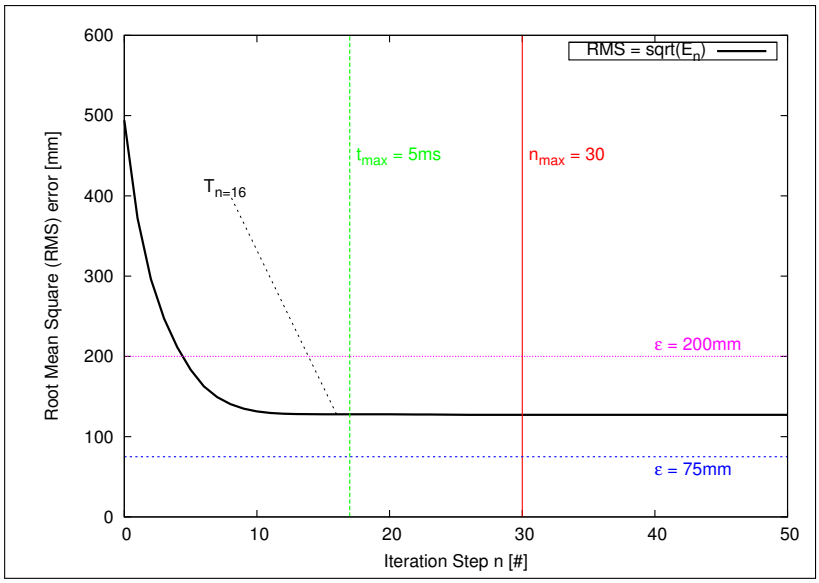


Figure 3.5: Typical error progression in the ICP algorithm. With an maximum allowable error $\epsilon = 75\,\text{mm}$ the ICP algorithm would have never terminated, whereas the larger threshold $\epsilon = 200\,\text{mm}$ would have caused early termination. Note that for visualization purposes it is not the RMS that is plotted here, but the RMS+100 mm.

Additional termination criteria that are used here are listed in the following:

- **Maximum number of iterations steps**: The most straightforward and commonly found workaround for the aforementioned termination problem, is the specification of a maximum number of iteration steps $N_{\max}$. Depending on the actual implementation of the algorithm, there is only one processing step that takes rather long, namely building the structure for later correspondence searches. This is, however, done only once in the first iteration step and subsequent iteration steps run only for a small fraction of a millisecond. Hence, it is suggestive to specify a rather large number of iteration steps, e.g. $N_{\max} = 100$ or $N_{\max} = 200$, especially when the initial estimates of $\mathbf{T}$ are known to be poor. In Figure 3.5, the algorithm would have terminated with the optimal solution after $N_{\max} = 30$ iterations steps.

- **Maximum allowable runtime**: All algorithms and components used in the context of the work presented here are intended to run online, possibly concurrent with many others. This fact and the essential requirement of real-time applicability suggest to also specify a maximum allowable runtime for a call of the ICP algorithm in addition to solely specifying a maximum number of iterations steps. Therefor, the implementation is augmented with an internal timer terminating the algorithm if the measured runtime exceeds some threshold $t_{\max}$. Further augmenting the implementation to keep track of the algorithm's state and return the reason for termination allows to turn the ICP algorithm into an anytime-algorithm (Zilberstein, 1996). That is, the user can repeatedly decide to call the ICP algorithm for small time fractions refining the results from the previous call until e.g. a new range image needs to be processed. As a simple example, calling the ICP algorithm five times with $t_{\max} = 1\,\text{ms}$ instead of one time with $t_{\max} = 5\,\text{ms}$ as in Figure 3.5 yields the same result but allows to carry out other processing steps in between.

- **Convergence of $\mathbf{T}$ / $E(\mathbf{T}_n)$**: In each iteration step n, the algorithm computes a local transformation $\mathbf{T}_n$ that is then applied to refine transformation $\mathbf{T}$. When the alignment can not be further improved by the algorithm, $\mathbf{T}$ does not need to be refined and the result of iteration step n is the identity matrix, i.e. $\mathbf{T}_n = I$. As the value of the error function $E(\mathbf{T}_n)$ also converges when the local transformations converge to the identity matrix, both can be checked for convergence as additional termination criteria:
  1. Terminate if $\Delta E = E(\mathbf{T}_n) - E(\mathbf{T}_{n-1})$ is approximately zero or falls below some threshold $\epsilon_{\Delta E}$.
  2. Terminate if $\mathbf{T}_n \approx I$, e.g. if the largest rotation angle θ is smaller than some threshold ϵ_θ (e.g. 0.25°) and if the magnitude of the translation vector $\|\Delta\mathbf{t}\|$ falls below another threshold $\epsilon_{\Delta\mathbf{t}}$ (e.g. 5 mm).

3.3.3 Generalizations and Extensions

Most extensions to the original formulation of the ICP algorithm that have been proposed address particular processing steps within the algorithm. A generalization of the ICP covering all these extensions and the aforementioned termination criteria can be formulated as follows (cf. Rusinkiewicz and Levoy, 2001):

1. Select features from D and M for the registration, i.e. build subsets $D' \subseteq D$ and $M' \subseteq M$.
2. Search for pairs of corresponding features $(\check{\mathbf{d}}'_k, \check{\mathbf{m}}'_k)$ in D' and M'.
3. Weight and reject correspondence pairs $(\check{\mathbf{d}}'_k, \check{\mathbf{m}}'_k)$.
4. Evaluate the defined error metric $E(\mathbf{T}_{n-1})$ for the remaining correspondence pairs $(\check{\mathbf{d}}'_k, \check{\mathbf{m}}'_k)$ and the current estimate $\mathbf{T}_n$ of transformation $\mathbf{T}$.
5. Calculate the transformation $\mathbf{T}_n$ that maps each $\check{\mathbf{d}}_k$ to its corresponding point $\check{\mathbf{m}}_k$ by minimizing $E(\mathbf{T}_n)$.
6. Apply transformation $\mathbf{T}_n$ to all points in D and to the global transformation matrix $\mathbf{T}_G$.

Repeat steps 1-6 until $(E(\mathbf{T}_n) < \epsilon) \vee (n \geq N_{\max}) \vee (t \geq t_{\max}) \vee (\mathbf{T}_n \approx I)$.

Feature Selection: When using *kd*-tree structures in the correspondence search as suggested by Besl and McKay, step 2 is the most complex processing step in the algorithm with $O(|D'| \log |M'|)$. where $|D'|$ and $|M'|$ denote the number of elements in D' and M'. As the expected runtime increases linearly in $|D'|$, several authors proposed methods to reduce the number of elements in $|D'|$ such as uniform subsampling (Turk and Levoy, 1994) in the first iteration or random re-sampling in each iterations step (Masuda et al., 1996). Zinßer et al. presented the *Picky ICP* algorithm, a collection of various extensions to the original ICP algorithm. Amongst these extension is a so-called *hierarchical point selection* (Zinßer et al., 2003). This subsampling technique selects only every 2^h-th point for the registration on hierarchy level h. The registration, using this subset, is carried out until convergence and the procedure is repeated on the next hierarchy level. Under the assumption that enough *good* data points are selected, the Picky ICP algorithm already converges to the global minimum in the early levels. On subsequent hierarchy levels, where the number of selected points is larger, the data sets are already coarsely aligned and the algorithm quickly terminates by reason of convergence.

When matching point sets, $|D'|$ can be reduced by replacing sets of points, whose maximum distance between each other falls below some threshold, by their centroids (Nüchter et al., 2003b). This procedure has the advantage that the spatial distribution of points remains almost unchanged whereas sampling might yield a subset D' with all points lying in the

same region. Lu and Milios project the model point set M into the coordinate frame $\{D\}$ of the data set D and then determine a subset $M' \subseteq M$ of points that are visible from the origin of D. A point $\mathbf{m} \in M$ is considered invisible if the points in M were ordered by their polar angle and this ordering changes direction around $\mathbf{m}$ or if $\mathbf{m}_i$ is occluded by another point $\mathbf{p}_j \in D \cup M$.

Correspondence Search: The actual search for correspondence pairs is then straightforward. In the case of point-to-point metric and using closest points as corresponding, a naive implementation is to build an any-to-any distance matrix or perform a *brute-force* search. Such an implementation turns the complexity of step 2 in the algorithm to $(|D'| \times |M'|)$. Besl and McKay already suggest the usage of *kd*-trees, a k-dimensional space partitioning data structure (Friedman et al., 1977). Building a static *kd*-tree from a set of n points requires $O(n \log n)$ (Mount and Arya, 1997), i.e. $O(|M'| \log |M'|)$. Note that this complexity highly depends on the means to calculate split axes and split values for the binary partitioning. If the model M is unchanged during iteration, the tree has to be constructed only once. Searching for correspondence pairs in subsequent iteration steps than takes $O(|D'| \log |M'|)$. The actual runtime can be further improved by e.g. approximate nearest neighbor search in *kd*-trees (Mount and Arya, 1997), searching with fixed radius bound or using *cached kd-trees* (Simon, 1996; Nüchter et al., 2007a) where corresponding points from iteration step $n-1$ are used as initial guesses for starting the search in iteration step n or to keep a correspondence pair from iteration step $n-1$ if the distance between $\check{\mathbf{m}}'_k$ and $\check{\mathbf{d}}'_k$ after transformation is still smaller than some threshold. Recently, Muja and Lowe published *FLANN*, a library for approximate nearest neighbor search that automatically selects the most suitable data representation, search algorithm and parameters based on the given data and the desired degree of precision (Muja and Lowe, 2009). The general drawback of approximate nearest neighbor search is that the approximation of the closest point might affect the convergence of the ICP algorithm. To avoid possible registration errors, e.g. Greenspan and Yurick (2003) suggested to use a hybrid search starting with a coarse approximate search in the first iteration steps and a classical nearest neighbor search for finetuning the registration in later steps. Here, it should be noted that for both search methods the same *kd*-tree can be used (cf. Greenspan and Yurick, 2003). Further refinements of *kd*-tree search techniques, regarding for example the partitioning planes, have been published in (Sproull, 1991), used e.g. by the Picky ICP algorithm by Zinßer et al. (2003). Another method for nearest neighbor search that is based on preprocessing and tracking correspondences under different constraints has been presented in (Greenspan and Godin, 2001).

Pair Rejection and Weighting: As already mentioned, false correspondences can have disastrous effects on the matching results. Several approaches have been proposed that reject certain point pairs to cope with false correspondences and partial overlap as well as to weight pairs according to their influence on the error $E(\mathbf{T}_n)$ and the calculated transformation $\mathbf{T}_n$. Pairs can, for example, be weighted according to their point-to-point distance in a point-to-point metric, the angle between surface normals in point-to-plane and plane-to-plane metrics or w.r.t to the difference in color or intensity (Godin et al., 1994).

In its extreme, certain pairs are completely rejected mainly to account for outliers in D and M. In Zhang's formulation of the algorithm constraints are imposed on the pairs in order to account for partial overlap, occlusions and outliers. Correspondence pairs that do not meet the requirements are rejected. Zhang proposes three constraints of which two are integrated in his algorithm: A pair $(\check{\mathbf{d}}'_k, \check{\mathbf{m}}'_k)$ is rejected if the distance between $\check{\mathbf{d}}'_k$ and $\check{\mathbf{m}}'_k$ exceeds a maximum tolerable distance $D_{\max}$ or if the angle between the tangent at $\check{\mathbf{d}}'_k$ and the tanget at $\check{\mathbf{m}}'_k$ is larger than the angle of rotation between the two images D and M. Whereas, the second constraint is rather specific to Zhang's approach, the first constraint forms a widely used extension to the original formulation of the ICP algorithm. Common ways to define this constraint are using a fixed distance threshold, e.g. a multiple of the standard deviation (Masuda et al., 1996), or some function changing $D_{\max}$ over the course of iterations. A larger value in early iterations steps pulls D' upon M' over larger distances, whereas smaller values

in later iterations steps allow for fine tuning the alignment. Minguez (2005) halves $D_{\max}$ each time when the error function $E(\mathbf{T}_n)$ converges, i.e. if $E(\mathbf{T}_n) - E(\mathbf{T}_{n-1}) < \epsilon_{\Delta E}$.

Another possibility is to reject a certain percentage of pairs, i.e. the pairs are first ordered according to their point-to-point distance and the worst n pairs are rejected, where n corresponds e.g. to the estimated overlap between D and M (Pulli, 1999; Chetverikov et al., 2002, 2005). The approach of Chetverikov et al. is referred to as the *Trimmed ICP algorithm (TrICP)*. Rejecting those pairs $(\check{\mathbf{d}}'_k, \check{\mathbf{m}}'_k)$ where $\check{\mathbf{m}}'_k$ lies on the border of a surface in M has been proposed in (Turk and Levoy, 1994), again to cope with partial overlap. Pajdla and Gool (1995) proposed a reciprocal rejection method called the *Iterative Closest Reciprocal Point algorithm (ICRP)*. Here $\check{\mathbf{m}}'_k$ is back-projected onto D finding the closest point $\mathbf{d}^\star_k$. The pair $(\check{\mathbf{d}}'_k, \check{\mathbf{m}}'_k)$ is rejected if the distance between $\check{\mathbf{d}}'_k$ and $\mathbf{d}^\star_k$ is larger than some threshold. Another formulation of this idea is to sort the pairs according to the model points. If multiple pairs contain the same model point $\check{\mathbf{m}}'_k$, only the pair with the smallest point-to-point distance is kept, the others are rejected. By this means it is prevented that a model point is present in more than one correspondence pair (Zinßer et al., 2003). This procedure has similar effects at borders in partially overlapping data sets, but can have unwanted effects with noisy data on surfaces (Chetverikov et al., 2005). Furthermore, Zinßer et al. apply a procedure for outlier detection and rejection presented in (Rousseeuw and LeRoy, 2003). They first calculate the standard deviation of the point-to-point distances in the found correspondence pairs and, second, reject all pairs whose point-to-point distance is larger than a multiple of the standard deviation. Montesano et al. (2005) proposed a probabilistic measure of the compability of corresponding points including the rejection of outliers.

Error Function Evaluation and Minimization: The core of the ICP algorithm is the evaluation of the error function $E(\mathbf{T}_{n-1})$ using the transformation $\mathbf{T}_{n-1}$ from the last iteration step or, in case of the first iteration, the initial estimate $\mathbf{T}_0$ and the remaining correspondence pairs $(\check{\mathbf{d}}'_k, \check{\mathbf{m}}'_k)$ as well as the calculation of the transformation $\mathbf{T}_n$ minimizing $E(\mathbf{T}_n)$. Besl and McKay (1992) proposed a point-to-point error metric for defining the error function $E(\mathbf{T})$ and to calculate the transformation $\mathbf{T}$ minimizing $E(\mathbf{T}_n)$ in a Least-Mean-Square (LMS) error sense. When using the sum of squared distances between corresponding points as the error function to minimize, there exist closed-form solutions for determining the rigid transformation minimizing the error, e.g. that of Arun et al. (1987) using Singular Value Decomposition (SVD) or that of Horn (1987) using unit quaternions. A comparison of four different algorithms for calculating 3D rigid transformations can be found in (Lorusso et al., 1995). For matching sets of different geometric objects Besl and McKay also provide the means for evaluating $E(\mathbf{T})$ based on the points defining the objects.

Chen and Medioni (1992) do not use the closest point in M' as the corresponding point for a point $\check{\mathbf{d}}'_k \in D'$, but the closest point along the local surface normal around $\check{\mathbf{d}}'_k$ thereby forming a point-to-plane error metric. For an error function $E(\mathbf{T})$, defined using a point-to-plane metric, no closed-form solutions exist. Here, the problem of calculating the transformation $\mathbf{T}$ minimizing $E(\mathbf{T})$ can be solved by e.g. generic non-linear methods such as Levenberg-Marquardt, non-linear optimization frameworks with or without derivatives such as NEWUOA (Powell, 2004) or by completely linearizing the problem (cf. Rusinkiewicz and Levoy, 2001). Masuda et al. (1996) define the error function $E(\mathbf{T})$ using a Least-Median-of-Squares error metric while only considering random subsets of D'.

Besides the iterative refinement of $\mathbf{T}$ with or without extrapolation of transformations, i.e. using the trend in $\mathbf{T}_{n-1}, \mathbf{T}_{n-2}, \ldots$ as an initial guess for $\mathbf{T}_n$ and starting with an initial estimate $\mathbf{T}_0$, as proposed by Besl and McKay, several methods have been proposed for searching for the optimal transformation T. These include applying Genetic Algorithms (Jankó et al., 2007), Simulated Annealing (Blais and Levine, 1995) or repeated iterative refinement for different initial estimates $\mathbf{T}_0$ (Simon, 1996).

In the ICP algorithm, the translational part of the transformation normally converges quickly while the rotational part is refined over several iteration steps. Lu and Milios (1997b) propose a two-steps transformation estimation to speed up the convergence of the rotational part of

the transformation. They combine the closest-point rule of the original ICP algorithm by Besl and McKay with a so-called *matching-range-point rule* (Lu and Milios, 1997b). The resulting *iterative dual correspondences (IDC)* algorithm, first, determines the transformation $\mathbf{T}_1 = (\mathbf{R}_1, \mathbf{t}_1)$ by means of the correspondence pairs $\left(\mathbf{d_i}, \mathbf{m}_j^1\right)$ resulting from nearest neighbor search. Without applying $\mathbf{T}_1$ to the data set, a new set of correspondence pairs $\left(\mathbf{d_i}, \mathbf{m}_j^2\right)$ is computed using the matching-range-point rule. Compared to the nearest neighbor search of the ICP algorithm, this second search is carried out in polar space. The corresponding point $\mathbf{m}_j^2$ results from searching in an angular interval, $[\theta_z - \epsilon_\theta, \theta_z + \epsilon_\theta]$ in the two-dimensional case. The measurement angle $\mathbf{d}_i^{\theta_z}$ of a point $\mathbf{d}_i$ forms the center of the angular interval while ϵ_θ forms an upper bound for the size of the interval. The model point $\mathbf{m}_j^2$ corresponding to $\mathbf{d}_i$ is then chosen according to the deviation in the polar range coordinate r of the points.

$$\mathbf{m}_j^1 = \underset{\mathbf{m}_k \in M}{\arg\min} \|\mathbf{d}_i - \mathbf{m}_k\| \tag{3.14}$$

$$\mathbf{m}_j^2 = \underset{\mathbf{m}_k \in M}{\arg\min} |\mathbf{d}_i^r - \mathbf{m}_k^r| \text{ with } \left(\mathbf{d}_i^{\theta_z} - \epsilon_\theta\right) \leq \mathbf{m}_k^\theta \leq \left(\mathbf{d}_i^{\theta_z} + \epsilon_\theta\right) \tag{3.15}$$

In other words, $\mathbf{m}_j^2$ is the model point in the angular interval around $\mathbf{d}_i^{\theta_z}$ that has the most similar range coordinate. Based on the new set of correspondence pairs, a second transformation $\mathbf{T}_2 = (\mathbf{R}_2, \mathbf{t}_2)$ is calculated. The transformation that is finally applied to the data set is composed of the translational part $\mathbf{t}_1$ resulting from the closest-point rule of the ICP and the rotational part $\mathbf{R}_2$ based on the matching-range-point rule:

$$\mathbf{T}_{IDC} = (\mathbf{R}_2, \mathbf{t}_1)\,. \tag{3.16}$$

Lu and Milios show that, with this combination, the registration procedure can converge fast in both the translational and the rotational part. The inherent drawback of the IDC algorithm is that the polar coordinates (θ, r) need to be known. Since D and M are normally referenced to different coordinate frames, the polar coordinates (θ, r) have to be computed for every point and the algorithm is not likely to scale very well for larger point sets. It should, furthermore, be noted that for registering 3D point sets, the angular interval needs to be replaced by a rectangle around the latitudinal and longitudinal angles in polar space.

Very recently, Segal et al. (2009) presented the combination of the classic point-to-point error metric from the original ICP formulation and a point-to-plane error metric. They integrate both approaches into a probabilistic framework and consider both point sets instead of matching the data set onto the model set. This *Generalized-ICP* thereby minimizes a plane-to-plane error metric (Segal et al., 2009).

Comprehensive overviews on extensions to the ICP algorithm can be found in (Simon, 1996) and (Rusinkiewicz and Levoy, 2001). Although some of the abovementioned extensions have already been implemented, it is a matter of future work to evaluate their effect on robustness and speed as well as to integrate those that are found to be appropriate into the currently used matching algorithm. The inherent problem here is that the majority of extensions to the ICP algorithm is designed for a specific purpose and, thus, only positively affects the registration results in certain situations. In other situations the same extension can negatively affect the registration result. As it cannot be simply determined which set of extension yields the best registration result in which situation, it is reasonable to keep the resulting registration algorithm as general as possible accepting e.g. slower convergence.

For the remainder of this thesis, the following composition of methods will be used: Matched are raw range measurements, i.e. no special feature selection is carried out and all points are used for the registration just like in the original formulation of the ICP algorithm by Besl and McKay, i.e. $D' = D$ and $M' = M$. Furthermore, the original error function with a point-to-point metric will be used where correspondence pairs $(\check{\mathbf{d}}_k', \check{\mathbf{m}}_k')$ are again defined as in Besl and McKay, i.e. $\check{\mathbf{m}}_k'$ is the closest point in M' to point $\check{\mathbf{d}}_k'$ in D'. The correspondence search is carried out using an approximate search in *kd*-trees (Mount and Arya, 1997)[3] and a distance threshold $D_{\max}$ between

[3] A C++ library for approximate nearest neighbor search by means *kd*-trees and *Bd*-trees can be found at `http://www.cs.umd.edu/~mount/ANN`.

corresponding points that exponentially decays during iteration (Holz, 2007; Holz et al., 2008). The error function that is evaluated and minimized in steps 4 and 5 is the one defined in Eq. (3.12) and (Besl and McKay, 1992). Calculating the rigid transformation $\mathbf{T}$ that minimizes $E(\mathbf{T})$ as defined in Eq. (3.12), given the set of corresponding points, for matching two-dimensional and three-dimensional point sets is addressed in the following two sections.

3.3.4 Matching 2D Point Sets

When matching two-dimensional point sets in the Euclidean xy-plane (right-handed coordinate system), model set M and data set D are defined as

$$M = \{\mathbf{m}_i \mid \mathbf{m}_i \in \mathbb{R}^2, i = 1, \ldots, N_m\} \tag{3.17}$$

$$D = \{\mathbf{d}_j \mid \mathbf{d}_j \in \mathbb{R}^2, j = 1, \ldots, N_d\} \tag{3.18}$$

with N_d and N_m being the number of raw range measurements stored, respectively, in D and M. Here, the transformation that maps D onto M consists of a rotation $\mathbf{R}_{\Delta\theta}$ by an angle $\Delta\theta$ around the z-axis and a translation $\Delta\mathbf{t} = (\Delta x, \Delta y)^T$. The error function $E(\mathbf{T})$ as provided in Eq. (3.12) can thus be written as:

$$E\left(\mathbf{R}_{\Delta\theta}, \Delta\mathbf{t}\right) = \sum_{i=1}^{N_m} \sum_{j=1}^{N_d} w_{i,j} \left|\left|\mathbf{m}_i - \left(\mathbf{R}_{\Delta\theta}\mathbf{d}_j + \Delta\mathbf{t}\right)\right|\right|^2 \tag{3.19}$$

Assuming that $\check{M} = \{\check{\mathbf{m}}_i\}_{i=1,\ldots,N} \subseteq M$ and $\check{D} = \{\check{\mathbf{d}}_i\}_{i=1,\ldots,N} \subseteq D$ are known from processing steps 1-3, Eq. (3.19) can be simplified to

$$E\left(\mathbf{R}_{\Delta\theta}, \Delta\mathbf{t}\right)^{(1)} = \sum_{i=1}^{N} \left|\left|\check{\mathbf{m}}_i - \left(\mathbf{R}\check{\mathbf{d}}_i + \Delta\mathbf{t}\right)\right|\right|^2. \tag{3.20}$$

As proposed in (Horn, 1987), determining the rotation $\mathbf{R}_{\Delta\theta}$ can be decoupled from determining the translation $\Delta\mathbf{t}$ by considering variants $\hat{M}$ and $\hat{D}$ of the point sets $\check{M}$ and $\check{D}$, both being shifted by their centroids $\check{\mathbf{c}}_m$ and $\check{\mathbf{c}}_d$.

$$\hat{M} = \{\hat{\mathbf{m}}_i \mid \hat{\mathbf{m}}_i = \check{\mathbf{m}}_i - \check{\mathbf{c}}_m\}_{i=1,\ldots,N} \tag{3.21}$$

$$\hat{D} = \{\hat{\mathbf{d}}_i \mid \hat{\mathbf{d}}_i = \check{\mathbf{d}}_i - \check{\mathbf{c}}_d\}_{i=1,\ldots,N} \tag{3.22}$$

Again rewriting $E(t)$ using $\hat{M}$ and $\hat{D}$ yields:

$$\begin{aligned} E\left(\mathbf{R}_{\Delta\theta}, \Delta\mathbf{t}\right)^{(2)} &= \sum_{i=1}^{N} \|\hat{\mathbf{m}}_i - \mathbf{R}_{\Delta\theta}\hat{\mathbf{d}}_i - \underbrace{\left(\Delta\mathbf{t} - \check{\mathbf{c}}_m + \mathbf{R}_{\Delta\theta\check{\mathbf{c}}_d}\right)}_{=\Delta\tilde{\mathbf{t}}}\|^2 \\ &= \sum_{i=1}^{N} \left|\left|\hat{\mathbf{m}}_i - \mathbf{R}_{\Delta\theta}\hat{\mathbf{d}}_i\right|\right|^2 && (3.23a) \\ &- 2\Delta\tilde{\mathbf{t}} \cdot \sum_{i=1}^{N} \left(\hat{\mathbf{m}}_i - \mathbf{R}_{\Delta\theta}\hat{\mathbf{d}}_i\right) && (3.23b) \\ &+ \sum_{i=1}^{N} \left|\left|\Delta\tilde{\mathbf{t}}\right|\right|^2 && (3.23c) \end{aligned}$$

The second term Eq. (3.23b) is zero since $\hat{\mathbf{m}}_i$ and $\hat{\mathbf{d}}_i$ are referred to the respective centroids (cf. Horn, 1987):

$$\begin{aligned}
\sum_{i=1}^{N}\left(\hat{\mathbf{m}}_i - R_{\Delta\theta}\hat{\mathbf{d}}_i\right) &= \sum_{i=1}^{N}\left[(\check{\mathbf{m}}_i - \check{\mathbf{c}}_m) - \mathbf{R}_{\Delta\theta}\left(\check{\mathbf{d}}_i - \check{\mathbf{c}}_d\right)\right] \\
&= \sum_{i=1}^{N}\left[\check{\mathbf{m}}_i - \frac{1}{N}\sum_{k=1}^{N}\check{\mathbf{m}}_k - \mathbf{R}_{\Delta\theta}\check{\mathbf{d}}_i + \frac{1}{N}\sum_{l=1}^{N}\mathbf{R}_{\Delta\theta}\check{\mathbf{d}}_l\right] \\
&= \sum_{i=1}^{N}\check{\mathbf{m}}_i - \underbrace{\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{N}\check{\mathbf{m}}_k}_{\sum_{i=1}^{N}\check{\mathbf{m}}_i} - \sum_{i=1}^{N}\mathbf{R}_{\Delta\theta}\check{\mathbf{d}}_i + \underbrace{\frac{1}{N}\sum_{i=1}^{N}\sum_{l=1}^{N}\mathbf{R}_{\Delta\theta}\check{d}_l}_{\sum_{i=1}^{N}\mathbf{R}_{\Delta\theta}\check{\mathbf{d}}_i} \\
&= 0
\end{aligned}$$

The third term (3.23c) can not be negative and finds its minimum at $\Delta\tilde{\mathbf{t}} = 0$, i.e. we can solve for the translation $\Delta\mathbf{t}$:

$$\Delta\mathbf{t} = \check{\mathbf{c}}_m - \mathbf{R}_{\Delta\theta}\check{\mathbf{c}}_d \tag{3.24}$$

The translation can be determined once the rotation angle $\Delta\theta$ is known. Furthermore, with $\Delta\tilde{\mathbf{t}} = 0$, Eq. (3.23) can be minimized by minimizing the first term Eq. (3.23a). The corresponding error function $E(\mathbf{R}_{\Delta\theta})^{(3)}$ is, however, no longer depending on the translation $\Delta\mathbf{t}$:

$$E\left(\mathbf{R}_{\Delta\theta}\right)^{(3)} = \sum_{i=1}^{N}\left|\left|\hat{\mathbf{m}}_i - \mathbf{R}_{\Delta\theta}\hat{\mathbf{d}}_i\right|\right|^2 \tag{3.25}$$

$$= \sum_{i=1}^{N}||\hat{\mathbf{m}}_i||^2 - 2\sum_{i=1}^{N}\left(\hat{\mathbf{m}}_i \cdot \mathbf{R}_{\Delta\theta}\hat{\mathbf{d}}_i\right) + \sum_{i=1}^{N}\left|\left|\mathbf{R}_{\Delta\theta}\hat{\mathbf{d}}_i\right|\right|^2 \tag{3.26}$$

Since rotation is length preserving, i.e. $\left|\left|\mathbf{R}_{\Delta\theta}\hat{\mathbf{d}}_i\right|\right|^2 \equiv \left|\left|\hat{\mathbf{d}}_i\right|\right|^2$, we obtain:

$$E\left(\mathbf{R}_{\Delta\theta}\right)^{(3)} = \underbrace{\sum_{i=1}^{N}||\hat{\mathbf{m}}_i||^2}_{S_{\hat{M}}} - 2\underbrace{\sum_{i=1}^{N}\left(\hat{\mathbf{m}}_i \cdot \mathbf{R}_{\Delta\theta}\hat{\mathbf{d}}_i\right)}_{S_{\hat{M}\hat{D}}} + \underbrace{\sum_{i=1}^{N}||\hat{\mathbf{d}}_i||^2}_{S_{\hat{D}}} \tag{3.27}$$

with $S_{\hat{M}}$ and $S_{\hat{D}}$ being the sums of squared lengths of the point vectors $\hat{\mathbf{m}}_i$ and $\hat{\mathbf{d}}_i$ and $S_{\hat{M}\hat{D}}$ being the sum of the respective dot products. Since $S_{\hat{M}}$ and $S_{\hat{D}}$ can not be negative, a maximization of

$S_{\hat{M}\hat{D}}$, in the following referred to as $E(\mathbf{R}_{\Delta\theta})^{(4)}$, minimizes $E(\mathbf{R}_{\Delta\theta})^{(3)}$:

$$
\begin{aligned}
E(\mathbf{R}_{\Delta\theta})^{(4)} &= \sum_{i=1}^{N}\left(\hat{\mathbf{m}}_i \cdot \mathbf{R}_{\Delta\theta}\hat{\mathbf{d}}_i\right) \qquad (3.28)\\
&= \sum_{i=1}^{N}\left(\underbrace{\begin{pmatrix}\hat{\mathbf{m}}_i^x\\ \hat{\mathbf{m}}_i^y\end{pmatrix}}_{\hat{\mathbf{m}}_i} \cdot \underbrace{\begin{pmatrix}\cos\Delta\theta & -\sin\Delta\theta\\ \sin\Delta\theta & \cos\Delta\theta\end{pmatrix}}_{\mathbf{R}_{\Delta\theta}} \underbrace{\begin{pmatrix}\hat{\mathbf{d}}_i^x\\ \hat{\mathbf{d}}_i^y\end{pmatrix}}_{\hat{\mathbf{d}}_i}\right)\\
&= \sum_{i=1}^{N}\left(\begin{pmatrix}\hat{\mathbf{m}}_i^x\\ \hat{\mathbf{m}}_i^y\end{pmatrix}^T \begin{pmatrix}\hat{\mathbf{d}}_i^x\cos\Delta\theta - \hat{\mathbf{d}}_i^y\sin\Delta\theta\\ \hat{\mathbf{d}}_i^y\cos\Delta\theta + \hat{\mathbf{d}}_i^x\sin\Delta\theta\end{pmatrix}\right)\\
&= \sum_{i=1}^{N}\left(\hat{\mathbf{m}}_i^x\hat{\mathbf{d}}_i^x\cos\Delta\theta - \hat{\mathbf{m}}_i^x\hat{\mathbf{d}}_i^y\sin\Delta\theta + \hat{\mathbf{m}}_i^y\hat{\mathbf{d}}_i^y\cos\Delta\theta + \hat{\mathbf{m}}_i^y\hat{\mathbf{d}}_i^x\sin\Delta\theta\right)\\
&= \sum_{i=1}^{N}\left(\cos\Delta\theta\left(\hat{\mathbf{m}}_i^x\hat{\mathbf{d}}_i^x + \hat{\mathbf{m}}_i^y\hat{\mathbf{d}}_i^y\right) + \sin\Delta\theta\left(\hat{\mathbf{m}}_i^y\hat{\mathbf{d}}_i^x - \hat{\mathbf{m}}_i^x\hat{\mathbf{d}}_i^y\right)\right)\\
\Longrightarrow\ E(\mathbf{R}_{\Delta\theta})^{(4)} &= \cos\Delta\theta\sum_{i=1}^{N}\left(\hat{\mathbf{m}}_i^x\hat{\mathbf{d}}_i^x + \hat{\mathbf{m}}_i^y\hat{\mathbf{d}}_i^y\right) + \sin\Delta\theta\sum_{i=1}^{N}\left(\hat{\mathbf{m}}_i^y\hat{\mathbf{d}}_i^x - \hat{\mathbf{m}}_i^x\hat{\mathbf{d}}_i^y\right) \qquad (3.29)
\end{aligned}
$$

Now, the rotation angle $\Delta\theta$ maximizing $E(\mathbf{R}_{\Delta\theta})^{(4)}$, and thereby minimizing $E(\mathbf{R}_{\Delta\theta})^{(3)}$, can be derived as follows:

$$
\begin{aligned}
-\sin\Delta\theta\sum_{i=1}^{N}\left(\hat{\mathbf{m}}_i^x\hat{\mathbf{d}}_i^x + \hat{\mathbf{m}}_i^y\hat{\mathbf{d}}_i^y\right) &+ \cos\Delta\theta\sum_{i=1}^{N}\left(\hat{\mathbf{m}}_i^y\hat{\mathbf{d}}_i^x - \hat{\mathbf{m}}_i^x\hat{\mathbf{d}}_i^y\right) = 0\\
\sin\Delta\theta\sum_{i=1}^{N}\left(\hat{\mathbf{m}}_i^x\hat{\mathbf{d}}_i^x + \hat{\mathbf{m}}_i^y\hat{\mathbf{d}}_i^y\right) &= \cos\Delta\theta\sum_{i=1}^{N}\left(\hat{\mathbf{m}}_i^y\hat{\mathbf{d}}_i^x - \hat{\mathbf{m}}_i^x\hat{\mathbf{d}}_i^y\right)\\
\tan\Delta\theta &= \frac{\sum_{i=1}^{N}\left(\hat{\mathbf{m}}_i^y\hat{\mathbf{d}}_i^x - \hat{\mathbf{m}}_i^x\hat{\mathbf{d}}_i^y\right)}{\sum_{i=1}^{N}\left(\hat{\mathbf{m}}_i^x\hat{\mathbf{d}}_i^x + \hat{\mathbf{m}}_i^y\hat{\mathbf{d}}_i^y\right)}\\
\Longrightarrow\qquad \Delta\theta &= \arctan\left(\frac{\sum_{i=1}^{N}\left(\hat{\mathbf{m}}_i^y\hat{\mathbf{d}}_i^x - \hat{\mathbf{m}}_i^x\hat{\mathbf{d}}_i^y\right)}{\sum_{i=1}^{N}\left(\hat{\mathbf{m}}_i^x\hat{\mathbf{d}}_i^x + \hat{\mathbf{m}}_i^y\hat{\mathbf{d}}_i^y\right)}\right) \qquad (3.30)
\end{aligned}
$$

Given $\Delta\theta$, substituting Eq. (3.30) into Eq. (3.24) yields:

$$
\begin{aligned}
\Delta\mathbf{t} &= \check{\mathbf{c}}_m - \mathbf{R}_{\Delta\theta}\check{\mathbf{c}}_d\\
&= \begin{pmatrix}\check{\mathbf{c}}_m^x\\ \check{\mathbf{c}}_m^y\end{pmatrix} \cdot \begin{pmatrix}\cos\Delta\theta & -\sin\Delta\theta\\ \sin\Delta\theta & \cos\Delta\theta\end{pmatrix}\begin{pmatrix}\check{\mathbf{c}}_d^x\\ \check{\mathbf{c}}_d^y\end{pmatrix} \qquad (3.31)
\end{aligned}
$$

The overall algorithm for matching 2D point sets is formulated in Algorithm 1. An example on registering two 2D laser scans taken in a typical indoor environment using the algorithm described above is shown in Figure 3.6. Both laser scans consist of 181 data points measured in a height of approximately 35 cm, i.e. $|D| = |M| = 181$. Matching one scan onto the other took approximately 3.5 ms. The only termination criterion used in this example is a maximum number of iterations $N_{\max} = 25$. For the correspondence search an efficient *kd*-tree implementation (Mount and Arya, 1997) has been used. The only constraint put on found correspondence pairs was a maximum allowable point-to-point distance $D_{\max}$ exponentially decaying over the course of iterations starting with $D_{\max,1} = 7.5\,\text{m}$ in the first iteration step and ending $D_{\max,25} = 10\,\text{cm}$ in the 25th iteration step.

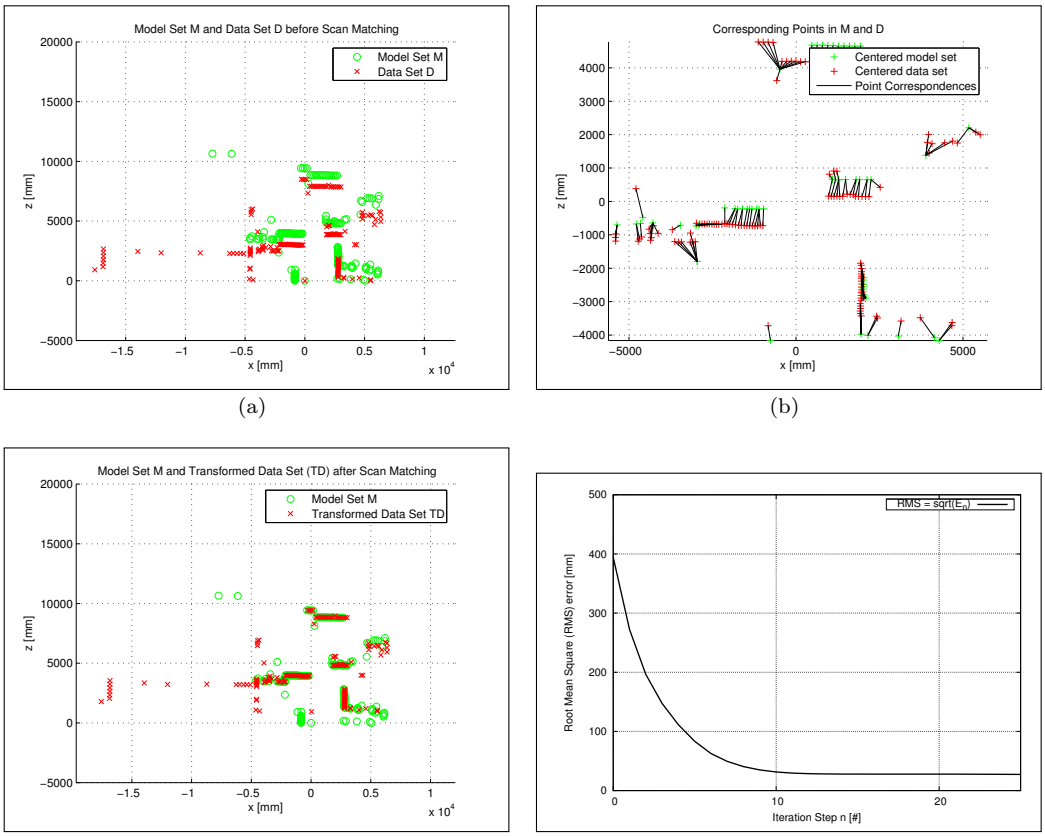


Figure 3.6: Example of an ICP-based 2D-Matching. Shown are model set M (green circles) and data set D (red crosses) before registration (a), the pairs of corresponding points $(\check{\mathbf{d}}_k, \check{\mathbf{m}}_k)$ (black lines) in the first iteration step (b) as well as M and D after successful registration (c) and the evolution of $E(\mathbf{T}_n)$ (d).

Algorithm 1: Algorithm for ICP-based Matching of 2D Point Sets

Require: $D = \{\mathbf{d}_i \mid \mathbf{d}_i \in \mathbb{R}^2, i = 1, \ldots, N_d\}$, $M = \{\mathbf{m}_j \mid \mathbf{m}_j \in \mathbb{R}^2, j = 1, \ldots, N_m\}$
Require: max. number of iteration steps $n_{\max}$, max. allowable mapping error ϵ
Require: initial estimate $\mathbf{T}_0$, $\mathbf{I}(3,3)$ if unknown

$n = 0$, $E_0 = \epsilon + 1$
$D_0 = \mathbf{T}_0\ D, \mathbf{T} = \mathbf{T}_0$
while $((n < n_{\max}) \wedge (E_n < \epsilon) \wedge \neg\,(\mathbf{T}_n \approx \mathbf{I}(3,3) \wedge n \neq 0))$ **do**
  $determinePairs\ \{(\check{\mathbf{d}}'_k, \check{\mathbf{m}}'_k) \mid k = 1, \ldots, N_k, N_k \leq N_d, \check{\mathbf{d}}'_k \in D, \check{\mathbf{m}}'_k \in M\}$
  $n = n + 1$
  $\check{\mathbf{c}}_m = \frac{1}{N_k} \sum_{k=1}^{N_k} \check{\mathbf{m}}_k$
  $\check{\mathbf{c}}_d = \frac{1}{N_k} \sum_{k=1}^{N_k} \check{\mathbf{d}}_k$
  $\hat{M} = \{\hat{\mathbf{m}}_i \mid \hat{\mathbf{m}}_i = \check{\mathbf{m}}_i - \check{\mathbf{c}}_m\}_{i=1,\ldots,N}$
  $\hat{D} = \{\hat{\mathbf{d}}_i \mid \hat{\mathbf{d}}_i = \check{\mathbf{d}}_i - \check{\mathbf{c}}_d\}_{i=1,\ldots,N}$
  $\Delta\theta = \arctan\left(\frac{\sum_{i=1}^{N}\left(\hat{\mathbf{m}}_i^y \hat{\mathbf{d}}_i^x - \hat{\mathbf{m}}_i^x \hat{\mathbf{d}}_i^y\right)}{\sum_{i=1}^{N}\left(\hat{\mathbf{m}}_i^x \hat{\mathbf{d}}_i^x + \hat{\mathbf{m}}_i^y \hat{\mathbf{d}}_i^y\right)}\right), \quad \mathbf{R}_{\Delta\theta} = \begin{pmatrix} \cos\Delta\theta & -\sin\Delta\theta \\ \sin\Delta\theta & \cos\Delta\theta \end{pmatrix}$
  $\Delta\mathbf{t} = \begin{pmatrix} \Delta x & \Delta y \end{pmatrix}^T = \check{\mathbf{c}}_m - \mathbf{R}_{\Delta\theta}\ \check{\mathbf{c}}_d$
  $\mathbf{T}_n = \begin{pmatrix} \mathbf{R}_{\Delta\theta} & \Delta\mathbf{t} \\ 0 & 1 \end{pmatrix}$
  $E = \sum_{k=1}^{N_k} \left|\left|\check{\mathbf{m}}_k - \left(\mathbf{R}\check{\mathbf{d}}_k + \Delta\mathbf{t}\right)\right|\right|^2$
  $\check{D}_n = \mathbf{T}_{n-1}\ \check{D}_{n-1}$
  $\mathbf{T} = \mathbf{T}_n\ \mathbf{T}$
end while

3.3.5 Matching 3D Point Sets

When matching 3D points sets, model set M and data set D are defined as

$$M = \{\mathbf{m}_i \mid \mathbf{m}_i \in \mathbb{R}^3, i = 1, \ldots, N_m\} \tag{3.32}$$
$$D = \{\mathbf{d}_j \mid \mathbf{d}_j \in \mathbb{R}^3, j = 1, \ldots, N_d\}, \tag{3.33}$$

i.e. according to Eq. (3.17) and Eq. (3.18) with the points $\mathbf{m}_i$ and $\mathbf{d}_j$ lying in Cartesian xyz-space (right-handed coordinate frame). The error function $E(\mathbf{T})$ is defined as in Eq. (3.19). Furthermore, the same decomposition can be applied to first calculate the rotation matrix $\mathbf{R}$ by minimizing $E(\mathbf{R})^{(3)}$ in Eq. (3.25) and then to calculate $\Delta\mathbf{t} = \check{\mathbf{c}}_m - \mathbf{R}\ \check{\mathbf{c}}_d$. The derivation for the three-dimensional case is up to Eq. (3.25) the same as that for the two-dimensional case presented in the last section.

$$E\left(\mathbf{R}, \Delta\mathbf{t}\right) = \sum_{i=1}^{N_m} \sum_{j=1}^{N_d} w_{i,j} \left|\left|\mathbf{m}_i - \left(\mathbf{R}\mathbf{d}_j + \Delta\mathbf{t}\right)\right|\right|^2 \tag{3.34}$$

$$E\left(\mathbf{R}\right)^{(3)} = \sum_{i=1}^{N} \left|\left|\hat{\mathbf{m}}_i - \mathbf{R}\hat{\mathbf{d}}_i\right|\right|^2 \tag{3.35}$$

In the three-dimensional case the rotation $\mathbf{R}$ consists of three rotations $\mathbf{R}_{\theta_x}$, $\mathbf{R}_{\theta_y}$ and $\mathbf{R}_{\theta_z}$ around the x, y and z axes and the translation $\Delta\mathbf{t}$ contains an additional translation along the z-axis, i.e. $\Delta t = \left(\Delta_x\ \Delta_y\ \Delta_z\right)^T$.

For calculating the 3D rotation $\mathbf{R}$ minimizing Eq. (3.35) four algorithms exist (Lorusso et al., 1995). The one used here is based on Singular Value Decomposition (SVD). As shown in (Arun et al., 1987), the rotation R minimizing Eq. (3.35) is

$$\mathbf{R} = \mathbf{V}\mathbf{U}^T \tag{3.36}$$

where $\mathbf{V}$ and $\mathbf{U}$ are orthonormal 3×3 matrices derived from the singular value decomposition

$$\mathbf{H} = \mathbf{U}\mathbf{\Lambda}\mathbf{V}^T \tag{3.37}$$

of the 3×3 correlation matrix $\mathbf{H}$ defined as

$$\mathbf{H} = \sum_{k=1}^{N_k} \check{\mathbf{d}}'_k \check{\mathbf{m}}'^T_k = \begin{pmatrix} S_{xx} & S_{xy} & S_{xz} \\ S_{yx} & S_{yy} & S_{yz} \\ S_{zx} & S_{zy} & S_{zz} \end{pmatrix} \tag{3.38}$$

with $S_{xx} = \sum_{k=1}^{N_k} \check{\mathbf{d}}'^x_k \check{\mathbf{m}}'^x_k$, $S_{xy} = \sum_{k=1}^{N_k} \check{\mathbf{d}}'^x_k \check{\mathbf{m}}'^y_k$, ..., $S_{zz} = \sum_{k=1}^{N_k} \check{\mathbf{d}}'^z_k \check{\mathbf{m}}'^z_k$. Detailled descriptions of this solution and proofs for Eq. (3.36) can be found in (Arun et al., 1987), (Lorusso et al., 1995) and (Nüchter et al., 2007b).

Algorithm 2: Algorithm for ICP-based Matching of 3D Point Sets

Require: $D = \{\mathbf{d}_i \mid \mathbf{d}_i \in \mathbb{R}^3, i = 1, \ldots, N_d\}$, $M = \{\mathbf{m}_j \mid \mathbf{m}_j \in \mathbb{R}^3, j = 1, \ldots, N_m\}$
Require: max. number of iteration steps $n_{\max}$, max. allowable mapping error ϵ
Require: initial estimate $\mathbf{T}_0$, $\mathbf{I}(4,4)$ if unknown
1: $n = 0$, $E_0 = \epsilon + 1$
2: $D_0 = \mathbf{T}_0\ D, \mathbf{T} = T_0$
3: **while** $((n < n_{\max}) \wedge (E_n < \epsilon) \wedge \neg(\mathbf{T}_n \approx \mathbf{I}(4,4)\ \wedge n \neq 0))$ **do**
4: $determinePairs\ \{(\check{\mathbf{d}}'_k, \check{\mathbf{m}}'_k) \mid k = 1, \ldots, N_k, N_k \leq N_d, \check{\mathbf{d}}'_k \in D, \check{\mathbf{m}}'_k \in M\}$
5: $n = n + 1$
6: $\check{\mathbf{c}}_m = \frac{1}{N_k} \sum_{k=1}^{N_k} \check{\mathbf{m}}_k$
7: $\check{\mathbf{c}}_d = \frac{1}{N_k} \sum_{k=1}^{N_k} \check{\mathbf{d}}_k$
8: $\hat{M} = \{\hat{\mathbf{m}}_i \mid \hat{\mathbf{m}}_i = \check{\mathbf{m}}_i - \check{\mathbf{c}}_m\}_{i=1,\ldots,N}$
9: $\hat{D} = \{\hat{\mathbf{d}}_i \mid \hat{\mathbf{d}}_i = \check{\mathbf{d}}_i - \check{\mathbf{c}}_d\}_{i=1,\ldots,N}$
10: $\mathbf{H} = \sum_{k=1}^{N_k} \hat{\mathbf{d}}'_k \hat{\mathbf{m}}'^T_k = \begin{pmatrix} S_{xx} & S_{xy} & S_{xz} \\ S_{yx} & S_{yy} & S_{yz} \\ S_{zx} & S_{zy} & S_{zz} \end{pmatrix}$
11: $(\mathbf{U}, \mathbf{\Lambda}, \mathbf{V}) = \mathrm{SVD}(\mathbf{H})$
12: $\mathbf{R} = \mathbf{V}\mathbf{U}^T$
13: $\Delta\mathbf{t} = \begin{pmatrix} \Delta x & \Delta y & \Delta z \end{pmatrix}^T = \check{\mathbf{c}}_m - \mathbf{R}\ \check{\mathbf{c}}_d$
14: $\mathbf{T}_n = \begin{pmatrix} \mathbf{R} & \Delta\mathbf{t} \\ 0 & 1 \end{pmatrix}$
15: $E = \sum_{k=1}^{N_k} \left|\left|\check{\mathbf{m}}_k - \left(\mathbf{R}\check{\mathbf{d}}_k + \Delta\mathbf{t}\right)\right|\right|^2$
16: $\check{D}_n = \mathbf{T}_{n-1}\ \check{D}_{n-1}$
17: $\mathbf{T} = \mathbf{T}_n\ T$
18: **end while**

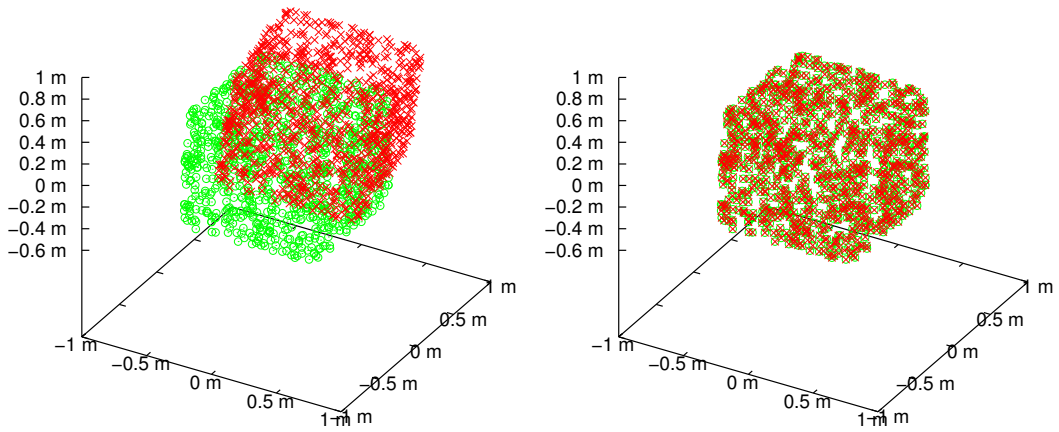
The final algorithm for matching 3D point sets is the same as that for matching 2D points sets with only a few changes being highlighted with underlain gray boxes (see Algorithm 2).

Here, light-gray boxes correspond to **minor changes**, e.g. points are given in $\mathbb{R}^3$ and the homogeneous transformation matrix $\mathbf{T}$ is a 4×4 matrix because of the additional rotations around x- and y-axes and the additional translation along the z-axis. The initial estimate is, if unknown, a 4×4 Identity matrix which is also used for the convergence test in the condition of the while-loop. Note that in the actual implementation, points are always represented in $\mathbb{R}^3$ with the z-component being zero in the case of a point lying in the Euclidean xy-plane and transformation matrices are, for the sake of compability, always 4×4 homogeneous transforms. Hence, these minor differences do not differ in the actual implementation.

Dark gray boxes correspond to **major changes**. First, the correspondence search is again carried out using kd-trees where here $k = 3$ and the construction of the tree gets slightly more complex (Mount and Arya, 1997). However, as the implementation by Mount and Arya allows

for high-dimensional space partitioning, it can also be used here and the only difference to the algorithm for 2D matching is that of specifying the dimension k. The second major change is that of calculating the rotation minimizing Eq. (3.35) by means of singular value decomposition in lines 10-12. In the actual implementation both are handled with a switch, where the user can define whether the registration of D and M should be carried out in $\mathbb{R}^2$ or $\mathbb{R}^3$.

An example for matching two 3D points sets is given in Figure 3.7. In this example, $N_m = 1000$ points have been generated on the surface of a unit cube centered at the origin of the coordinate frame forming the model set M. The data set D is an exact copy of M rotated around z by 10°, y by 7.5° and x by 5° and translated by $\begin{pmatrix}0.2\,\text{m} & 0.1\,\text{m} & 0.4\,\text{m}\end{pmatrix}^T$. The resulting transformation T_{example} has been used to generate D, i.e. $D = \{\mathbf{d}_i \mid [d_i\ 1]^T = \mathbf{T}_{\text{example}}\ [\mathbf{m}_i\ 1]^T, i = 1, \ldots, N_m\}$ but its inverse is *not* provided to the algorithm as an initial estimation $\mathbf{T}_0$. By this means, the registration of D and M is successful if the multiplication $\mathbf{T}_{\text{example}}\ \mathbf{T}$ is equal to the Identity matrix, where $\mathbf{T}$ is the transformation matrix determined by the matching algorithm. Note, that in this example only the translation $\Delta\mathbf{t}$ is fully determined whereas the rotation $\mathbf{R}$ is under-determined due to the geometrical shape of M and D. That is, due to the symmetry in D and M, once correctly translated, rotations about the x-, y- or z-axes by integral multiples of $\frac{\pi}{2}$ changing $\mathbf{T}$ lead to local minima in the registration. Therefore, small rotation angles have been used in $\mathbf{T}_{\text{example}}$ guaranteeing that the determined transformation $\mathbf{T}$ is the exact inverse of $\mathbf{T}_{\text{example}}$ assuming correct alignment. In typical environments both parts of the solution are fully determined.



(a) (b)

Figure 3.7: Example for matching two 3D point sets. Shown are data set D (red crosses) and model set M (green circles) before registration (a) and after successful registration (b).

The matrices $\mathbf{T}_{\text{example}}$ to generate D, the determind transformation $\mathbf{T}$ and the result of the matrix multiplication $\mathbf{T}_{\text{example}}\ \mathbf{T}$ are

$$\mathbf{T}_{\text{example}} = \begin{pmatrix} +0.9764 & -0.1618 & +0.1432 & +0.2000 \\ +0.1722 & +0.9830 & -0.0633 & +0.1000 \\ -0.1305 & +0.0864 & +0.9877 & +0.4000 \\ +0.0000 & +0.0000 & +0.0000 & +1.0000 \end{pmatrix}$$

$$\mathbf{T} = \begin{pmatrix} +0.9764 & +0.1722 & -0.1305 & -0.1603 \\ -0.1618 & +0.9830 & +0.0864 & -0.1005 \\ +0.1432 & -0.0633 & +0.9877 & -0.4174 \\ +0.0000 & +0.0000 & +0.0000 & +1.0000 \end{pmatrix}$$

$$\mathbf{T}_{\text{example}}\ T = \begin{pmatrix} +1.0000 & +0.0000 & +0.0000 & -0.0000 \\ -0.0000 & +1.0000 & -0.0000 & +0.0000 \\ +0.0000 & -0.0000 & +1.0000 & -0.0000 \\ +0.0000 & +0.0000 & +0.0000 & +1.0000 \end{pmatrix} \approx \mathbf{I}(4,4)$$

The registration over 25 iteration steps, including the generation of the *kd*-tree, took approximately

$(90 \pm 15)\,ms$ measured over 100 registrations. The evolution of the root mean square error is shown in Figure 3.8. Note that there is a residual RMS due to inaccuracies in floating point arithmetic shown in detail in Figure 3.8.b. Here, the residual mean square error is $2.5 \times 10^{-15}\,\mathrm{mm}^2$.

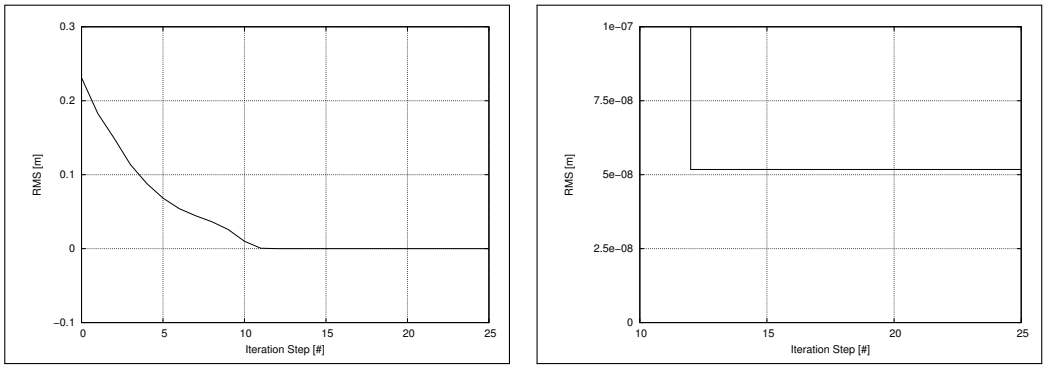


Figure 3.8: Evolution of the Root Mean Square Error over the 25 iteration steps.

3.3.6 Avoiding Correspondence Search – the Grid Closest Point Algorithm

In 1997, Ahmed et al. proposed a registration algorithm closely related to the ICP algorithm by Besl and McKay. The core of this algorithm is the so-called *grid closest point (GCP)* transform and the application of a genetic algorithm (GA) to minimize a cost function similar to the error function of the ICP algorithm in Eq. (3.11). Ahmed et al. present results showing that GCP/GA is slightly more accurate than the ICP and Zhang's algorithm (or at least equally accurate) while being significantly faster (cf. Ahmed et al., 1997; Yamany et al., 1998). The cost function of GCP/GA is defined as being

$$\mathcal{F}(\mathbf{R}, \mathbf{t}) = \sum_{i=1}^{N} d^2\left(\mathbf{R}\mathbf{d}_i + \mathbf{t}, S\right) \tag{3.39}$$

where rotation $\mathbf{R}$ and translation $\mathbf{t}$ form the rigid transformation being searched for and $d(\mathbf{d}_i, S)$ denotes the distance of point $\mathbf{d}_i$ to a surface S.

The combination of both leads to the algorithm's name *GCP/GA* (Ahmed et al., 1997). The algorithm is applicable for the registration of two-dimensional and three-dimensional point sets. To avoid the computationally expensive nearest neighbor search, Ahmed et al. first construct a k-dimensional grid that serves as a lookup-table for retrieving closest model points in the actual minimization (matching) step. This is done before starting the registration. First, the axis-aligned bounding box for the model set M is computed and then decomposed into equally sized grid cells with a specified side length. Then for every grid cell, a vector is computed pointing, from the cell's center, towards the closest model point. In the actual registration, each point $\mathbf{d}_i$ in the data set D that falls into one of the cells contributes to minimizing Eq. (3.39). The vector contained in the grid cells that $\mathbf{d}_i$ falls into, is used to calculate the squared distance $d^2\left(\mathbf{R}\mathbf{d}_i + \mathbf{t}, S\right)$ used in the error function. A nice characteristic of the GCP transform is that the grid only spans over the model set and that points not overlapping with the model set are not considered in the registration. However, in situations where the partial overlap between D and M is only small or the initial alignment inaccurate, this might also cause that the registration gets caught in a local minimum and the data sets do not get correctly aligned. Neglecting the aforementioned nice characteristic of dealing with partial overlaps, Ahmed et al. propose to build the grid structure over the composite bounding box around D and M. By this means, every point in D falls into one of the grid cells and contributes to the error function in Eq. (3.39).

To further speed up the registration, Yamany et al. (1998) proposed an approximate variant, where the vector pointing to the closest model point is directly used in the error function, instead

of calculating the real distance between $\mathbf{d}_i$ and the closest model point. Especially when applying this approximation, the size of the grid cell is an important parameter specifying the accuracy of the registration. Similar to the aforementioned idea of decaying distance thresholds between corresponding points for rough alignments in early iterations steps and finetuning of the registration result in later iteration steps, Yamany et al. propose to repeatedly apply GCP/GA for registration with varying grid cell sizes. That is, computing the GCP transform with a large grid cell size, e.g. 1 m, for a rough pre-alignment, and, once the optimal transformation has been found, to repeat the registration with a smaller grid cell size, e.g. 25cm. However, when repeatedly computing the grid, the benefit in terms of runtime from avoiding nearest neighbor search vanishes. The latter is the main reason for not applying this algorithm in the context of this thesis. However, it forms the basis for the registration algorithm based on Normal Distribution Transforms (NDT) described in the following chapter. Another drawback of the GCP transform is that for accurate alignments, i.e. for small cell sizes, the size of the grid exceeds dimensions that can no longer be handled online. The 10 cm-resolution GCP transform of a larger 3D model, for example, can potentially have a size of $(100\,\text{m}/0.1\,\text{m}) \times (100\,\text{m}/0.1\,\text{m}) \times (30\,\text{m}/0.1\,\text{m}) = 300\,000\,000$ cells.

For the actual registration and finding the transformation T minizing Eq. (3.39), Ahmed et al. apply a genetic algorithm. In principle, however, the correspondences determined by $\mathbf{d}_i$ and the vector contained in the grid cell that $\mathbf{d}_i$ falls into, could also be used in the point-to-point metric of Eq. (3.11) so that the registration can be carried out using the original ICP algorithm. Still, the idea of applying evolutionary algorithms in range image registration has been adopted in many other approaches, e.g. (Robertson and Fisher, 2002). An approach similar to the GCP transform called *pre-computed voxel nearest neighbor* has been proposed by Yan and Bowyer. To reduce possibly large grids, they apply a Principal Component Analysis (PCA) on the three-dimensional data of the model point set M for computing dominant variances representing M which are then used to reduce the volume of the model (Yan and Bowyer, 2007).

3.4 NDT-based Range Image Registration

Another registration algorithm based on the *Normal Distributions Transform (NDT)* has been proposed by Biber and Strasser (2003). Compared to the ICP algorithm by Besl and McKay, the key idea of NDT-based range image registration is to represent the model point set by a piecewise continuous and differentiable probability density, i.e. a linear combination of normal distributions comparable to a Gaussian mixture model. Matching two point sets D and M is then no longer carried out on the basis of correspondence pairs and a point-to-point error metric, but by deriving, from that representation, the probability of finding a point from a range image D in the model M. By this means, there is no need for the computationally expensive nearest neighbor search as in the ICP algorithm (Magnusson and Duckett, 2005). Instead, the registration problem becomes maximizing the sum, that the aligned points in D *"score"* on the NDT-representation of M (Biber and Strasser, 2003). As the NDT-representation of M is a piecewise continuous and differentiable function, numerical optimization methods, like for instance Newton's method or gradient descent, can be used for the registration.

This chapter will first cover the central transformation of two-dimensional and three-dimensional point clouds into NDT-representations, i.e. grid-based linear combinations of normal distributions as proposed by Biber and Strasser. Matching point clouds on the basis of this representation will be discussed in the remainder of this chapter.

3.4.1 The Normal Distributions Transform (NDT)

As already mentioned, the Normal Distributions Transform (NDT) models the distribution of points in a range image by a collection of local normal distributions. For this purpose, the space that is occupied by the model point set is subdivided into uniform grid cells of constant size, i.e. squares or cubes for, respectively, two-dimensional and three-dimensional point sets just like in the Grid Closest Point algorithm described in the previous chapter. The NDT representation NDT(P) for a point set P is then built by, first, assigning each point $\mathbf{p} \in P$ to its corresponding grid cell

and, second, computing the normal distribution $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ for each grid cell that contains more than a certain number of points $n_{\min}$, e.g. $n_{\min} = 3$ for the two-dimensional case (Biber and Strasser, 2003) and $n_{\min} = 5$ for the three-dimensional case (Magnusson, 2006). In both two-dimensional and three-dimensional case, the normal distribution is multivariate with expected value vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$. After collecting all N points $\mathbf{p}_k \in P_i \subseteq P, k = 1, \ldots, N_i$ that fall into the same grid cell c_i, the cell's contribution to the NDT-representation is derived from the normal distribution of P_i with

$$\boldsymbol{\mu}_i = \frac{1}{N_i} \sum_{k=1}^{N_i} \boldsymbol{p}_k \tag{3.40}$$

$$\boldsymbol{\Sigma}_i = \frac{1}{N_i - 1} \sum_{k=1}^{N_i} (\boldsymbol{p}_k - \boldsymbol{\mu}_i)(\boldsymbol{p}_k - \boldsymbol{\mu}_i)^T \tag{3.41}$$

The probability $p(\mathbf{x})$ of measuring or finding a point at position $\mathbf{x}$ in cell c_i is then modeled by the normal distribution $\mathcal{N}_i(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$:

$$p(\mathbf{x}) \sim \exp\left(-\frac{(\boldsymbol{x} - \boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}_i^{-1} (\boldsymbol{x} - \boldsymbol{\mu}_i)}{2}\right) \tag{3.42}$$

Compared to occupancy grid maps, this does not only provide the probability of measuring a point in a certain cell, but also the probability of measuring a point for each position in this cell (Biber and Strasser, 2003). The algorithm for constructing the kD-NDT for k-dimensional point clouds with cell size l is summarized in Algorithm 3.

Algorithm 3: Algorithm for building the NDT of n-dimensional point sets: NDT(P, l)

Require: Point set $P = \{\boldsymbol{p}_i \mid \boldsymbol{p}_i \in \mathbb{R}^n, i = 1, \ldots, N_P\} \neq \emptyset$
Require: grid cell size $l > 0$
Ensure: $C = \text{NDT}(P, l)$
1: Build cell structure C {Construct grid representation.}
2: **for all** cells $c_i \in C$ **do**
3: $P_i = \emptyset$ {Initialize all subsets as being empty.}
4: **end for**
5: **for all** points $\boldsymbol{p}_k \in P$ **do**
6: $c_i = findCell(\boldsymbol{p}_k)$ {Find the cell c_i that contains $\boldsymbol{p}_k$}
7: $P_i = P_i \cup \boldsymbol{p}_k$ {Add $\boldsymbol{p}_k$ to the subset $P_i \subseteq P$}
8: **end for**
{loop through all cells that contain at least $n_{\min}$ points}
9: **for all** cells $c_i \in C$ with $|P_i| \geq n_{\min}$ **do**
10: $\boldsymbol{\mu}_i = \frac{1}{|P_i|} \sum_{k=1}^{|P_i|} \boldsymbol{p}_k$ with $\boldsymbol{p}_k \in P_i$ {Calculate mean.}
11: $\boldsymbol{\Sigma}_i = \frac{1}{|P_i|-1} \sum_{k=1}^{|P_i|} (\boldsymbol{p}_k - \boldsymbol{\mu}_i)(\boldsymbol{p}_k - \boldsymbol{\mu}_i)^T$ {Calculate covariance matrix}
12: **end for**

As a simple example, Figure 3.9 shows the NDT representation for a three-dimensional point set. The points were sampled from a normal distribution. According to different Bernoulli distributions, constant offsets have been applied to the x-, y- and z-coordinates resulting in eight clusters of points. All these clusters are itself normal distributed as the constant offsets forming a cluster only affect the mean vector. This is reflected by the spherical error ellipsoids representing the normal distributions for grid cells that contain more than $n_{\min} = 5$ points (see Figure 3.9.b). Only those cells that contain less than $n_{\min}$ points do not contribute to the NDT-representation.

3.4.2 Registration using Normal Distribution Transforms

To actually match a data set D against a model set M, an NDT representation is constructed for the points in M. Just like in the ICP algorithm, the transformation mapping D onto M is

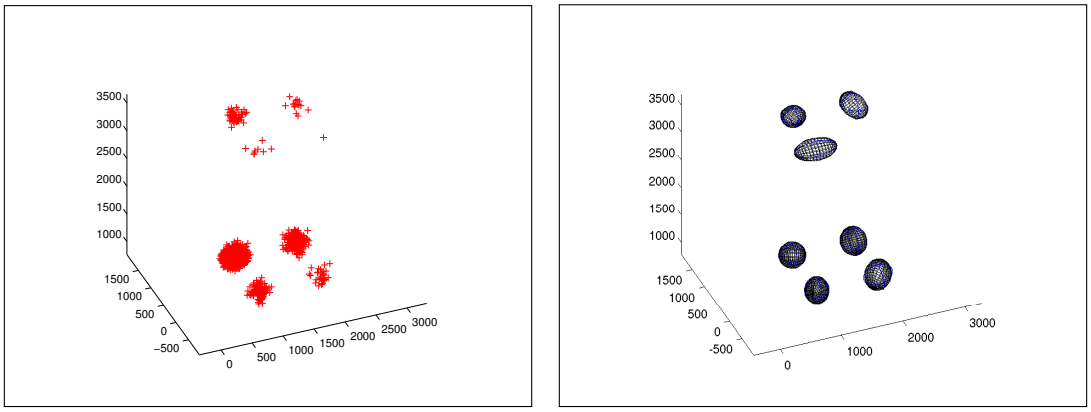


Figure 3.9: Example of a 3D-NDT Representation. Shown are a set of randomly generated 3D points (a) and the corresponding NDT representation (b). Note that the single point in the upper right of (a) is the only point in the corresponding grid cell and is thus ignored in the NDT representation.

iteratively refined. However, instead of carrying out a nearest neighbor search in each iteration step, the score of the currently used transformation is evaluated, possibly together with the values of the first and second derivative of the score function. The score s of a transformation $\mathbf{p}$ is defined as being the sum of the score of all points:

$$s(\mathbf{p}) = \sum_{k=1}^{|D|} p\left(T\left(\mathbf{p}, \mathbf{d}_k\right)\right) \tag{3.43}$$

$$\text{with} \quad p(\mathbf{x}) = \frac{1}{2} \exp\left(-\frac{\left(\boldsymbol{x} - \boldsymbol{\mu}_i\right)^T \boldsymbol{\Sigma}_i^{-1} \left(\boldsymbol{x} - \boldsymbol{\mu}_i\right)}{2}\right) \tag{3.44}$$

where $T(\mathbf{p}, \mathbf{d}_k)$ corresponds to a transformation of a point $\mathbf{d}_k$ according to the values in the parameter vector $\mathbf{p}$. In this context, $\mathbf{p}$ can be an arbitrary represention for a rigid transformation, e.g. a homogeneous transformation matrix as in the formulation of the ICP algorithm.

The registration procedure is the same for both the two-dimensional and the three-dimensional case. For every point $\mathbf{d}_k$ in the data set D, the cell c_i that contains $\mathbf{d}_k$ is looked up in the NDT-representation of M. If c_i contains a valid normal distribution, i.e. $|P_i| > n_{\min}$, the probability $p(\mathbf{d}_k)$ of measuring a point on the model surface at $\mathbf{d}_k$ is evaluated according to mean value and covariance matrix of cell c_i and contributes to the score of the currently used transformation $\mathbf{p}$. The refinement of $\mathbf{p}$ and the alignment of the data sets D and M is carried out in an optimization procedure, respectively, maximizing the score $s(\mathbf{p})$ or minimizing the negative score $-s(\mathbf{p})$. In the context of this thesis two optimization frameworks have been used, namely OPT++ (Meza et al., 2007) and NEWUOA (Powell, 2004). Since the NDT-representation is a piecewise differentiable function not only the score function but also its first and second derivatives can be used in this optimization. Evaluating the score function and computing its first and second derivatives for both the two-dimensional and three-dimensional case is described in the following sections.

3.4.3 Matching 2D Point Sets

In the two-dimensional case, the transformation aligning a data set D with a model set M consists of a translation $\mathbf{t}$ in the xy-plane and a rotation $\mathbf{R}$ about the z-axis. That is, there are three parameters to optimize and the parameter vector is $\mathbf{p} = [\mathbf{t}^x \ \mathbf{t}^y \ \theta_z]^T$ where $\mathbf{t}^x$ and $\mathbf{t}^y$ correspond to a translation along the x-axis and y-axis respectively. The transformation function $T(\mathbf{p}, \mathbf{d}_k)$ can

then be formulated as

$$T(\mathbf{p}, \mathbf{d}_k) = \underbrace{\begin{pmatrix} \cos\theta_z & -\sin\theta_z \\ \sin\theta_z & \cos\theta_z \end{pmatrix}}_{\mathbf{R}} \begin{pmatrix} \mathbf{d}_k^x \\ \mathbf{d}_k^y \end{pmatrix} + \underbrace{\begin{pmatrix} \mathbf{t}^x \\ \mathbf{t}^y \end{pmatrix}}_{\mathbf{t}} \tag{3.45}$$

just like in the ICP algorithm. As optimization problems are generally formulated as minimization problems, registration of point sets using the NDT-representation is formulated as minimizing the negative score function, i.e.

$$\begin{aligned} s_{NDT}(\mathbf{p}) &= -\sum_{k=1}^{|D|} p\left(T\left(\mathbf{p}, \mathbf{d}_k\right)\right) && (3.46) \\ &= -\sum_{k=1}^{|D|} \begin{pmatrix} \cos\theta_z & -\sin\theta_z \\ \sin\theta_z & \cos\theta_z \end{pmatrix} \begin{pmatrix} \mathbf{d}_k^x \\ \mathbf{d}_k^y \end{pmatrix} + \begin{pmatrix} \mathbf{t}^x \\ \mathbf{t}^y \end{pmatrix}. && (3.47) \end{aligned}$$

The goal of the optimization is to iteratively compute a refinement $\Delta\mathbf{p}$ for the parameter vector $\mathbf{p}$, such that $\mathbf{p}$ converges to a transformation that minimizes the negative score function. Using Newton's algorithm, $\Delta\mathbf{p}$ can be computed by solving the equation

$$\mathbf{H}\Delta\mathbf{p} = -\mathbf{g} \tag{3.48}$$

where $\mathbf{H}$ and $\mathbf{g}$ are the Hessian and gradient of the score function $s_{NDT}(\mathbf{p})$. The increment $\Delta\mathbf{p}$ is then added to the current parameter vector $\mathbf{p}$ to refine the estimate of the transformation, i.e.

$$\mathbf{p} \leftarrow \mathbf{p} + \Delta\mathbf{p}. \tag{3.49}$$

The entries of the gradient $\mathbf{g}$ can, thereby, be written as:

$$\begin{aligned} \mathbf{g}_i &= \frac{\delta s}{\delta \mathbf{p}_i} && (3.50) \\ &= \sum_{k=1}^{|D|} \check{\mathbf{d}}_k^T \boldsymbol{\Sigma}^{-1} \frac{\delta \check{\mathbf{d}}_k}{\delta \mathbf{p}_i} \exp\left(\frac{-\check{\mathbf{d}}_k^T \boldsymbol{\Sigma}^{-1} \check{\mathbf{d}}_k}{2}\right) && (3.51) \end{aligned}$$

where $\check{\mathbf{d}}_k$ is the deviation of the transformed point $\mathbf{d}_k$ from the mean vector $\boldsymbol{\mu}$. Both $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ are looked up in the cell of the NDT-representation that contains the point $\mathbf{d}_k$, i.e.

$$\check{\mathbf{d}}_k = T(\mathbf{p}, \mathbf{d}_k) - \boldsymbol{\mu}. \tag{3.52}$$

Accordingly, the entries of the Hessian $\mathbf{H}$ are:

$$\begin{aligned} \mathbf{H}_{ij} &= \frac{\delta^2 s}{\delta \mathbf{p}_i \delta \mathbf{p}_j} && (3.53) \\ &= \sum_{k=1}^{|D|} \exp\left(\frac{-\check{\mathbf{d}}_k^T \boldsymbol{\Sigma}^{-1} \check{\mathbf{d}}_k}{2}\right) \\ &\quad \left[\left(\check{\mathbf{d}}_k^T \boldsymbol{\Sigma}^{-1} \frac{\delta \check{\mathbf{d}}_k}{\delta \mathbf{p}_i}\right) \times \left(-\check{\mathbf{d}}_k^T \boldsymbol{\Sigma}^{-1} \frac{\delta \check{\mathbf{d}}_k}{\delta \mathbf{p}_j}\right) + \check{\mathbf{d}}_k^T \boldsymbol{\Sigma}^{-1} \frac{\delta^2 \check{\mathbf{d}}_k}{\delta \mathbf{p}_i \delta \mathbf{p}_j} + \frac{\delta \check{\mathbf{d}}_k^T}{\delta \mathbf{p}_j} \boldsymbol{\Sigma}^{-1} \frac{\delta \check{\mathbf{d}}_k}{\delta \mathbf{p}_i}\right] && (3.54) \end{aligned}$$

The first-order partial derivatives $\frac{\delta \check{\mathbf{d}}_k}{\delta \mathbf{p}_i}$ of $\check{\mathbf{d}}_k$ in Equations (3.51) and (3.54) are given by the i-th column of the Jacobian matrix $\mathbf{J}$ with

$$\mathbf{J} = \begin{pmatrix} 1 & 0 & -\check{\mathbf{d}}_k^x \sin\theta_z - \check{\mathbf{d}}_k^y \cos\theta_z \\ 0 & 1 & \check{\mathbf{d}}_k^x \cos\theta_z - \check{\mathbf{d}}_k^y \sin\theta_z \end{pmatrix} \tag{3.55}$$

and the second-order partial derivatives $\frac{\delta^2 \check{\mathbf{d}}_k}{\delta \mathbf{p}_i \delta \mathbf{p}_j}$ are

$$\frac{\delta^2 \check{\mathbf{d}}_k}{\delta \mathbf{p}_i \delta \mathbf{p}_j} = \begin{cases} \begin{pmatrix} -\check{\mathbf{d}}_k^x \cos\theta_z + \check{\mathbf{d}}_k^y \sin\theta_z \\ -\check{\mathbf{d}}_k^x \sin\theta_z - \check{\mathbf{d}}_k^y \cos\theta_z \end{pmatrix} & \text{if } i = j = 3 \text{ and} \\ \begin{pmatrix} 0 \\ 0 \end{pmatrix} & \text{otherwise.} \end{cases} \tag{3.56}$$

The resulting algorithm for matching two-dimensional point sets using the normal distributions transform is formulated in Algorithm 4.

Algorithm 4: Algorithm for matching 2D points using NDT: $ndtMatch(D, M, \mathbf{p}, l)$

Require: Data point set $D = \{\boldsymbol{d}_i \mid \boldsymbol{d}_i \in \mathbb{R}^2, i = 1, \ldots, N_D\} \neq \emptyset$
Require: Model point set $M = \{\boldsymbol{m}_j \mid \boldsymbol{m}_j \in \mathbb{R}^2, j = 1, \ldots, N_M\} \neq \emptyset$
Require: Initial transformation estimate $\mathbf{p}$
Require: grid cell size $l > 0$
1: $ndt(M, l)$ {Build NDT-representation for M. See Algorithm 3.}
2: **while** termination criteria not met **do**
3: $score \leftarrow 0$
4: $\mathbf{g} \leftarrow 0$
5: $\mathbf{H} \leftarrow 0$
6: **for all** $\boldsymbol{d}_i \in D$ **do**
7: $(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \leftarrow$ find the cell c_k that containts $T(\mathbf{p}, \mathbf{d}_i)$
8: $\check{\mathbf{d}}_i \leftarrow T(\mathbf{p}, \mathbf{d}_i) - \boldsymbol{\mu}_k$
9: $score \leftarrow score - p(\check{\mathbf{d}}_i)$ {see Eq. (3.46)}
10: update gradient $\mathbf{g}$ {see Eq. (3.51)}
11: update Hessian $\mathbf{H}$ {see Eq. (3.54)}
12: **end for**
13: $\Delta\mathbf{p} \leftarrow solve(\mathbf{H}\Delta\mathbf{p} = -\mathbf{g})$ {e.g. using OPT++}
14: $\mathbf{p} \leftarrow \mathbf{p} + \Delta\mathbf{p}$
15: **end while**

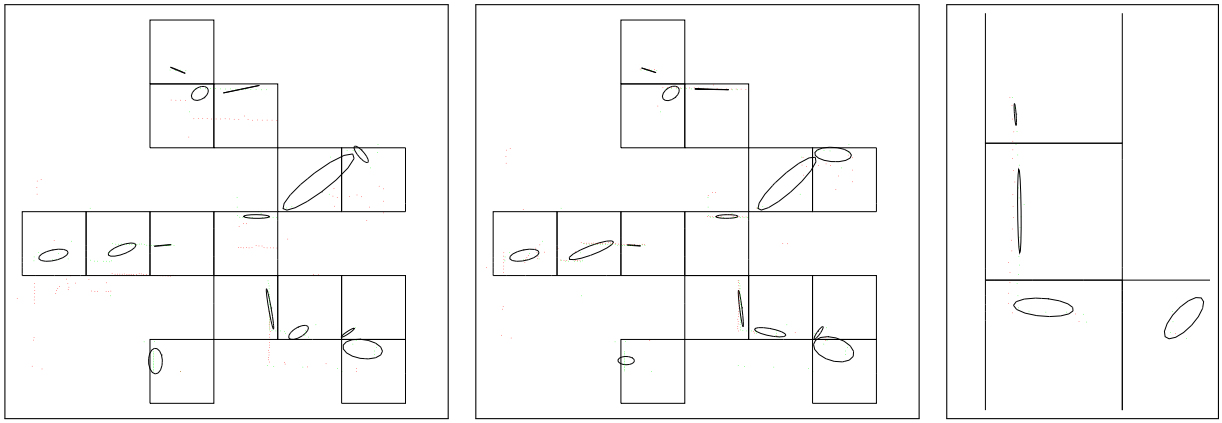
Since analytic first-order and second-order derivatives are available, the actual implementation uses Newton's method from OPT++ (Meza et al., 2007) and the optimization problem is formulated as a *NLF2* problem, i.e. a nonlinear function that is to be minimized using both first-order and second-order derivatives. A typical result of applying the resulting matching algorithm based on normal distribution transforms to register two two-dimensional laser range scans is shown in Figure 3.10. It can also be seen in the figure that for some regions the grid cell size seems to be not appropriate as the according normal distribution does not well model the environmental structure, whereas in other cells the error ellipses almost turn into line segments oriented according to the sensed environmental surface in the corresponding region.

Biber and Strasser (2003) presented results that, amongst others, show that NDT-based registration is less sensitive to noise than using the ICP algorithm by Besl and McKay. This, however, only holds true when the normal distributions of the NDT accurately model environmental structures. This issue will be further addressed in Chapter 3.4.5.

3.4.4 Matching 3D Point Sets

Just like for the ICP, the registration algorithm for NDT in the three-dimensional case only slightly differs from the two-dimensional case as formulated in Algorithm 4. Besides the dimensionality of the grid and the transformation function, only the evaluation of the score functions as well as the computation of its derivatives change.

Several extensions have been proposed to NDT-based scan matching that allow for registering spatial information even without changing the registration algorithm. Ripperda and Brenner (2005)



(a) (b) (c)

Figure 3.10: Matching 2D point sets using the normal distributions transform. Shown are two two-dimensional laser range scans before registration (a) and after registration (b). The NDT-representation of the first scan is visualized using boxes for the cells and error ellipses for the normal distributions. A detail view from after registration is shown in (c).

proposed a registration procedure that cuts three-dimensional point clouds into slices and then applies the NDT-matching algorithm as formulated in Algorithm 4 to each of these slices. The score of a transformation is thereby given by the sum of the scores of all slices (Dold et al., 2007). The resulting registration algorithm can, however, only be applied if both data sets are almost well aligned to the same horizontal plane. In the case of larger deviations in height or orientations about x- and y-axes, the algorithm can not be applied, as the slices of D and M model different regions in the environment.

Magnusson et al. (2007) propose a generalization of NDT-based scanmatching for the three-dimensional case. Starting from a 3D representation of the rigid transformation mapping D onto M, they derive appropriate score functions as well as their derivatives for different types of transformation representations. Just like for the ICP algorithm in Chapter 3.3.5, here we consider only the case where the transformation being searched for consists of rotations $\mathbf{R}_x$, $\mathbf{R}_y$ and $\mathbf{R}_z$ about and the translation $\mathbf{t} = (\mathbf{t}^x\ \mathbf{t}^y\ \mathbf{t}^z)^T$ along the x-,y- and z-axes. That is, compared to the two-dimensional case with three parameters, here six parameters need to be optimized in order to move D onto M, i.e. $\mathbf{p} = [\mathbf{t}^x\ \mathbf{t}^y\ \mathbf{t}^z\ \theta_x\ \theta_y\ \theta_z]^T$. Using X-Y-Z Euler angles (cf. Craig, 1989, Chapter 2) and this six-dimensional parameter vector, the transformation function from Eq. (3.45) turns into

$$T(\mathbf{p}, \mathbf{d}_k) = \mathbf{R}_x\ \mathbf{R}_y\ \mathbf{R}_z\ \mathbf{d}_k + \mathbf{t} \tag{3.57}$$

$$= \begin{bmatrix} c_y c_z & -c_y s_z & s_y \\ s_x s_y c_z + c_x s_z & -s_x s_y s_z + c_x c_z & -s_x c_y \\ -c_x s_y c_z + s_x s_z & c_x s_y s_z + s_x c_z & c_x c_z \end{bmatrix} \mathbf{d}_k + \begin{bmatrix} \mathbf{t}^x \\ \mathbf{t}^y \\ \mathbf{t}^z \end{bmatrix} \tag{3.58}$$

where c_x is shorthand for $\cos\theta_x$ and s_x for $\sin\theta_x$ etc. Note that the transformation depends not only on the actual rotation representation but also on the order of application. In principal, any representation can be chosen here as long as it is used and applied in a consistent way. Under the assumption that only small angles are considered, respectively, in the transformation and the refinement vector $\Delta\mathbf{p}$, trigonometric simplifications can be applied in order to further simplify the transformation function. For small θ, $\sin\theta \approx \theta$, $\cos\theta \approx 1 - 0.5\theta^2$ and $\theta^2 \approx 0$.

$$T(\mathbf{p}, \mathbf{d}_k) \approx \begin{bmatrix} 1 & -\theta_z & \theta_y \\ \theta_z & 1 & -\theta_x \\ -\theta_y & \theta_x & 1 \end{bmatrix} \mathbf{d}_k + \begin{bmatrix} \mathbf{t}^x \\ \mathbf{t}^y \\ \mathbf{t}^z \end{bmatrix} \tag{3.59}$$

This is of particular interest here, as by means of these simplifications, most of the terms in the derivatives reduce to zero. The gradient and the Hessian forming the first-order and second-order

derivatives do not differ from Equations (3.51) and (3.54); only the partial derivatives change. All second-order partial derivatives reduce to zero when the small angle assumption is applied. The first-order partial derivatives are given by the i-th column of the 3×6 Jacobian $\mathbf{J}$ with

$$\mathbf{J} = \begin{bmatrix} 1 & 0 & 0 & 0 & \check{\mathbf{d}}_k^z & -\check{\mathbf{d}}_k^y \\ 0 & 1 & 0 & -\check{\mathbf{d}}_k^z & 0 & -\check{\mathbf{d}}_k^x \\ 0 & 0 & 1 & \check{\mathbf{d}}_k^y & -\check{\mathbf{d}}_k^x & 0 \end{bmatrix}. \tag{3.60}$$

A typical result of applying this generalization to the NDT-based registration algorithm is shown in Figure 3.11. The figures also shows the weak point of the NDT-based registration algorithm. After registration, D and M are still not well aligned as the used grid cell size did not allow for a perfect refinement of the transformation matrix. However, for decreasing the cell size the number of points on the surface of the cubes is not enough. That is, with a smaller cell size the majority of cells do not contain enough points for calculating covariance matrix and mean vector for a valid normal distribution.

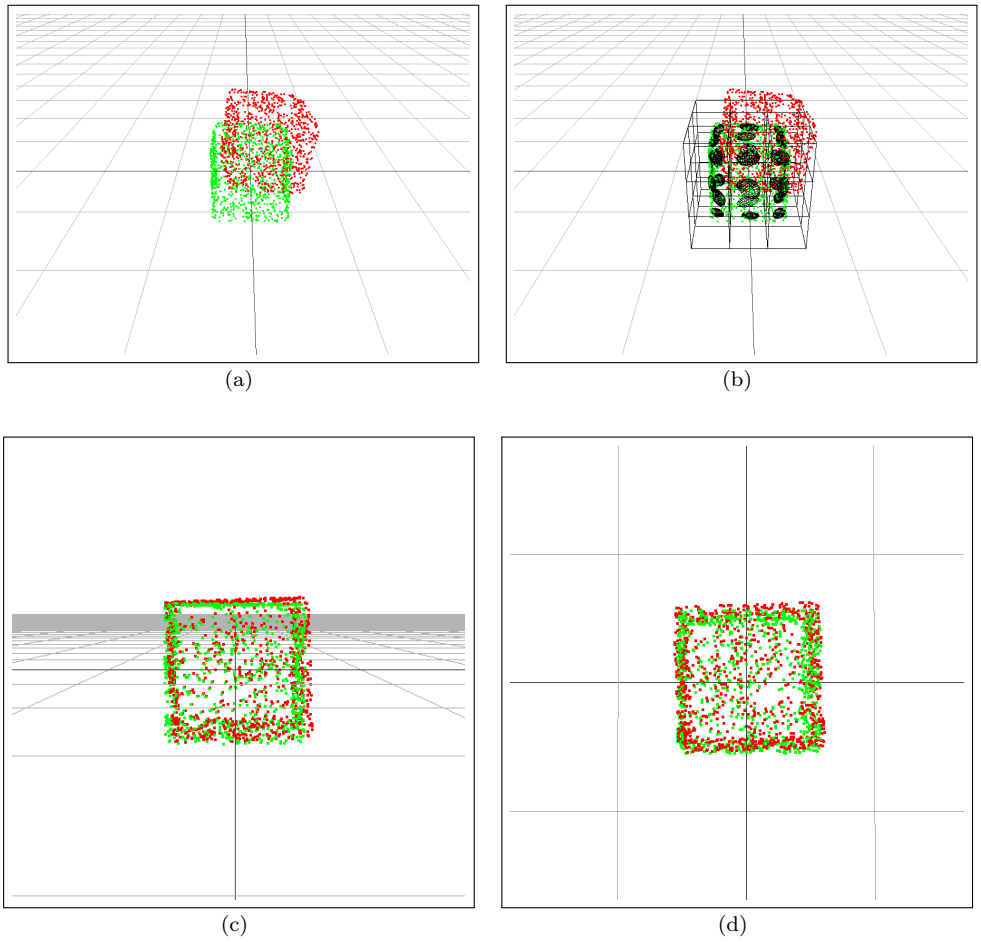


Figure 3.11: Matching 3D point sets using the normal distributions transform. Shown here are points uniformly distributed over the surface of a unit cube (same as in Figure 3.7) before registration (a) and the corresponding NDT-representation (b). The snapshots in (c) and (d) show the registration result in a sideview and a topview.

Magnusson and Duckett (2005) report that, by avoiding the computationally expensive nearest neighbor search, NDT-based range image registration outperforms ICP-based registration in terms of runtime while providing comparable results. For more details on the 3D generalization of NDT-based range image registration as well as the score function and its derivatives for a complete (seven-dimensional) axis-angle representation it is referred to (Magnusson, 2006) and (Magnusson

et al., 2007). In (Huhle et al., 2008) and (Andreasson, 2008) the 3D version of NDT-based range image registration is augmented with additional color information to construct colored 3D models of the robot's workspace by means of a 3D laser range finder and a monocular camera. In very recent work Magnusson et al. (2009a) extract local feature histograms from three-dimensional NDT-representations to detect loop closures in the robot's trajectory.

3.4.5 Alternative Subdivision Methods and Extensions

As already mentioned, the accuracy of the registration result and the level of detail kept by the NDT-representation primarily depend on the grid cell size. Whereas a larger cell size allows to register data sets where the initial alignment is poor, only smaller cell sizes allow for a fine-grained alignment. Although NDT-based range image registration is quite new compared to the Iterative Closest Point algorithm by Besl and McKay, a large variety of subdivision methods have been proposed that address this issue. In this context, using a constant cell size to construct a NDT-representation and subsequently register two point sets can be referred to as a *fixed subdivision* (cf. Biber and Strasser, 2003; Magnusson et al., 2007).

The most straightforward way of dealing with the discretization effects is to apply an *iterative subdivision* (Magnusson, 2006). Here, the NDT-based registration is repeated several times with decreasing cell sizes just like in the ICP-based registration using a decaying threshold for the point-to-point distance in correspondence pairs. Starting with a large cell size allows for a rough pre-alignment of the point sets whereas small cell sizes at the end of the iterative registration result in fine-tuning the rough estimate. Here, it has to be paid attention that the cell sizes are not getting too small resulting in the fact that the majority of cells no longer contain enough points for a valid normal distribution, i.e. that the number of points in a cell falls below $n_{\min}$.

Alternatively, one can use quad-trees (Biber and Strasser, 2003) and oct-trees (Magnusson et al., 2007) to recursively partition the space occupied by the model point set until no cell contains a covariance matrix that induces an almost spherical error ellipse. That is, the grid cell size is no longer constant and varies from region to region. Especially in the vicinity of corners, the space is subdivided until the error ellipses in adjacent cells are almost "flat" and oriented along the local surface. In the actual implementation, this extension augments the iterative subdivision. First, the NDT representation is built with a fixed cell size. Then it is checked, for every cell, whether the normal distribution well models the local surface. This is determined by means of the corresponding error ellipse. Each cell where the error ellipse does not well model the local surface, i.e. the error ellipse is almost spherical, is replaced by an oct-tree partitioning the space occupied by that cell. The recursive partitioning is continued until either no leave has a normal distribution with a spherical error ellipse or none of the cells contain enough points for a valid normal distribution. While this extension allows to keep a high detail in the NDT-representation, it also negatively affects the runtime of the registration algorithm as cell contents can no longer be directly looked up in the grid structure. Instead, the cell that is to be evaluated for a point $\mathbf{d}$ needs to be searched for in the corresponding oct-tree. That is, the NDT-representation with a uniform grid cell size turns into a grid where cells intersecting environmental structures form a forrest of oct-trees. Once the leave containing $\mathbf{d}$ is found, it is used to calculate the point's contribution to the score function and its derivatives like for any regular grid cell.

As the leaves of an oct-tree might represent only small portions of the environment, Magnusson (2006) proposed another subdivision method called *additive subdivision*. Here, the space occupied by the model set is partitioned using the aforementioned oct-tree subdivision, but instead of only evaluating the leave that contains a point $\mathbf{d}$, all leaves of the corresponding oct-tree are evaluated to determine $\mathbf{d}$'s contribution to the score function. Note that by this means, the runtime is not only increased by searching in oct-trees but also by evaluating more than one cell for a single point.

Regardless of which subdivision method is used, a point $\mathbf{d}$ can always fall into a cell that does not contain a valid normal distribution. This is especially the case when the data set D and model set M overlap only partially. The NDT-representation only spans over the region occupied by the model point set. Points in the data set that do not lie in this region are, by this means, ignored and do not contribute to the score function. Magnusson (2006) proposed two extensions to address this issue – namely *infinite outer bounds* and *linked cells*. The concept of infinite outer bounds is

quite simple: instead of completely ignoring points that lie outside of the NDT-representation of the model set, the nearest cell in the grid is used. In other words, the cells at the border of the grid extend to infinity and *catch* all points in the corresponding rows and columns of the grid. This is visualized in Figure 3.12 where dashed lines extend outer cells to infinity. Considering points outside of the portion of the environment that is covered by the model set is, however, often not desirable especially if data set and model set overlap only partially. Hence, it is suggestive to avoid the usage of infinite outer bounds when using range sensors that have a smaller apex angle such as 3D time-of-flight cameras like the SwissRanger or PMD cameras.

The 2D scans visualized in Figure 3.12 are poorly pre-aligned and the majority of data set falls into regions that are not occupied by the model set although they lie within the grid structure. To handle the problem of poor pre-alignment, Magnusson (2006) uses linked cells. Here, an additional grid structure, with the same grid cell size as the NDT-representation, is used that containts links to the closest occupied cells. If a point from the data set falls into a cell that does not contain a valid normal distribution, the NDT-based registration follows the link and evaluates the closest occupied cell. Without this fundamental extension it would be impossible to register the two scans shown in Figure 3.12. However, this extension drastically increases the algorithm's runtime. Especially if an iterative subdivision is used, the linked cell structure needs to be re-built for every cell size, i.e. every time the NDT-representation is constructed.

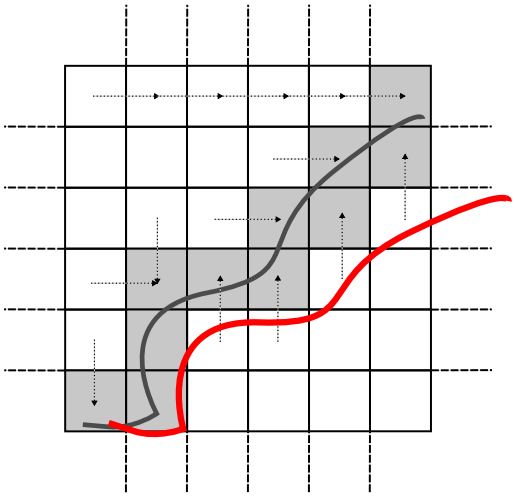


Figure 3.12: Visualization of infinite outer bounds and linked cells. Shown here is a 2D laser scan (dark gray curve). Occupied cells are shaded. When registering another range scan (thick red curve) only points in shaded cells are taken into account. Infinite outer bounds extend outer cells according to the dashed lines. Linked cells (dotted arrows, only partially visualized) allow to consider points in unshaded cells. Figure adapted from Magnusson (2006).

Assuming that the initial alignment is not too poor, the problem of not considering points in unoccupied regions can be addressed by additionally evaluating adjacent cells instead of only that very cell where a data point falls into. In the situation visualized in Figure 3.12 this allows to correctly register data set and model set without the need of additionally constructing the linked cell structure. Still, even this simple extension considerably affects the runtime of the registration. In the two-dimensional case the runtime increases by a factor of 9 as eight additional cells have to be evaluated for every point. In the three-dimensional case with 26 neighbors the runtime is increased by a factor of 27. Furthermore, evaluating multiple cells for the same point can also negatively affect the registration by adding contradictory contributions to gradient and Hessian. In very recent work, Magnusson et al. (2009b) also used neighbor evaluation and showed that evaluating the eight cells that are closest to a point can considerably improve the registration result. They additionally use a trilinear interpolation weight function that has a smoothing effect on the individual contributions of the eight evaluated cells.

However, appropriately handling situations like the one visualized in Figure 3.12 or those with

even worse initial alignments while preserving the low runtime complexity of NDT-based scan matching is a matter of future work.

3.5 Incremental ICP-Based Matching and Sparse Point Maps

This chapter will describe the implemented approach to SLAM based on incremental range image registration using the ICP algorithms presented in Chapter 3.3 and sparse point maps briefly introduced in (Holz, 2007) and (Holz et al., 2008). The next section will focus on the concept of and the idea behind sparse point maps. The remainder of the chapter will address their construction by means of incrementally matching acquired range scans.

3.5.1 Sparse Point Maps

The idea of incremental range image registration (see Chapter 3.1.3) is to incrementally construct a so-called *metaview* or *metascan* in the context of scan-matching. Herefore, the first acquired laser scan D_0 forms the model set M. All subsequent scans $D_i, i > 0$ are matched against M to determine the location (in M) where D_i belongs to and use the corresponding homogeneous transformation matrix $\mathbf{T}_i$ to transform all points $\mathbf{d}_{i,j} \in D_i, j = 1, \ldots, N_{D_i}$ yielding the transformed point set $\check{D}_i = \{\check{\mathbf{d}}_{i,j} | \check{\mathbf{d}}_{i,j} = \mathbf{R}\mathbf{d}_{i,j} + \mathbf{t}\}$. The model M is then updated to account for possibly new information in D_i by adding all points $\check{\mathbf{d}}_{i,j}$ to M, i.e. after matching range image D_i, the model set M_{i-1} from the last registration is updated to

$$M_i = M_{i-1} \cup \{\check{\mathbf{d}}_{i,j} \mid \check{\mathbf{d}}_{i,j} \in \check{D}_i\}. \tag{3.61}$$

By this means, a model M_N constructed by incrementally registering N range images, contains all points measured in the environment, i.e.

$$M_N = \bigcup_{i=[1,N]} \{\check{\mathbf{d}}_{i,j} \mid \check{\mathbf{d}}_{i,j} \in \check{D}_i\} \tag{3.62}$$

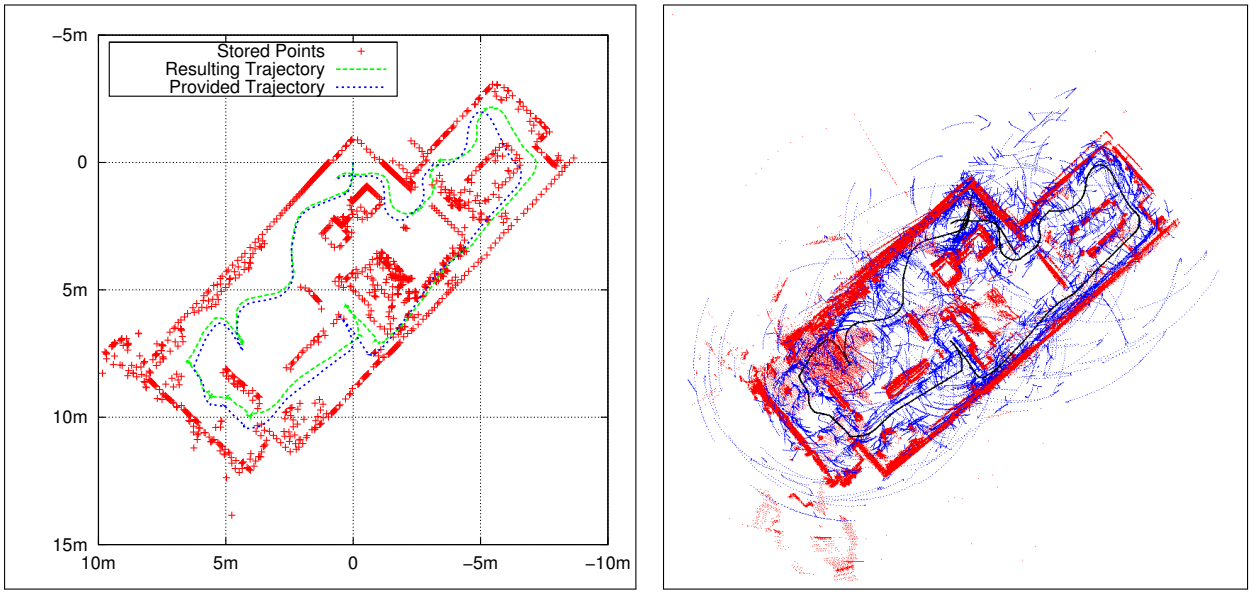
The main problem of this approach is its scalability with respect to the size of the environment and the number of range images taken. To fully cover a large environment, a lot of range images might be needed. When registering and completely adding all acquired range images, the model set M can get quite large, e.g. several million points for 3D scans taken in a larger outdoor environment (Nüchter et al., 2007b; Wulf et al., 2008). Furthermore, when acquiring range images in already modeled parts of the environment, a lot of points can be added to M without providing any new information about the environmental structures. Again, especially when matching 3D laser scans that typically contain 10 000 to 100 000 points, this can have disastrous effects regarding the memory consumption of the constructed model.

The key idea of sparse point maps is to avoid duplicates by not adding a point that corresponds to the same point in the physical environment as a point already stored in the model. Hereby, correspondence is defined just like in the ICP algorithm, i.e. a point $\mathbf{d}_{i,j} \in D_i$ is not added to M_{i-1} if the point-to-point distance of the transformed point $\check{\mathbf{d}}_{i,j}$ to its closest point $\mathbf{m}_{i-1,k} \in M_{i-1}$ is smaller than a minimum allowable distance $D_{\min}$.

$$M_i = M_{i-1} \cup \{\check{\mathbf{d}}_{i,j} \mid \check{\mathbf{d}}_{i,j} \in \check{D}_i, \nexists \mathbf{m}_{i-1,k} \in M_{i-1} : \|\check{\mathbf{d}}_{i,j} - \mathbf{m}_{i-1,k}\| < D_{\min}\} \tag{3.63}$$

The threshold $D_{\min}$ spans regions in the model, spheres with radius $D_{\min}$, in which the number of points is limited to 1. That is, $D_{\min}$ provides an upper bound on the point density in a sparse point map M. Choosing a value of $D_{\min}$ according to the accuracy of the used range sensor for acquiring the range images will exactly neglect duplicate storage of one and the same point assuming correct alignment of range images. Due to measurement noise, a value smaller than the sensor's accuracy might still lead to duplicate entries. Choosing, however, a larger value allows to reduce the number of points stored in the map. Altough some details of the environmental structures might not be modeled, a map constructed by this means still provides a coarse-grained model of the environment.

An example for a sparse point map is shown in Figure 3.13.a. The map models the example scenario of (Zivkovic et al., 2007) and is the result of applying the SLAM algorithm presented in the following section. By using a distance threshold of $D_{\min} = 20\,\text{cm}$ the sparse point map, constructed from 910 2D laser scans each containing 361 points, stores only 1303 points, that is only 0.39 % of the $910 \times 361 = 328\,510$ points in the data set.



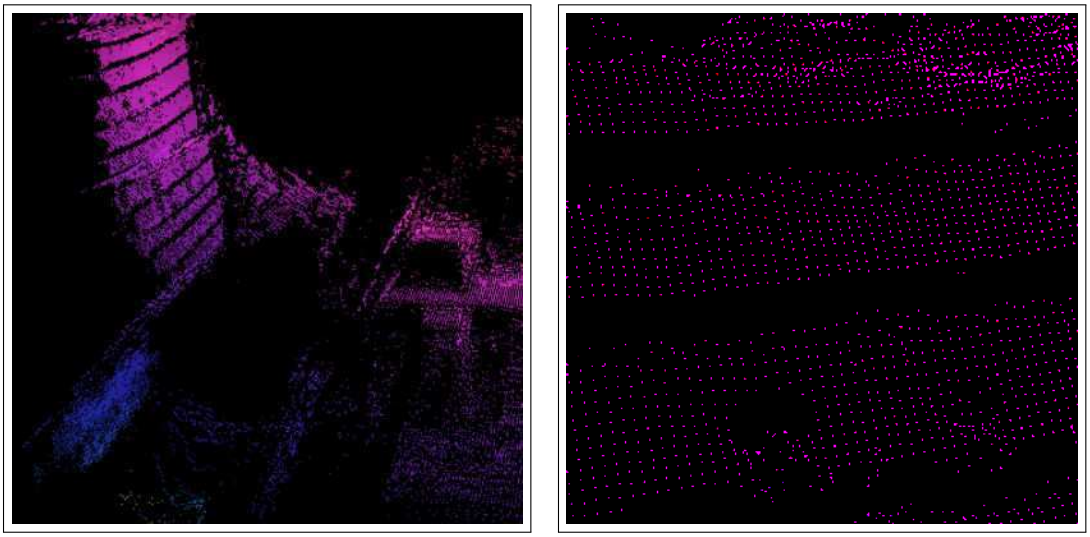
(a) (b)

Figure 3.13: Sparse point map (a) of an example scenario. The map contains only 1303 points to represent the complete environment, whereas the map (b) provided with data set (Zivkovic et al., 2007) contains all 328 510 points.

Note, that at some modeled structures in the map in Figure 3.13.a the point density is higher than that at others. This is caused by the fact that $D_{\min}$ is only taken into account when determining correspondences between points D_i and M_{i-1}, but not to M_i. A point $\mathbf{d}_{i,j}$ already being added to M_i is not checked for correspondence with a point $\mathbf{d}_{i,k}, k > j$ that is to be checked later. Hence, although the point-to-point distance $\|\mathbf{d}_{i,j} - \mathbf{d}_{i,k}\|$ is smaller than $D_{\min}$, both points $\mathbf{d}_{i,j}$ and $\mathbf{d}_{i,k}$ might be added to M_i. An example for this unwanted behavior is shown in Figure 3.14. Shown is a sparse point map built from three-dimensional range images acquired in a typical indoor environment. The wall in the upper left of Figure 3.14.a does not show a uniform point distribution but seems to be composed of several fractions. The borders between these fractions result from partial overlap between the already modeled surface in M_{i-1} and range image D_i. Those points corresponding to already modeled points are neglected leading to empty regions of width $D_{\min}$, whereas not corresponding points, i.e. all points with a distance larger than $D_{\min}$ to the already modeled surface are added to M with the same point density as in the range image D_i.

There are two possibilities to overcome this unwanted behavior. The first is to also take those points from D_i into account that have been already added to M instead of only checking correspondences with M_{i-1}. This, however, would require that the points in D_i need to be ordered with respect to their distance to the nearest border of the already modeled part of the surface they belong to. The major disadvantage of this approach is that ordering the points and especially determining the distances for the ordering is computationally expensive as, first, all borders and edges in M_{i-1} have to be determined, second, for all points in D_i the closest border as well as the point's distance to it have to be determined and, third, D_i has to be sorted. Other points in D_i can also be taken into account when conducting the nearest neighbor search in M_i by either re-building the *kd*-tree after every addition of a point and by conducting brute-force search.

The second possibly is to use the same threshold $D_{\min}$ for reducing the point density in D_i, e.g. to replace a set of points, whose point-to-point distances between each other fall below $D_{\min}$, by their centroid, just like in the feature selection mechanism for the ICP algorithm in (Nüchter et al., 2003b).



(a) (b)

Figure 3.14: Topview on a (three-dimensional) sparse point (a) and a detail view (b) of the upper left wall showing that only checking correspondences with the model neglecting just added points from the same range image results in the storage of points not meeting the constraint of a minimum allowable point-to-point distance in the model.

3.5.2 Constructing Sparse Point Maps

The construction of sparse point maps is straightforward given a sequence of range images $D_i, i = 1, \ldots, N$ and a sequence of transformations $\mathbf{T}_i, i = 1, \ldots, N$ determining the views where the range images have been taken. An update of the map M_{i-1} using range image D_i consists of the following processing steps:

1. **Reducing the Point Density**: Reduce the point density in D_i by replacing clusters of points, whose point-to-point distance between each other do not exceed the minimum allowable point distance $D_{\min}$ of the sparse point map, by their centroid.
2. **Transforming the Image**: Transform the measured points in range image D_i into the map's (global) coordinate frame using $\mathbf{T}_i$.
3. **Correspondence Search**: Determine the closest point in M_{i-1} for every point $\mathbf{d}_{i,j} \in D_i$. Add the correspondence pair $(\mathbf{d}_{i,j}, \mathbf{m}_{i-1,k})$ to the set of found correspondences if their point-to-point distance is smaller than $D_{\min}$ and add those points in D_i that did not find a corresponding point in M_{i-1} to the set of non-corresponding points $D_i^\star$.
4. **Map Update**: Add all points $\mathbf{d}_{i,j}^\star \in D_i^\star$ to M_{i-1} resulting in the updated map M_i.

The actual correspondence search, after reducing the point density in the range image, is implemented using *kd*-tree search just like in the ICP algorithm. In fact, the idea behind sparse point arose from the implementation of the ICP algorithm: After successful matching, a range image D_i is correctly aligned with the so far built model M_{i-1} and according to Eq. (3.61) all points

$\mathbf{d}_{i,j} \in D_i$ could be used to update the model M_i. However, from the last iteration step, there is still the set of K correspondence pairs $\{(\check{\mathbf{d}}'_k, \check{\mathbf{m}}'_k) \mid k = 1, \ldots, K\}$. Now building the subset $\check{D}^\star \subseteq \check{D}$ of points that did not have a corresponding point in M_{i-1} and adding only points $\check{\mathbf{d}}^\star \in \check{D}^\star$ to M_i results in the behavior of Eq. (3.63) where the distance threshold $D_{\max}$ used in the last iteration step of ICP algorithm becomes the minimum allowable point-to-point distance $D_{\min}$ of the sparse point map.

$$M_i = M_{i-1} \cup \{\check{\mathbf{d}}^\star_l \mid \check{\mathbf{d}}^\star_l \in \check{D}^\star\} \tag{3.64}$$

However, the correspondences $(\check{\mathbf{d}}'_k, \check{\mathbf{m}}'_k)$ between points in D_i and M_{i-1} only result from the ICP algorithm when following the procedure of incremental registration, but not for that of pairwise registration since there it is the last range image D_{i-1} forming the model set M in the ICP. For allowing the construction of sparse point maps independent of the used matching procedure, it needs to be decoupled from the ICP. After determining the correct transformation $\mathbf{T}_i$ mapping D_i onto the so far built model M_{i-1}, an additional correspondence search is carried out to determine the set of non-corresponding points $\check{D}^\star$. This decoupling also decouples the distance thresholds $D_{\max}$ used in the ICP algorithm and $D_{\min}$ for the sparse point map. Hence, the map's maximum point density is no longer dependend on the parametrization of the ICP algorithm. In cases where the *kd*-tree partitioning M_{i-1} can not be directly taken from the ICP, it has to be constructed within the update procedure of the map. Building the *kd*-tree is, however, only necessary if the sparse point map has been changed since the last update, e.g. due to adding and removing points or transformations of the points' coordinates.

Note, that in the update procedure described above points are only added to the model. Up to now, there is no automated procedure for moving or completely removing points from a sparse point map that are no longer valid. Points measured on the surface of an object that has been moved, for example, also need to be moved inside the point map. If this object has been completely removed from the environment, then the corresponding points in the map also need to be removed. Altough the functionality of moving and deleting points has already been implemented, it is a matter of future work to develop an algorithm for determining whether a point in the map is still valid or not and how the map needs to be corrected.

Currently there is only a heuristic validity check that assumes that a probabilistic reflection grid (see Chapter 3.2.2) map is constructed based on the registration results. In some fixed interval, it is then checked for every point in M, whether the reflection probability of the corresponding region is larger than some threshold, e.g. ≥ 0.5. Points, where the reflection probability is lower than the threshold, are removed from the point map. By this means, phantom effects, e.g. caused by people moving by, are avoided. Since the runtime complexity of the update procedure for grid maps is linear in the number of laser beams and the maximum measureable distance and not dependend on the grid map size, the additional update can easily be carried out after registration without noticeably influencing the overall runtime of the SLAM algorithm. For updating reflection models with laser range scans containing 181 range readings with a maximum distance of 30 m, runtimes in the range of 0.5 ms to 2 ms have been measured. The validity check itself is linear in the number of points stored in the map, since the reflection maps allow access to the reflection probability of the corresponding region in constant time. This validity check as well as directly performing the presented incremental registration procedure on grid maps is addressed in Chapter 3.6. For this reason, the experiments presented in this chapter, have been carried out without the additional validity check.

3.5.3 Pairwise and Incremental Matching

The update procedure of sparse point maps described above necessitates the knowledge about the robot's positions and orientations along the trajectory where the range images D_i have been taken, i.e. for each range image D_i the transformation matrix $\mathbf{T}_i$ correctly aligning the points in D_i to already stored points in M_{i-1} has to be given. Such a problem is referred to as *mapping with known poses* in (Stachniss, 2006). Determing the poses and transformations $\mathbf{T}_i$ is, however, a problem of SLAM that is here to be addressed using pairwise and incremental registration of range images by means of the ICP algorithm.

Think of a robot starting to explore a completely unknown environment. Without loss of generality and regardless of the used registration procedure, we can assume (or define) that the robot starts in the origin of the map's coordinate frame, i.e. the transformation $\mathbf{T}_0$ representing the robot's pose at time $i = 0$ is the Identity matrix:

$$\mathbf{T}_0 = \mathbf{I}(4,4).$$

Hence, the first range image D_0 taken at time $i = 0$ can be directly be taken to form the initial environment model M_0. The only processing step necessary here, is reducing the range image's point density according to step 1 in the update procedure above described.

For the registration of all subsequent range images $D_i, i > 0$, the transformation $\mathbf{T}_i$ has to be determined according to some registration procedure. Here, we will focus only on pairwise and incremental registration using the ICP algorithm. Incremental registration using NDT-based scanmatching is addressed in Chapter 3.7.

Incremental Registration

For incremental registration the so far built point map M_{i-1} (M_0 for the first registration) forms the data set M for the ICP algorithm. The registration of a range image D_i is then carried out in the coordinate frame of the map, i.e. all transformations $\mathbf{T}_i$ and models M_i are referred to the frame $\{M\} = \{M_0\}$. Although the difference between views of two successive range images might be small, an initial estimate of $\mathbf{T}_i$ needs to be known as all points $\mathbf{d}_{i,j} \in D_i$ are referred to the sensor's local coordinate frame $\{D_i\}$. However, as the registration is applied online and the range images can be assumed to arrive in the order of acquisition, the transformation $\mathbf{T}_{i-1}$ from registering the last range image D_{i-1} already provides a good initial estimate of $\mathbf{T}_i$. In cases where the difference between the poses $\mathcal{P}_{i-1}$ and $\mathcal{P}_i$ where D_{i-1} and D_i have been acquired is large, an additional refinement of the initial guess for $\mathbf{T}_i$, e.g. by means of odometry, might be necessary for convergence to the global minimum in the ICP algorithm. In the registration of 2D range images, this additional refinement is straightforward when simply storing the odometry estimates $\mathcal{P}_{i,\text{Odo}}$ as reference poses. That is, for the registration of range image D_i the current odometric pose estimate $\mathcal{P}_{i,\text{Odo}}$ and the reference estimate $\mathcal{P}_{i-1,\text{Odo}}$, for the pose where the last registered range image D_{i-1} has been taken, are used to calculate the estimated pose shift $\Delta\mathcal{P}_{i,\text{Odo}}$.

$$\mathcal{P}_{i,\text{Odo}} = \mathcal{P}_{i-1,\text{Odo}} + \Delta\mathcal{P}_{i,\text{Odo}} \tag{3.65}$$

Using matrix notation to express the relation of coordinate frames (cf. Craig, 1989, Chapter 2), Eq. (3.65) can be expressed as:

$$\begin{Bmatrix}\{M\}\\\{D_i\}\end{Bmatrix}\mathbf{T}_{\text{Odo}} = \begin{Bmatrix}\{M\}\\\{D_{i-1}\}\end{Bmatrix}\mathbf{T}_{\text{Odo}} \begin{Bmatrix}\{D_{i-1}\}\\\{D_i\}\end{Bmatrix}\mathbf{T}_{\text{Odo}} \tag{3.66}$$

$$\begin{Bmatrix}\{D_{i-1}\}\\\{D_i\}\end{Bmatrix}\mathbf{T}_{\text{Odo}} = \begin{Bmatrix}\{M\}\\\{D_{i-1}\}\end{Bmatrix}\mathbf{T}^{-1}_{\text{Odo}} \begin{Bmatrix}\{M\}\\\{D_i\}\end{Bmatrix}\mathbf{T}_{\text{Odo}} \tag{3.67}$$

where $\begin{Bmatrix}\{D_{i-1}\}\\\{D_i\}\end{Bmatrix}\mathbf{T}_{\text{Odo}}$ is the transformation matrix corresponding to the wanted pose shift $\Delta\mathcal{P}_{i,\text{Odo}}$ and $\begin{Bmatrix}\{M\}\\\{D_i\}\end{Bmatrix}\mathbf{T}_{\text{Odo}}$ and $\begin{Bmatrix}\{M\}\\\{D_{i-1}\}\end{Bmatrix}\mathbf{T}_{\text{Odo}}$ are derived from $\mathcal{P}_{i,\text{Odo}}$ and $\mathcal{P}_{i-1,\text{Odo}}$ respectively. For simplicity, we will refer to $\begin{Bmatrix}\{D_{i-1}\}\\\{D_i\}\end{Bmatrix}\mathbf{T}_{\text{Odo}}$ as $\Delta\mathbf{T}_{\text{Odo}}$ in the following. An refined estimate $\hat{\mathbf{T}}_i$ for the transformation $\mathbf{T}_i$ can than be calculated as:

$$\hat{\mathbf{T}}_i = \mathbf{T}_{i-1}\ \Delta\mathbf{T}_{\text{Odo}} \tag{3.68}$$

For the registration of 3D range images, odometry estimates have to be extrapolated to 3D space (Nüchter et al., 2007b).

The newly acquired range image D_i is then given to the ICP algorithm together with the so far built sparse point map M_{i-1} and the estimate $\hat{\mathbf{T}}_i$ to correctly align D_i to M_{i-1} and to refine the transformation $\hat{\mathbf{T}}_i$. An alternative way is to first transform D_i with respect to $\hat{\mathbf{T}}_i$ and to use the transformed data set in the ICP algorithm neglecting the initial transformation estimation for the ICP, i.e. applying $\hat{\mathbf{T}}_i$ to D_i and the Identity matrix $I(4,4)$ to the ICP algorithm. By this means, the ICP only calculates a refinement matrix $\Delta\mathbf{T}_{\text{ICP}}$ whose application can be made

dependend e.g. on the number of corresponding points or the remaining mapping error $E(\Delta\mathbf{T}_{\text{ICP}})$. Furthermore, mapping by means of pure odometry (see Figure 3.17) is inherently implemented by never applying $\Delta\mathbf{T}_{\text{ICP}}$. For the purpose of incremental registration, the transformation matrix $\mathbf{T}_i$ correctly aligning D_i to M_{i-1} results from correcting the initial estimate $\hat{\mathbf{T}}_i$ by means of the refinement matrix $\Delta\mathbf{T}_{\text{ICP}}$ determined by the ICP algorithm.

$$\mathbf{T}_i = \Delta\mathbf{T}_{\text{ICP}}\ \hat{\mathbf{T}}_i = \Delta\mathbf{T}_{\text{ICP}}\ \mathbf{T}_{i-1}\ \Delta\mathbf{T}_{\text{Odo}} \tag{3.69}$$

The overall procedure of incremental registration based on the ICP algorithm including the update of the sparse point map used as the *metascan*, can be formulated as follows:

1. **Estimating the Initial Transformation**: Based on the transformation $\mathbf{T}_{i-1}$ resulting from registering range image D_{i-1}, the current pose estimate $\mathcal{P}_{i,\text{Odo}}$ and the reference pose estimate $\mathcal{P}_{i-1,\text{Odo}}$, calculate an initial estimation $\hat{\mathbf{T}}_i$ of the transformation matrix $\mathbf{T}_i$.
$$\hat{\mathbf{T}}_i = \mathbf{T}_{i-1}\ \Delta\mathbf{T}_{\text{Odo}}$$

2. **Transforming the Image**: Transform the measured points in range image D_i into the map's (global) coordinate frame using $\hat{\mathbf{T}}_i$.
$${}^{\{M\}}D_i = \{\hat{\mathbf{T}}_i\ \mathbf{d}_{i,j} \mid \mathbf{d}_{i,j} \in D_i\}$$

3. **Registration using ICP**: Determine the refinement matrix $\Delta\mathbf{T}_{\text{ICP}}$ and the remaining mapping error $E(\Delta\mathbf{T}_{\text{ICP}})$ by calling the ICP algorithm with map M_{i-1} being the model set, D_i the data set and the Identity matrix $\mathbf{I}(4,4)$ the initial transformation estimation.
$$(\Delta\mathbf{T}_{\text{ICP}}, E(\Delta\mathbf{T}_{\text{ICP}})) = ICP\,(M_{i-1}, D_i, \mathbf{I}(4,4))$$

4. **Updating Map and Pose**: Calculate the transformation $\mathbf{T}_i$ based on the initial estimate $\hat{\mathbf{T}}_i$ and calculated refinement transformation
$$\mathbf{T}_i = \Delta\mathbf{T}_{\text{ICP}}\ \hat{\mathbf{T}}_i$$
and call the map's update procedure with D_i and $\mathbf{T}_i$.

An example showing the result of applying the above described procedure based on the ICP algorithm and the concept of sparse point maps is shown in Figure 3.15. Here a sequence of laser scans, taken during a run of a mobile robot through an office environment at the University of Zaragoza, has been incrementally registered to construct the shown point map and determine the trajectory of the robot resulting from the sequence of transformations in step 4. Here, no odometry estimations have been used, i.e. the initial estimation of transformation $\hat{\mathbf{T}}_i$ aligning D_i to M_{i-1} is solely composed of the last transformation $\mathbf{T}_{i-1}$

$$\hat{\mathbf{T}}_i = \mathbf{T}_{i-1}$$

and every single laser scan is matched and used to update the model. Thereby, the ICP algorithm had to determine the poseshift between $\mathcal{P}_{i,\text{Odo}}$ and $\mathcal{P}_{i-1,\text{Odo}}$. The data set (Minguez, 2005) consists of 778 laser scans with 361 points per scan. The constructed sparse point contains 2802 points with a minimum allowable point-to-point distance of $D_{\min} = 20\,\text{cm}$. The robot starts in the room at the bottom right, traverses along a corridor and takes scans in neighboring rooms before returning to the start pose. The constructed map is globally consistent, i.e. by incrementally matching newly acquired range scans against the so far built point map, the robot correctly localizes itself during the complete run, and all range images are registered at the correct location. When the robot re-enters an already modeled part of the environment, e.g. the room in which exploration has started and ended, the robot correctly localizes itself in the point map. When no new environmental

structures are observed, no points from newly acquired range images are added to the map as all points in the map are used for the correspondence search in the ICP algorithm. Exactly this issue makes incremental registration more robust than pairwise registration as will be shown in the following.

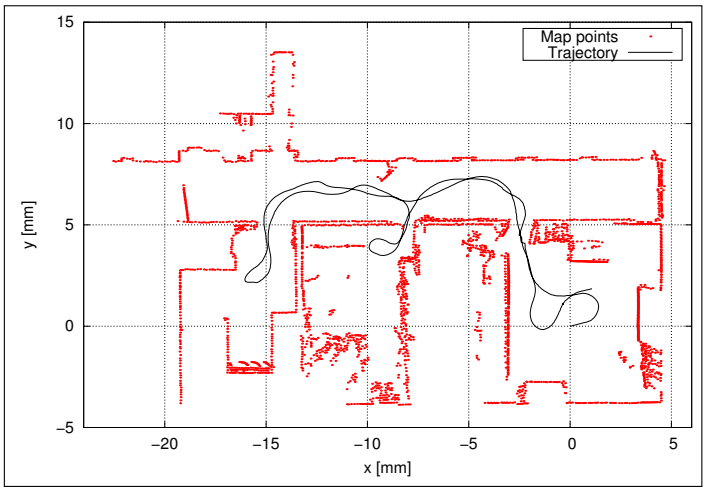


Figure 3.15: Example of incremental range image registration. Shown is the result of applying the incremental registration using the ICP algorithm and sparse point maps to the dataset of Minguez (2005).

Pairwise Registration

In the case of pairwise registration, a newly acquired range scan is not matched against the so far built model but against the last range scan – the *reference scan*. Here, keeping track of the robot's pose in the map's global coordinate frame (represented by the sequence of transformations T_i in incremental registration) is decoupled from the actual registration procedure. Instead of refining an estimate for the global transformation ${}^{\{M\}}_{\{D_i\}}\mathbf{T}$, the ICP algorithm is used to refine an estimate of the pose shift, i.e. the location of $\{D_i\}$ relative to $\{D_{i-1}\}$ represented by ${}^{\{D_{i-1}\}}_{\{D_i\}}\mathbf{T}$. By this means, the robot's pose in the environment model is incrementally updated with every new pose shift estimation:

$$ {}^{\{M\}}_{\{D_N\}}\mathbf{T} = {}^{\{M\}}_{\{D_0\}}\mathbf{T}\ {}^{\{D_0\}}_{\{D_1\}}\mathbf{T}\ \cdots\ {}^{\{D_{N-1}\}}_{\{D_N\}}\mathbf{T} \tag{3.70} $$

This procedure is quite similar to that of incremental registration, where $\mathbf{T}_{i-1}$ is used as an additional estimate to calculate a refinement leading to $\mathbf{T}_i$. Although, calculating this refinement is comparable to determining ${}^{\{D_{N-1}\}}_{\{D_N\}}\mathbf{T}$, the primary difference between pairwise and incremental matching is that here only the points in D_{i-1} are used to register D_i. By this means, even small registration errors accumulate what can lead to large localization errors and inconsistencies in the constructed map just like in odometric pose estimations as all information about the environment, contained in the map but not in the reference scan, is not taken into account in the matching process. When the robot re-enters a previously modeled part of the environment, for example, this information is neglected and the robot solely localizes itself based on the current and the reference range scan. Infact, the robot does not even need to construct a map for localizing itself.

Referring to ${}^{\{D_{N-1}\}}_{\{D_N\}}\mathbf{T}$ as $\Delta\mathbf{T}_i$ and $\Delta\hat{\mathbf{T}}_i$ as its initial estimate, the overall algorithm for pairwise registration using the ICP algorithm and sparse point maps can be formulated as follows. Note that here map and pose update are, respectively, not needed and completely decoupled from the actual matching process.

1. **Estimating the Initial Transformation**: Based on the current pose estimate $\mathcal{P}_{i,\text{Odo}}$ and the reference pose estimate $\mathcal{P}_{i-1,\text{Odo}}$, calculate an initial estimate $\Delta\hat{\mathbf{T}}_i$ of the transformation matrix $\Delta\mathbf{T}_i$.

$$\Delta\hat{\mathbf{T}}_i = \Delta\mathbf{T}_{i,\text{Odo}}$$

2. **Registration using ICP**: Determine $\Delta\mathbf{T}_{i,\text{ICP}}$ and the remaining mapping error $E(\Delta\mathbf{T}_{i,\text{ICP}})$ by calling the ICP algorithm with the reference scan D_{i-1} being the model set, D_i the data set and $\Delta\hat{\mathbf{T}}_i$ as the initial transformation estimation.

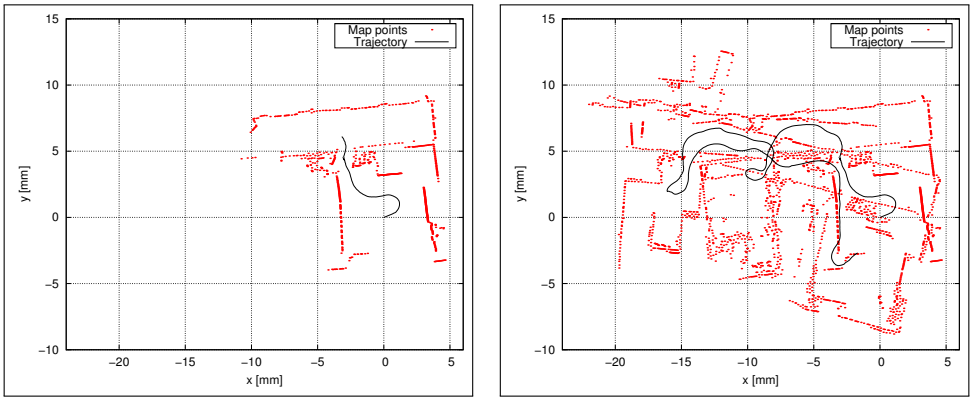
$$(\Delta\mathbf{T}_{i,\text{ICP}}, E(\Delta\mathbf{T}_{i,\text{ICP}})) = ICP\left(D_{i-1}, D_i, \Delta\hat{\mathbf{T}}_i\right)$$

3. **Updating Map and Pose**: Calculate the transformation $\mathbf{T}_i$ based on the initial estimate $\hat{\mathbf{T}}_i$ and calculated refinement transformation

$$\mathbf{T}_i = \mathbf{T}_{i-1}\ \Delta\mathbf{T}_{i,\text{ICP}}$$

and call the map's update procedure with D_i and $\mathbf{T}_i$.

The result of applying this algorithm to the same data set (Minguez, 2005) is shown in Figure 3.16 with the aforementioned inconsistencies due to accumulated localization errors.

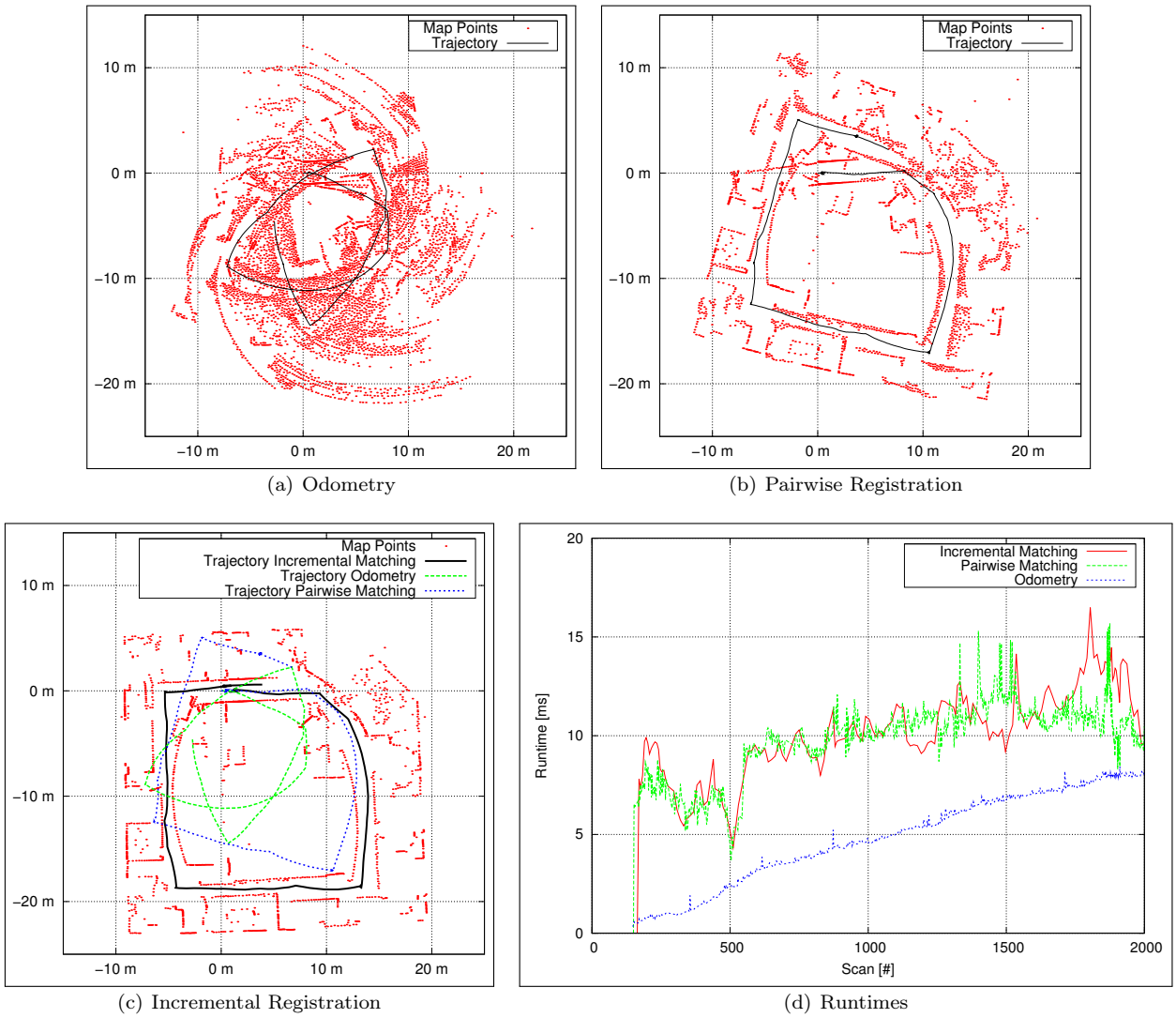


(a) (b)

Figure 3.16: Example of pairwise range image registration. Shown is the result of applying the pairwise registration using the ICP algorithm and sparse point maps to the dataset of (Minguez, 2005). Due to only very small registration errors the map in (a) is consistent. As the distance traversed by the robot gets larger, accumulated registration errors lead to inconsistencies in the final map (b).

Pairwise vs. Incremental Matching

It is clear that both pairwise and incremental registration procedures have their right to exit as both show particular advantages with respect to the application. For tracking a robot's pose only over short distances, for example, pairwise matching might be preferable as a map does not need to be constructed at all leading to lower runtimes of the matching algorithm and accumulated registration errors are negligible over short distances. That pairwise registration does not show its advantage regarding speed concerns in the runtime comparison of Figure 3.17.d, lies in the fact that step 3 of the pairwise registration procedure is completely carried out and a map is constructed for the sake of a qualitative comparison to incremental registration and pure odometry in Figure 3.17.c.



(a) Odometry

(b) Pairwise Registration

(c) Incremental Registration

(d) Runtimes

Figure 3.17: Pairwise vs. Incremental Matching. Shown are maps and trajectories constructed based on data from the first loop through the Intel Research Lab (first 2000 scans) using (a) pure odometry, (b) pairwise matching and (c) incremental matching. The corresponding runtimes are plottet in (d).

Figure 3.17 shows the results of applying the procedures for pairwise and incremental registration as well as for constructing sparse point maps on pure odometry to a data set by Dirk Hähnel recorded at the Intel Research Lab in Seattle (Hähnel, 2005). The map constructed by means of pure odometry (Figure 3.17.a) shows that the odometry estimations used for pairwise and incremental registration are poor. True trajectory and environment structure can not be derived from the map. The map constructed by means of pairwise registration (Figure 3.17.b) already gives a clue on how the path of the robot and the sensed environmental structures look like. However, due to accumulated registration the robot does not end up in the same position where it started, i.e. the loop was not closed. Here, incremental registration (Figure 3.17.c) shows its predominance as the robot correctly localizes itself throughout the complete loop by considering all points stored in the map in the registration of a newly acquired range image. The runtime for mapping using pure odometry in Figure 3.17.d is that for constructing and searching in the *kd*-tree spatially partitioning the map. Note that all runtimes are only measured for a single run through the first 2000 laser scans of the data set, i.e. the measured runtime for each laser scan is influenced, for example, by other running threads. Still they provide an impression on expected runtimes. It is also to note, that pairwise registration is not faster than incremental registration due to the correspondence search in the map update procedure.

Odometry Thresholding

Although incremental registration is able to correct minor localization and alignment errors from preceding registrations by taking all points in the so far built model into account, larger errors can still have disastrous effects like for instance having several parts of the environment being modeled locally consistent but the overall structure and relative locations of the locally consistent parts to each other inconsistent. An example for such an inconsistency is given in the remainder of this chapter where several results and open problems will be discussed. Such inconsistencies can only be corrected using e.g. loop detection mechanisms and algorithms for global relaxation of the constructed model.

What can, however, be done without any further algorithm being involved, is to simply reduce the possible number of misregistrations by not matching every single laser scan but only a certain fraction of acquired scans. Instead of some random selection, the most reasonable way is to make the decision of whether or not a laser range scan should be used for registration dependend on the pose shift measured by means of odometry, here referred to as *odometry thresholding*. Figure 3.18 shows two maps constructed from the same quite noisy data set to visualize this idea.

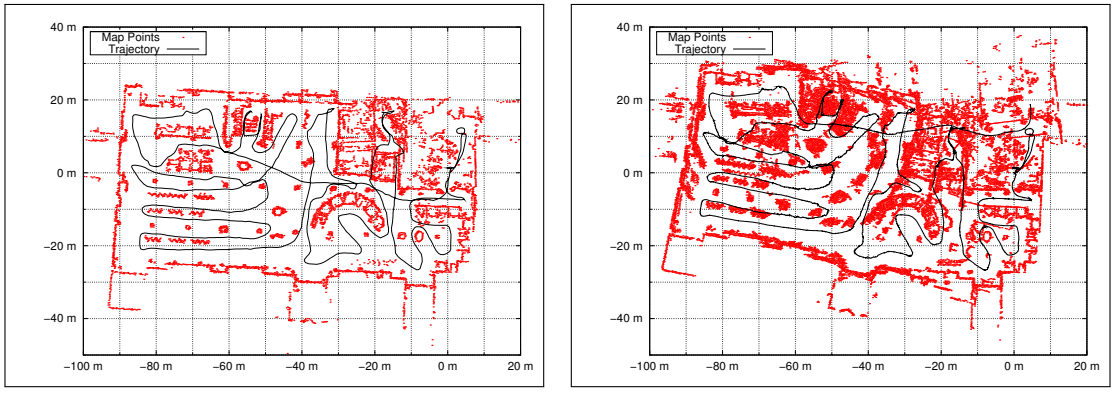


Figure 3.18: Sparse point map and trajectory of a robot moving through the Acapulco Convention Center in Acapulco, Mexico. Map (a) was constructed using odometry and only matching laser scans after travelling a distance of more than 50 cm or turning more than 25°. Map (b) was constructed without odometry and matching every single (noisy) laser scan. Map (a) contains 6621 points and is globally consistent, map (b) contains 16 735 points and shows inconsistencies. The data set was recorded by Nicholas Roy.

Figure 3.18.b shows a map of the Acapulco Convention Center using incremental registration using the ICP algorithm and sparse point maps. Here, odometry estimations are not taken into account and the initial transformation $\hat{\mathbf{T}}_i$ for registering every single range scan is solely estimated from the last registration $\mathbf{T}_{i-1}$. The range measurements in this data set are quite noisy leading to several minor misalignments. As the map is only extended in the course of registration but not corrected with respect to these minor errors, the resulting map is blurred and deformed.

Taking, however, odometry estimations into account in calculating the initial transformation estimation, i.e. $\hat{\mathbf{T}}_i = \mathbf{T}_{i-1}\ \Delta\mathbf{T}_{\text{Odo}}$ according to Eq. (3.68), and additionally matching only those range scans where the robot has at least traversed a distance of more than 50 cm or turned more than 25° since the last registration, allows for constructing an accurate and consistent model of the convention center. Note, that this thresholding can also be applied to pairwise registration with similar effects. The registration errors still accumulate but, as the number of possible misalignments is reduced, the distance that the robot can traverse until the accumulated error causes major inconsistencies gets larger. In fact, the aforementioned thresholding has been applied to the registration procedures in the comparison of Figure 3.17.

3.5.4 Results and Open Problems

This section will provide some qualitative results on applying the presented SLAM algorithm based on ICP-based scan-matching an sparse point maps. Here, the focus lies on larger data sets where the trajectory of the robot contains several possibly nested loops as these are especially hard to model. Furthermore, some open problems will be discussed in this section.

2D Mapping in Indoor Environments

Figures 3.19 and 3.20 show two-dimensional sparse point maps constructed from data sets recorded by Cyrill Stachniss at the University of Freiburg.

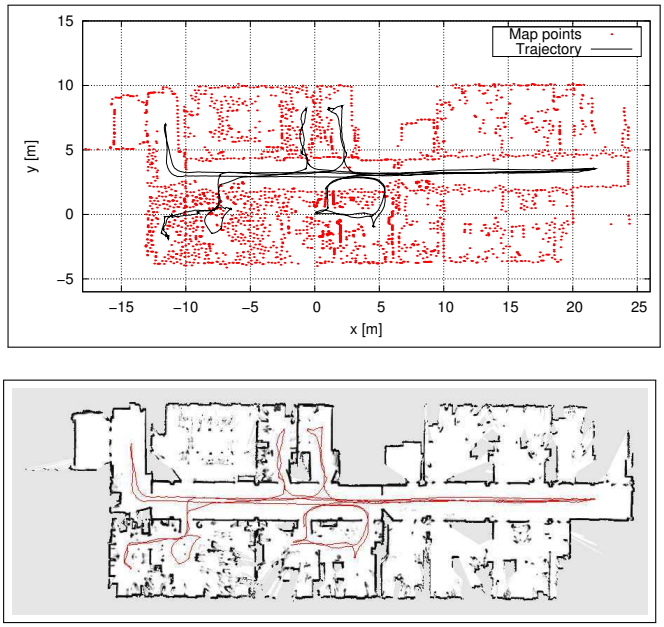


Figure 3.19: Sparse Point Map (top) of building 79 of the University of Freiburg and an occupancy grid map (bottom) provided with the data set by Cyrill Stachniss. The sparse point map contains 3092 points.

The data set recorded in building 79 of the University of Freiburg (Figure 3.19) consists of 3120 2D laser scans each containing 361 measurements. The trajectory with a total length of approx. 210 m contains several loops where the robot entered and left several smaller offices. The constructed sparse point map contains 3092 points. Compared to the probabilistic occupancy grid

map provided with the data set, it seems to be a little bit more cluttered. This is due to the fact that in a sparse point map a single measurement has a much larger effect on the resulting map as in occupancy grid maps. That is, repetitive scans not showing measurements at the same place, altough it would have been visible, cancel out single outliers. In a sparse point map, an outlier does not correspond to any point already stored in the environment model and is, thus, added just like any other point measured on the surface of environmental structures. This issue is addressed in Chapter 3.6.

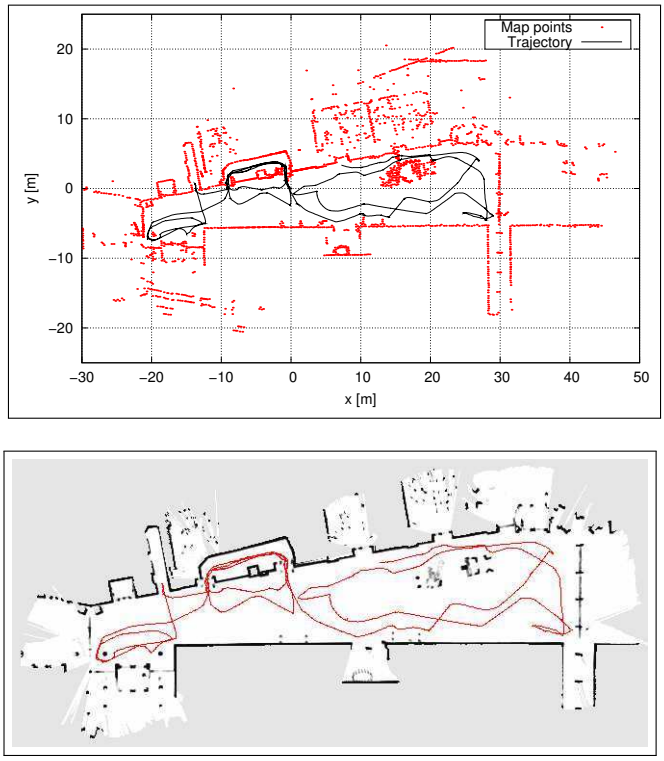


Figure 3.20: Sparse Point Map (top) of building 101 of the University of Freiburg and an occupancy grid map (bottom) provided with the data set by Cyrill Stachniss. The sparse point map contains 2364 points.

The data set recorded in building 101 of the University of Freiburg (Figure 3.20) consists of 5487 2D laser scans, again, each containing 361 measurements. The trajectory with a total length of approx. 280 m contains several loops as the robot was driven by an exploration strategy actively closing loops to decrease map uncertainty (Stachniss et al., 2004). Here, the constructed map contains 2364 points. In both cases the robot is correctly localized during the complete run and both maps constructed from the acquired range scans are globally consistent.

Figure 3.21 shows a sparse point map modeling a larger environment – namely the Shaw Conference Center in Edmonton, Canada. The data set was recorded by Nicholas Roy and consists of 6013 2D laser scans each containing 181 range measurements. The robot's path has a total length of approx. 330 m. The constructed sparse map contains 4544 points.

What all three environments have in common, is that the loops in the robot's trajectory are rather small. That is, the robot only travelled over short distances in "unknown" terrain before re-entering an already modeled part of the environment. When travelling along larger loops, even small inconsistencies in the map, i.e. minor registration errors, can lead to situations where points measured on environmental structures that have already been modeled are not correctly aligned with the map and added at some other place resulting in a global inconsistency of the model. This issue will be addressed further below. That global inconsistencies can be avoided by actively closing smaller loops is used by Stachniss et al. (2004) in the context of an exploration strategy. This strategy forces the robot to repeatedly re-visit already modeled places during the exploration

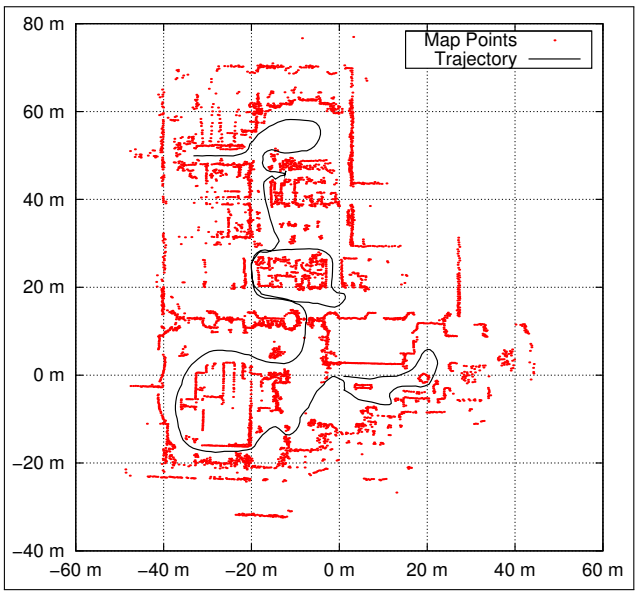


Figure 3.21: Sparse point map and trajectory of a robot moving through the Shaw Conference Center in Edmonton, Canada. The map contains 4544 points. The data set was recorded by Nicholas Roy.

of unknown terrain. Instead of exploring two consecutive unknown rooms, for example, the robot might leave the first room after its exploration and before entering the second unknown room, to decrease the uncertainty in its position estimation by entering an already modeled room.

Floor Points, Erroneous Measurements and Outliers

Up to now, there is no mechanism implemented in the registration algorithm to deal with erroneous measurements such as floor points in 2D scans and any kind of outliers in general. That these points can lead to larger registration errors finally leading to global inconsistency of the map is shown in Figure 3.22.

The partial map in Figure 3.22.a contains only a small number of floor points not causing noticeable localization errors. While traversing the doorway to the next room, the robot sensed a larger amount of floor points causing registration errors as shown in the partial map in Figure 3.22.b. The errors are shown in detail in Figure 3.22.c identifiable by jumps in the robot's trajectory (black polyline) and multiple displaced wall structures. When finally re-entering the room in which the robot has started, the loop in the robot's trajectory is not closed, i.e. points measured on already modeled environmental structures are not correctly registered. It is a matter of future work to implement mechanisms for identifying erroneous measurements, outliers and dynamics for being ignored in the registration process. Hähnel (2005), for example, iteratively calculates expectations of laser beams using a probabilistic sensor model given an initial map and, based on the calculated estimations, constructs a new map of the environment. This is repeated until no improvement of the map is measurable. Another approach could be the application of *virtual structure maps* (Holz et al., 2008) to aggregate a sequence of laser scans filtering out maximum measured distances in each direction relative to the robot. It has been shown in the context of 3D scan analysis, that, by this means, the structure maps only model the environmental structures whereas smaller and dynamic objects are filtered out. The aggregation of 2D scans to form partial maps for 2D scan registration is e.g. used in (Eliazar and Parr, 2003, 2004). For the data set of the Belgioioso Castle used here as an example, it is sufficient to simply take other thresholds in the decision whether or not a range scan is registered. To depict the problem described here, *odometry thresholding* was not applied to construct the maps shown in Figure 3.22 and every single laser scan was taken into account in the registration procedure. Applying the odometry thresholds, i.e. only matching

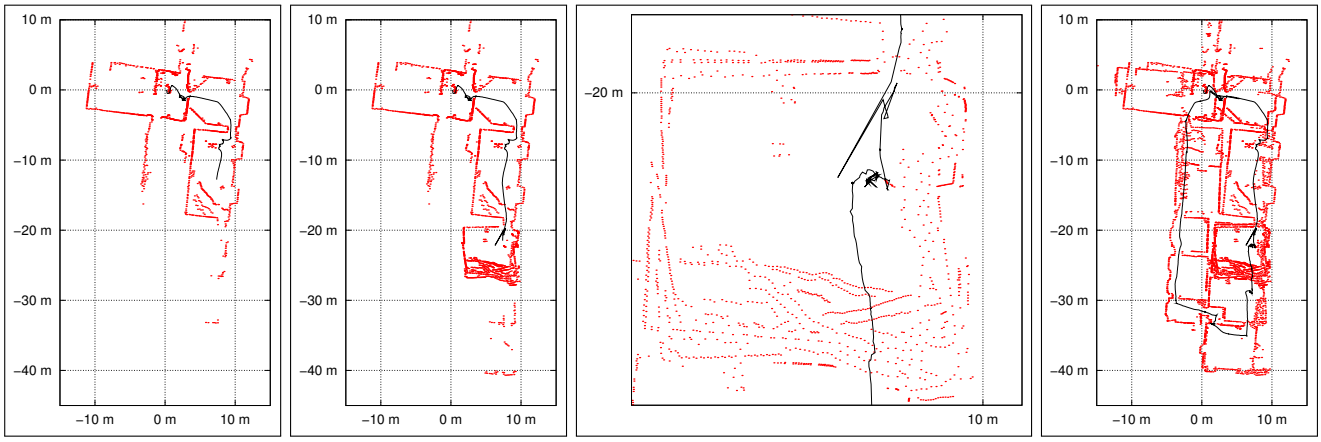


Figure 3.22: Sparse point map of Belgioioso Castle with erroneous points caused by sensing floor structures. The partial maps in (a) and (b) contain a large amount of floor points causing localization errors shown in detail in (c) and finally disallowing loop closure (d). The data set was recorded by Dirk Hähnel.

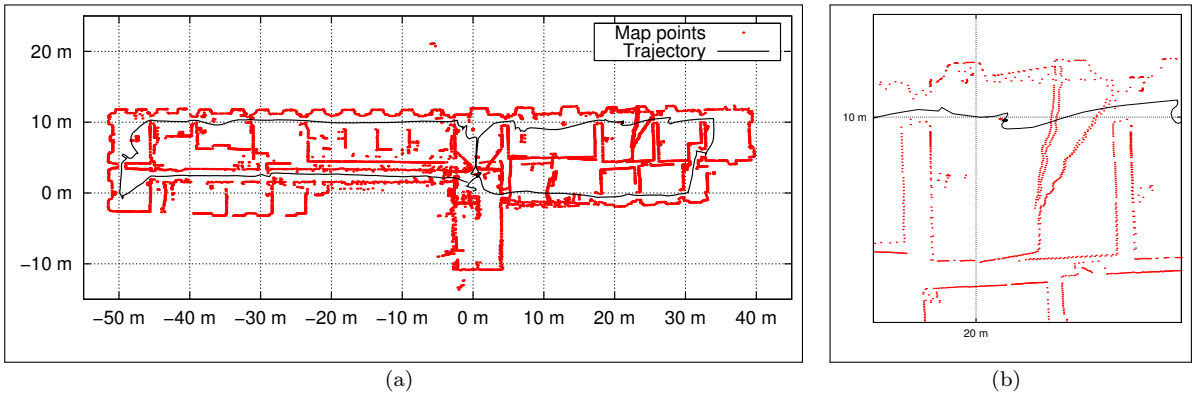


Figure 3.23: Sparse point map of Belgioioso Castle containing 7503 points. Here, the map (a) is globally consistent since only a small number of floor points is taken into account in the registration procedure not affecting the localization as shown in (b). The data set was recorded by Dirk Hähnel.

a newly acquired range scan when the robot has moved at least 50 cm or turned more than 25° since the last registration, results in the sparse point map shown in Figure 3.23. Here, only a small fraction of sensed floor points is taken into account in the registration not affecting the localization of the robot and yielding a globally consistent model.

Global Consistency

Still, a major problem of incremental registration is that larger registration errors can have disastrous effects on the resulting map. Although a sequence of range images can be incrementally registered to form a locally consistent partial map, the incremental registration procedure, as described in Chapter 3.5.3, always operates on the complete environment model. Once, a range scan is not correctly aligned with the model, subsequent registrations might not correct this registration error leading to two locally consistent partial maps. The environment model, however, will be globally inconsistent as the locally consistent partial maps are not correctly aligned at that very location where the incorrect registration happened to take place. Such a situation is shown in Figure 3.24.

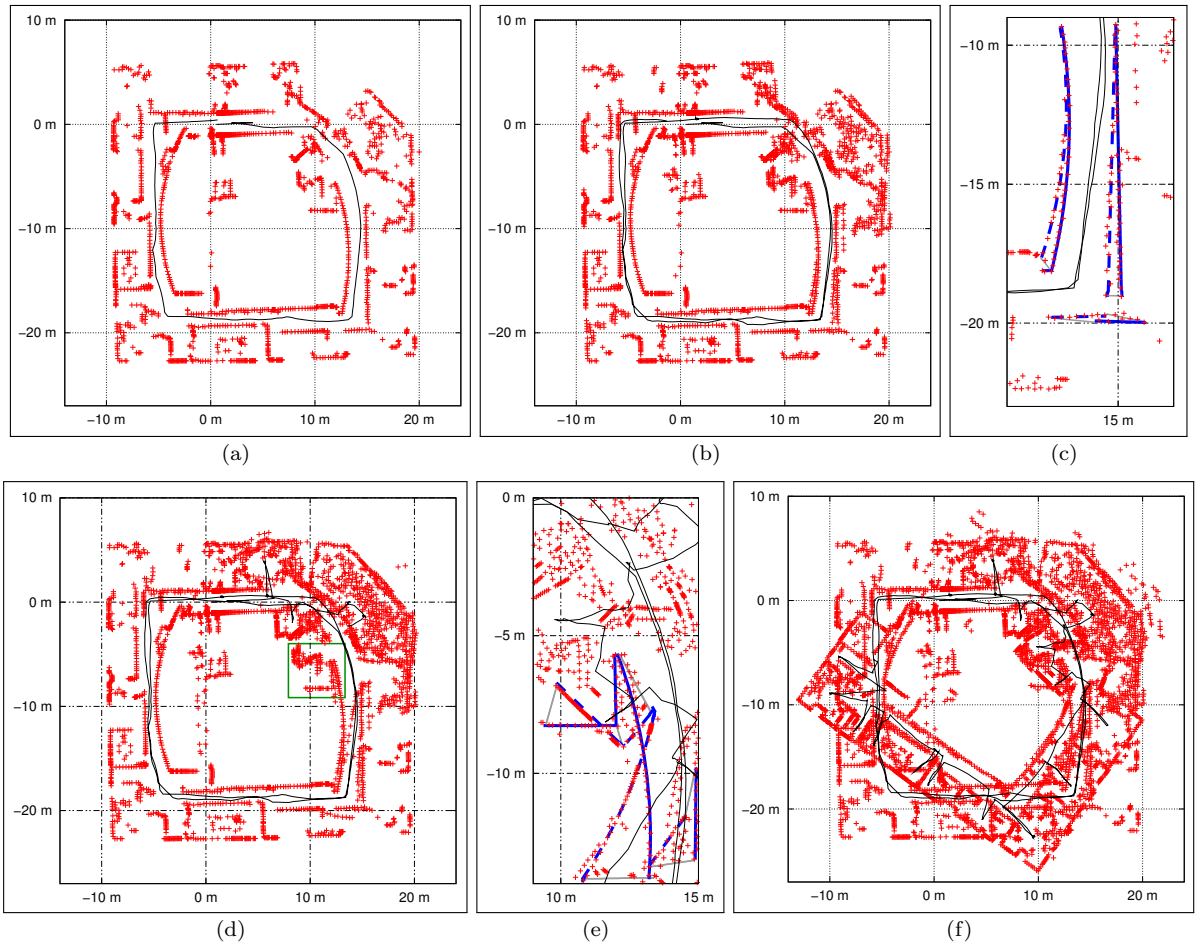
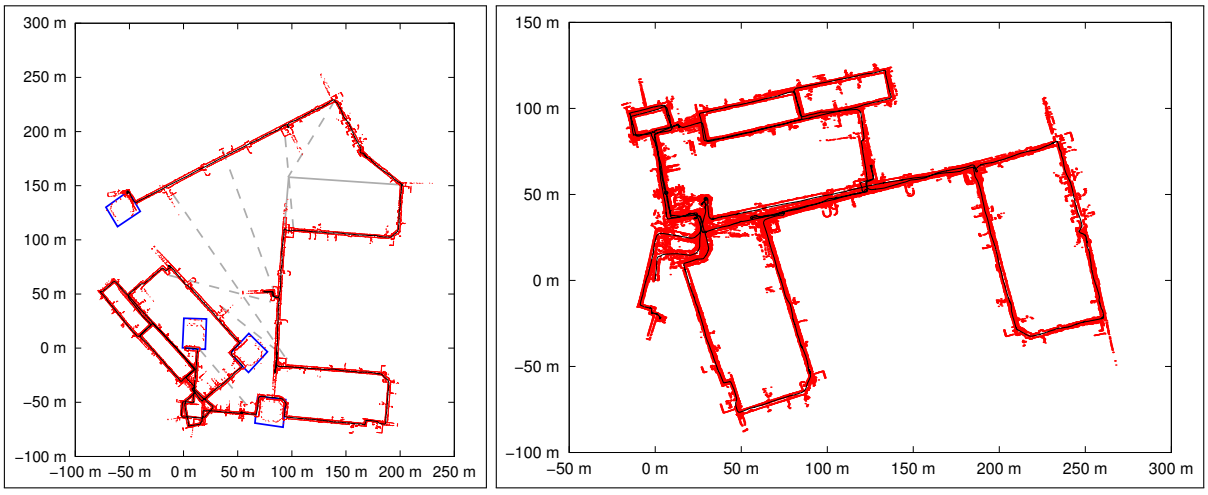


Figure 3.24: Map of the Intel Research Lab. The map is globally consistent after the first loop closure (a), but shows some minor inconsistencies after the second loop closure (b). These inconsistencies are shown in detail in (c) where the dashed blue lines correspond to continuous blue lines. A serious localization error in the third loop (d) (area is marked with a green rectangle) leads to major inconsistencies shown in detail in (e). The final map (f) shows two partial maps that are both in itself locally consistent. The data set was recorded by Dirk Hähnel.

Figure 3.24 shows the incremental construction of a sparse point map based on a data set recorded by Dirk Hähnel at the Intel Research Lab in Seattle. The data set contains 13 633 scans each containing 181 range measurements. By means of incremental registration the first loop is closed and the robot is correctly localized as being near its starting location (Figure 3.24.a). When traversing the same (already modeled) corridor again in the second loop (Figure 3.24.b), a minor inconsistency arises due to a small registration error. The minor inconsistency is shown in detail in Figure 3.24.c. The corresponding registration error is, however, corrected in subsequent registrations and also the second loop is correctly closed. In the third loop (Figure 3.24.d) the robot successively enters small offices along the corridor to fully cover the complete environment. In a small storage room (marked by a green rectangle in Figure 3.24.d) the robot turns on the spot where a sequence of registrations is not correct probably due to odometry errors and clutter modeled in the map. When the robot leaves the storage room, its orientation (angle about the z-axis) is shifted by approx. 40°. This is depicted in detail in Figure 3.24.e where just like in Figure 3.24.c environmental structures belonging together are marked with dashed and continuous blue lines. As a result, the map is not globally consistent (Figure 3.24.f) but consists of two locally consistent partial maps rotated relative to each other about the shift in the robot's orientation (approx. 40°) at that very location in the map where the major registration errors happened to take place. This experiment shows, that registration errors can have disastrous effects on the metascan used in an incremental registration procedure. It also shows, that incremental registration on its own can not cope with larger environments and trajectories containing larger loops. It is a matter of future work to augment the registration procedure with loop detection and global relaxation mechanisms, e.g. to add a graph-based SLAM algorithm to refine the constructed model and correct inconsistencies like the one shown in Figure 3.24.f.

A good example for larger environments containing larger loops is the *Infinite Corridor*, a hallway that runs through the main buildings of the Massachusetts Institute of Technology. Figure 3.25 shows two sparse point maps constructed from a data set recorded by Mike Bosse and John Leonard where a mobile robot travelled along this hallway and corridors of adjacent buildings.



(a) (b)

Figure 3.25: Maps of the "Infinite Corridor" at MIT. The map in (a) is globally not consistent, but shows better alignments in local regions. The map in (b) is almost globally consistent but several regions within the map show minor local inconsistencies. The data set was recorded by Mike Bosse and John Leonard.

The map shown in Figure 3.25.a was constructed with the same set of parameters, if not mentioned otherwise, as all the maps constructed before. These include odometry thresholding with 50 cm and 25°, an exponentially decaying function determining the distance threshold $D_{\max}$ in the ICP algorithm and a minimum allowable point-to-point distance of 25 cm for the sparse point

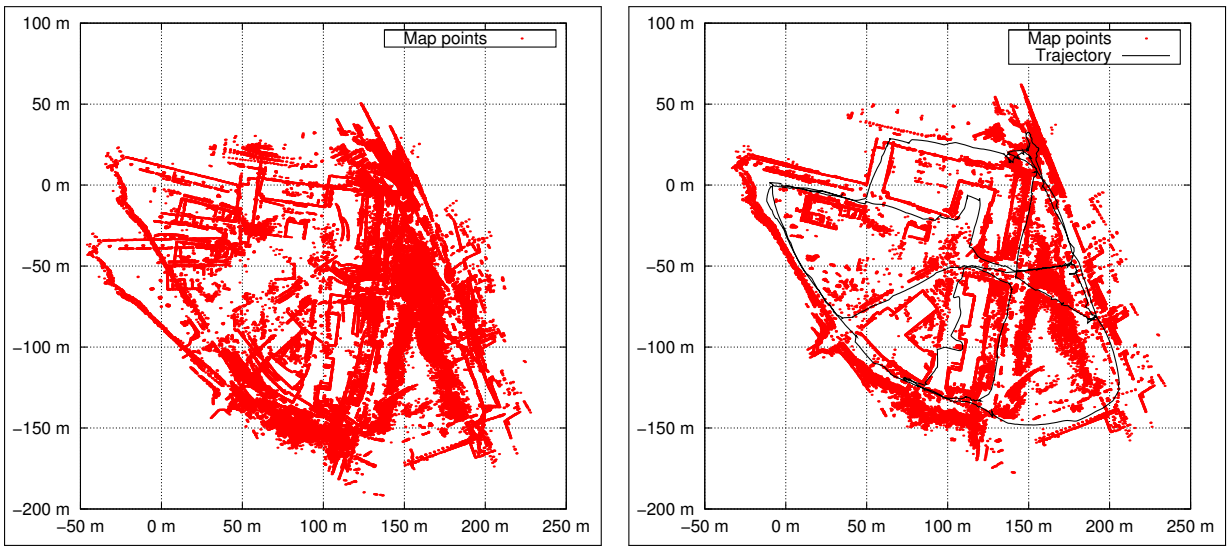
map. Here, local regions of the map are accurately modeled and locally consistent. Larger loops could, however, not be closed. Regions in the map that belong to the same location in the physical environment are marked with dashed gray lines and blue rectangles. Due to the length of the corridors, rotational errors lead to large displacements and it is questionable whether loop detection and global relaxation mechanisms solely based on heuristics (e.g. Borrmann et al., 2008) can correct the map. Additionally using visual features might help to close these loops (Newman and Ho, 2005; Kunze et al., 2007). What is, however, definitely one of the next steps is the integration of a graph-based SLAM algorithm (Olson et al., 2006; Grisetti et al., 2007b) into the incremental registration procedure to construct globally consistent maps even for large scale environments.

The map shown Figure 3.25.b, constructed from the same data, is almost globally consistent. In the left most part of the map, rotational errors lead to multiple copies of one and the same corridor and the hallway is modeled twice. In comparison to Figure 3.25.a, local regions within the map are, however, not modeled as accurate and details on the modeled environmental structures seem to be blurred. The map was constructed while trying out different parameter settings. Here, the refinement matrices determined by the ICP algorithm have only been applied to correct initial estimates when the number of corresponding points was at least 60 % of the total number of points in the range image and the corrected transformation matrix did not deviate too much from the estimated transformation. In some cases the registration procedure trusted the odometry and not the matching result. Note, that this parameter setting could only be used here. For any other data set the maps constructed with these settings were not correct and showed multiple major registration errors finally leading to an environment model that was almost as bad as that constructed from pure odometry. This shows that appropriate parameters for the matching procedure highly depend on the used sensor, the quality of odometry estimations and especially the environment. Up to now, no systematic way of determining appropriate parameters as been found. However, the parameter settings used throughout the report for producing most of the environment models have been found to produce reasonable and sufficiently good results for any given data set.

2D Mapping in Outdoor Environments

Up to now, the environments in the presented results were constrained to indoor environments. These are structured and the robot normally moves on plain flat floor. The size of these environments is often limited to single buildings or rooms. Typical outdoor environments are cluttered and not constrained in size. As the robot can not be assumed to move on flat floor, its position and orientation is no longer limited to the xy-plane, but has to be determined in 3D space. Range images taken in outdoor environments often show a large amount of clutter due to e.g. measurement points on vegetation or roughness of the terrain.

Figure 3.26 shows two maps of the campus of the University of Freiburg. This data set, recorded by Cyrill Stachniss and Giorgio Grisetti, contains 15 689 2D laser scans each containing 361 measurements. The robot travelled a total distance of approx. 1.9 km. The main challenge of this data set lies in the fact that at several locations the distance of the robot to surrounding environmental structures is larger than the maximum measurement range of the used laser scanner. The majority of measurements in some of the laser scans consists of maximum range readings, i.e. most of the points are not valid and the registration of these laser scans can only use a small amount of measurements possibly leading to major misregistrations as shown in Figure 3.26.a. Here, thresholds of 50 cm and 25° for odometry thresholding, i.e. the decision of whether or not a scan is registered based on the distance travelled and angle turned since the last registration, is not appropriate for an environment of this size. Increasing these thresholds to 2 m and 45° results in the globally consistent map shown in Figure 3.26.b. This reflects the assumption that at corners where not enough environmental structure is perceived, turning about 45° allows the robot to, again, perceive enough distinctive details for a successful registration. Note that for smaller indoor environments these thresholds are too large and a lot of details of perceived environmental structures are not modeled.



(a) (b)

Figure 3.26: Sparse Point Maps of the Freiburg Campus. The maps in (a) and (b) are constructed using different thresholds in the odometry thresholding mechanism of the registration procedure. In addition, for the map in (b) odometric estimates have not been corrected when the number of corresponding points fell below a threshold. The data set was recorded by Cyrill Stachniss and Giorgio Grisetti.

3D Mapping in Outdoor Environments

To construct a complete terrain model of an outdoor environment, 2D sensors are not appropriate. As described in Chapter 2.3, the inherent drawback of 2D laser scanners for the purpose of mapping and collision avoidance is that objects not intersecting the 2D scan plane are not detected by the laser scanner and, thus, cannot be perceived by the robot. When using 3D sensors, the virtual maps from Chapter 2.3.4 provide efficient means for constructing 2.5D maps based on rich 3D data.

However, up to now, the registration algorithm has not been used to construct 3D sparse point maps. Since the matching-algorithm based on the ICP algorithm can handle 3D data, the incremental registration does not even have to be changed in order to construct 3D models. Figure 3.27 shows a three-dimensional sparse point map constructed from a data set recorded by Oliver Wulf containing 468 3D laser scans each containing approx. 20 000 points. The trajectory of the robot contains several larger loops and has a total length of approx. 1 km. With a minimum allowable point-to-point distance of 2 m the shown sparse point map containts only 10 060 points. The map is globally consistent and altough the map does not model details in the visited parts of the environment, its structure is accurately modeled. Heavily reducing the point density of the acquired range scans and the sparse point map by using the threshold of 2 m allows for matching the range scans in less than 150 ms. When using a smaller minimum allowable point-to-point distance, e.g. 15 cm, the registration of a single range scan can take more than 1 s. Figure 3.28 shows detail views on a model constructed from the data set with a minimum allowable point-to-point distance of 15 cm. Here, the point density of the range scans has also been reduced according to this threshold.

3.6 Incremental ICP-Based Matching and Grid Maps

Especially in the context of robotic exploration it makes sense to construct a grid map structure that distinguishes between free and unexplored regions of the robot's workspace. This information is, for example, provided by the probabilistic reflection maps described in Chapter 3.2.2. In principal, there are two ways of incorporating reflection maps in the SLAM approach introduced in the last

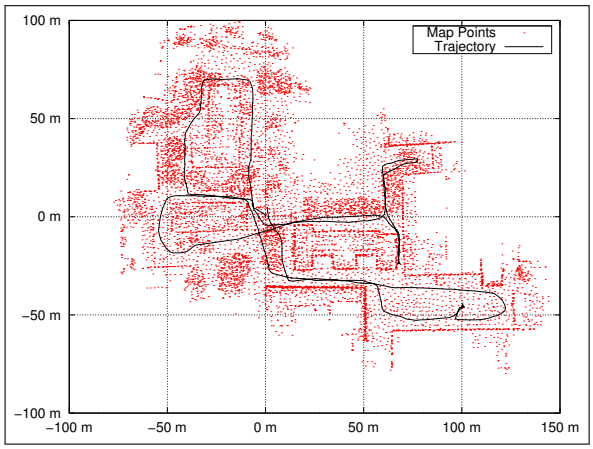


Figure 3.27: Topview on a 3D model of the Hannover Campus. Shown is the sparse point map with a minimum point distance of 2 m and the robot's trajectory of approx. 1 km length. The map contains 10 060 points

Figure 3.28: Detail views on a sparse point map of the Hannover Campus with a minimum point-to-point distance of 15 cm.

section. The first possibility is to not change the SLAM approach at all and to use the determined robot poses to update a reflection map as described in Chapter 3.2.2. That is, the reflection map is constructed in addition to the sparse point map built by the incremental registration. The second possibility is to replace sparse point maps by reflection maps in the formulation of the SLAM algorithm, i.e. instead of incrementally building a sparse point map, the reflection map is built inherently in the SLAM algorithm.

3.6.1 Additional Reflection Map Construction

Here the idea is to perform SLAM by means of the ICP algorithm and sparse point maps. Every scan that is registered to the map is then used to update an initially empty reflection map. That is, the trajectory of the robot determined by the aforementioned SLAM approach is used to construct the reflection as described in Chapter 3.2.2. In fact, the maps shown there were constructed by, first, determining the trajectory of the robot by incrementally building a sparse point map using the ICP algorithm and, second, constructing a reflection map based on the sequence of determined robot poses. The only information that is necessary to update the reflection map is the robot pose where the latest range scan has been taken as well as the range scan itself. That is, both the SLAM algorithm using sparse point maps and the construction of the reflection map can be run online. Compared to offline SLAM problems where all the information acquired be the robot is available, online algorithms only process the so far gathered information. Using an online approach allows the robot to construct an environment model while exploring its workspace. Furthermore, it should be mentioned that conducting updates to the reflection map does not affect the incremental structure of the SLAM algorithm. An according formulation for the procedure of using incremental registration with the ICP algorithm and the concept of sparse maps while building a probabilistic reflection map can be summarized as follows:

For every range image D_i that is to be registered **do**

1. **Determine robot pose**: Use the ICP algorithm to register D_i with the so far built sparse point map M_{i-1}. As an initial estimate for the resulting transformation $\mathbf{T}_i$ use the transformation $\mathbf{T}_{i-1}$ from the last registration (possibly updated using the odometric estimate of the pose shift between poses $\mathbf{x}_{i-1}$ and $\mathbf{x}_i$).

   $$\mathbf{T}_i = ICP(M_{i-1}, D_i, \mathbf{T}_{i-1})$$

   If M_{i-1} is empty, i.e. $M_{i-1} = M_0$, skip this step.

2. **Transforming the Image**: Transform the measured points in range image D_i into the map's (global) coordinate frame using $\mathbf{T}_i$.

3. **Update the sparse point map**: Determine the set $D_i^\star$ of transformed points for which no corresponding point can be found in M_{i-1} (see Chapter 3.5.2). Add all points $\mathbf{d}_{i,j}^\star \in D_i^\star$ to the sparse point map:

   $$M_i = M_{i-1} \cup D_i^\star$$

4. **Update the reflection map**: Use the robot's pose represented in form of the transformation $\mathbf{T}_i$ and the transformed range image D_i to update the reflection map (see Chapter 3.2.2).

end

An example of applying the above procedure, i.e. constructing a probabilistic reflection map in addition to the inherent construction of a sparse point map in the registration algorithm, is shown in Figure 3.29. Which scans should be used for registration depends on whether or not odometry thresholding is used as well as the corresponding thresholds. By registering a scan the pose of the

robot is determined and the scan is transformed into the world coordinate frame. Both the robot's pose and the transformed scan are then used to update the reflection map.

The additional construction of the reflection map only slightly increases the runtime of the registration procedure. The worst-case complexity of the update procedure of the reflection map solely depends on maximum measurable distance, apex angle and angular resolution of the scanner as well as the map's resolution. In experiments runtimes of 0.5 ms to 3 ms have been measured for the update of the reflection map. Furthermore, it should be mentioned, that the above procedure can, in the same way, also be applied to probabilistic occupancy maps or other types of grid maps.

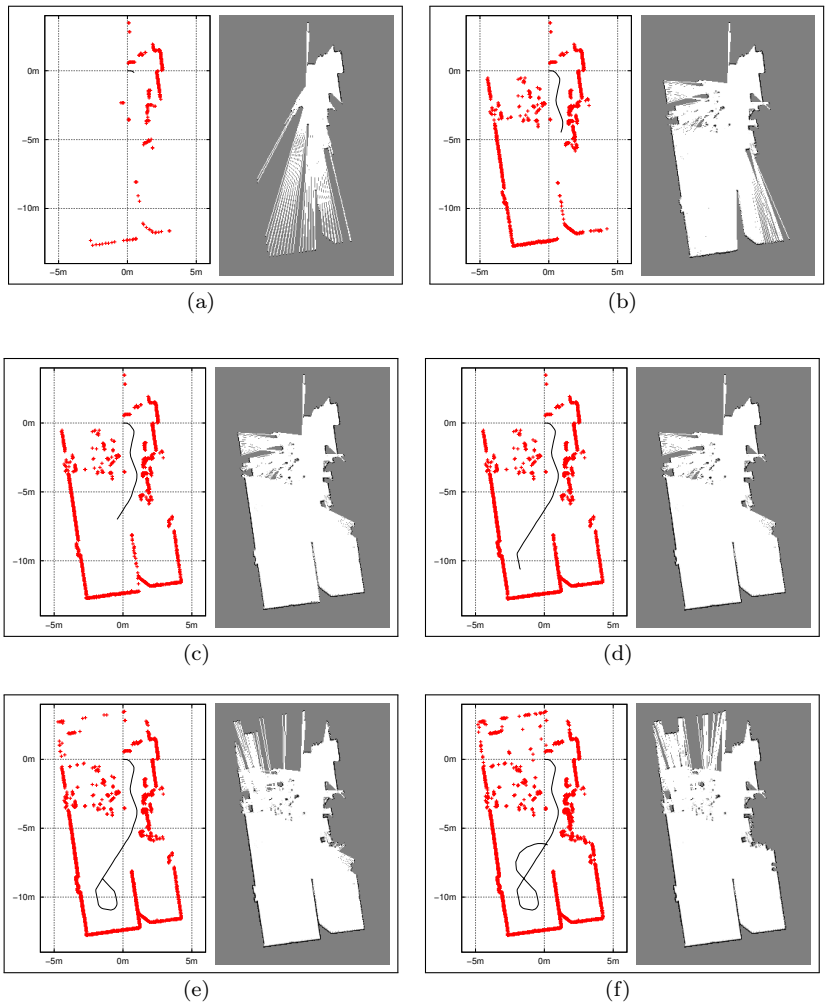


Figure 3.29: Additionally constructing a reflection map (right) based on the trajectory determined on a sparse point map (left) over time (a-f). The aligned laser range scans are used to incrementally update the reflection map.

3.6.2 Removing Points On Dynamic Objects

The inherent drawback of incrementally constructing point maps is that, once a point is added to the model, it is no longer modified or removed from the map. Especially in dynamic and populated environments a larger amount of points might have been measured on the surface of dynamic objects. That is, a point in the point map might longer correspond to an object in the actual environment. As already mentioned in Chapter 3.5 there is no straightforward way of determining which points need to be moved or removed from a sparse point map. However, when additionally constructing a reflection map, the robot is provided with the means of directly

determining whether it is probable or certain to measure a point in the map again. That is, it can check in regular intervals whether a model point lies in a region where the probability of reflecting a laser beam is high. For dynamic objects, the reflection probability will be lower than that of regions being block by a static object, as more laser beams will pass the corresponding cells and not cause a reflection when the dynamic object is no longer present in that region. Points that lie in regions of low reflection probability can be simply removed from the point map. By this means measurements caused by dynamic objects are not present in the final map. As the grid map structure of the reflection map allows for directly looking up the reflection probability of a particular region, i.e. $O(1)$, the complexity of this validity check is linear in the number of map points, i.e. $O(n)$ for n points. Furthermore, this check does not need to be carried out before or after every registration, but can instead be conducted in regular intervals, e.g. after a certain time $t_{\max}$ since the last check or after registering n range scans. In the actual implementation, the above procedure is simply extended by a 5-th step performing the validity check (gray box) after every registration.

For every range image D_i that is to be registered **do**

1. **Determine robot pose**: Use the ICP algorithm to register D_i with the so far built sparse point map M_{i-1}. As an initial estimate for the resulting transformation $\mathbf{T}_i$ use the transformation $\mathbf{T}_{i-1}$ from the last registration (possibly updated using the odometric estimate of the pose shift between poses $\mathbf{x}_{i-1}$ and $\mathbf{x}_i$).

   $$\mathbf{T}_i = ICP(M_{i-1}, D_i, \mathbf{T}_{i-1})$$

   If M_{i-1} is empty, i.e. $M_{i-1} = M_0$, skip this step.

2. **Transforming the Image**: Transform the measured points in range image D_i into the map's (global) coordinate frame using $\mathbf{T}_i$.

3. **Update the sparse point map**: Determine the set $D_i^\star$ of transformed points for which no corresponding point can be found in M_{i-1} (see Chapter 3.5.2). Add all points $\mathbf{d}_{i,j}^\star \in D_i^\star$ to the sparse point map:

   $$M_i = M_{i-1} \cup D_i^\star$$

4. **Update the reflection map**: Use the robot's pose represented in form of the transformation $\mathbf{T}_i$ and the transformed range image D_i to update the reflection map (see Chapter 3.2.2).

5. **Remove dynamic points**: For every point $\mathbf{m}_{i,j} \in M_i$ check whether the reflection probability of the corresponding cell $p\left(c^{[\mathbf{m}_{i,j}]}\right)$ is larger than some threshold $p_{\min}$, e.g. 10 %:

   $$\check{M}_i = M_i \setminus \{\mathbf{m}_{i,j} \mid p\left(c^{[\mathbf{m}_{i,j}]}\right) < p_{\min}\}$$

   and use $\check{M}_i$ for the next registration instead of M_i.

end

By applying the additional validity check, points measured on the surface of dynamic objects are removed as soon as the object disappears. This is of particular interest in populated environments as points measured e.g. on a human's leg might cause phantom effects that, in later applications of the map, might hinder the robot of traversing the corresponding regions of its workspace. Consider, for example, the situation visualized in Figure 3.30. Here, a human crossed the trajectory of the robot while it was incrementally building an environment model. The measurements taken at the legs of that human cause phantom effects in the sparse point map (Figure 3.30.a). However, as other range beams have passed the respective cells before and after the human passed by, the corresponding regions in the reflection map (Figure 3.30.b) show a low probability of reflecting a

range beam. By conducting the aforementioned validity check and removing points in cells with low reflection probability, these phantom effects (Figure 3.30.c) are identified and the points measured on the (dynamic) legs of the human are removed from the sparse point map (Figure 3.30.d). In this example, the validity check has been carried out after every 10-th registration of a range scan. Checking the reflection probability after every registration, as described in the procedure above, directly removes these points. That is, phantom effects are not taken into account in the registration. Both the update of the reflection map and the validity check have a low complexity and the runtime of the overall registration algorithm is only slightly increased.

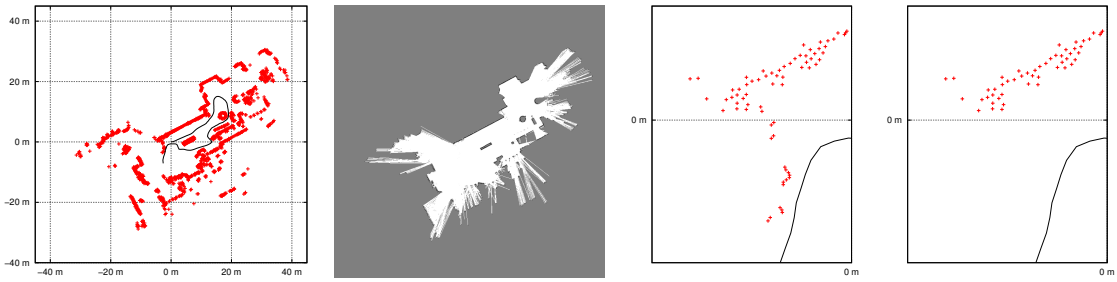


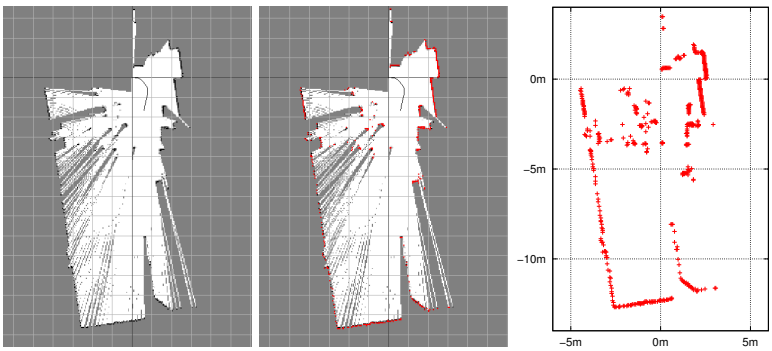
Figure 3.30: Removing less probable points from point maps. Shown here are a sparse point (a) and the additionally constructed reflection map (b). The detail views in (c) and (d) show a region where a human passed by before and after the validity check. The data set used in this example has been recorded by Nick Roy.

3.6.3 Inherent Reflection Map Construction

Up to now, a probabilistic reflection map has been only built in addition to the sparse point map generated by the incremental registration algorithm. Another possibility of constructing reflection maps is to base the complete registration algorithm on grid maps. That is, the reflection map is inherently built within the SLAM algorithm. The model point set for registering a range scan with the ICP algorithm is then generated from the grid map, for example, by extracting the centers of all cells whose reflection probability is larger than some threshold $p_{\min}$, e.g. $p_{\min} = 0.5$. 0.5 is the value that is used to initialize the reflection probability for all cells. Hence, choosing a threshold that is smaller than 0.5 will result in the fact that the centers of all previously untouched cells are added to the model point set. Thus, the runtime of the ICP algorithm considerably increases since the construction of the *kd*-tree for nearest neighbor search considers all model points. Furthermore, larger portions of the model set are uniformly distributed over unknown regions of the environment what can lead to registration errors. On the other extreme, choosing a rather large value, e.g. $p_{\min} = 0.9$, might, due to the scanner's inaccuracies, neglect cells that are occupied by static objects. Furthermore, the SLAM procedure can no longer benefit from the fact that reflection maps model even less reflective objects.

Figure 3.31 shows an example of extracting a model point set from a probabilistic reflection map (Figure 3.31.a). Here, a threshold of $p_{\min} = 0.5$ is used to extract cells being relevant for registration (Figure 3.31.b). Cells whose reflection probability is larger than $p_{\min}$ are extracted and their cell centers are added the model point set (Figure 3.31.c). The behavior of this extracted model set in the presence of dynamic objects is similar to that of additionally constructing a reflection map and subsequently applying the aforementioned validity check to remove dynamics from the point map. As a side note it is to remark that, respectively, the resolution and the cell side length of the reflection map have a similar effect on the extracted point set as the distance threshold between neighboring points in a sparse point map. That is, the cell size forms an upper bound on the density in the extracted point set.

For the inherent construction of reflection maps, the update of the sparse point map and the subsequent validity check are simply replaced by an extraction step to retrieve a point map from



(a) (b) (c)

Figure 3.31: Extracting point set from grid maps. Shown here are a reflection map (a) and the relevant cells (b, red pixels). Extracting the centers of relevant cells results in the model point set shown in (c).

the reflection map. A SLAM procedure based on the ICP algorithm inherently constructing a probabilistic reflection map can be formulated as follows:

For every range image D_i that is to be registered **do**

1. **Extract model point set**: Add the centers of all cells c whose reflection probability $p(c)$ is larger than a threshold $p_{\min}$ to the initially empty model point set M:

$$M = \{(x\ y)^T \mid p(c^{[xy]}) > p_{\min}\}$$

   If the reflection map is empty, skip steps 1 and 2.

2. **Determine robot pose**: Use the ICP algorithm to register D_i with the extracted model point set M. As an initial estimate for the resulting transformation $\mathbf{T}_i$ use the transformation $\mathbf{T}_{i-1}$ from the last registration (possibly updated using the odometric estimate of the pose shift between poses $\mathbf{x}_{i-1}$ and $\mathbf{x}_i$).

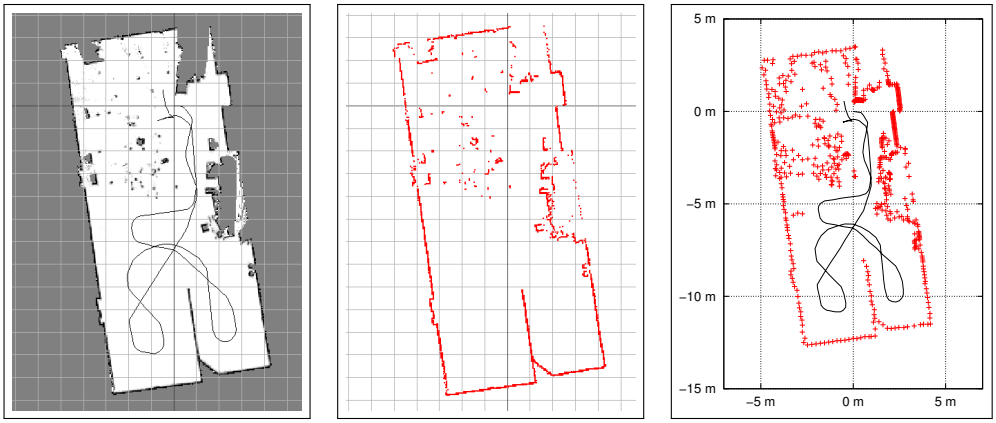
$$\mathbf{T}_i = ICP(M, D_i, \mathbf{T}_{i-1})$$

3. **Update the reflection map**: Use the robot's pose represented in form of the transformation $\mathbf{T}_i$ and the range image D_i to update the reflection map (see Chapter 3.2.2).

end

The complexity of the extraction step is linear in the number of cells. Figure 3.32 shows the result of applying the proposed SLAM procedure for inherently constructing probabilistic reflection maps. The cell size in the reflection map is 5 cm and the threshold $p_{\min}$ for extracting the model point set is 0.5, i.e. $M = \{(x\ y)^T \mid p(c^{[xy]}) > 0.5\}$. Laser range scans have been registered after either travelling more than 30 cm or turning more than 15°. The extracted point map (Figure 3.32.b) contains 1822 points. Compared to incrementally built sparse point maps, the extracted map contains less points corresponding to clutter and dynamic objects. However, they normally contain more points for one and the same environmental structure. A wall, for example, can, depending on the grid resolution and the accuracy of the registration algorithm, occlude a wider range of cells instead of only a single row. The environment modeled in this example is partially cluttered but not dynamic. Using the ICP algorithm to directly construct a sparse point map as described in Chapter 3.5 results in a set of only 600 points (Figure 3.32.c). For the sparse point map a threshold of $d_{\min} = 20$ cm for the distance between neighboring points has been used. With a threshold of $d_{\min} = 10$ cm, the resulting sparse point map contains 1048 points. Using a

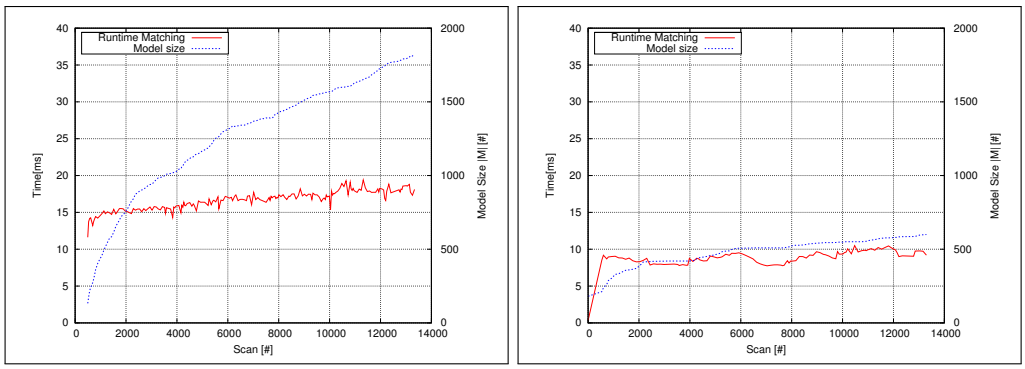
threshold of $d_{\min} = 5\,\text{cm}$ that is comparable to the resolution of the reflection map, the resulting map contains 2073 points. This increase of points primarily occurs in the vicinity of table legs and chairs. In more dynamic environments, the sparse point map built without the aforementioned validity check and the removal of dynamic points, might contain considerably more points than the point map extracted from reflection map.



(a) (b) (c)

Figure 3.32: Inherent reflection map construction. Shown here is the resulting reflection map (a) obtained using the proposed SLAM procedure. The point map (b) has been extracted from the final reflection map and contains 1822 points. For comparison, the sparse point map (c) originating from incremental registration contains only 600 points.

The runtimes for this example, measured over a single run, are shown in Figure 3.33. As the complexity of both algorithms for registering a range scan D is logarithmic in the size of the model set M, $O(|D| \log |M|)$, the difference in the measured runtimes is primarily caused by the additional extraction step that needs to be carried out when inherently building a reflection map.



(a) ICP + Reflection Map (b) ICP + Sparse Point Map

Figure 3.33: Runtimes and model set sizes for the registration against an incrementally built reflection map (a) and against an incrementally built sparse point maps (b).

3.6.4 Grid Resolution and Discretization Effects

Since the points in the model set used in the registration are formed by the centers of occupied cells, their coordinates are discretized and not in the continuous range of values like the raw measurements of the laser range finder or the points in a sparse point map. Hence, the accuracy of the registration is bound by, respectively, the cell size and the distance between the model points and

the actually modeled surface. Using larger cells in the grid representations might speed up the registration and has advantages e.g. in terms of memory consumption, but the resulting inaccuracies can lead to minor misalignments. If these misalignments are not corrected in the incremental registration procedure, by visiting already modeled portions of the environment, inconsistencies may arise. With smaller grid cell sizes, this discretization effect has a lower impact on the registration. That even minor inconsistencies might not be directly visible in a visual inspection of the resulting map is shown in Figure 3.34.

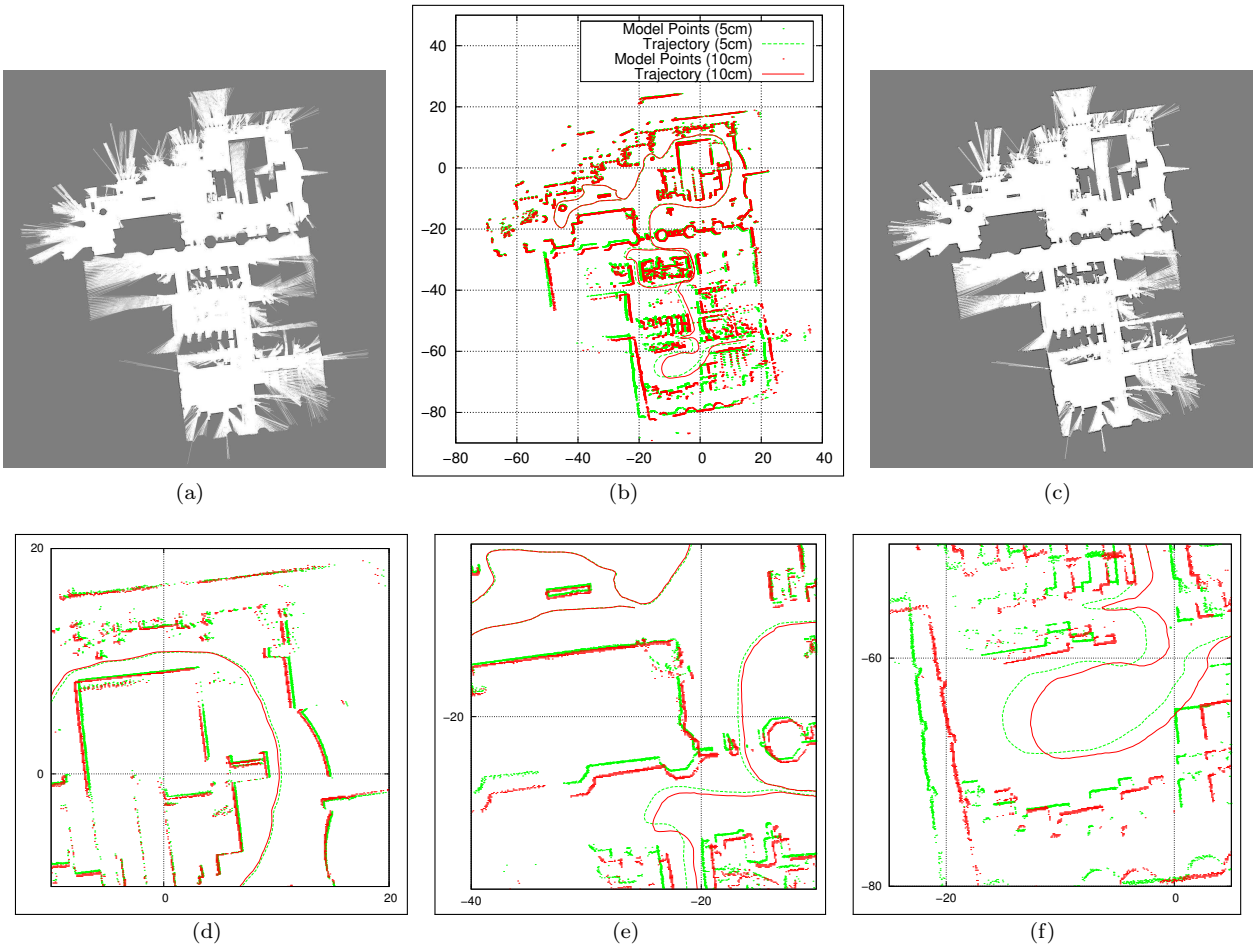
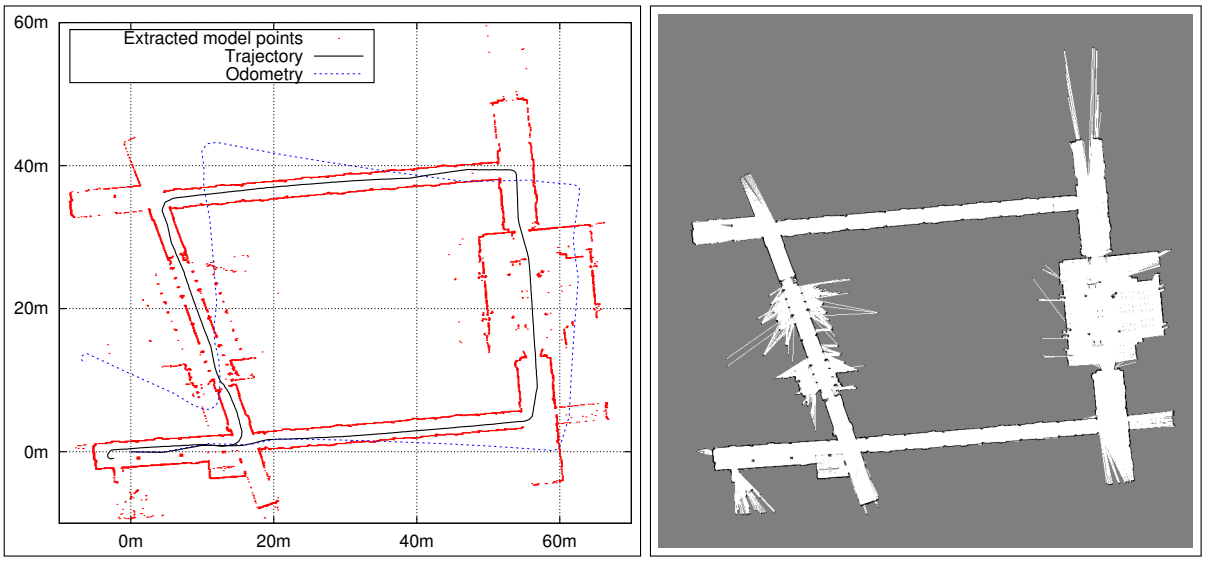


Figure 3.34: Discretization effects. Shown here are the results of applying ICP-based incremental registration on points extracted from probabilistic reflection maps. The differences between the two reflection maps (a) and (c) can only only be seen when directly comparing the extracted point sets (b). The detail views in (d-f) show that minor misalignments in early registrations can lead to major differences in later registrations. The data set used here was recorded by Nick Roy.

3.6.5 Results

The inherent problem of the ICP algorithm is determining the transformation aligning a data set D and a model set M in underconstrained situations, e.g. when M is a corridor or a circular environment. Furthermore, the inherent problem of the proposed incremental registration procedure is the fact that minor misalignments can cause major inconsistencies in the result map. Especially larger loops in the robot's trajectory might not correctly be closed requiring additional post-processing. In a final experiment, a mobile robot was remotely driven along a loop consisting of four corridors at the campus of the Bonn-Rhein-Sieg University of Applied Sciences. This experiment was meant

to help in examining the behavior of the ICP algorithm directly operating on grid maps, i.e. when inherently constructing a probabilistic reflection map, and in the aforementioned underconstrained situations. Figure 3.35 shows the results of incrementally registering the acquired range scans against, respectively, an incrementally updated reflection map and the extracted cell centers.



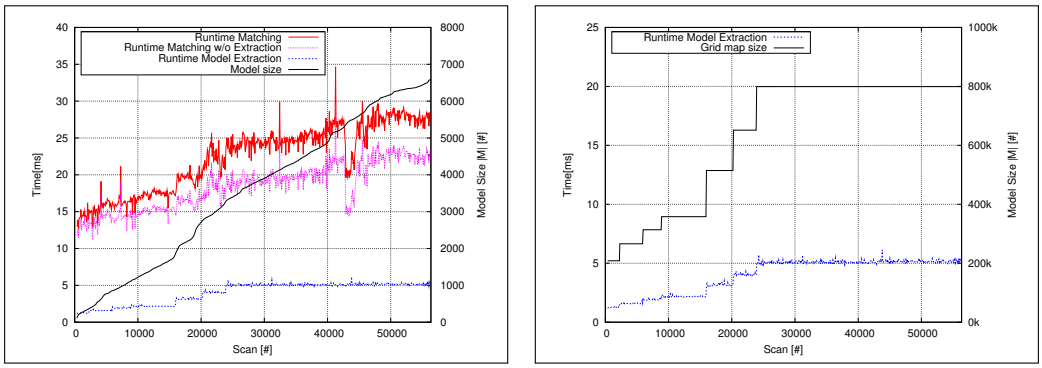
(a) Point map and Trajectory (b) Reflection grid map

Figure 3.35: Modeling a larger loop of corridors. Shown here is the trajectory determined by incrementally registering acquired range scan against a point set (a) extracted from the inherently built probabilistic reflection map (b). Although two corridors in the loop do not specific features, the loop is successfully closed without further post-processing.

The shown maps have been built online while the robot was driven through the environment. The cell size of the reflection map is 10 cm × 10 cm. Artificial beam widening has not been used and laser beams have been cut at 15 m in the update procedure of the reflection map (see Chapter 3.2.2). A range scan has been registered only when the robot has either moved more than 50 cm or turned more than 25° since the last registration of a range scan. For the actual registration, the ICP algorithm was configured to carry out 30 iterations with a distance threshold linearly decaying from 2.5 m to 0.1 m in the first 15 iterations and being constant at the side length of the grid cells (10 cm) in the last 15 iterations. The size of the resulting probabilistic reflection map is 833 × 829 cells (Figure 3.35.b). The point map (Figure 3.35.a) extracted from the final reflection map contains 6587 points. As a side note it is to remark that the measured runtimes (Figure 3.36.a) exceed the interval in which range scans have been taken (13.32 ms, 75 Hz). However, as the generic implementation of the ICP algorithm (see Chapter 3.3) allows for interrupting the registration after a certain time, consecutive scans can still be processed and the registration is refined in later control loop cycles just like in an anytime-algorithm (Zilberstein, 1996). The step function (Figure 3.36.b) also shows another characteristic of the implemented probabilistic reflection map. It dynamically resizes to the actually modeled environment. That is, one can start with a small grid map, e.g. 10 m × 10 m, taking the benefit of lower runtimes for the extraction of the model set. If a valid measurement or the robot's pose lie outside of the grid, the map is enlarged so that the corresponding cells are existent and can be updated. Note, that such a procedure is not necessary for point maps that solely store a set of points in the continuous range of values.

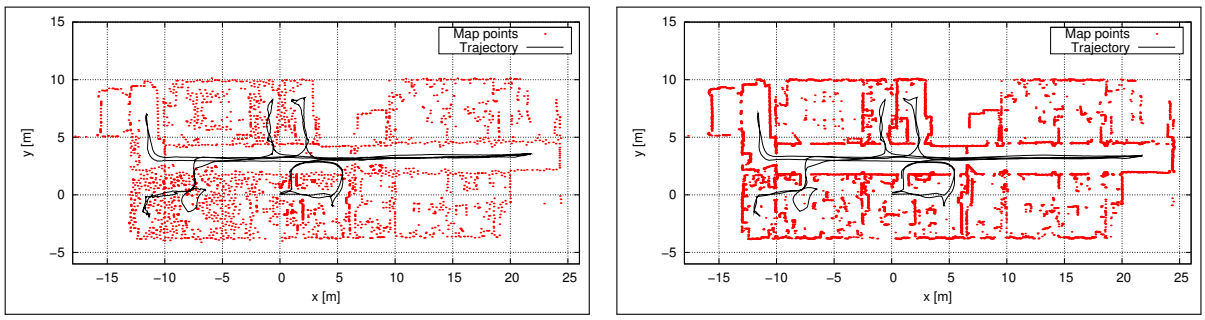
The main drawback of incrementally registering range scan against an incrementally built sparse point map is that dynamic objects can cause phantom effects, i.e. points that do not correspond to objects that are still present at the position where the points have been measured. Figure 3.37.a shows a sparse point map that has been constructed in an early experiment of Chapter 3.5.4. Here, the human operator following the robot caused a number of phantom effects especially in the

vicinity of doors. The corresponding measurements are inherently filtered out in the reflection map shown in Figure 3.37.b. Note that the same effect can be achieved by conducting the additional validity check and the removal of points that lie in cells with low reflection probability.



(a) (b)

Figure 3.36: Registration runtimes for the corridor loop. Shown here are the runtimes (a), measured over a single run, of the overall SLAM algorithm and the individual processing steps. What can be seen is that the runtimes only slightly increase for larger grid maps. The dependency of the runtime of the extraction step on the grid size is visualized in (b).



(a) ICP + Sparse Point Map (b) ICP + Reflection Map

Figure 3.37: Sparse point maps vs. reflection maps. Shown are a sparse point map (a) constructed from incrementally registering range scans recorded by Cyrill Stachniss and Giorgio Grisetti at Building 079 of the University of Freiburg. The point map extracted from the incrementally built reflection map (b) contains considerable less clutter and dynamic objects, especially in the vicinity of doors.

3.7 Incremental NDT-Based Matching and Point Maps

In recent work, e.g. (Magnusson et al., 2007) and (Magnusson et al., 2009b), it has been shown that range image registration using the Normal Distributions Transform (Biber and Strasser, 2003) produces registration results comparable to that of the ICP algorithm (Besl and McKay, 1992) in considerable less runtime. This was the primary reason for introducing NDT-based registration as an alternative to the ICP in Chapter 3.4. Magnusson et al. evaluated the performance and runtimes of NDT- and ICP-based pairwise registration in a 3D mine mapping scenario. This section will cover incremental registration using the Normal Distributions Transform for addressing SLAM just like with the ICP algorithm for constructing sparse point maps and probabilistic reflection maps.

3.7.1 Configuration and Qualitative Evaluation of NDT-based Registration

A set of experiments, comparable to that for registering range scans using the ICP algorithm against an incrementally built sparse point map in Chapter 3.5, has been carried out to evaluate the quality of registration results depending on the different NDT-extensions and subdivision methods. It turned out that an iterative subdivision repeatedly re-building the NDT-representation with decreasing cell sizes, in general, yields a more accurate registration comparable to course registration and fine-tuning the transformation using decaying distance thresholds in the ICP algorithm. In the experiments four different cell sizes have been used: 2 m and 1 m for an initial coarse registration as well as 0.75 m and 0.5 m for refining the resulting registration. However, the benefits of the NDT-based registration in terms of runtimes, compared to that of the ICP algorithm, almost vanishes with iterative subdivision. In the iterative subdivision of NDT, the NDT-representation needs to be re-computed four times. The decaying distance threshold in the ICP algorithm only affects the pair rejection while the *kd*-tree for performing nearest neighbor searches is computed only once.

Regarding the initial coarse registration in cases where initial estimates for the transformation are only rough, NDT may be trapped in a local minimum, even when using infinite outer bounds, neighbor evaluation and linked cells. Especially in larger outdoor environments it was sometimes necessary to start the registration with a cell size of up to 3.5 m to make the registration converge at all towards the optimal solution. With smaller cell sizes, different parts of the data set to register caused different and contradictory directions of gradient and Hessian in the optimization process. In these situations the data set was only minimally transformed with a local minimum in the transformation's score. In other situations, a particular cell size caused that the NDT-representation did not appropriately model environmental structures and the data set was moved away from the global minimum, although earlier registrations with larger cell sizes and later registrations with smaller cell sizes correctly transformed the data set towards the optimal solution.

With the ICP algorithm highly accurate registration results could be achieved by fine-tuning an initially coarse registration using smaller distance thresholds in later registrations. Similarly, small cells can used in the NDT algorithm. However, the problem of the NDT-representation is that cells need to contain a minimum number of points (e.g. 3 points in 2D and 5 points in 3D) to calculate a valid covariance matrix for the normal distribution. Especially in unstructured environments, rather smaller cell sizes need to be chosen to appropriately model the robot's surroundings. For having, respectively, valid normal distributions and enough points in the cells, the point density in the model needs to be in accordance with the cell size. That is, arbitrarily small cell sizes can not be used as larger portions of the model might not be dense enough to produce valid normal distributions and are, thus, neglected in the registration. When moving around corners, for example, only the latest range images contain information about the so-far unseen region behind the corner. Hence, the point density of the model in these regions is comparably low. In the conducted experiments, cell sizes smaller than 0.5 m lead to major misalignments in these situations.

Using *infinite outer bounds* for the NDT-representation can improve the registration result for rough initial estimates but may also lead to situations where non-overlapping regions are taken into account. That is, parts of the data set to register that are not contained in the model, can be moved towards the modeled regions. This behavior is quite similar to using larger distance thresholds in early registrations of the ICP algorithm without further rejecting correspondence pairs. Here, a solution is to use infinite outer bounds and to additionally apply distance thresholding just like in the correspondence search of the ICP algorithm to not evaluate points if there distance to the closest region in the model is too large, e.g. larger than 2 m. Not using infinite outer bounds, on the other hand, can cause that the NDT-based registration is directly trapped in a local minimum. In the case of poor initial estimates, larger portions of the data set might lie outside of the modeled environment and are, thus, ignored in the registration.

The evaluation of neighboring cells in addition to a point's corresponding cell only slightly improves the registration results. This fact needs to be further evaluated and is a matter of future work. For a point that falls into an empty cell, the evaluation of the cell's neighbors, of course,

improves the registration. Especially for only rough initial estimates, neighbor evaluation can also cause transforming the data set into the wrong direction just like false correspondences in the ICP algorithm. Very recently, Magnusson et al. (2009b) started to also evaluate neighboring cells using a trilinear interpolation weight function to address the effect that points are erroneously aligned with other environmental structures in a neighboring cell. The inherent drawback of evaluating neighboring cells is that it requires more lookups in the NDT-representation, i.e. 9 cells in the 2D case and 27 cells in the 3D case. Magnusson et al. (2009b) limit the number of evaluated neighbor cells to 8 and observe an increase in the measured runtimes of 450 %.

Regarding the different subdivision methods, it does not seem to play a role which subdivision method is used as long as the model set is "well" represented by the NDT-representation. In this context, iterative subdivision, i.e. starting with a larger cell size and iteratively scaling down over multiple registrations seems to be most reliable. That is, the model set is normally well represented in later registrations when the transformation is refined. Results like this have also been reported by Magnusson et al. (2007).

Here, an iterative subdivision is used computing the NDT-representation four times, with cell sizes of 2 m, 1 m, 0.75 m and 0.5 m. For the actual evaluation of a point' score and the derivatives of the score function, infinite outer bounds are used, i.e. the the nearest cell in the grid is used when the point lies outside of the modeled region. Furthermore, for all four cell sizes the linked cell structure is built in addition allowing for allowing to evaluate the score of a point that falls into an empty cell. The evaluation of neighboring cells is also enabled. Deviating from (Magnusson et al., 2009b) the linked cell structure is included in this evaluation. That is, if a neighboring cells does not contain a valid normal distribution the closest valid cell, determined on the linked cell structure, is used. By this means, unique correspondences have a higher impact on the final registration. A points that falls into a cell whose neighbors are all empty, for example, will cause that the same cell is evaluated multiple times. Hence, the cell's contribution to score function, gradient and Hessian is higher compared to that of cells with neighbors containing valid normal distributions.

3.7.2 Incremental NDT-based Registration

In their comparison of performance and reliability of ICP- and NDT-based range image registration, Magnusson et al. (2009b) only consider a pairwise registration of 3D laser range scans and apply a post-processing step for global relaxation, i.e. distributing accumulated errors at detected loop closures over individual robot poses to remove global inconsistencies. Biber and Strasser (2003) follow a similar approach for the registration of 2D laser range scans. They represent the environment by a set of (raw) laser range scans and associate a translation vector and a rotation angle that describe the pose where a scan was taken in the global coordinate frame. When registering a new range scan against this type of environment representation it is, for every point, determined which scan in the map yields the maximum score for that point. Only this very scan is then used in the registration of that point. That is, instead of matching a point against another point set, Biber and Strasser match each point in the data set to one point set in the vector of range scans forming the map. They refer to the scans in the map as *keyframes*. The drawback of this representation is that the same portion of the environment might me modeled by multiple keyframes. Biber and Strasser propose to not add every acquired range scan as a keyframe to the map, but only those whose overlap with the region modeled by the so-far added keyframes is smaller than some threshold. By this means they considerably decrease the number of keyframes and, thus, the runtime for registering a newly acquired range scan with the vector of keyframes. To further speed up the registration, they propose to organize the vector of keyframes in a graph-like structure and to consider only those keyframes in the registration that can be reached from the last vehicle pose where a scan was registered. Whereas decreasing this area speeds up the registration it also increases the change of misregistrations with poor initial estimates of the robot's pose. Here, the idea is to use the NDT-based scan matching algorithm in an incremental registration procedure just like for the ICP algorithm. That is, instead of representing the robot's environment by a vector of range scans with associated vehicle poses, the environment model should solely consist of a point set where all coordinates are given in the world coordinate frame. An according

registration procedure can be formulated as follows:

For every range image D_i that is to be registered **do**

1. **Determine robot pose**: Use the NDT-based registration algorithm to register D_i with the model point set M_{i-1}. As an initial estimate for the resulting transformation $\mathbf{T}_i$ use the transformation $\mathbf{T}_{i-1}$ from the last registration (possibly updated using the odometric estimate of the pose shift between poses $\mathbf{x}_{i-1}$ and $\mathbf{x}_i$).

   $$\mathbf{T}_i = NDT(M_{i-1}, D_i, \mathbf{T}_{i-1})$$

   If M_{i-1} is empty, skip this step.

2. **Transform the range image**: Use the robot's pose represented in form of the transformation $\mathbf{T}_i$ to transform the range image D_i into the world coordinate frame, yielding the transformed data set $\check{D}_i$.

   $$\check{D}_i = \{\check{\mathbf{d}}_{i,j} \mid \begin{bmatrix}\check{\mathbf{d}}_{i,j} & 1\end{bmatrix}^T = \mathbf{T}_i \begin{bmatrix}\mathbf{d}_{i,j} & 1\end{bmatrix}^T, \mathbf{d}_{i,j} \in D_i\}$$

3. **Update the map**: Add the transformed range image $\check{D}_i$ to the so-far constructed point map M_{i-1} yielding the updated point map M_i:

   $$M_i = M_{i-1} \cup \check{D}_i$$

end

The drawback of this formulation is that the resulting point map contains all points measured in the environment, i.e. $M_i = \bigcup_k D_k$, which led to the concept of sparse point maps in Chapter 3.5. Of course, this registration procedure can be simply extended with the ideas of Biber and Strasser (2003) by only adding the transformed data set $\check{D}_i$ to the model set M_{i-1} if the overlap of the corresponding regions in the environment is small. This can be achieved by simply counting the points in D_i that fall into cells with valid normal distributions in the registration procedure. Only if the ratio between these points and all points in D_i is smaller than some threshold, e.g. 50 %, the transformed image $\check{D}_i$ is added to M_{i-1}. Of course, those points falling into valid cells only when enabling infinite outer bounds in the NDT-representation should not be considered in this ratio. The problem with this extension is that even small portions of the transformed data set might contain relevant information for future registrations, e.g. when moving around corners.

3.7.3 Discretization Effects and Inaccurate Registrations

In first experiments, the aforementioned registration procedure has been used without further extensions for decreasing the number of model points. That is, all points are used in the registration to avoid unnecessary misregistrations and the emergent high runtime complexity has been neglected. An inherent drawback of NDT-based registration seems to be aforementioned problem of not having enough points in regions that have only been perceived in the latest range scans. Especially when moving around corners, newly acquired points fall into empty cells. Hence, these cells do not contain enough points for calculating a valid normal distribution in the latest registration. Due to this fact, newly acquired information about these regions in the environment can not be taken into account in the registration. Using larger cell sizes results in having enough points in cells representing these regions, but also cause discretization effects as shown for the ICP-based registration inherently constructing probabilistic reflection maps in Chapter 3.6. A typical results obtained in these first experiments is shown in Figure 3.38. Here, again, the data set recorded by Nick Roy in the Edmonton Convention Centre has been used. In this run, the robot traverses larger open rooms as well as narrow passages. Minor misalignments in the NDT-based registration occur primarily in these narrow passages due to the aforementioned problem. The resulting map is clearly distorted as these minor misalignments could not be corrected in the incremental

registration procedure. Larger open rooms, on the other hand, are rather accurately modeled and seem to be locally consistent. In fact, it turned out in all experiments conducted that, using the NDT-algorithm, registering range scans taken while moving around corners or traversing and leaving narrow passages seems to be a major problem.

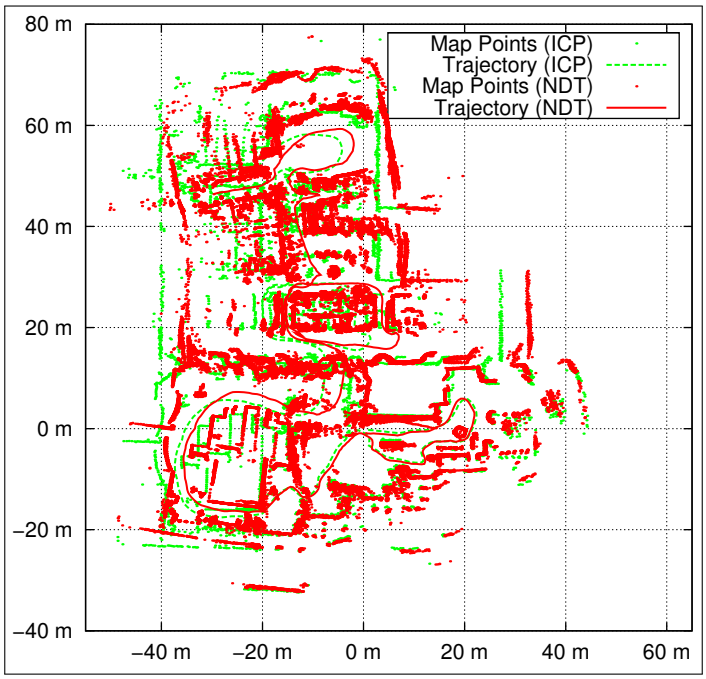


Figure 3.38: Incremental NDT-registration vs. incremental ICP-registration. Shown here are the results of applying both registration algorithms to a data set recorded by Nick Roy. Where as the ICP-based registration provided an accurate and globally consistent sparse point map, the point map resulting from NDT-based registration is clearly distorted. Runtime and space complexity have not been considered in this example.

In the experiments carried out in (Magnusson et al., 2007) and (Magnusson et al., 2009b) the robot moved through a sandstone mine near Örebro in Sweden. The traversed mine tunnels have, due to the natural layer of sandstone, a characteristic shape with flat ceiling and relatively straight walls (see Figure 3.39). Troughout the experiments, the robot acquired range scans solely in wide open spaces compared to the narrow passages as shown in Figure 3.38. In these open spaces, the aforementioned problem did not arise. In this context, it should also be mentioned that registering 3D scans seems to be less problematic in these situations compared to registering 2D range scans. With increasing cell sizes, multiple objects in the robot's environment are represented by the same cell leading to the aforementioned inaccuracies in the registrations and misalignments of newly acquired range scans. With 3D scans the cell size also increases in height and more points that have been measured on the surface of the same object (but in different heights) fall into the same cell. Because of multiple objects in one cell, the resulting normal distribution might still not adequately represent the distribution of points, but the 3D error ellipsoid better reflects the distribution than a 2D error ellipse in the two-dimensional top-view of the scene as used in 2D NDT-registration. The result of matching two typical range scans from the data sets used in (Magnusson et al., 2007) and (Magnusson et al., 2009b) is shown in Figure 3.39.

3.7.4 Concluding Remarks

Because of the occurring misregistrations in the vicinity of narrow passages and corners as well as transitions to so-far unmodeled regions in the environment, NDT-based registration is not further considered in the context of this thesis. The different extensions proposed by Biber and Strasser

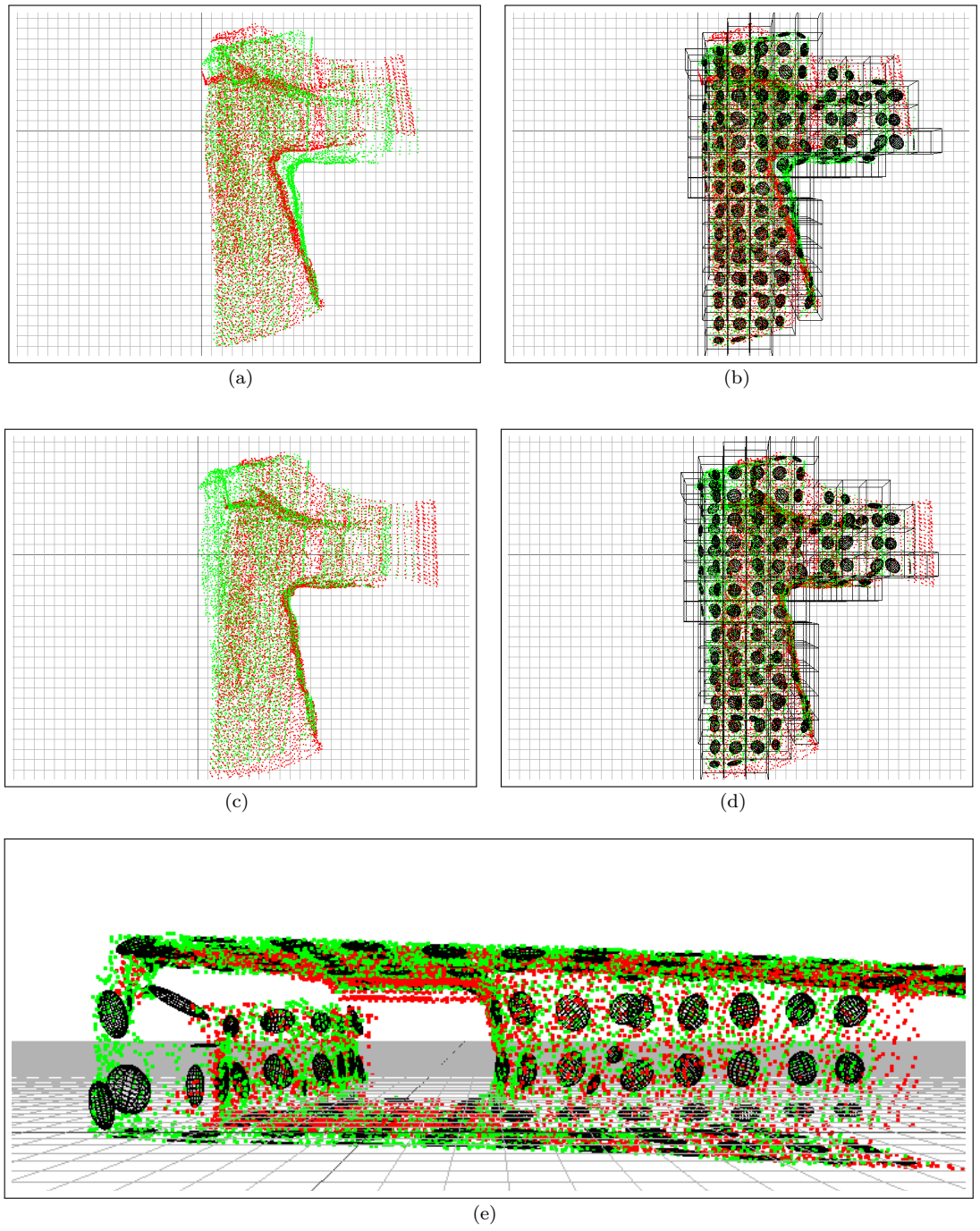


Figure 3.39: 3D NDT-based Registration in structured environments. Shown here a two points sets (left) together with the corresponding NDT-representation (right) from range scans taken by Magnusson et al. in the Kvarntrop mine before registration (a+b) and after registration (c+d). The scanned tunnels have flat floor and ceiling as well as relatively straight walls (e).

as well as Magnusson et al. also seem to provide no solution to the aforementioned problem and it is a matter of future work to address this issue.

Regarding the incremental construction of a point map, an efficient way to reduce the memory consumption of the map is to approximate the covariance matrix for the normal distributions in the NDT-representation in an incremental way (Pébay, 2008). By computing the NDT-representation for a newly acquired range scan and merging the contained normal distributions with those of the so-far built NDT-representation, the actually measured points do no longer need to be stored at all. By this means the NDT-representation becomes an environment representation in its own right. Of course, for using iterative subdivision, multiple NDT-representations need to be incrementally constructed with different cell sizes. Another possibility is organize the complete model in a single oct-tree where not only the leaves but all nodes contain an incrementally updated normal distribution for the points measured in the corresponding region of the environment. It is, however, a matter of future work to implement and evaluate these ideas.

Furthermore, it still needs to be evaluated whether the concept of sparse point maps helps to reduce the number of points in the model for an incremental registration procedure as described above while still allowing for accurate registrations. Here, first tests suggest that the NDT-representation built from the points in a sparse points adequately represents environmental structures although a larger portion of actually measured points is not considered. An example of such a map is shown in Figure 3.40.

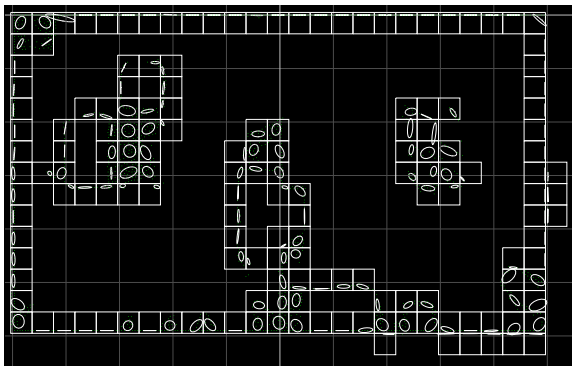


Figure 3.40: Example of a NDT-representation built for a sparse point map. The plot shows a sparse point map modeling the RoboCup@Home arena of the GermanOpen 2008 in Hannover as well as the NDT-representation for the 1904 points in the sparse point map.

However, an interesting characteristic of the NDT is that the normal distributions can be used to classify the cells, e.g. whether the contained points lie on a planar surface or not. Magnusson et al. (2009a) exploit this characteristic and use, for example, the eigenvalues of the covariance-matrix to distinguish different classes of cells. By means of the orientation of a distribution's error ellipse e.g. vertical and horizontal planes can be distinguished. The classified cells are then used to extract local features subsequently used for detecting loop closures in the robot's trajectory. Jensen et al. (2005) follow a similar approach. They partition the space occupied by the 3D model into equally sized cubes with a side length of 25 cm. After assigning all data points to their corresponding cell, a RANSAC approach is used together with least-squares fitting to approximate the points in a cube by a planar patch. In (Weingarten et al., 2004) the same space partitioning is used and planes are fitted to the points within a cube by means of probabilistic approach using the covariance matrix just like Magnusson et al. (2009a).

Chapter 4

Path Planning and Motion Control

As already mentioned, a procedure for simultaneous localization and mapping is purely passive and the robot is not actively driven. Instead, it is assumed that some other task, e.g. exploration or inspection, specifies where the robot has to move. The challenge of SLAM lies only on processing the information that accumulates during that movement. That is information about the robot's surrounding environment and information about the movement carried out. The focus of this chapter of the thesis lies on the question of how to control the robot's movements so that it reaches the locations that are specified by the other task. An exploration strategy, for example, determines, given the robot's current world model, a particular vehicle pose that needs to be approached by the robot in order to gain new information about the environment and update its world model. Reaching this vehicle pose, e.g. by planning shortest obstacle-free paths and controlling the robot's velocities, will be described in the following sections. In this context, central problems are path and motion planning, motion control as well as collision avoidance.

4.1 Introduction and Related Work

Before going into the details of the proposed approaches, this first section will give a brief overview on related work and the used approach that is described in the remainder of the chapter. In principal, two classes of approaches can be distinguished according to their type of environment representation and the workspace of the robot – namely *static* and *dynamic planning* (LaValle, 2006). The first one assumes that the environment of the robot does not change during the robot's operation, i.e. that the environment is static. Here, the general procedure is split into two phases: in the first phase the robot constructs a static internal environment representation or is provided with such a representation, whereas in the second phase the robot moves within its environment representation without performing any updates to it or at least new information is not taken into account for controlling the robot's movements. In this class, navigational tasks like for instance moving to a certain position can be solved by planning the robot's path or trajectory once, and, afterwards, controlling its movements in order to follow the plan possibly together with a collision avoidance strategy like the well-known *bug strategies* (Choset et al., 2004; LaValle, 2006). These strategies, however, are often purely reactive, e.g. turning away from or slowing down in front of obstacles. In the presence of obstacles that were not taken into account in the planning phase, reactive collision avoidance mechanisms might steer the robot in a way that it is no longer able to follow the planned path or makes larger detours until it reaches the goal. That is, if the assumption of a static environment is not met, this class of approaches can be only used with caution. The second class of approaches takes environmental changes into account. That is, the assumption of a static environment does no longer need to hold. Instead, paths or trajectories of the robot are planned in a continuous way and adapted to changes in the environment. This enhancement, however, often comes with a drastically increased complexity. The differentiation between the two classes is not strict and hybrid forms exist that, for example, plan paths on static environment models that are only locally adapted when an object appears, respectively, in the robot's vicinity and its sensors (Stachniss and Burgard, 2002). Another more drastic option is to update the robot's environment representation during operation and to completely re-plan if an object, not taken into account in the planning phase, blocks the planned path (cf. Choset et al., 2004, Ch. 5). Other hybrid approaches, e.g. (Zavlangas and Tzafestas, 2002), utilize multi-layered environment

representations by planning a shortest path on a topological map at the beginning and continuously re-planning local paths and trajectories in a metric map.

In the context of this thesis only the basic motion planning problem is addressed where the environment is completely known and the environment is assumed to be static. That these assumptions can be relaxed will be shown in the following sections. The basic motion planning problem can be informally stated as getting from a start position to a target position without colliding with any obstacles in the robot's workspace (cf. Choset et al., 2004, Ch. 1). A formal problem description is provided in Chapter 4.3. In principal, the basic motion planning problem can be solved by following two steps:

1. Define a graph that represents the geometric structure of the environment and the objects contained therein.

2. Perform a graph search to find a path, preferably the shortest, from the node containing or representing the start position to the node containing or representing the target position.

A related problem is calculating the velocities or movement commands that, upon successful execution, drive the robot along the planned path to the target position. Approaches addressing this basic motion planning problem can be roughly classified into three classes – namely *roadmap*, *cell-decomposition* and *potential field approaches*. It should, however, be noted that these approaches are not mutually exclusive and some practical implementations of a motion planner, e.g. (Masehian and Amin-Naseri, 2004), use a combination of them.

4.1.1 Roadmap-based Approaches

Roadmap-based approaches utilize a network of valid paths, each connecting two positions in the robot's environment (Latombe, 1991). In roadmap-based approaches the path from a start position to a goal position simply results from connecting the robot's initial position or start position as well as the target position to the roadmap and adding edges from the roadmap so that start position and target position are connected. Adding these edges is normally carried out by searching for the shortest path in the graph structure of the roadmap. This is described in more detail in Chapter 4.3. The main problem lies in finding appropriate roadmaps for which a large variety of approaches have been proposed. The roadmaps originate, for example, from visibility graphs (Nilsson, 1969; Siméon et al., 2000; Wooden and Egerstedt, 2006) or Voronoi diagrams (Choset et al., 1997; Garrido et al., 2006; Bhattacharya and Gavrilova, 2008).

Another important issue in roadmap-based approaches is how to handle dynamic environments and especially obstacles that are not modeled in the robot's internal environment representation at the time of computing the roadmap. Here, completely recomputing the roadmap might be computationally too expensive. Kavraki et al. (1995) and Hsu et al. (1999) apply random sampling to avoid that the roadmap is completely recomputed in cases of suddenly appearing obstacles. In probabilistic roadmap approaches, samples are randomly drawn from the configuration space. It is then checked whether or not these samples lie in free space and a local planner is used that tries to connect free configurations to other free configurations nearby. The roadmap is then formed by the resulting connections (cf. LaValle, 2006, Ch. 2). Different sampling techniques and procedures for adding nodes to probabilistic roadmaps are addressed in (Geraerts and Overmars, 2006) and (Hsu et al., 2006).

Especially in environments that contain narrow passages, the efficiency of probabilistic roadmap methods decreases dramatically since, depending on the distribution where samples are drawn from, having enough samples to find paths trough narrow passages might require a vast amount of samples in the overall configuration space. This issue is addressed in (Saha et al., 2005). Saha et al. artificially widen the free configuration space and construct a probabilistic roadmap. By widening the free space more samples are drawn in narrow passages. Once the roadmap is constructed, they retract the contained configurations so that they lie in the original free configuration space. A comprehensive overview on other extensions can be found in (Choset et al., 2004, Ch. 7).

4.1.2 Potential Field Methods and Vector Field Histograms

Another group of approaches is based on the potential field method introduced by Oussama Khatib (Khatib, 1986). These artificial potential fields are similar to those found in physics, e.g. the Coulomb field. The idea is that, for every position in the field, a vector or the gradient of a function determines the desirable behavior of the robot by modeling, for example, velocities or torque applied to the robot. The key concept is based on attractive and repulsive potential: the goal attracts the robot while obstacles repel it (Choset et al., 2004, Ch. 4). The goal thereby forms a global minimum. The inherent drawback of this approach is that the field might contain local minima that are not identical to the goal. That the robot might get stuck in local minima is a general problem in potential field methods. Strategies for determining whether or not the robot is stuck in a local minimum and how it should behave in this situation have been proposed by Ratering and Gini (1993).

A mechanism, that avoids that the robot gets stuck in local minima, called *forward chaining* has been presented by Bell and Weir (2004). They create a set of subgoals between start and target position. The robot continues to travel to each of these subgoals in turn until it has reached the goal. Mabrouk and McInnes (2008) model the internal state of the robot as a dynamical system in terms of first order differential equations. This dynamic model is used to manipulate local minima in the potential field by transforming stable equilibria into unstable ones.

Another idea of avoiding persistent local minima are *randomized potential fields*. Balch and Hybinette (2000), for example, add local random fields of low potential to the global potential field. Furthermore, they take into account the state of other robots and refer to the resulting field as a *social potential field*. Other approaches add random movements and walks to escape local minima (LaValle, 2006, Ch. 5).

Borenstein and Koren (1991) introduced the concept of *vector field histograms (VFH)*. In a multi-step procedure they calculate Cartesian and polar histograms representing portions and local characteristics of the configuration space. From these histograms they derive velocities for steering the robot (Ulrich and Borenstein, 1998). In (Ulrich and Borenstein, 2000) they combine the vector field approach with $A^\star$ search to achieve a more goal-directed behavior of the robot and to avoid that it gets stuck in local minima.

4.1.3 Cell Decomposition-based Approaches

Another group of approaches is based on a *cellular decomposition* of the configuration space (Schwartz and Sharir, 1983a,b; Marchand, 1988; Canny, 1989). In these approaches the free configuration space is decomposed into cells, e.g. quadrats or cells of a quadtree in 2D space. A connected subset of cells, that contain both the start position and target position, then forms a so-called *channel* that contains valid paths from start to target position. Representing the channel in form of a graph then allows to apply any search algorithm for shortest paths in graphs, e.g. *Dijkstra's algorithm* (Dijkstra, 1959), $A^\star$ (Hart et al., 1968), *value iteration* and *greedy best-first search* (LaValle, 2006), $D^\star$ (Stentz, 1994), *Focused* $D^\star$ (Stentz, 1995), *Learning Real-Time* $D^\star$ (Furcy and Koenig, 2000) and *Focused* $D^\star$ *Lite* (Koenig and Likhachev, 2002). The runtime of these algorithms primarily depends on the size of the emerging search space. Regarding the limited computational power on autonomous mobile robots, the processing time in fast control loops might not be enough to sufficiently explore the search space to find an optimal solution. To improve the real-time applicability of the search, approximations are used or the planning problem is divided into achieving several subgoals.

A common variant is to apply an approximate cell decomposition (Latombe, 1991, Ch. 6). In this context, representing the environment in form of a quadtree is also referred to as approximate cell decomposition. Here, a recursive procedure continues to subdivide the cells until the cell size falls below a lower bound or all cells lie either completely in free space or completely in space occupied by obstacles. Katevas et al. (1998) use an approximate cell decomposition together with a path refinement procedure that takes into account local characteristics of the workspace.

4.1.4 Decoupling Path Planning and Motion Control

A commonly found class of approaches decouples planning and control, e.g. by planning a path in two-dimensional space and applying reactive strategies or motion controllers for following the planned path. That is, the robot does not exactly follow the trajectory resulting from path or motion planning, but is controlled, for instance, by reactive collision avoidance that tries to maximize some navigation function that depends on the planned path (Latombe, 1991, Ch. 9). By this means, the computationally expensive path planning is conducted only once or at least not for every update of the robot's movement commands. In this context, path planning is often addressed in terms of cell decomposition as it is inherently provided by grid maps when only planning 2D vehicle poses. A common solution to control the robot so that it moves along the planned path is the *Dynamic Window Approach* by Fox et al. (1997). The dynamic window approach only considers the portion of the search space that can be reached by the robot in a certain time interval. Another approach in this context is the *Curvature-Velocity Method* by Simmons (1996). The possible trajectories in the examined local space are evaluated and selected in a way so that the robot tries to follow the planned path. A generalization of the dynamic window approach combining concepts from path planning and reactive collision avoidance has been proposed in (Brock and Khatib, 1999). Ko and Simmons (1998) combine the curvature-velocity method with a roadmap-based approach to improve the behavior of the robot when traversing longer corridors.

4.1.5 Overview on the presented Approach

In the context of this thesis, two particular problems are addressed:

Problem 1.) Reaching a designated position in the robot's immediate vicinity (possibly under a designated orientation) and

Problem 2.) reaching a designated position somewhere in the robot's workspace.

Assuming that there is no obstacle between the robot and the goal position, the first problem can be coped with by merely applying a motion controller based on the robot's kinematic or dynamic model. Chapter 4.2 describes the simple motion controller used here. It drives the robot along a smooth trajectory from the robot's initial vehicle pose to a target pose. Upon successful completion of the trajectory the robot ends up at the target position under the designated orientation.

The second problem involves planning an obstacle-free path ending up in the goal position and a controller enabling the robot to follow that path (preferably avoiding obstacles blocking its way). Here, it should be noted that this approach is highly simplified in the context of motion planning since neither kinematic and dynamic constraints nor additional goal specifications like for instance velocities or accelerations at certain states of the robot are taken into account in the planner. However, such a decomposition into a low-level controller dealing with these constraints and a planner operating on a more abstract layer is quite common in plan-based robot control (Beetz et al., 2001; Beetz, 2003) and will suffice in the context of the work presented here. Problems where the goal specification does not only include position but also orientation of the robot can be solved by first planning and following a path towards the goal position (solving Problem 2) and, once the robot has arrived in the vicinity of the goal position, applying a motion controller that also takes into account the orientation (solving Problem 1). Planning paths on the different types of environment representations used in the last chapter, namely (sparse) point maps and probabilistic grid map, is described in Chapter 4.3 and Chapter 4.4 respectively. The motion controller for following planned paths is presented in Chapter 4.5.

4.2 Motion Control for Reaching Target Poses

Driving the robot towards a designated target pose in the robot's vicinity is a quite simple task. In fact, if the robot is able to turn on the spot and no obstacles are blocking its way, the desired behavior can be achieved by simply

1. turn the robot on the spot towards the target position,

2. drive the robot along a straight line until it reaches the target position and
3. turn the robot on the spot to reach the target orientation.

However, this fragmented movement is rather awkward and might lead to problems especially in situations where the target position is right in front of an obstacle where the robot can not freely turn on the spot. Furthermore, this simple technique can not be applied to robots that can not turn on the spot, i.e. moving with zero linear velocity on an infinite curvature. This class of vehicles is often referred to as having *bicycle-like kinematics* as opposed to the *unicycle model* that generalizes, amongst others, the common differential drive (Aicardi et al., 2000).

In 1999, Indiveri proposed a motion controller that yields smooth trajectories while taking the aforementioned inability to move on infinite curvatures and other kinematic constraints of the vehicle, like for instance linear velocity saturation, into account. That is, his controller can be used for a wide class of *non-holonomic* vehicles what fosters the desired general applicability of algorithms used in the context of the work presented in this thesis. The key idea underlying Indiveri's controller is to choose a discontinuous state representation to overcome *Brockett's Theorem* (Brockett, 1983) that would otherwise prevent the global asymptotic stability of such a feedback controller (c.f. Indiveri, 1999).

4.2.1 Pose Representation and Kinematic Model

The general concept of a closed-loop controller is to monitor a system's output and to use this feedback to determine the control error in form of the deviation between the designated and the actual state of the system. In order to reach and maintain a designated state, this error has to be minized by appropriately changing a control signal, the input to the system. In the context of motion control, the designated state can be for example the specification of a target pose $\mathcal{P}_{\text{target}}$ given, respectively, in the world coordinate frame $\{W\}$ and the coordinate frame of an allocentric world representation like sparse point maps as:

$$^{\{W\}}\mathcal{P}_{\text{target}} = \begin{pmatrix} x & y & z & \theta_x & \theta_y & \theta_z \end{pmatrix}^T \tag{4.1}$$

The deviation between the target pose and the robot's current pose forms the error signal of the motion controller. The controller's output used to steer the robot is a vector containing linear and angular velocities. The controller changes the velocities over time according to the error signal and thereby drives the robot towards the designated target pose. Depending on the specification of designated state, error and output signal, other types of motion controllers can be defined, for example, for maintaining a certain vehicle speed or to drive along a given path. For reaching a target pose, the error signal $E(t)$ can be defined as

$$E(t) = \Delta\mathcal{P}(t) = {}^{\{W\}}\mathcal{P}_{\text{target}} - {}^{\{W\}}\mathcal{P}(t) \tag{4.2}$$

where $\mathcal{P}(t)$ is the robot's pose at time t in frame $\{W\}$. The design goal for a motion controller approaching a target pose is the convergence of $E(t)$ to zero. Hence, the controller needs to choose appropriate velocities, i.e. linear velocity $v(t)$ and angular velocity $\omega(t)$, such that $E(t)$ becomes zero ($E(t_1) = 0$) within some finite time t_1 and to maintain the state in which $E(t)$ became zero such that $E(t_1 + t_2) = 0$ for some positive duration t_2.

The basic concept underlying Indiveri's controller is an alternative state space representation that is directly used in the error signal. Therefor, the controller is not operating in the world coordinate frame $\{W\}$ but in the local coordinate frame of the designated target pose, the *goal frame* $\{G\}$. Mapping between these two frames involves simple rigid transformations (cf. Craig, 1989, Chapter 2). The transformation ${}^{W}_{G}\mathbf{T}$ mapping a point in the goal frame $\{G\}$ into the world coordinate frame $\{W\}$ is

$${}^{W}_{G}\mathbf{T} = \begin{bmatrix} & {}^{W}_{G}\mathbf{R} & & {}^{W}\mathbf{P}_{GORG} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.3}$$

where ${}^{W}_{G}\mathbf{R}$ is given by the orientation of the designated target pose and ${}^{W}\mathbf{P}_{GORG}$ is the origin of $\{G\}$ specified in $\{W\}$ and given by the target position. Under the assumption that the robot

moves on the xy-plane, the rotation ${}^{W}_{G}\mathbf{R}$ involves a rotation $\mathbf{R}_{\theta_z}$ around the z-axis with the angle θ_z specified as the heading angle of the robot in the target pose and the translational component ${}^{W}\mathbf{P}_{GORG} = ({}^{W}x_G \ {}^{W}y_G)^T$ is given by the robot's position on the plane. Using the inverse ${}^{W}_{G}\mathbf{T}^{-1}$ allows to map the robot's current pose in the world frame $\{W\}$ into the goal frame $\{G\}$ (using matrix notation):

$$
\begin{aligned}
{}^{W}_{R}\mathbf{T} &= {}^{W}_{G}\mathbf{T}\ {}^{G}_{R}\mathbf{T} && (4.4)\\
\Longrightarrow \quad {}^{G}_{R}\mathbf{T} &= {}^{W}_{G}\mathbf{T}^{-1}\ {}^{W}_{R}\mathbf{T} && (4.5)
\end{aligned}
$$

where ${}^{G}_{R}\mathbf{T}$ is the homogeneous transformation matrix representing, respectively, the robot's current pose and its local coordinate frame $\{R\}$ in the goal frame $\{G\}$.

In addition to this mapping, Indiveri uses a bicycle-like kinematic model and a polar-like representation. The bicycle-like kinematic model is given as

$$
\begin{aligned}
\dot{x} &= v\cos\phi && (4.6)\\
\dot{y} &= v\sin\phi && (4.7)\\
\dot{\phi} &= \frac{\tan\psi}{l} = vc && (4.8)
\end{aligned}
$$

and models the changes in the robot's pose $(x\ y\ \phi)^T$ w.r.t. time given the robot's linear velocity v in the direction ϕ, the steering angle ψ, the length of the vehicle l and the curvature c, that is the reciprocal of the radius of the arc the robot is traversing.

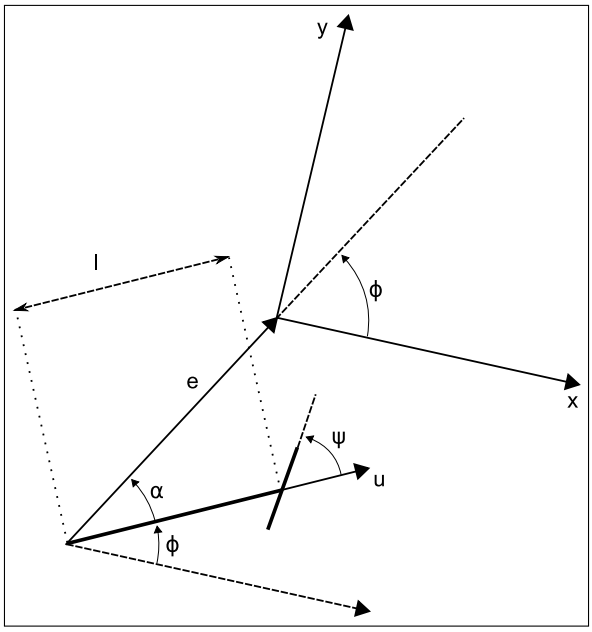


Figure 4.1: Visualization of the bicycle-like kinematic model.

This Cartesian model is then transformed into a polar-like representation (see Figure 4.1)

$$
\begin{aligned}
\dot{e} &= -v\cos\alpha && (4.9)\\
\dot{\alpha} &= -v\left(c - \frac{\sin\alpha}{e}\right) && (4.10)\\
\dot{\theta} &= v\frac{\sin\alpha}{e} && (4.11)
\end{aligned}
$$

with the triple (e, α, θ) representing the state of the robot and the triple $(0, 0, 0)$, being the space's origin, is formed by the goal pose. That is, (e, α) defines the robot's current location in the polar-like goal frame and θ its heading. Here, e is the Euclidean distance and α the direction to the target pose.

4.2.2 Steering and Velocity Control Law

The intention of the controller is to guarantee that the robot's pose (e, α, θ) asymptotically converges to, respectively, the origin $(0, 0, 0)$ and the target pose. Instead of directly controlling v and ω, an appropriate linear velocity v depending on the Euclidean distance e to the target position and a smooth curvature c for reaching the target position under the designated orientation is calculated. In a Lyapunov-like synthesis, Indiveri derives the following two control laws:

$$v = \gamma e \tag{4.12}$$

$$c = \frac{\sin\alpha}{e} + h\frac{\theta}{e}\frac{\sin\alpha}{\alpha} + \beta\frac{\alpha}{e} \tag{4.13}$$

with γ, h and β being positive parameters determining the controller's behavior.

Indiveri suggests to use $\gamma = 1$ turning Eq. (4.12) into a simple proportional controller (P-Controller) with the Euclidean distance to the target position being the error signal. Furthermore, the following three kinematic constraints are integrated:

1. The linear velocity should never be negative, as a wide variety of vehicles can not move backwards.

2. The linear velocity should be bounded by some upper value to avoid large lateral acceleration cv^2 and actuator saturation.

3. The curvature should be bounded, so that $\dot{\phi} = 0$ whenever $v = 0$ to take into account that a wide variety of vehicles can not turn on the spot.

The first constraint necessitates that $\gamma > 0$. The second constraint is met by turning Eq. (4.12) into

$$v = \begin{cases} \gamma e & \text{if } \gamma e < \bar{v} \\ \bar{v} & \text{otherwise} \end{cases} \tag{4.14}$$

saturating v with some upper bound $\bar{v}$. To meet the third constraint, h and β must fulfill:

$$h < 1;\ 2 < \beta < h + 1$$

Indiveri also provides a proof of convergence and stability and suggests the following values for the control parameters that will be also used here:

$$\gamma = 1, \beta = 2.9, h = 2$$

In cases where $\alpha \approx 0$, i.e. when the robot is oriented towards the goal, c can be approximated to follow a straight line turning Eq. (4.13) into

$$c = \begin{cases} \frac{\sin\alpha}{e} + h\frac{\theta}{e}\frac{\sin\alpha}{\alpha} + \beta\frac{\alpha}{e} & \text{if } |\alpha| > \epsilon_\alpha \\ \frac{\alpha}{e}(1+\beta) + h\frac{\theta}{e} & \text{otherwise} \end{cases} \tag{4.15}$$

with some small angle threshold ϵ_α. Note that neither functions are defined for $e = 0$. However, with $e = 0$ the robot has already reached the target position and, under the assumption of a successful application of the controller outputs, the robot should have reached the target position under the designated orientation. An orientation at the target position that deviates from the designated orientation is an error that needs to be handled outside of the controller.

The output of the controller, i.e. $(v, \omega = vc)$ is then given to the robot as a control input that results in driving the robot towards the goal position and finally reaching the goal position under the designated orientation. For a proof of stability and convergence it is referred to (Indiveri, 1999).

4.2.3 Implementation Details

In the actual implementation, Indiveri's controller is embedded into a reactive behavior within the robot control architecture (see Holz, 2007). Once initialized by specifying the goal pose, the behavior and the motion controller are updated with every update of the robot's pose estimation. The controller outputs (v, ω) are temporarily stored as designated velocities in the robot's state representation. Note that these values can be changed by other reactive behaviors e.g. for collision avoidance (see Chapter 2.4). In predefined intervals according to the used robot platform, the velocities are used to calculate the actual set speeds for the motor controllers.

When initializing the behavior by specifying a new target pose, the error states are reset and the homogeneous transformation matrix ${}^{G}_{R}\mathbf{T}$ mapping the robot's current pose into the Cartesian goal frame is calculated. For subsequent updates and re-calculations of (v, ω), the robot's pose is, first, transformed into $\{G\}$ according to Eq. (4.5) and then transformed into the polar-like representation according to Equations (4.9 - 4.11). Based on the resulting triple (e, α, θ), the linear velocity v, the curvature c and the angular velocity ω are re-calculated.

For applying the controller in a real-world application, i.e. where ideal robot movement can not be assumed and, furthermore, the accuracy of pose estimations is limited, the target pose should have a certain tolerance. Instead of waiting for the convergence $(e, \alpha, \theta) \longrightarrow (0, 0, 0)$, the target pose is assumed to be reached if the Euclidean distance to the target position is smaller than some threshold ϵ_e and the deviation between goal orientation and the robot's current heading is smaller than some threshold ϵ_θ. This thresholding, i.e. checking whether $((e < \epsilon_e) \wedge (\theta < \epsilon_\theta))$ is met, is conduceted in the behavior's update procedure before re-calculating the controller outputs. Once the above termination criterion is met, the behavior stops the execution of the motion controller and deactivates itself. Appropriate values for ϵ_e and ϵ_θ depend on the task and the accuracy of the robot's pose estimation.

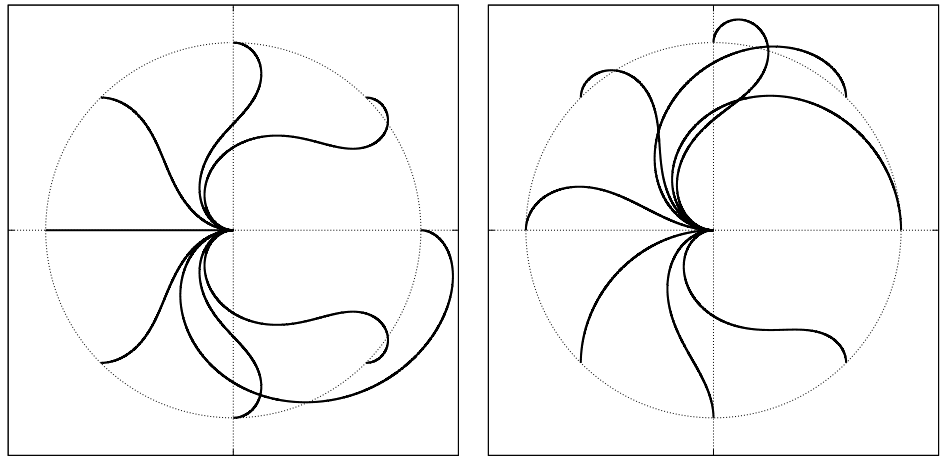
Also to account for the aforementioned situation that the robot reached the target position under an orientation other than the designated, the behavior implements a state machine. As long as $(e \geq \epsilon_e)$ and $(\theta \geq \epsilon_\theta)$ the motion controller is applied and the velocities v and ω are determined according to Eq. (4.14) and Eq. (4.8) respectively. As soon as the Euclidean distance to the target position e gets smaller than the distance threshold ϵ_e, the target position is marked as being reached. If the robot is able to turn on the spot, it is turned in the direction of the target orientation until θ is smaller than the angular threshold ϵ_θ and the target pose is reached. If the robot is not able to turn on the spot, it is moved somewhere into the free space and the application of the controller is re-started from the robot's new location.

4.2.4 Results and Open Problems

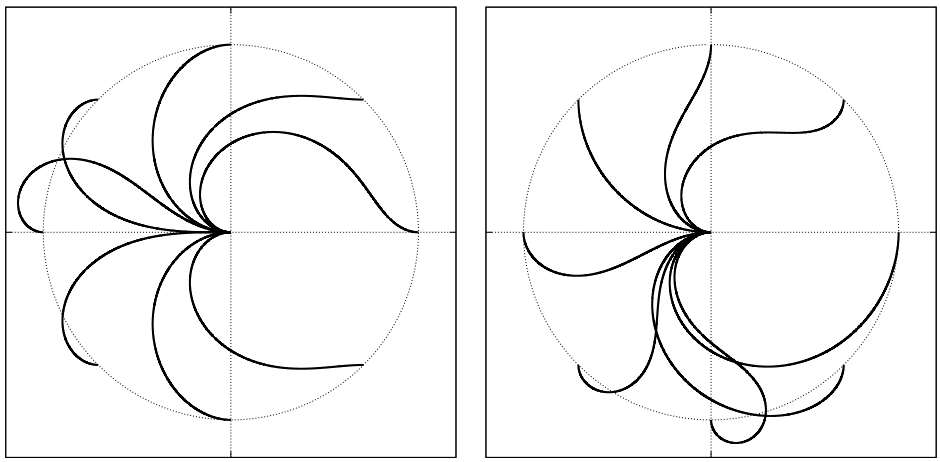
Indiveri presented a set of simulated tracks where the robot was starting from different positions on the unit circle in the goal frame $\{G\}$ under different initial orientations θ_z. This experiment has also been carried out here, as it nicely depicts the behavior of the control laws. The simulated paths of the robot are shown in Figure 4.2.

The implemented controller does not behave as the control laws proposed by Indiveri. When not explicitly limiting the range of all involved angles to the interval $[-\pi, \pi)$, the controller can oscillate in certain situations, due to the unavoidable limitation of the robot's heading angle θ_z. This can lead to oscillations of α between approximately $-\pi$ and π causing sign changes in the steering control law. It, however, reflects the behavior of the proposed control laws, that show much larger outputs in the steering angle $\alpha' = \alpha + n\pi, n \in \mathbb{N}$, than for $\alpha \in [-\pi, \pi)$. When explicitly limiting the range of all angles, including α, the controller becomes stable in all situations. This, however, further bounds the curvature corresponding to the maximum angular deviations at $\alpha = -\pi$ and $\alpha = \pi$. It is a matter of future work to overcome these problems in the implementation, possibly cooperating with Giovanni Indiveri.

The workaround used here, is to only apply the controller when the goal position is in front of the robot, i.e. $-\frac{\pi}{2} < \alpha_0 < \frac{\pi}{2}$, where α is the initial angle deviation. This, however, does not pose any limitation in the context of the work presented here as the controller's implementation was intended to be used only for approaching goal poses in the robot's immediate vicinity e.g. in



(a) $\theta_z = 0$ (b) $\theta_z = +\pi/2$



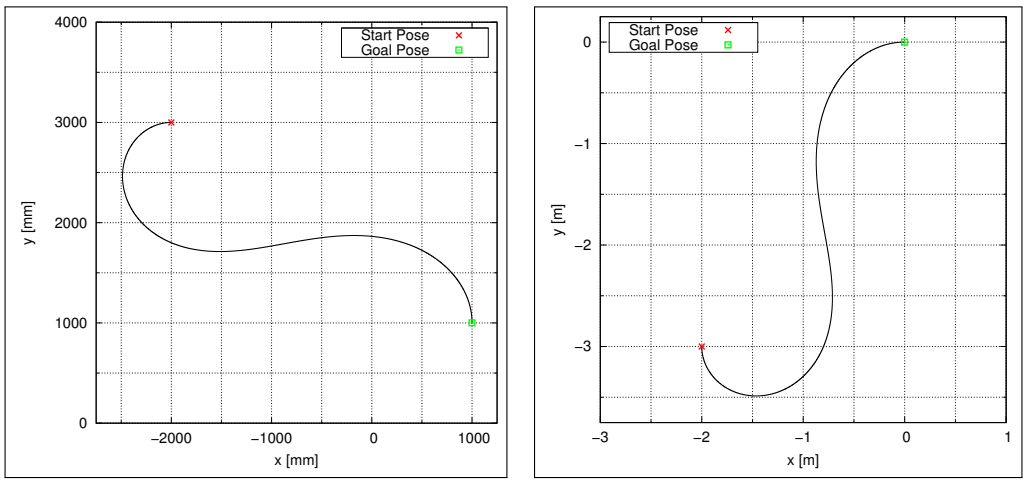
(c) $\theta_z = +\pi$ (d) $\theta_z = -\pi/2$

Figure 4.2: Resulting trajectories for an ideal robot starting on the unit circle in the goal frame $\{G\}$ under different initial orientations θ_z. The controller outputs yield smooth trajectories ending in G's origin, being the equilibrium point.

preparation of a manipulation task using a robot arm or the acquisition of new range measurements in the process of an exploration task. Another restriction used here is that the distance to the goal position is below a threshold $d_{\max}$ of e.g. 1.5 m and the expected track is obstacle free. The expected track can be simulated before the actual application by projecting controller outputs in time under the assumption of ideal robot movement. For the projected poses it can be tested whether the robot's bounding box, transformed w.r.t. its pose in the world frame $\{W\}$, intersects modeled environmental structures. Such procedure is also referred to as *Oriented Bounding Box Intersection Tests* (Fares and Hamam, 2005).

Approaching a target pose, where α_0 is outside of the aforementioned angular range, can be accomplished by turning the robot on the spot until α satisfies the constraint before applying the control laws. The advantage of this approach is that it does not directly impair the controller.

A final experiment, again adopted from (Indiveri, 1999), shows the behavior of a robot applying the implemented controller to reach the goal pose $(1m \quad 1m \quad -\pi/2)^T$ from $(-2m \quad 3m \quad 0)^T$ both specified in $\{W\}$. Figure 4.3 shows the resulting trajectory of a simulated ideal robot controlled via (v, ω) determined by Indiveri's controller both in goal frame $\{G\}$ and world coordinate frame $\{W\}$. The start pose is marked by a red cross, the goal pose by a green rectangle.



(a) Trajectory in $\{W\}$ (b) Trajectory in $\{G\}$

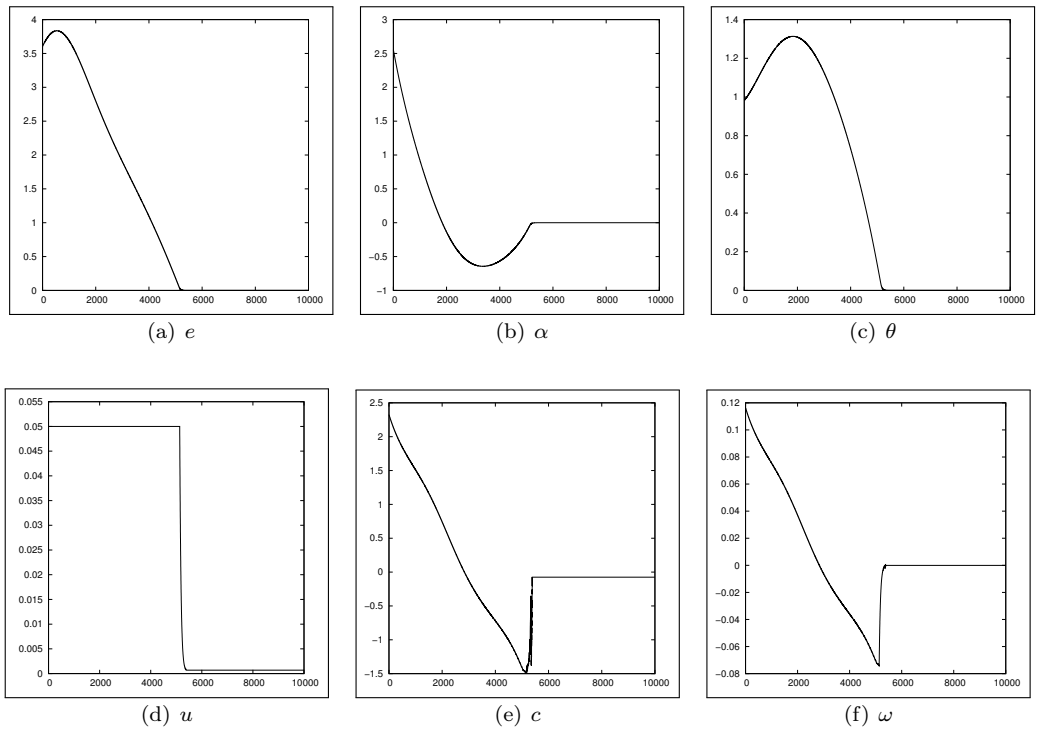
Figure 4.3: Example Application of the Motion Controller. Trajectory of the robot in the global world coordinate frame $\{W\}$ (a) and the goal frame $\{G\}$ (b) used by Indiveri's controller.

The evolution of the state triple (e, α, θ), i.e. the error signal, as well as the calculated velocities u and ω and the curvature c for this experiment are shown in Figure 4.4. With $\bar{u} = 0.05\,\mathrm{m\,s^{-1}}$, 10 000 controller updates finally led to $(e, \alpha, \theta) = (0, 0, 0)$.

Note that this controller is only applied if the target pose is in the immediate vicinity of the robot. Moving to a target position not in the robot's immediate vicinity but somewhere in its workspace will be addressed in the remainder of this chapter.

4.3 Path Planning in Point Maps

A simple motion controller like the one described in the previous section is not applicable for moving towards a target position not in the robot's immediate vicinity as obstacles, i.e. environmental structures and objects contained therein, are not taken into account in the calculation of the curvature determining the robot's path. In fact, the trajectory only depends on the initial conditions, i.e. position and orientation of the robot when starting the controller. To approach an arbitrary position in the robot's workspace, an obstacle-free path need to be planned which upon successful traversal leads the robot to the target position. Once the goal position is in the robot's immediate vicinity, a simple motion controller can be applied to also reach a target orientation.



(a) e (b) α (c) θ

(d) u (e) c (f) ω

Figure 4.4: Evolution of error (e, α, θ), and outputs u, c and ω.

The problem of path planning can be formulated as follows: Given the specifications of start pose $\mathbf{x}_{\text{start}}$, goal pose $\mathbf{x}_{goal}$ and a model of the environment M, find a finite sequence of states, state transitions or actions ending up with the robot's current state $\mathbf{x}$ fulfilling the goal specification. Compared to *motion planning*, here only the path is planned and velocities, accelerations or time are not taken into account. Still, path planning is a highly complex problem as, up to now, the search space being the robot's configuration space is continuous and possibly infinite. The configuration space $\mathcal{Q}$ of a robot system, also called *C-space*, is the space of all possible configurations q of the system where q is a complete specification of all points of the system (cf. Choset et al., 2004, Chapter 3.1). In 3D space a configuration q fully determines the position of each point on the robot system in $\mathbb{R}^3$. Assuming that the position of each point on the system relative to the robot's center of rotation is known, the configuration space is $\mathcal{Q} \subseteq \mathbb{R}^3 \times [-\pi, \pi)^3$ including the position of the robot's center of rotation in $\mathbb{R}^3$ and the robot's orientation about the three coordinate axes $\hat{X}$, $\hat{Y}$ and $\hat{Z}$ (cf. Craig, 1989, Chapter 2).

Note that the state of a *non-holonomic* systems is path-dependent and the number of coordinates required to represent a system's configuration q completely is more than its (*controllable* or *differentiable*) degrees of freedom (DDOF) since a return to an initial configuration q_0 does not guarantee the return to the initial state $\mathbf{x}_0$, i.e. $(q_t = q_0) \rightarrow (\mathbf{x}_t = \mathbf{x}_0)$ does not hold (cf. Choset et al., 2004, Chapter 12). In the context of path planning, this characteristic and other kinematic or dynamic constraints of the vehicle can be neglected.

Throughout this thesis it is assumed that the mobile robot is wheeled and its movement is limited to almost planar environments. Using e.g. 2.5-dimensional environment representations such as *elevation maps* (Pfaff et al., 2007) to account for slightly uneven instead of fully planar terrain, allows to reduce complete specification of q to the robot's position in the $\hat{X}\hat{Y}$-plane, i.e. in $\mathbb{R}^2$ and its orientation around the $\hat{Z}$-axis, i.e. $\mathcal{Q} \subseteq \mathbb{R}^2 \times [-\pi, \pi)$. As this reduced configuration space is still continuous, existing approaches often apply randomized sampling, e.g. by using *Rapidly-exploring random trees (RRTs)* (Kuffner and LaValle, 2000), or approximate the problem. A commonly used approximation is to extract a set of distinct locations within the environment and their topological structure to construct a graph where the distinct locations form the nodes or vertices and their

connectivity or *traversability* the edges in the graph. Path planning in the continuous state space thereby becomes the search for a shortest path in a graph. In the following, we will refer to such a graph as the *path graph*. For a comprehensive overview on path planning problems and existing approaches it is referred to (Bennewitz, 2004).

4.3.1 Path Planning in Graphs

Instead of searching in the continuous configuration space of the robot, the search space of graph-based path planning algorithms is limited to a discrete search graph. Each node in the graph represents not only some distinct location within the environment as mentioned above, but a valid configuration of the robot system, i.e. the robot's bounding box transformed w.r.t the configuration is not allowed to intersect any objects in the environment model. Under the assumption that a location is a valid configuration regardless of the robot's orientation θ_z, θ_z can be completely neglected and each node in the graph simply represents a state $\mathbf{x}$ solely containing the robot's position $(x\ y)^T \in \mathbb{R}^2$. Assuming that the start position $\mathbf{x}_{\text{start}}$ and the goal position $\mathbf{x}_{\text{end}}$ are nodes in the graph, the problem of searching for a shortest path between $\mathbf{x}_{\text{start}}$ and $\mathbf{x}_{\text{end}}$ can be formulated as follows (cf. Diestel, 2005):

> Given a graph $G = (V, E)$, where V or $V(G)$ is the set of vertices or nodes of the graph G and E or $E(G)$ the set of all edges in G, a start node $\mathbf{x}_{\text{start}} \in V(G)$ and a goal node $\mathbf{x}_{\text{goal}} \in V(G)$, find a sequence of states/nodes $< \mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_N >$, with $\mathbf{x}_0 = \mathbf{x}_{\text{start}}$, $\mathbf{x}_n = \mathbf{x}_{\text{goal}}$ and where every state $\mathbf{x}_i, i \in 0 \ldots N$ is represented by a node in graph G, i.e. $\mathbf{x}_i \in V(G) \forall i$, end every state transition $\dot{\mathbf{x}}_{i,j}, i \in 0 \ldots N-1, j = i+1$ is represented as an edge in G, i.e. $\dot{\mathbf{x}}_{i,j} = (\mathbf{x}_i, \mathbf{x_j}) \in E(G) \forall i, j$.

Note that here a node n_i and the state $\mathbf{x_i}$ it is representing are both simply named $\mathbf{x_i}$, although the node not only contains a state but also information about cost and neighboring or adjacent nodes.

The idea of graph search algorithms is to assign a cost $f(n)$ to every node n according to the actual path planning problem. As it is searched for the shortest path $f(n)$ is commonly defined by using some distance metric. By evaluating the cost of travelling from one node to another the graph is extended by assigning weights to the edges according to the cost of adjacent nodes forming a weighted graph (see Diestel, 2005).

4.3.2 Graph Representations

A path graph is nothing else but a topological environment representation, where relative distances between nodes, i.e. positions (x, y) in the robot's environment model, play an important role as it is searched for the shortest path w.r.t. travelled distance. Nodes are formed by positions and the corresponding configurations in state space, whereas an edge represents that the robot can travel along that very edge between the connected nodes. Distances between connected nodes are either directly encoded as weights on the edges of the graph or can be derived from the states the connected nodes are representing. A simple example for such a graph is shown in Figure 4.5.

The graph contains six nodes as well as directed and undirected (bidirectional) edges. Note, that the (dashed) self-connection at node '6' can be neglected here, but is for example used in dynamic path planning, where edge weights vary over time, to account for the cost of weighting at a certain position. Basically two design choices influence the complexity and runtime of path planning in graphs. That is the choice of the graph representation and the choice of, respectively, the search algorithm and in which order nodes in the search tree are expanded. There are many different ways of representing such graphs briefly summarized in the following. For an comprehensive overview on graph representations and graph theory it is referred to (Diestel, 2005, Chapters 3, 4 and 6) and the Wikipedia article[1] on *graph theory*. Again, here the node representing a position (x_i, y_i)

[1] The Wikipedia article on graph theory (`http://en.wikipedia.org/wiki/Graph_theory`, accessed December, 2008) provides a brief introduction into graph representations and a list of links to related articles.

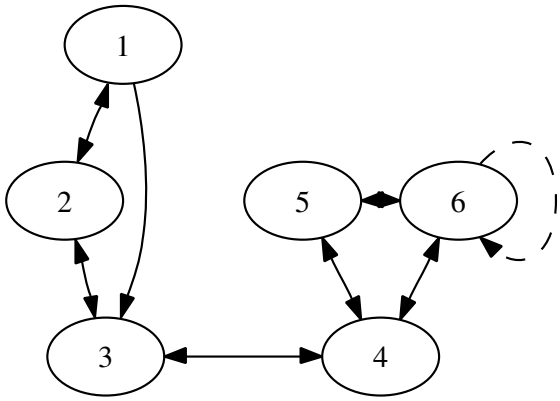


Figure 4.5: Example Graph

in a vehicle state $\mathbf{x}_i$ is named after that state, e.g. node '1' is named $\mathbf{x}_1$ in the following whereas the edge connecting nodes '1' and '2' is $(\mathbf{x}_1, \mathbf{x}_2)$.

An *Incidence List* is a list of pairs $(\mathbf{x}_i, \mathbf{x}_j)$ representing a directed edge from node $\mathbf{x}_i$ to node $\mathbf{x}_j$. Undirected edges are normally represented by the existence of both $(\mathbf{x}_i, \mathbf{x}_j)$ and $(\mathbf{x}_j, \mathbf{x}_i)$, e.g. $(\mathbf{x}_1, \mathbf{x}_2)$ and $(\mathbf{x}_2, \mathbf{x}_1)$ in Figure 4.5. An incidence list representing the complete example graph would contain 14 edges including the self-connecting edge at node '6'. In a *weighted incidence list* the pairs are augmented with a cost w_{ij} to $(\mathbf{x}_i, \mathbf{x}_j, w_{ij})$, where the cost w_{ij} can be e.g. the Euclidean distance $\|\mathbf{x}_i - \mathbf{x}_j\|$. For retrieving a set of nodes $\mathbf{X}_N$ that are connected to a node $\mathbf{x}_n$, the complete list need to be searched requiring $O(e)$ for a linear search in the unordered list where e is the number of edges, i.e. $O(n^2)$ for a fully connected graph with n nodes. In addition, if the weights are not encoded in the pairs, they have to be derived for all elements in $\mathbf{X}_n$. The space complexity for incidence lists is, $O(n^2)$ for a fully connected graph containing n nodes.

When using *Adjacency Lists* to represent the graph, a search for $\mathbf{X}_n$ is, respectively, not necessary and has complexity $O(1)$. Here, each node $\mathbf{x}_i$ has its own list $\mathbf{X}_i$ of *adjacent* nodes, e.g. $\mathbf{X}_4 = \{\mathbf{x}_3, \mathbf{x}_5, \mathbf{x}_6\}$ in Figure 4.5. An undirected edge between $\mathbf{x}_i$ and $\mathbf{x}_j$, again, causes redundancy with $\mathbf{x}_j \in \mathbf{X}_i$ and $\mathbf{x}_i \in \mathbf{X}_j$, i.e. the space complexity is $O(n^2)$ for a fully connected graph containing n nodes. Using *weighted adjacency lists*, a node $\mathbf{x}_i$ stores not only the set of adjacent nodes $\mathbf{X}_i$ but also a weight vector $\mathbf{w}_i$ storing the weights w_{ij} for each $\mathbf{x}_j \in \mathbf{X}_i$.

Note that basically all graph representation show this space complexity for fully connected graphs. Furthermore, when all connections between nodes are bidirectional, i.e. traversable for the robot in both directions, and have the same cost in both directions, the path graph turns into a (weighted) undirected graph. In an incidence list, this will reduce the number of elements, whereas for adjacency lists the redundant storage is still necessary. That is, here is a trade-off between fast look-ups and memory requirement. Still, both incidence and adjacency lists have advantages in memory consumption compared to matrix representations when the connectivy of the nodes in the graph is sparse. Compared to adjacency lists, matrix representations provide a centralized storage instead of individual lists for each node but, except for incidence matrices, still allow for a direct look-up of sets of adjacent nodes instead of searching as in incidence lists. An *incidence matrix* $\mathbf{M}_{\text{Inc}}$ is a $|V| \times |E|$ $(0,1)$-matrix where $|V|$ is the number of vertices or nodes and $|E|$ the number of edges, i.e. the matrix has a row for each node and a column for each edge. A value $m_{ij} = 1$ represents that node $\mathbf{x}_i$ is incident upon edge e_j. If node $\mathbf{x}_i$ is not incident upon edge e_j, $m_{ij} = 0$. An incidence matrix is more or less only one possibility for storing an incidence list. The graph in

Figure 4.5 can be represented by the incidence matrix

$$\mathbf{M}_{\text{Inc}} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

The redundancy in columns for undirected edges is necessary to express the directed edge $(3, 4)$ in Figure 4.5. The self-connecting edge e_{14} of node 6 is represented by having only one 1-value in column 14. Note that some authors define the incidence matrix as being the transpose of $\mathbf{M}_{\text{Inc}}$ or as not being a $(0, 1)$-matrix but containing the node on the other side of an edge instead of '1' for representing incidence.

Storing adjacency lists in the rows of the matrix leads to adjacency matrices. An *adjacency matrix* $\mathbf{M}_{\text{Adj}}$ is a $|V| \times |V|$ $(0, 1)$-matrix that just like adjacency lists allow a direct look-up of $\mathbf{X}_n$ in its rows instead of searching through all columns in incidence matrices. If there is an edge $(\mathbf{x}_i, \mathbf{x}_j)$ in the graph, $m_{ij} = 1$. If no such edge exists, $m_{ij} = 0$. The adjacency matrix $\mathbf{M}_{\text{Adj}}$ representing the graph in Figure 4.5 is

$$\mathbf{M}_{\text{Adj}} = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

Note that despite for node 6 having a self-connecting edge, the diagonal elements are zero. An adjacency matrix augmented with a degree matrix, i.e. storing the number of connections in diagonal elements, is referred to as a *Laplacian* graph representation. It is defined as $\mathbf{M}_{\text{Deg}} - \mathbf{M}_{\text{Adj}}$ where $\mathbf{M}_{\text{Deg}}$ is the diagonal vertex degree matrix. Hence, off-diagonal elements m_{ij} are negative if an edge $(\mathbf{x}_i, \mathbf{x_j}) \in E(G)$ exists. For representing weighted edges by means of a *weighted adjacency matrix*, an edge $(\mathbf{x}_i, \mathbf{x}_j)$ is not represented with $m_{ij} = 1$ in the matrix but with $m_{ij} = w_{i,j}$. Using distances as weights results in $|V| \times |V|$ distance matrices where $w_{i,j} = dist(\mathbf{x}_i, \mathbf{x_j})$, with $dist(\mathbf{x}_i, \mathbf{x_j})$ being some distance matrix, if $E(G)$ contains the edge $(\mathbf{x}_i, \mathbf{x_j})$ and $m_{ij} = \infty$ if no such edge exists.

In addition to space complexity and the complexity of searching for the set of adjacent nodes $\mathbf{X}_i$ for a particular node $\mathbf{x}_i$, another important issue affects the choice for an appropriate representation, namely its administrability. For adding and removing nodes or edges both list representations and incidence matrices require searching through the stored edges or adjacency lists. In an adjacency matrix, adding or removing a node $\mathbf{x}_k$ corresponds to, respectively, adding or removing the k-th row and k-th column of the matrix. Adding or removing an edge $(\mathbf{x}_i, \mathbf{x}_j)$ necessitates only the modification of the value m_{ij}. In the case of distance matrices, adding an edge $(\mathbf{x}_i, \mathbf{x}_j)$ additionally requires the calculation of $m_{ij} = w_{ij} = dist(\mathbf{x}_i, \mathbf{x_j})$ For detailled descriptions of graphs and line graphs, the relation between their representations as well as applications and interesting problems in graph theory, it is referred to (Diestel, 2005).

4.3.3 Constructing Pathgraphs from Sparse Point Maps

Before being able to address path planning as searching for shortest paths in graphs, the question of how to obtain a path graph from a sparse point map need to be addressed.

Grid maps, for example, already have an internal structure that can directly be used to form a path graph by e.g. adding a node in the graph for every cell c_{ij} in the grid map M and edges for

every neighboring node of c_{ij}.

$$G = (V, E) \tag{4.16}$$
$$V = \{c_{i,j} \mid c_{i,j} \in M\} \tag{4.17}$$
$$E = \{(c_{i,j}, c_{k,l}) \mid c_{i,j} \in M, c_{k,l} \in M, c_{k,l} \in N_{r=1}^{M}(c_{i,j}) \setminus c_{i,j}\} \tag{4.18}$$

where $N_{r=1}^{M}(c_{i,j})$ is the *Moore neighborhood* of range $r = 1$. With respect to the center $(c_{i,j}^{x}\ c_{i,j}^{y})^T$ of cell $c_{i,j}$, the Moore neighborhood of range r for cell $c_{i,j}$ is defined as:

$$N_{r=1}^{M}(c_{i,j}) = \{c_{k,l} \mid (|c_{k,l}^{x} - c_{i,j}^{x}| \leq r), (|c_{k,l}^{y} - c_{i,j}^{y}| \leq r)\}. \tag{4.19}$$

As the Moore neighborhood of a cell includes the cell itself, $c_{i,j}$ is removed from the set of neighboring cells in Eq. (4.18). Note that in actual implementations such a graph is not explicitly build, but the structure of the grid map, e.g. an array, is directly used by manipulating indices for accessing neighboring cells and simply checking whether a cell is occupied or not (see Chapter 4.4).

Such an internal structure is, however, not given in the case of continuous, i.e. not discretized, geometric representations like the sparse point maps used here. These representations contain nothing more but an unordered set of features, a two-dimensional point cloud in the case of sparse point maps. The most obvious way to obtain such a structure, that can directly be used to form a path graph, is constructing the Voronoi diagram for the set of points contained in the map.

Voronoi Diagrams

The Voronoi diagram is a spatial decomposition determined by a set of points and a distance metric. The points are referred to as *Voronoi sites*. Each site $\mathbf{s}_i$ has a Voronoi cell $V(\mathbf{s}_i)$ containing all points $\mathbf{p}$ that are closer to $\mathbf{s}_i$ than to any other site $\mathbf{s}_j, j \neq i$. Those points $\mathbf{p}$ whose distance to the two closest Voronoi sites is equal, i.e. $\|\mathbf{p} - \mathbf{s}_i\| = \|\mathbf{p} - \mathbf{s}_j\|$ using Euclidean distances, form the *Voronoi segments*. Points $\mathbf{p}$ that are equidistant to the three closest sites, form the *Voronoi nodes*. Using the Voronoi nodes and segments to form a graph for path planning, benefits from the facts that

1. Voronoi tessellation, i.e. constructing Voronoi diagrams, is a well-known problem and several effective algorithms exist and
2. each edge in the graph will have the maximum distance to its neighboring points, i.e. environmental structures in the map.

The latter causes the fact that path planning algorithms might not find the optimal shortest path, but if a path is found it is the safest path given the knowledge about obstacles in the map. This, of course, only holds true under the assumption that all edges that can not be traversed by the robot are pruned (see e.g. Bhattacharya and Gavrilova, 2007). A spatial decomposition of a set of points by means of a Voronoi diagram is shown in Figure 4.6. The figure shows a set of points in the 2D plane together with the edges of the constructed Voronoi diagram. Those edges whose distances to the closest data points are smaller than the width of the robot need to be pruned in order to get a graph that can be used for path planning purposes.

Constructing Voronoi Diagrams

As constructing the Voronoi diagram for a set of points and its dual the *Delaunay Triangulation* are well known problems, e.g. in the field of Computational Geometry, several algorithms have been proposed over the last decades.

The well-known *Bowyer-Watson algorithm* (Bowyer, 1981; Watson, 1981) allows for incrementally constructing Delaunay triangulations and Voronoi diagrams as their dual. In contrast to static triangulation methods such as *divide-and-conquer* or *recursive split* algorithms, incremental or dynamic triangulation algorithms like *Bowyer-Watson* always carry a valid Delaunay triangulation during processing. It sequentially adds points to the set of already processed points, deletes neighboring triangles and corrects the triangulation. For a detailled description of the Bowyer-Watson algorithm and for an overview on several static and dynamic triangulation algorithms it is

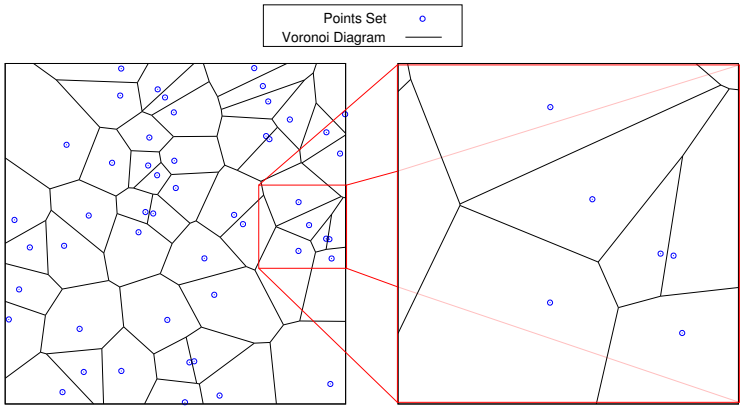


Figure 4.6: Voronoi diagram example. The figure shows (left) an example data set with 50 points (blue circles) drawn from a uniform distribution together with the 125 edges (black line segments) of the constructed Voronoi diagram. The scaled-up section (right) depicts how close edges of the diagram can lie to data points.

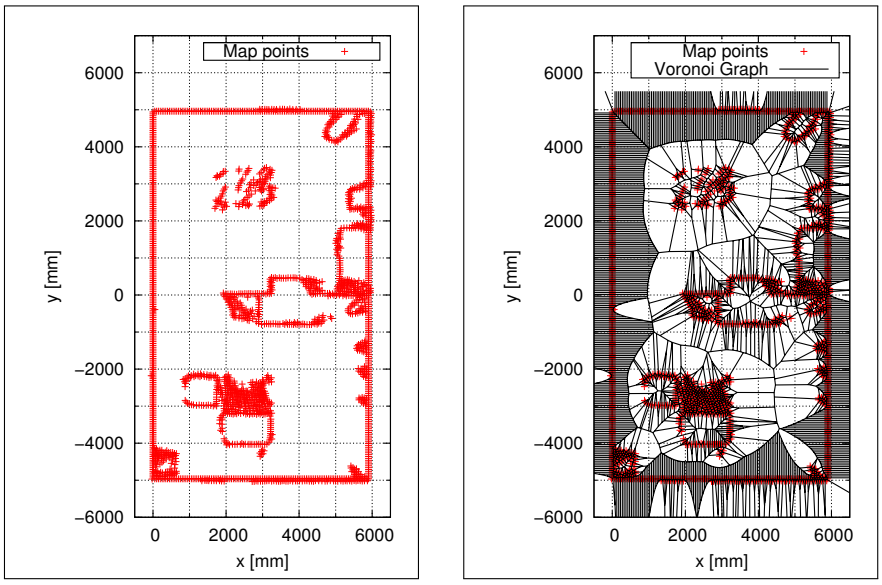
referred to (O'Rourke, 1994). The extensive software library *QHull*[2], primarily implementing the *QuickHull* algorithm (Barber et al., 1996) for computing convex hulls in high-dimensional spaces, contains several algorithms for computing Voronoi diagrams and Delaunay triangulations even in high-dimensional spaces. An incremental algorithm for determining the Delaunay triangulation for points in the 2D plane has been proposed in (Guibas et al., 1990). It, however, requires for determining the triangle in the current triangulation in which a point, used in the incremental update, is lying and how the triangulation needs to be corrected in order to consider that point.

In 1987, Fortune proposed the *sweepline algorithm*, that runs in $O(n \log n)$ with space complexity $O(n)$ to construct the planar Delaunay triangulation (and the Voronoi diagram) of n points in the $2D$ plane (Fortune, 1987, 1998). An efficient implementation in C++ of Fortune's algorithm has been proposed[3] by O'Sullivan. The algorithm, also referred to as *plane sweep*, maintains a *sweepline* and a *beach line*. Depending on the implementation, the sweepline moves along the $\hat{X}$ or $\hat{Y}$-axis in the coordinate frame of the map. Those points that lie behind the sweepline have already been considered forming a valid Delaunay triangulation. The points in front of the beach line have not yet been considered. As soon as the sweepline hits a point not yet considered, it is used to correct and update the so far built triangulation and Voronoi diagram. The beachline is a chain of connected parabolas each being defined as a parabola of points being equidistant to an already considered point behind the sweep line and the sweep line itself. Thereby, the beach line is moving behind the sweepline. The points in which two parabolas intersect (the connections of the chain) exactly fulfill the above definition of Voronoi segments, i.e. they are equidistant to the closest two points. In an actual implementation, the beach line is a priority queue of future events in the progression of the algorithm. Two types of events are distinguished: either a parabola is added to the chain when the sweepline hits a not yet considered point or a parabola is removed from the chain when the distance of the point, defining the parabola, to the sweepline is larger than the distance to the intersection of the two adjacent parabolas in the chain. The priority or event queue is initialized once before actually running the algorithm by sorting the points in the map according to the direction in which the sweepline and beachline are moving.

For a detailed description of Fortune's algorithm and a comprehensive overview on algorithms for Voronoi diagrams and Delaunay triangulations it is referred to (de Berg et al., 2008, Chapters 7, 9 and 11). A typical result of a Voronoi diagram constructed from a sparse point map using Fortune's algorithm is shown in Figure 4.7. The sparse point map in Figure 4.7.a has been constructed from a sequence of 2D laser scans taken in a small indoor environment by means of the incremental

[2]QHull is available at `http://www.qhull.org`
[3]Shane O'Sullivan's C++ implementation of Steven Fortune's sweepline algorithm is available at `http://sourceforge.net/projects/mapmanager`.

registration procedure presented in Chapter 3.5. The Voronoi diagram in Figure 4.7.b has been obtained by applying Fortune's algorithm. It is unbounded, i.e. the Voronoi edges that do not intersect any other edge are extended to infinity, as formulated in the algorithm (Fortune, 1987), and lie outside of the map, and unpruned, i.e. all Voronoi edges determined by the algorithm are shown in the graph. Except for some theoretical path planning problems where the robot is only a point in space (see e.g. Latombe, 1991), the resulting graph can not be used for path planning in the context of this thesis. For this purpose the Voronoi diagram needs to be bounded and pruned to form a path graph.



(a) Sparse Point Map (b) Voronoi diagram

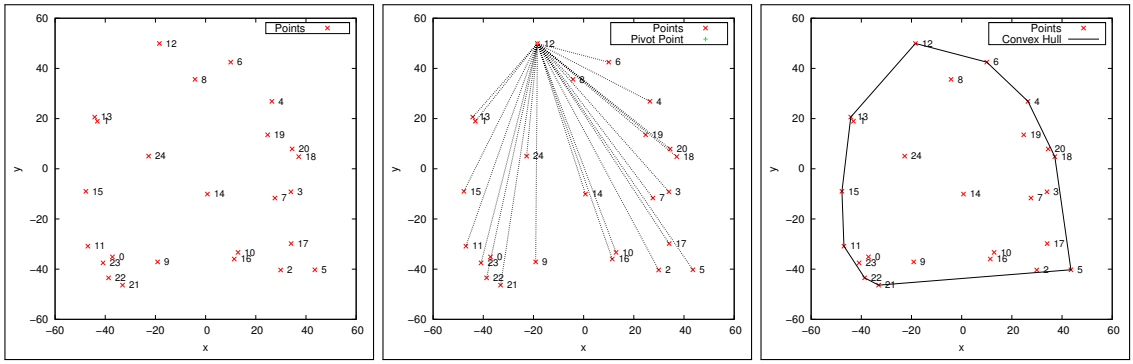
Figure 4.7: A sparse point map (a) constructed during an explorative run during the 2008 RoboCup @Home German Open in Hannover and a Voronoi diagram (b) constructed by means of Fortune's plane sweep algorithm. The Voronoi diagram is both unbounded and unpruned. It contains 5209 edges for 1904 2D points in the map.

Bounding and Pruning the Voronoi Diagram

Pruning the Voronoi diagram is necessary in order to guarantee that the resulting path graph contains only edges that can be traversed by the robot. As can clearly be seen in Figure 4.7.b, the Voronoi diagram contains edges intersecting objects and environmental structures where no measurements are stored in the sparse point map. These edges can not be traversed by the robot and should, thus, not be considered in path planning as the path that is searched for needs to be obstacle-free. Furthermore, the Voronoi diagram contains edges outside of the map, i.e. edges lying in not yet explored terrain.

The latter edges can be pruned quite easily by removing all edges where one or both endpoints lie outside of the modeled environment. For this purpose, the boundary of already modeled environmental structures needs to be determined. In the case of sparse point maps the boundary of stored points is the two-dimensional or three-dimensional convex hull of the points stored in the map. The convex hull $\mathcal{CH}(S)$ of a set S is the smallest convex set that contains S where a set P is defined to be convex if for any two points $p, q \in P$ the line segment $\overline{pq} \subseteq P$ (cf. O'Rourke, 1994, Chapter 3). Several efficient algorithms for computing the convex hull of two-dimensional and three-dimensional point sets have been proposed over the last decades, e.g. the *Quickhull algorithm* (Barber et al., 1996). In the context of this thesis, navigation is restricted to 2D point maps and 2.5D point maps extracted from 3D information. For computing the convex hull of these point maps, we use Graham's scan algorithm (Graham, 1972). It consists of three phases. In the

first phase it selects a point that is definitely a part of the convex hull, e.g. the with smallest or largest coordinate in either x-or y. This point is then referred to as the *pivot* point. In the second phase, all other points from the point set are ordered with respect to their orientation to the pivot point. Points with the same orientation to the pivot point are ordered w.r.t. to the distance to the pivot point. Sorting the points is the computationally most expensive step yielding an overall complexity of $O(n \log n)$ for n points using standard search algorithms, e.g. *sort* from the C++ Standard Template Libraries (STL). STL sort implements the *introsort* algorithm (Musser, 1997) whose worst case complexity is $O(n \log n)$. In the third phase, Graham's scan runs over the sorted point set and checks which points are contained in the convex hull. Actual implementations use a stack or queue. The pivot and the next point in the ordered point set are added to the queue and form the initial hull. If point i is examined, it lies outside of the so far constructed hull and it is examined how the hull changes when adding the point. If the points lies, respectively, left and right, according to whether the points are ordered clockwise or anti-clockwise, it is either added as a new point on the hull or the last point on the hull is removed from the queue, i.e. *backtracking*. For a detailled description of the algorithm it is referred to (Graham, 1972). Typical results for the three phases of the algorithm applied to a random point set is shown in Figure 4.8.



(a) (b) (c)

Figure 4.8: Example for computing the convex hull of a 2D point set by means of Graham's scan. Shown are a randomly generated set of points in the 2D plane (a), the found pivot point (b) and the computed convex hull (c).

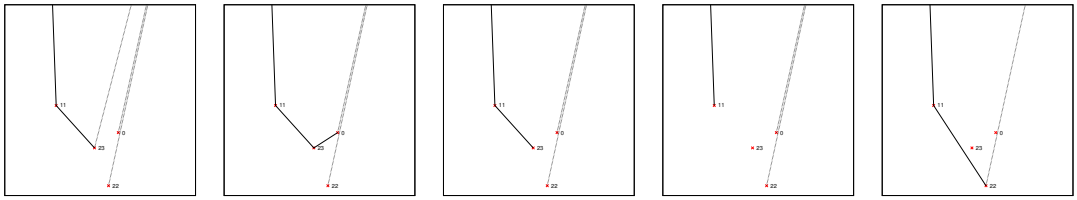
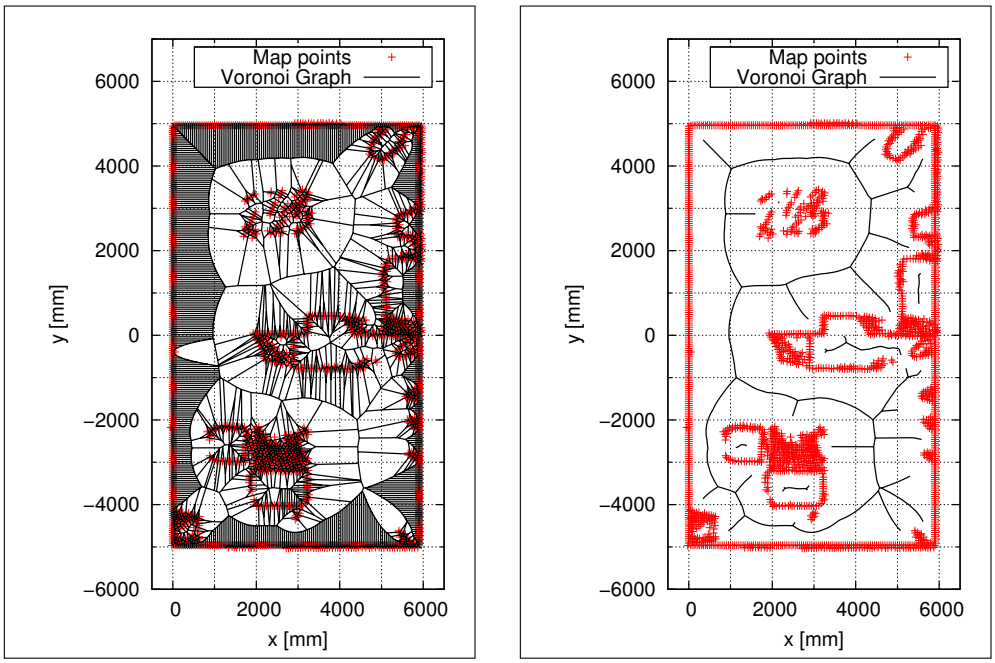


Figure 4.9: Backtracking in Graham's scan. Points $\mathbf{p}_{23}$ and $\mathbf{p}_0$ are removed from the queue (backtracking) and points $\mathbf{p}_{11}$ and $\mathbf{p}_{22}$ form a new segment on the convex hull.

The aforementioned backtracking is not complex here, as points are removed only once from the queue (in constant time $O(1)$, i.e. in $O(n)$ for n points to be removed) and the point ordered behind the removed ones is considered next. An example for this backtracking is shown in Figure 4.9. To actually determine whether a node in the Voronoi diagram lies inside or outside of the convex hull, a simple check can be carried out: If a direct straight line from the point trough the centroid of the convex hull intersects the hull at only one point, the point is contained in the polygon formed by the convex hull. If there are two intersections, the point lies outside. An important issue here is to only check intersections in the direction to the centroid. It should also be noted, that modeled parts of the environment are, for sure, not convex and pruning the nodes outside of the convex hull is just an approximation.



(a) Bounded Voronoi Diagram (b) Bounded and Pruned Voronoi diagram

Figure 4.10: The bounded Voronoi diagram (a) constructed by pruning all edges that intersect or lie outside of the bounding box of the point set stored in the map. It contains 5116 edges. Further pruning those edges in the diagram whose distance to the closest neighboring point falls below 500 mm (the largest distance between the robot's center of rotation and its boundary projected into the xy-plane) results in the final path graph (b) with 564 nodes and 560 edges.

An even more efficient alternative is pruning the edges outside of the modeled environment by means of the minimum axis-parallel bounding box containing all points in the sparse point map M. Computing the bounding box is quite efficient as a single loop through M suffices to determine the minimum and maximum values of x, y and z coordinates spanning the bounding box. However, removing all edges where one or both end-points lie outside of the bounding box will not necessarily remove all edges whose endpoints are outside of the modeled region of the environment as the bounding box might contain unmodeled regions. Furthermore, the Voronoi diagram might contain edges that completely lie inside objects. These edges will also not be removed when pruning edges outside of the convex hull. In general, if the shape of the environmental structures is not convex, not all edges in unmodeled terrain can be pruned neither using the convex hull nor the bounding box. This is mainly due to the fact, that in sparse point maps there is no distinction between unmodeled terrain and free space as in probabilistic grid maps. Exactly this issue will be addressed in Chapter 4.4.1.

Pruning edges of a Voronoi diagram can directly be integrated into Fortune's algorithm, by, first, ignoring points outside of the bounding box, i.e. neglect the event adding a new parabola to the chain, and, second, by not extending edges to infinity but until they intersect the bounding box. This is already taken into account in the implementation of O'Sullivan used here. The result of pruning all edges of the Voronoi diagram in Figure 4.7.b outside of the bounding box of the sparse point is shown in Figure 4.10.a.

The second pruning step addresses those edges in the Voronoi diagram that run through objects and environmental structures and can, thus, also be not traversed by a mobile robot. A straightforward way for removing these edges from the graph is to define a minimum allowable distance $D_{\min}$ to neighboring points and to keep only those edges whose distance to the closest point in the map, i.e. the closest Voronoi site, is larger than $D_{\min}$. Setting $D_{\min}$ to the width of the robot plus

some additional safety distance, e.g. 5 cm, will guarantee that the robot can traverse along every edge in the residual path graph under the assumption that all objects possibly blocking its way are modeled in the sparse point map. Using the longest diagonal of the robot's bounding box in the plane instead of its width additionally allows a robot to turn on the spot at every position on the edges of the graph. It should, however, be noted that, in any case, $D_{\min}$ needs to be larger than half of the minimum allowable point-to-point distance of the used sparse point map. If $D_{\min}$ is smaller not all edges will be pruned. The result of applying both pruning techniques is shown in Figure 4.10.b.

Whereas the unpruned Voronoi diagram is a connected graph, i.e. there exists a path between any two nodes, a path graph constructed from the pruned Voronoi diagram may contain multiple unconnected graphs. Here, path planning can only be applied to search for nodes that are contained within the same (connected) graph. Otherwise no solution will be found.

Resulting Algorithm for Constructing Path Graphs

The overall algorithm for constructing the bounded and pruned Voronoi diagram for a sparse point map is summarized in the following.

1. Construct Voronoi diagram by means of Fortune's Sweep Line Algorithm (Fortune, 1987, 1998) and a previously constructed point map.
2. Prune those edges of the Voronoi diagram that start and/or end outside of the known environment, i.e. whose connected nodes lie outside of the map's bounding box or convex hull.
3. Prune those edges of the Voronoi diagram whose minimum distance to neighboring points is less than a given threshold (e.g. the robot's width).
4. Transform the residual nodes and edges into a graph representation.

For the purpose of path planning, the residual nodes and edges in the Voronoi diagram are used to form a path graph. Here adjacency lists are used in the C++ implementation of the presented algorithms. The path graph structure maintains a list of nodes, where each node stores an own list of C++ pointers to adjacent nodes. This avoids the redundant storage of node information, but still allows for efficient lookups e.g. of the set of adjacent nodes for a particular node. For a fully connected graph this nested list structure stores information about n nodes and $n(n-1)$ (neglecting loops) node pointers for representing directed edges between n nodes. As path graphs constructed here can be assumed to be sparse and, in any case, not fully connected, the list stores only $2|E|$ directed edges, where $|E|$ is the number of edges in the Voronoi diagram. Note that the original Voronoi diagram contains only undirected edges, but the redundant storage of 2 directed edges allows for faster lookups (see Chapter 4.3.2). The density D (see Diestel, 2005, Chapter 7) of the bounded and pruned Voronoi diagram in Figure 4.10 with $|V| = 564$ nodes and $|E| = 560$ edges is

$$D = \frac{2|E|}{|V|\ (|V|-1)} = \frac{2 \times 560}{564 \times 563} \approx 0.0035$$

compared to $D = 1$ for a complete graph, i.e. an undirected fully connected graph without loops.

Generating the path graph by means of the Voronoi diagram of a set of points measured on the surface of objects and environment structures, guarantees that all edges and nodes in the graph are traversable by the robot given its current knowledge about objects in its workspace. That is, traversable free space is determined by means of knowledge about obstacles and the path graph is composed of the nodes and edges in the Voronoi diagram. The exact opposite of this means is followed by Vlassis et al. (1997), who sample the robot's poses during the exploration phase to determine configurations in free space. These samples are then clustered using a probabilistic algorithm where the resulting cluster centers are comparable to the centers of a Voronoi diagram (Vlassis et al., 1997). The path graph is then composed of the Voronoi diagram's dual, i.e. the Delaunay triangulation.

Compared to grid maps, both have the inherent drawback that searching in the resulting path graph might neglect a large amount of valid configurations. The Delaunay triangulation of Vlassis et al. takes only sampled configurations into account, i.e. regions that have already been traversed by the robot, whereas the Voronoi diagram used here only takes into account already modeled structures and the searchable configuration space is constrained to the set of configurations with a maximum distance to closest points in the model. When traversing along the edges of the Voronoi diagram the robot keeps a maximum distance to surrounding obstacles. The resulting trajectory might thus be by far longer than the shortest path planned on a grid map or when searching in the continuous unconstrained configuration space.

4.3.4 Path Finding Algorithms for Graphs

Path planning in graphs becomes the problem of searching for the shortest path between two graph nodes. Therefore, the graph is weighted according to some distance metric. That is, every edge is assigned a cost according to the distance between connected nodes. The length of a path between two nodes is, thus, the sum of the costs of all edges along the path. Searching for the shortest path can be carried out by means of any tree-search algorithm, i.e. by means of a simulated exploration of the sampled state space, where every sample is represented by a node in the graph (cf. Latombe, 1991, Ch. 1-2). These algorithms can be distinguished regarding their search direction, and the order in which nodes are expanded during exploration. When searching forward from the start node to the goal node, the start node forms the root of the tree and its adjacent nodes are expanded in the order determined by the search strategy. Backward search starts from the goal node forming the root of the tree and expands the nodes in the tree until it has found the shortest path to the start node or the list of expandable nodes is empty and the sampled workspace is fully explored without finding a solution. In a bidirectional search, two trees are used, one starting at the start node and the other starting at the end node. A solution is found, if the trees intersect (cf. LaValle, 2006, Ch. 2). The search strategy determines the order in which nodes are expanded. Besides time and space complexity, important characteristics of a search strategy are completeness and optimality, i.e. whether the strategy always finds a path if one exists and whether it always finds the least-cost path.

In the well-known *Dijkstra's algorithm* (Dijkstra, 1959) named after Edsger Wybe Dijkstra, the cost for reaching node $f(\mathbf{x}_i)$ from the start node $\mathbf{x}_{\text{start}}$ is the accumulated cost of all edges along the shortest path from $\mathbf{x}_{\text{start}}$ to $\mathbf{x}_i$. Dijkstra's algorithm assumes the weight of all edges as being non-negative. The weight of an edge $(\mathbf{x}_i, \mathbf{x_j})$ connecting $\mathbf{x}_i$ and $\mathbf{x_j}$ thereby corresponds to the cost of travelling from $\mathbf{x}_i$ to $\mathbf{x_j}$, e.g. the Euclidean distance $\|\mathbf{x}_i - \mathbf{x_j}\|$. Under the assumption that all edge weights are non-negative and known (or can be directly derived from adjacent nodes), Dijkstra's algorithm searches for the shortest path between nodes representing $\mathbf{x}_{\text{start}}$ and $\mathbf{x}_{\text{goal}}$ in a forward manner starting from $\mathbf{x}_{\text{start}}$ and successively evaluating the cost of neighboring unvisited nodes in decreasing order w.r.t. the cost of their predecessors or parents until $\mathbf{x}_{\text{goal}}$ is reached. In actual implementations of tree-search algorithms, priority queues are used determining the order in which neighboring nodes are visited. The representation of this queue highly influences the runtime complexity of the strategy. Starting from $\mathbf{x}_{\text{start}}$, the adjacent nodes to $\mathbf{x}_{\text{start}}$ are ordered w.r.t. the weight of the corresponding edges and added to the priority queue. The algorithm then successively removes the node with the highest priority from the queue and adds its neighboring nodes (except for the node's parent) to the queue before, again, re-ordering the queue. For each node the predecessor, i.e. the node from which it is reached on the shortest path from the start node and the accumulated costs along the shortest path are stored. Once the node with the highest priority in the queue is the goal node, its cost can directly be looked up and the shortest path from the start to the goal node can be determined by going back to the start node along the stored predecessors. Referring to the accumulated cost to reach a node n from the start node as $g(n)$, the cost function $f(n)$ used in Dijkstra's algorithm is

$$f_{\text{Dijkstra}}(n) = g(n). \tag{4.20}$$

The search strategy of Dijkstra's algorithm is complete and optimal. Compared to classical search tree algorithms such as *breadth-first search* or *depth-first search*, the order in which nodes are

expanded in Dijkstra's algorithm is not predefined but takes the cost of the nodes into account when sorting the priority queue. Still it might expand a lot of paths that might not lead to the goal node as Dijkstra's algorithm is an uninformed search algorithm.

In an informed search strategy a problem-specific heuristic is used to guide the expansion of the search tree. An evaluation function $f(n)$ as that of Dijkstra's algorithm in Eq. (4.20) including an heuristic estimate is used to determine the most desireable unexpanded node (cf. Latombe, 1991, Ch. 2). The heuristic function $h(n)$ is problem-specific and the distance of the straight line between the position represented by node n (in $\mathbf{x}_n$) and the position represented by the goal node (in $\mathbf{x}_{\text{goal}}$) is commonly used in the context of path planning. In *greedy best-first search* (GBFS) the evaluation function $f(n)$ is solely based on an heuristic estimate (Russell and Norvig, 1995, Ch. 2):

$$\begin{aligned} f_{\text{GBFS}}(n) &= h(n) \\ \text{e.g. with } h(n) &= \|\mathbf{x}_n - \mathbf{x}_{\text{goal}}\| \end{aligned} \tag{4.21}$$

Whereas in Dijkstra's algorithm the most desireable node is the unexpanded node with the shortest distance to the start node, here the most desirable node is the unexpanded node with the shortest distance to the goal node. This search strategy is generally faster than uninformed search, but not complete and not optimal (cf. Russell and Norvig, 1995, Ch. 2).

A search strategy combining both, the $A^\star$ algorithm, has been presented by Hart et al. (1968). The value of the evaluation function $f(n)$ for a node n in the $A^\star$ algorithm is the sum of the cost for reaching node n, i.e. $g(n)$ as in Dijkstra's algorithm, from the start node and an estimated cost for reaching the goal node from node n, i.e. $h(n)$ as in greedy best-first search:

$$f_{A^\star}(n) = g(n) + h(n) \tag{4.22}$$

The heuristic $h(n)$ can be used to control the behavior of $A^\star$ search. With $h(n) = 0$, $A^\star$ turns into Dijkstra's algorithm, which is optimal and complete, whereas large values of $h(n)$ relative to $g(n)$, e.g. $g(n) = 0, h(n) > 0$, will turn $A^\star$ into greedy best-first search. If $h(n)$ is always lower than the true cost $h'(n)$ for reaching the goal node, i.e. when the heuristic is admissible, $A^\star$ is optimal and guaranteed to find a shortest path. However, the lower the value of $h(n)$, the more nodes are expanded making the search slower. In fact, over-estimating $h'(n)$ by choosing values $h(n) \geq h'(n)$ allows to speed up the search while loosing optimality. It should also be noted, that if the true distance $h'(n)$ is known and used as the heuristic, i.e. $h(n) = h'(n)$, $A^\star$ will only follow the shortest path and not expand any other node. This will be utilized in Chapter 4.4.6 allowing for fast re-planning of paths.

Here, $A^\star$ will be used with an heuristic $h(n)$ being equal to the Euclidean distance between the position of node n in the map, i.e. $(x_n \; y_n)^T \in \mathbf{x}_n$, and the goal position in $\mathbf{x}_{\text{goal}}$. The cost function $g(n)$ is the accumulated sum of edge weights along the shortest path from the start node representing $\mathbf{x}_{\text{start}}$ to node n. The edge weights are again chosen as being the Euclidean distance between the two connected nodes, e.g. $\|\mathbf{x}_i - \mathbf{x}_{i-1}\|$ for the edge $(\mathbf{x}_i, \mathbf{x}_{i-1})$. Hence, the evaluation function is

$$f(n) = g(n) + h(n) = \left(\sum_{i=1,\ldots,n} \|\mathbf{x}_i - \mathbf{x}_{i-1}\| \right) + \|\mathbf{x}_{\text{goal}} - \mathbf{x}_n\| \tag{4.23}$$

where $\mathbf{x}_0 = \mathbf{x}_{\text{start}}$ and the sequence $< \mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_n >$ represents the shortest path between $\mathbf{x}_{\text{start}}$ and $\mathbf{x}_n$.

For a more detailled description of the algorithms presented here and a comprehensive overview on other tree search and graph search algorithms it is referred to (Russell and Norvig, 1995, Chapters 3-4) and (LaValle, 2006). An overview on path planning algorithms for teams of multiple robots can be found in (Bennewitz, 2004).

4.3.5 Searching for the Shortest Path

Once a path graph establishing the basis for planning paths is constructed, basically all graph search algorithms can be used to determine an optimal or nearly optimal path from an initial

robot pose $\mathcal{P}_{\text{start}}$ to a target pose $\mathcal{P}_{\text{goal}}$. However, regardless of the used algorithm, the search can only take those positions into account that form the nodes in the path graph. Hence, in order to plan shortest obstacle-free paths between two arbitrary poses, not necessarily represented by nodes in the graph, appropriate representatives in the set of nodes $V(G)$ need to be determined in a preprocessing step. The most straightforward way of determining the representatives is to select the closest nodes in $V(G)$. Instead of directly planning a path between $\mathcal{P}_{\text{start}}$ and $\mathcal{P}_{\text{goal}}$, it is then searched for the shortest path between these representatives. Assuming the shortest path returned by $A^\star$ is the sequence $< \mathbf{x}_0, \ldots, \mathbf{x}_n >$, where $\mathbf{x}_0$ and $\mathbf{x}_n$ are the representatives of $\mathcal{P}_{\text{start}}$ and $\mathcal{P}_{\text{goal}}$ respectively, the according states are added to the beginning and the end of the sequence and the overall path between $\mathcal{P}_{\text{start}}$ and $\mathcal{P}_{\text{goal}}$ is

$$< \mathbf{x}_{\text{start}}, \mathbf{x}_0, \ldots, \mathbf{x}_n, \mathbf{x}_{\text{goal}} > .$$

A path obtained by these means is shown in Figure 4.11. Here a path between two positions in an indoor environment has been planned on a graph constructed from the bounded and pruned Voronoi diagram from Figure 4.10.b. The resulting sequence contains 196 positions including start and target poses. The planned path is thus a polyline consisting of 195 line segments. It can clearly be seen in the figure that the planned path is not the shortest path between start and goal position w.r.t. the modeled environmental structures, but the shortest path on the pruned Voronoi diagram. By following the planned path (black lines) the robot, however, always maintains a maximum distance to surrounding obstacles. For taking dynamic obstacles into account that might block the way of the robot, the overall algorithm, comprising the calculation of the Voronoi diagram and the application of $A^\star$ on the resulting path graph, can be repeatedly run online while the robot approaches the goal position. To not augment the map, e.g. obtained from exploring the deserted static environment, with measurements acquired on dynamic objects, it is suggestive to avoid changes to the map, but instead to use a point set composed of the sparse point map and the latest laser scan to construct the path graph.

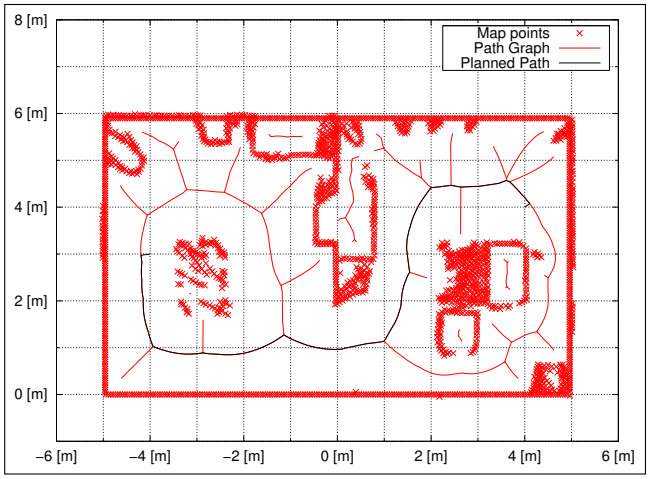


Figure 4.11: Planned path from $(4\,\text{m}\ 4\,\text{m})^T$ to $(-4\,\text{m}\ 3\,\text{m})^T$ containing 195 line segments forming the (optimal) path in the path graph. Four non-redundant paths can found by $A^\star$ whereas the depicted one is the shortest regarding its total cost if form of travelled distance. The search took approximately 4.4 ms.

In real-world experiments, it was observed that taking the closest nodes in the graph for carrying out the search and to explicitly add line segments connecting start and target pose with these nodes to the path, sometimes result in the unwanted behavior that the robot had to move in the opposite direction, i.e. away from the target pose, to reach the first node in the graph, although the direct line-of-sight to the graph is obstacle-free and could be traversed by the robot. This is handled by i) preferring nodes in the direction of the target if several nodes have similar distances to the robot and ii) checking for the first, up to ten, poses on the path whether or not they can be approached

directly. However, it is a matter of future work to replace this selection mechanism by a more reasonable metric. Another issue to smooth the planned path e.g. by fitting splines to the polyline and to completely shorten the planned path. Bhattacharya and Gavrilova (2007), for example, also use Voronoi diagrams for the purpose of path planning but retract the obtained paths towards the boundaries of obstacles blocking the direct line-of-sight to the goal. Shortening and smoothing a planned path using simple subsampling techniques is going to be presented in Chapter 4.4.8.

4.4 Path Planning in Grid Maps

The previous chapter focused on the searching for shortest paths in point maps. This chapter addresses path planning in grid maps. For being able to use the same algorithms as for planning paths in point maps, the next section describes how to calculate the pruned Voronoi diagram for the traversable cells in a grid map. The remainder of this chapter then addresses path planning on the internal structure of a grid map.

4.4.1 Voronoi-Based Path Planning in Grid Maps

Before going into details about planning paths directly on the structure of a grid map, this first section describes how to apply the algorithms presented in Chapter 4.3 on grid maps. The idea is quite straightforward by simply extracting the centers of all cells whose probability of being occupied is larger than some threshold, e.g. 0.5. The extracted centers then form a point map on which a path graph can be calculated. A typical result of computing a path graph on a grid map is visualized in Figure 4.12. What can be seen is that, here, the convex hull is not appropriate for pruning edges and nodes outside of the modeled environment. Instead, it is checked for each node whether or not it lies in a cell that is free, e.g. having a probability of being occupied below 0.3. Edges whose endpoints do not lie in free cells or that intersect other cells that are not free are pruned. This simple procedure is the one carried out for computing the pruned Voronoi diagram of Figure 4.12. The resulting path graph forms the basis for Voronoi-based path planning in grid maps and allows for using the path planning procedure presented in Chapter 4.3.

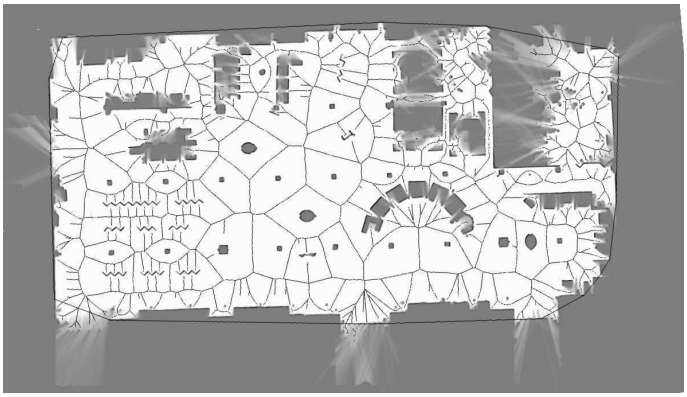


Figure 4.12: Computing Voronoi-based path graphs on grid maps. Shown here is a probabilistic occupancy grid constructed from a data set recorded by Nick Roy at the Acapulco Convention Center as well as the pruned Voronoi diagram and the convex hull computed for the center points of cells with an occupancy probability larger than 0.5.

4.4.2 Possible Movements and Movement Costs

A graph like the pruned Voronoi diagram does not need to be constructed explicitly for path planning in grid maps as these already have an internal structure. Instead of expanding a graph by visiting adjacent nodes, path planning in grid maps reduces to visit neighboring cells. Consider,

for example, a robot whose center of rotation is in the middle (white) cell of Figure 4.13. If all neighboring cells are traversable by the robot, there are eight possible movements and, thus, eight possibly unvisited cells. If, for example, the initial position falls into the middle cell and the target position into the upper left cell, the shortest path consists of a single diagonal movement. Determining representing nodes as in the case of the Voronoi diagram is not necessary here, since the corresponding cells can be directly looked up in the grid map. Because of this discretization according approaches are referred to as *cell decomposition methods* (Latombe, 1991; LaValle, 2006).

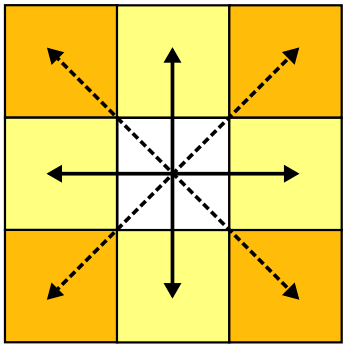
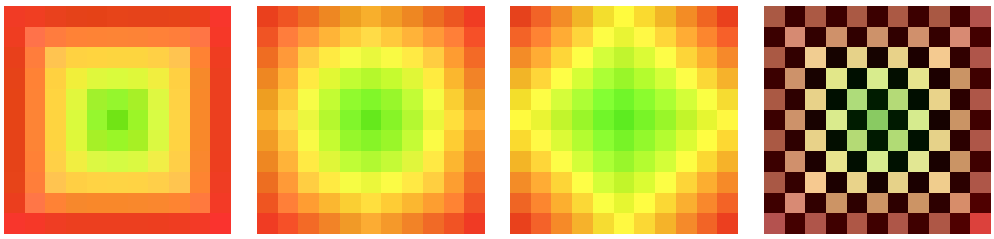


Figure 4.13: Possible movements in the grid structure. Being in the white cell, the robot can perform two types of actions: offdiagonal movements (solid black arrows) and diagonal movements (dashed black arrows).

Important questions, in this context, are which movements are allowed and which costs are involved in travelling to a neighboring cell by applying the different movements. For path planning in the continuous space, *cost* was defined as the sum of the lengths of path segments from one node to another. Here, a one-to-one mapping to the discrete case is accomplished by assigning the following costs to the two types of movements: the length of a grid cell to offdiagonal movements and the length of the diagonal through a cell to diagonal movements (refer to Figure 4.13). That is, the cost involved in travelling to a node from another node is defined as the the number of movements weighted by the individual costs for the types of movement. Assigning equal costs to diagonal and offdiagonal as in case 1 of Figure 4.14 might result in the unwanted behavior that the robot prefers diagonal movements since their cost is smaller than the actual distance between the centers of diagonally connected cells. Assigned costs according to this distance (case 2) correctly reflect the actually travelled distance. Cases 3 and 4 are merely of theoretical interest as it is not reasonable to disallow, in general, a certain type of movement. What makes sense, however, is e.g. to disallow certain movements according to the robot's heading direction. Consider, for example, that the robot moved from the center to the next cell on the left, i.e. the robot is heading west. To avoid abrupt changes in the robot's movement direction, it is reasonable to constrain possible movements to not deviate too much from the current heading direction, in this example e.g. west, north-west and south-west.

4.4.3 Computing Distance Transforms and Extracting Traversable Regions

Another important question is whether or not a cell is traversable by the robot. The path planner should, for example, not evaluate cells that are occupied by objects. Furthermore, it should be avoided that the robot travels through unexplored terrain and passages that are too narrow. When planning on the pruned Voronoi diagram, every edge can be traversed by the robot, i.e. the planner can expand the graph by visiting all adjacent nodes. To restrict the possible movements in grid map-based path planning, similar to pruning the Voronoi diagram, we compute a traversability map. A traversability map is a grid map where each cell contains a binary variable representing whether or not a cell is traversable. Here, we the define the traversability of a cell according to the following four constraints:



(a) Case 1 (b) Case 2 (c) Case 3 (d) Case 4

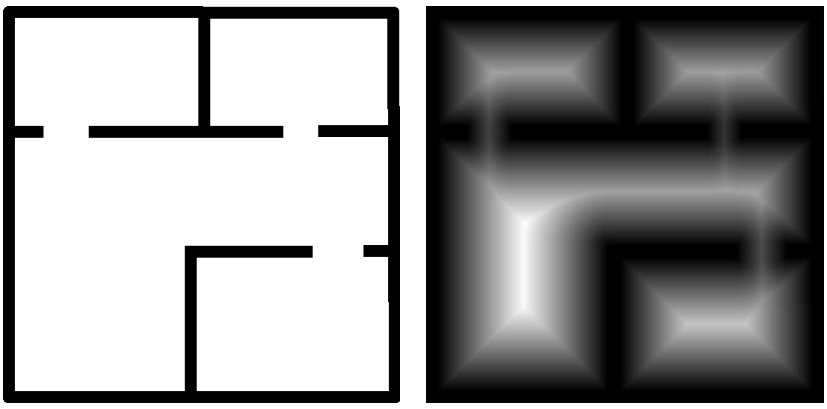
Case	Diagonal Movements		Offdiagonal Movements	
	allowed	cost	allowed	cost
1	yes	c	yes	c
2	yes	c	yes	$\sqrt{2}\ c$
3	yes	c	no	-
4	no	-	yes	$\sqrt{2}\ c$

(e) Cases

Figure 4.14: Possible movements and movement costs for different configurations (c = cell side length).

1. The robot should not traverse cells that are *occupied* by an object.
2. The robot should not traverse cells that are not yet explored, i.e. *unknown*.
3. The robot should maintain a minimum safety distance to surrounding objects.
4. The robot should maintain a minimum safety distance to unexplored regions.

Whereas the first two can be directly looked up in the grid map, the latter two necessitate determining, for each cell, the distance to the closest object. Determing the distance to the closest object for each cell can be efficiently done using the *Euclidean Distance Transform* (Rosenfeld and Pfaltz, 1968). Distance transforms form a widely used tool in image processing and pattern recognition, e.g. as a post-processing or matching step after local feature detection (Gavrila and Philomin, 1999). Applied to a binary input image, they determine for every pixel the distance to the closest non-zero pixel according to some metric, see Figure 4.15. Accordingly, different distance transforms are distinguished and a large variety of algorithms have been proposed to compute the (Squared) Euclidean Distance Transform (EDT), the Manhattan Distance Transform (MDT) and the Checkerboard Distance Transform (CDT).



(a) Binary Input Image (b) Distance Transform

Figure 4.15: Exemplary application of the Euclidean Distance Transform to a binary intput image.

A general algorithm for computing all of the aforementioned transforms has been proposed by Meijster et al. (2000). For a binary input image $B = \{b_{ij} \mid b_{ij} \in \{0,1\}, i \in [1,M], j \in [1,N]\}$, the distance transforms $\mathrm{DT}(x,y)$ for a pixel (x,y) are defined as (cf. Meijster et al., 2000):

$$\mathrm{EDT}(x,y) = \mathrm{MIN}(\ i,j\ :\ 1 \leq i \leq M\ \wedge\ 1 \leq j \leq N\ \wedge\ b_{ij}\ :\ (x-i)^2 + (y-j)^2\ ) \quad (4.24)$$

$$\mathrm{MDT}(x,y) = \mathrm{MIN}(\ i,j\ :\ 1 \leq i \leq M\ \wedge\ 1 \leq j \leq N\ \wedge\ b_{ij}\ :\ |x-i| + |y-j|\ ) \quad (4.25)$$

$$\mathrm{CDT}(x,y) = \mathrm{MIN}(\ i,j\ :\ 1 \leq i \leq M\ \wedge\ 1 \leq j \leq N\ \wedge\ b_{ij}\ :\ |x-i| \max |y-j|\ ) \quad (4.26)$$

They key idea of the algorithm by Meijster et al. is to decompose the minimization over i and j such that

$$\mathrm{EDT}(x,y) = \mathrm{MIN}(\ i,j\ :\ 1 \leq i \leq M\ :\ (x-i)^2 + G(i,y)^2\ ) \quad (4.27)$$

$$\mathrm{MDT}(x,y) = \mathrm{MIN}(\ i,j\ :\ 1 \leq i \leq M\ :\ |x-i| + G(i,y)\ ) \quad (4.28)$$

$$\mathrm{CDT}(x,y) = \mathrm{MIN}(\ i,j\ :\ 1 \leq i \leq M\ :\ |x-i| \max G(i,y)\ ) \quad (4.29)$$

with $G(i,y) = \mathrm{MIN}(\ 1 \leq j \leq N\ \wedge\ b_{ij}\ :\ |y-i|\ \}$. The algorithm is split into two phases: in the first phase $G(i,y)$ is determined by scanning each column of the image separately two times, from top to bottom and from bottom to top. In the second phase $\mathrm{EDT}(x,y)$, $\mathrm{MDT}(x,y)$ or $\mathrm{CDT}(x,y)$ is determined by scanning each row of the image separately two times, from left to right and from right to left resulting in a total of four loops over the image pixels, i.e. the runtime complexity of the algorithm is linear in the number of pixels with $O(NM)$ for a $N \times M$ binary image. A bigger part of other algorithms for computing distance transforms, e.g. (Bailey, 2004) and (Felzenszwalb and Huttenlocher, 2004), operare in a similar fashion. For more information on distance transforms and the algorithm of Meijster et al. it is referred to (Meijster et al., 2000) and (Felzenszwalb and Huttenlocher, 2004).

To guarantee that constraints 3 and 4 are satisfied when planning a path, the following procedure is applied:

1. Extract those cells from the grid map that contain objects and create a binary image, i.e. cells whose probability of occupancy is larger than some threshold.

2. Apply the Euclidean distance transform on the binary image resulting in a distance map storing, for each cell, the Euclidean distance to the closest object.

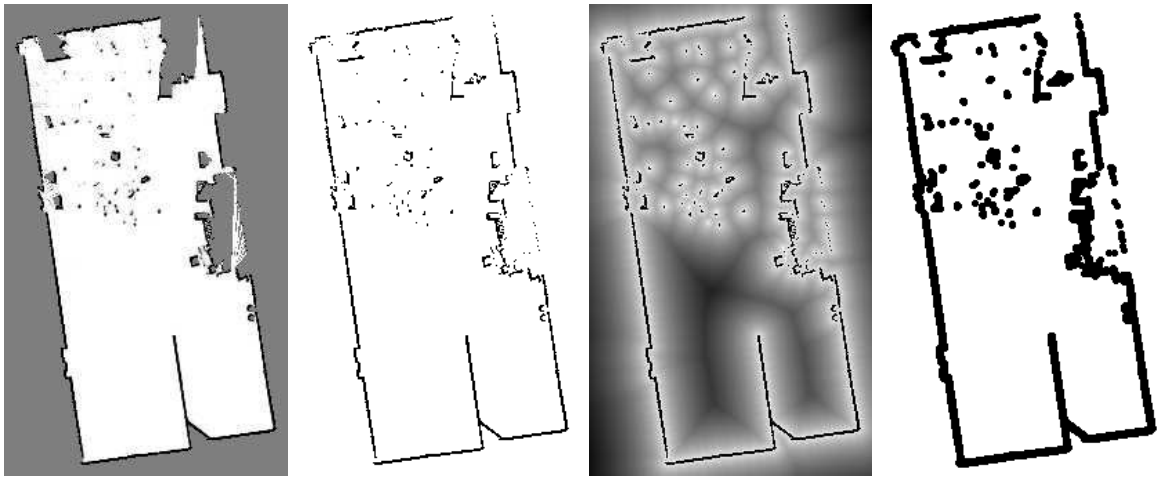
3. Mark all cells whose distance to the closest object is larger than some safety distance as being traversable, and all other cells as being not traversable by the robot.

The threshold in the first step depends on whether or not constraints 2 and 4 should be met. Like for instance, when the grid map is continuously updated and the path is repeatedly re-planned, there is no risk of planning and following a path through a so far not modeled obstacle.

Applying the above three steps to a grid map of the RoboCup lab at the Bonn-Rhein-Sieg University of Applied Sciences is shown in Figure 4.16. Once computed, the Euclidean distance transform allows for a direct lookup of traversability w.r.t. the distance to surrounding obstacles just like checking if a cell is occupied by an obstacle or not yet explored, i.e. validating whether constraints 1 and 2 are met.

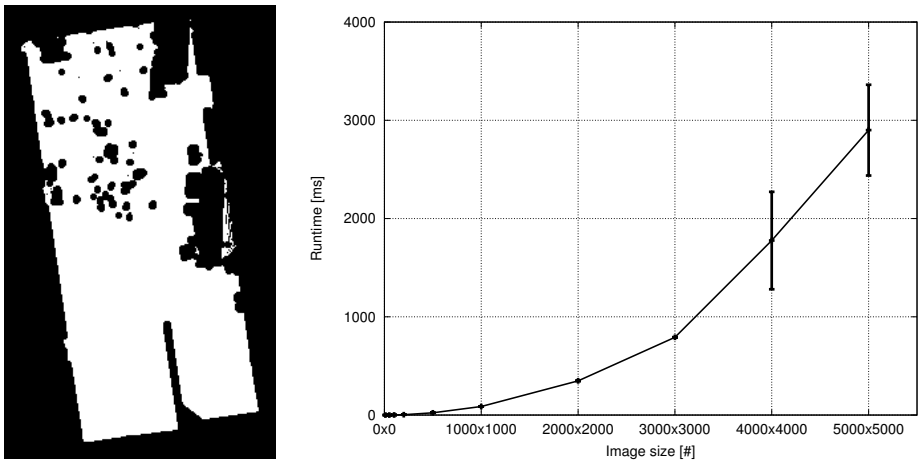
An important characteristic of the algorithm is that the runtime does not depend on the actual content of the grid map, i.e. how many cells are occupied by objects. Furthermore, since the columns and rows of the image are scanned separately, processing of the binary input image to compute the distance transform can be parallelized. Amongst others, Peikert and Sigg as well as Schneider et al., presented similar approaches to compute distance transforms on graphics hardware (Peikert and Sigg, 2006; Schneider et al., 2009).

The traversable region that results from extracting all cells from the grid map of the example scenario that satisfy the constraints of being known free cells and not having surrounding objects within a certain distance is shown in Figure 4.17.a. Figure 4.17.b shows runtimes of the overall extraction procedure measured over 100 runs for different map sizes with mean values and standard deviations.



(a) (b) (c) (d)

Figure 4.16: Computing the Euclidean distance transform for a grid map. Shown here is a (390×527) reflection-based grid map (a) together with the binary image (b) resulting from extracting occupied cells. Marking all cells in the Euclidean distance transform (c) whose distance to the closest object is larger than 30 cm as being traversable results in the white traversable region (d). Extracting the traversable region took approx. 5 ms, measured over 100 runs.



(a) (b)

Figure 4.17: Extracting traversable cells from a grid map. Shown here are those cells (white) satisfying all constraints and forming the traversable region (a), as well as the runtimes for extracting traversable regions (satisfying all four constraints) from grid maps of different sizes.

Note that the same result, i.e. traversable regions like the one shown in Figure 4.17.a, can also be achieved by simply

1. marking all cells whose probability of being occupied is lower than some threshold as being traversable,
2. marking all other cells as being not traversable and
3. marking traversable cells in the Moore-neighborhood of non-traversable cells as being not traversable.

Thereby, the Moore neighborhood around non-traversable cells is determined by the same safety distance used in the above approach. However, the Euclidean distance transform is used here, as the additional information, i.e. distances to surrounding objects, is of particular interest in, respectively, other algorithms being used and later sections of this thesis.

4.4.4 Searching for Shortest Path in Grid Maps

Once the traversable cells in a grid map have been determined, it can be searched for shortest paths from any traversable cell to any other traversable cell. Here, the three algorithms described in Chapter 4.3.4 are used for this task. These are:

- Dijkstra's algorithm solely accounting for actually involved movement costs: $f(n) = g(n)$,
- Greedy best-first search solely based on a heuristic function: $f(n) = h(n)$ and
- $A^\star$ accounting for both: $f(n) = g(n) + h(n)$.

The same implementation is used for all three algorithms and only $f(n)$ for sorting so far unvisited cells depends on the actually applied search strategy. The cost function $g(n)$ is defined as the cost involved in travelling to cell n, i.e. the number of offdiagonal movements plus the number of diagonal movements each weighted by the corresponding movement costs (see Chapter 4.4.2). The heuristic function $h(n)$ is defined as in Chapter 4.3.4. That is the Euclidean distance from the center of cell n to the target location. This heuristic is admissible, when the movement costs are set to the cell side length for offdiagonal movements and the length of the cell diagonal for diagonal movements.

The results obtained from the different path planning algorithms for a typical navigation task are shown Figure 4.18. The space searched by the different algorithms is visualized in Figure 4.19. What can be seen is, that the four different path planning algorithms, i.e. $A^\star$, Dijkstra, Greedy-Best First and the Voronoi-based approach from the previous chapter yield four different paths. Although the paths from $A^\star$ and Dijkstra look different, both paths have the same length (6.51 m) and are optimal solutions to the given problem. Driven by the heuristic, the greedy approach first searches in the direction of the goal until it reaches the nontraversable cells near the wall. Then it starts to visit neighboring nodes in a direction parallel to the wall. At the upper end of the wall, the direct path is free again and the greedy approach continues the search to the goal cell without further changing its search direction. The first half of the path is not shortened, i.e. cells are not re-evaluated, as the actually involved movement costs are not considered in the greedy approach. The second half, however, is an optimal path to the goal and for this part the path found by the greedy approach is not longer than the paths found by Dijkstra and $A^\star$.

The second half also nicely depicts the ordering in the priority queues of the algorithms. All three paths are, in this second half, composed of 19 diagonal and 25 offdiagonal movements. In principal, every direct path from top to bottom in the trapezoid spanned by the paths obtained from Dijkstra and the greedy approach, forms an optimal solution. In Dijkstra's algorithm, the cells are ordered solely based on the cost function $g(n)$. Since the cost assigned to diagonal movements is larger than the cost assigned to offdiagonal movements, Dijkstra's algorithm tends to use offdiagonal movements in the beginning and diagonal movements at the end when there is no optimal path obtainable using offdiagonal movements. In the greedy approach, the costs for the different movements are not taken into account. Driven by the heuristic, the algorithm applies

all 19 diagonal movements "for free" at the beginning, thereby minimizing the distance to the goal. The residual path to the goal is formed by a straight line composed of the 25 offdiagonal movements. $A^\star$ considering both, the involved costs and the heuristic, balances between cost and distance to the goal by applying diagonal and offdiagonal movements alternately. The path obtained from the Voronoi-based approach is, with a length of 10.71 m, by far the longest. However, the path along the edges of the pruned Voronoi diagram has a maximum distance to surrounding objects along the complete path. In practice, one has to choose whether the robot should follow a path while maintaining a maximum distance to obstacles or the shortest path from the start position to the goal position. In narrow environments or corridors it might be more reasonable to apply the Voronoi-based approach. In larger rooms this might lead to making larger detours and one of the other planners is preferable.

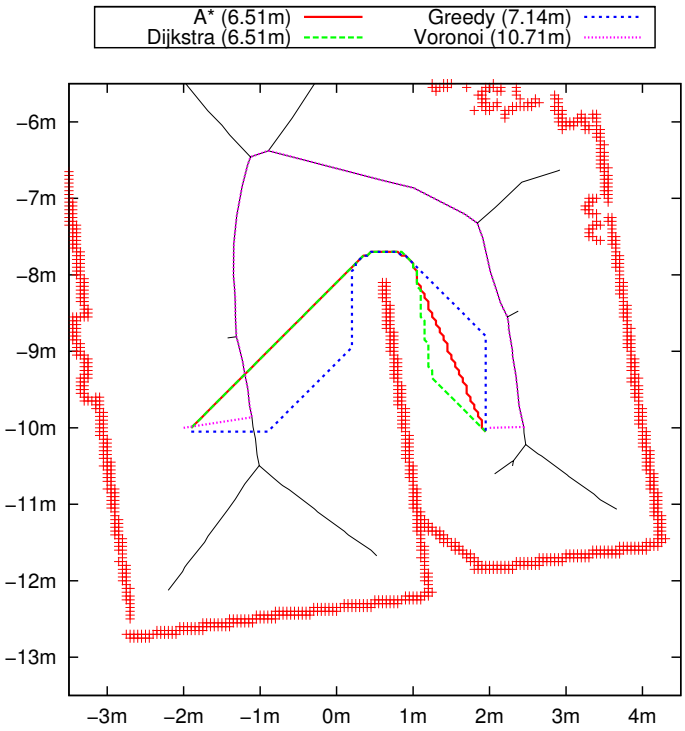
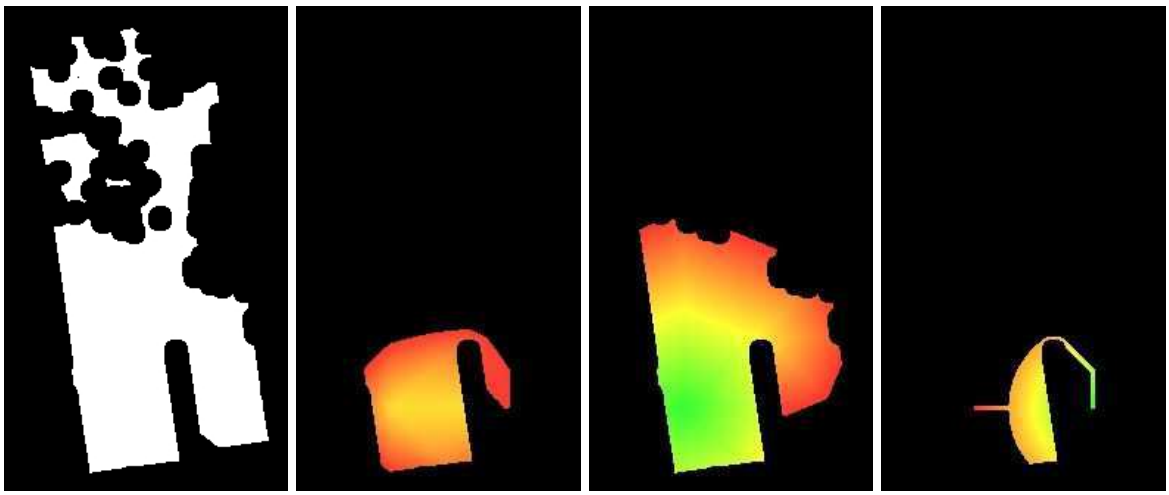


Figure 4.18: Paths obtained from the different path planning algorithms. Shown here is a point map (red crosses) extracted from the occupied cells of the grid map, the corresponding pruned Voronoi diagram (thin black lines) and the paths planned by $A^\star$ (continuous red lines), Dijkstra's algorithm (dashed green lines), Greedy Best-First search (dashed blue lines) as well as the path planned on the Voronoi diagram (dotted magenta lines).

The traversable cells in this example scenario are depicted in Figure 4.19.a. The spaces searched by $A^\star$, Dijkstra's algorithm and the greedy approach are visualized in Figure 4.19(b-d). The runtime of the algorithms solely depends on the number of cells that are visited during the search. In this example, a difference in runtimes is, respectively, not measureable and random. For larger environments, $A^\star$ is the preferred algorithm as it is complete and, by means of the heuristic function, the searched space is limited.

4.4.5 Calculating Complete Path Maps – Reachability Maps

The idea of (complete) path maps is to calculate a grid structure containing the same information as aggregated during the search of the path planning algorithms described above. That it, such a map contains for every cell the length of the shortest path to the start position together with cell's predecessor in the shortest path. A traversability map, as shown in Figure 4.20.a, contains only a binary value for each cell representing whether the according region is, in principal, traversable by



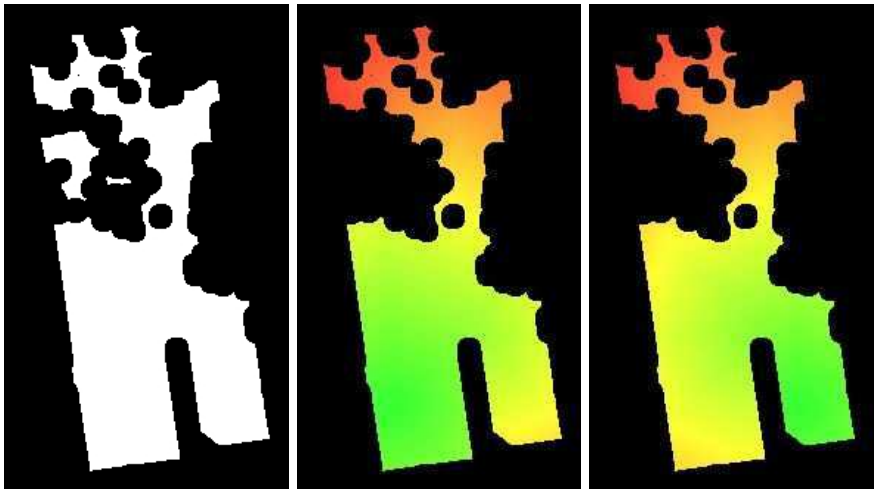
(a) Traversable cells (b) $A^\star$ (c) Dijkstra (d) Greedy

Figure 4.19: Traversable cells visited during path planning. Shown here are the traversable cells (a) for the path planning experiment in Figure 4.18 as well as the space searched by $A^\star$ (b), Dijkstra' algorithm (c) and Greedy Best-First search (d). The color encoding in (b-d) corresponds to the score of the nodes that is used for sorting in the priority queue.

the robot. However, not all of these cells might be *reachable* by the robot. Exactly this information is contained in a path map. If a cell is not reachable by the robot, the cost value of that cell is infinity just like in the initialization of the path planning algorithms. That a cell is reachable is represented by a finite cost value. Moreover, the shortest path to this cell can be directly read from the path map by recursively following the cell's predecessors. In the remainder, we will use the term *reachability map* when referring to these complete path maps.

The construction of a reachability map is quite straightforward by simply neglecting the goal specification in the path search, e.g. by running Dijkstra's algorithm until there is no traversable cell that has not been visited. Using $A^\star$ yields the same result here when defining the heuristic function to be 0 or at least equal for all nodes when no goal position is specified. With $h(n) = 0$, $A^\star$ turns into Dijkstra's algorithm. Figure 4.20 shows the results of constructing reachability maps for both the start position and the goal position from the example scenario used above. Two regions in the traversability map are black in the reachability map which denotes that, respectively, the cost of moving to the cells is infinite and the cells are not reachable from position where the search began.

There are different possible applications for this type of map. Especially in the context of robotic exploration, an exploration strategy might evaluate different candidate positions in the robot's workspace. A reachability map directly provides the movement cost to all candidates and whether or not they are reachable from the robot's current position. By this means, it is not necessary to search for the shortest path for every candidate. Another possible application of reachability maps is path planning with a set of goal positions. Consider, for example, a robot that is told to leave a room that has several doors. Instead of planning the shortest path to every door in order to decide which exit to take, the necessary information can be directly read from a reachability map, i.e. the result from a single search, just like for the evaluation of candidate positions in the exploration strategy. Searching backwards, i.e. starting from the target location, naturally yields shortest paths from all possible robot positions to the target location. Thus, a reachability map constructed by means of backward search, can be used as an optimal heuristic for path planning as described in the following section.



(a) Traversable Cells (b) Reachability Map $\mathbf{p}_a$ (c) Reachability Map $\mathbf{p}_b$

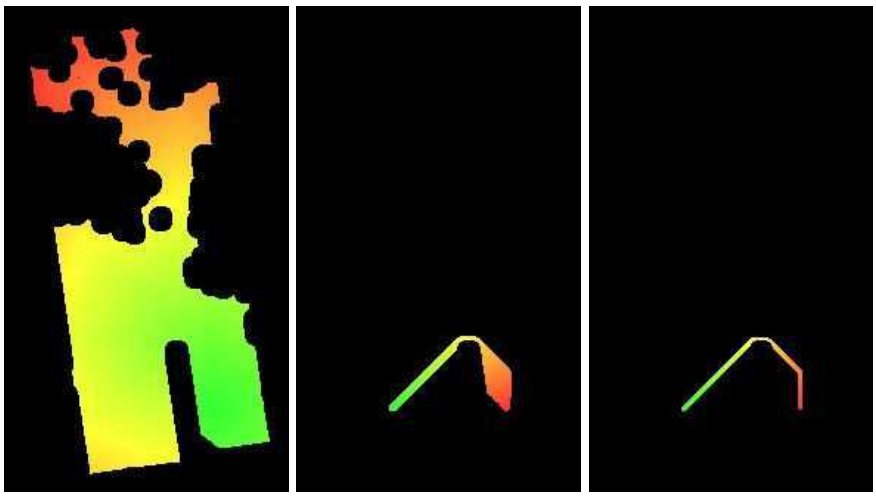
Figure 4.20: Examples of reachability maps. Shown are traversable cells (a) as well as complete path maps for the poses $\mathbf{p}_a = (-2m \ -10m)^T$ (b) and $\mathbf{p}_b = (2m \ -10m)^T$ (c). The color coding corresponds to the length of the shortest path to/from a cell.

4.4.6 Path Planning with Optimal Heuristic

The better the heuristic, the faster will the $A^\star$ algorithm find the optimal solution, at least if the heuristic is admissible. Constructing a reachability map starting the search from the target position naturally yields the optimal heuristic since it contains, for every cell, the length of the shortest past. When using this information in the heuristic function, $h(n)$ turns into the optimal heuristic $h'(n)$ (see Chapter 4.3.4). This allows to further speed up path planning, at least for $A^\star$ and greedy best-first search where the heuristic is taken into account. Figure 4.21 shows that, even in this simple example scenario, the search space can be considerably limited by applying the cost contained in the cells of the reachability map as the optimal heuristic. Since $A^\star$ balances cost and heuristic function in the priority queue, the space searched by $A^\star$ (Figure 4.21.b), again, shows the trapezoid mentioned Chapter 4.4.4.

Of course, the procedure of computing a reachability map before searching for the shortest path does only make sense when paths to the target location need to be planned multiple times. Here, this procedure is used in the context of navigating through dynamic environments, i.e. to relax the assumption of a static environment in the beginning of this chapter. An according approach has been presented by Stachniss and Burgard in (Stachniss, 2002) and (Stachniss and Burgard, 2002). Their primary contribution lies in a combined planner for, first searching the shortest path, second, compute a *channel* along that path and, finally, search for appropriate velocities to move the robot through that channel. Another idea regarding continuous re-planning is adopted here. When the robot is assigned to move to a certain location Stachniss and Burgard, first, search for the shortest path in a static environment model that is provided to the robot. In this model, the robot has the maximum number of possible paths, e.g. all doors are open and no obstacles are blocking the robot's way. That is, only environmental structures, like for instance walls, are modeled in this static map. They use a value iteration method to search for the initial path yielding reachability maps as those shown in Figure 4.20. During navigation, the static model is updated by simply setting all occupancy probabilities of cells, where laser beams end up, to the constant value 1.0, i.e. occupied with maximum certainty. To avoid collisions with obstacles in a goal-directed manner, they re-plan the path to the goal and use the initially computed reachability map as the optimal heuristic for the $A^\star$ search. If no obstacles are blocking the initially planned path, the optimal heuristic guarantees that the new path is found quickly and does not deviate from the initially planned path. If, however, a door that the robot plans to traverse is closed or an obstacle is

blocking the passage the new path, being the shortest for the current situation, will also be found quickly due to the information in the reachability map.



(a) Reachability Map (b) $A^\star$ (c) Greedy

Figure 4.21: Examples for path planning with optimal heuristic. The reachability map (a) calculated from the target position is used as the optimal heuristic to guide the search of $A^\star$ (b) and greedy best-first (c). By this means, the actually searched space is considerably limited.

Here, the following procedure is used: when a navigation task is assigned to the robot, it searches for the shortest path on the current grid map by, first, computing the reachability map from the target position and, second, extracting the shortest path from the robot's initial position to the target location. If, during navigation, an obstacle appears in front of the robot that blocks the planned path, a new path is planned from the robot's current position to the target position. Here $A^\star$ is used where the costs from the reachability map are used as the heuristic function as visualized in Figure 4.21. New information about the obstacle is then taken into account by either marking the corresponding cells as being not traversable (if and only if continuous updates to the grid map are disabled) or inherently in the continuous update procedure of the map. By this means, the robot avoids collisions in a goal-directed manner and reactive techniques for collision avoidance are, in principal, no longer necessary.

4.4.7 Additional Costs for Moving close to Obstacles

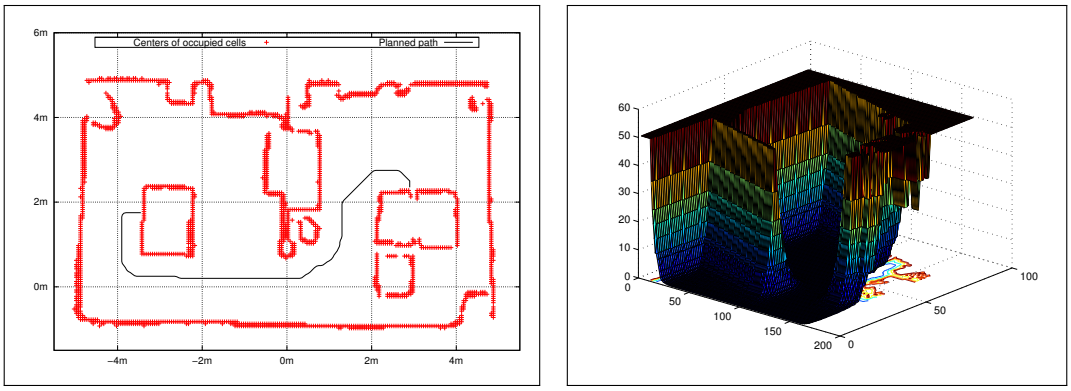
In Chapter 4.4.3, traversable cells have been determined that satisfy a set of constraints including that the robot maintains a minimum distance to occupied and unknown cells. For the majority of navigational tasks, these distance constraints inherently keep the robot away from obstacles and do not pose problems on the involved path planning problem. However, in the context of mobile manipulation, the robot might be forced to approach a vehicle pose in the immediate vicinity of an obstacle, like for instance a table or a shelf, to grasp an object. However, when using the aforementioned distance constraints, cells in the immediate vicinity of obstacles are marked as being not traversable. Hence, the path planner cannot find a solution. The same holds true if the robot itself is standing right in front of an obstacle and needs to move somewhere else. Solely decreasing the safety distance or completely disabling these constraints, makes such cells traversable but also allows the robot to move close to obstacles during navigation.

Here it is more reasonable to regularize the planner's score function according to the distances to closest obstacles instead of applying distance constraints. This can be achieved by assigning a penalty $p(n)$ for traversing s cell n that is close to an obstacle. By this means, cells close to obstacles are traversable by the robot, but are only contained in solutions if there is no *cheaper* way for reaching the target cell. To take the penalty function into account in the score function

$f(n)$, used for ordering cells in the planner's priority queue, $f(n)$ as reformulated to contain three summands: the actual cost to actually move to a cell $g(n)$, the heuristic estimate of the distance to the target position $h(n)$ and the penalty function $p(n)$ depending on the distance transform of the traversability map.

$$f(n) = g(n) + h(n) + p(n) \tag{4.30}$$

Note that in the actual implementation, $p(n)$ is directly added to $g(n)$ and is, thus, not taken into account in greedy best-first search. A typical path resulting from $A^\star$ search using the regularized score function from Eq. (4.30) is shown in Figure 4.22.a. Although both the robot's initial position and the target position lie in the immediate vicinity of objects, a solution can be found. Furthermore, the robot maintains a minimum clearance to obstacles along the path which is the desired behavior especially in the context of mobile manipulation. The inherent drawback of this approach is that the robot can, in principal, plan paths through passages that are too narrow for actually traversing them. To avoid this, it is suggestive to use rather high penalties for moving along cells that are close to obstacles, so that any other path to the target location through wide open space is preferred according to the regularized function. Figure 4.22.b shows the penalty function used for planning the path in Figure 4.22.a evaluated on some random grid map. The visualized penalty function assigns a penalty of 50 regular movements for moving to a neighboring cell that is directly adjacent to an object. The penalty function has a similar effect as repulsive potentials in a potential field approach (cf. Choset et al., 2004, Ch. 4).



(a) Example path

(b) Example penalty function

Figure 4.22: Path planning using additional costs in form of a penalty function depending on the distance transform of the traversability map.

4.4.8 Path Smoothing and Subsampling

Both the paths planned on the pruned Voronoi diagram and the paths planned on grid maps consist of a large number of small steps which may cause drastic changes in the robot's orientation especially when trying to follow the exact path. As shown in Figure 4.23, especially paths planned on grid maps can be quite jerky if the sequence of vehicle positions results of alternately applying diagonal and offdiagonal movements. Geraerts and Overmars (2007) present different methods for shortening planned paths in terms of the number of vehicle positions and for creating smooth trajectories. They present a simple algorithm for pruning a planned path by removing redundant motions. Two consecutive motions $\mathbf{m}_{i,i+1}$ and $\mathbf{m}_{i+1,i+2}$ are, thereby, defined as being redundant if the direct path from the predecessor pose $\mathbf{x}_i$ to the successor pose $\mathbf{x}_{i+2}$ is free of obstacles (Geraerts and Overmars, 2007). Furthermore, they propose algorithms for replacing path segments by direct or partial shortcuts. Waringo and Henrich (2006) propose a similar algorithm for incrementally removing positions from the path that do not have a significant influence on the overall path geometry. These are, for example, points resulting from the application of a sequence of movements

into the same direction. Path shortening by iteratively replacing path segments by direct shortcuts is, amongst others, applied by Kanehara et al. (2007). In addition, they slightly move poses from the residual path towards free space to gain a minimum clearance to surrounding objects. For finally smoothing the path in regions of infinite curvature Kanehara et al. replace consecutive poses, inducing a corner in the path, by arcs that result from fitting a circular sector through the points. Another common technique for smoothing is the application of *B-Splines*, a generalization of Bézier curves, and other spline types. Here, however, the path does not need to be a continuous curve and instead the piecewise linear structure of the planned path should be maintained as it allows for the application of a wider class of motion controllers without, again, sampling from a smooth curve.

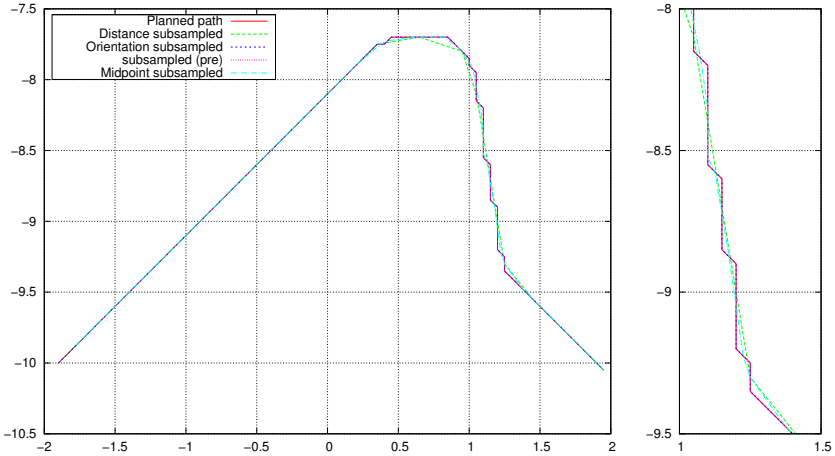


Figure 4.23: Subsampling planned paths as a simple and efficient smoothing technique.

The approach used here is similar to that of Geraerts and Overmars. The path resulting from the planner is simply subsampled or *pruned* in order to reduce the number of poses along the path while preserving its geometry. Therefore, different subsampling techniques have been implemented, again, with the focus on computationally efficient approaches that do not considerably affect the runtime of the path planning component. The following procedures are all based on a single run over the sequence of poses forming the planned path.

1. **Distance Subsampling**
   Here a pose $\mathbf{x}_i$ is removed from the sequence if the distance to its predecessor is smaller than some threshold $d_{\min}$, e.g. 30 cm:

   $$remove(\mathbf{x}_i) \text{ if } \|\mathbf{x}_i - \mathbf{x}_{i-1}\| < d_{\min}$$

   If $d_{\min}$ is larger than the grid cell size, this simple technique allows to effectively reduce the number of poses in the path. However, it might also change the paths geometry by conducting shortcuts.

2. **Orientation Subsampling**:
   Here a pose $\mathbf{x}_i$ is removed from the sequence if the change in orientation from its predecessor to its successor is smaller than some threshold $\theta_{\min}$, e.g. 25°:

   $$remove(\mathbf{x}_i) \text{ if } \arccos\left(\frac{\mathbf{m}_{i-1,i} \cdot \mathbf{m}_{i,i+1}}{|\mathbf{m}_{i-1,i}||\mathbf{m}_{i,i+1}|}\right) < \theta_{\min}$$

   where $\mathbf{m}_{i-1,i} \cdot \mathbf{m}_{i,i+1}$ is the cross product of the motion vectors from the pose's predecessor and to the pose's successor. With a value of $\theta_{\min} < 45°$, a sequence of motions into the same direction is replaced by a single straight line segment and the geometry of the path is unchanged.

3. **Midpoint Subsampling or *Midpoint Smoothing***
   Here two motions $\mathbf{m}_{i-1,i}$ and $\mathbf{m}_{i,i+1}$ and the corresponding poses $\mathbf{x}_{i-1}$, $\mathbf{x}_i$ and $\mathbf{x}_{i+1}$ are replaced by a single straight line connecting the midpoints from the two motion vectors, i.e. $1/2\,(\mathbf{x}_i + \mathbf{x}_{i-1})$ and $1/2\,(\mathbf{x}_{i+1} + \mathbf{x}_i)$. By this means, all motion vectors (and the corresponding poses) along the path are replaced by their midpoints. For longer path segments, this might cause unwanted shortcuts where a minimum clearance to surrounding objects is no longer maintained. On the other extreme, for very short segments, a single application of midpoint subsampling does not considerably smoothens the paths necessitating repetitive runs. What is applied here is a composite subsampling (presented in the following) in a preprocessing step and the subsequent application of midpoint smoothing.

4. **Composite Subsampling w.r.t. Distance and Orientation**
   Here a pose $\mathbf{x}_i$ is only removed from the sequence if the distance to its predecessor is smaller than some threshold $d_{\min}$, e.g. 30 cm *and* if the change in orientation from its predecessor to its successor is smaller than some threshold $\theta_{\min}$, e.g. 25°:

$$remove(\mathbf{x}_i) \text{ if } (\|\mathbf{x}_i - \mathbf{x}_{i-1}\| < d_{\min}) \wedge \left(\arccos\left(\frac{\mathbf{m}_{i-1,i} \cdot \mathbf{m}_{i,i+1}}{|\mathbf{m}_{i-1,i}||\mathbf{m}_{i,i+1}|}\right) < \theta_{\min}\right).$$

   This combination of distance subsampling and orientation subsampling yields that the straight line segments resulting from pure orientation subsampling are kept short enough so that midpoint smoothing can not considerably shortcut corners.

A typical result of applying the different subsampling strategies is visualized in Figure 4.23. For the motion controller used for following planned paths and presented in the following chapter, the path obtained from the subsequent application of the composite subsampling strategy and midpoint subsampling are smooth enough. The trajectory of the robot is then, again, inherently smoothed in the motion controller. It should also be noted, that the same procedure can also be applied to paths planned on the pruned Voronoi diagram (see Chapter 4.3).

4.5 Following Planned Paths

Whereas the problem of path planning was to determine a path between a start position and a target position, the problem of path following is to determine movement commands $u = (v, \omega)$ along the track to make the robot follow the planned path. Just as the simple motion controller for reaching a certain pose presented in Chapter 4.2, a path following controller determines the linear velocity v and angular velocity ω that drive the robot to the target position by traversing the curvature specified by the course of the planned path.

4.5.1 Problem Formulation

Regarding the robot's ability to follow a given path, two problems can be distinguished (cf. Coulter, 1992):

1. **Regaining a path:** The robot is not on the given path, its distance d to the nearest point on the path is rather large and exceeds a certain threshold δ_d, i.e. $d \geq \delta_d$. Here the control problem is to bring the robot back on the path.

2. **Maintaining a path:** The robot is already on the given path, i.e. $d < \delta_d$. Here the control problem is to steer the robot in a way that it stays on the path, e.g. by orienting the robot towards the end point of the currently traversed path segment. In (Indiveri and Corradini, 2004) this problem statement is again divided into two subproblems:

   2.1 The robot is on the path but, respectively, is not orientated towards the end point of the currently traversed path segment and its orientation θ towards the (partial) goal pose exceeds a certain threshold δ_θ, i.e. $d < \delta_d$ and $\theta \geq \delta_\theta$.

2.2 The robot is on the path and orientated towards the end point of the currently traversed path segment, i.e. $d < \delta_d$ and $\theta < \delta_\theta$.

As described in the previous chapters, both paths planned on the pruned Voronoi diagram and paths planned on grid maps consist of a list of line segments. This chapter will thus focus on following linear paths. This, however, does not form a restriction, since any given curvature can be approximated by a polyline, i.e. a sequence of line segments.

4.5.2 Simple Motion Controllers for Path Following

The possibly simplest path following controller is *follow-the-carrot* (see e.g Wit et al., 2004). It controls only the angular velocity ω, whereas the linear velocity v is assumed to be constant ($v = v_0$ with $v_0 > 0$) or set by another controller ($v = v(t)$). That is, it solely consists of a steering control law. Note that this basic concept is also used in many other path following controllers. The underlying controller of follow-the-carrot is a purely proportional controller (P-controller) with gain K_P and feedback error $\tilde{\theta}$ (refer to Figure 4.24.a).

$$\omega = K_P \cdot \tilde{\theta} \tag{4.31}$$

$$\text{with} \quad \tilde{\theta} = \tan^{-1}\left(\frac{g_y}{g_x}\right) \tag{4.32}$$

$$\text{and} \quad \tilde{\theta} \in [-\pi, \pi) \tag{4.33}$$

In the simple case, distances and position deviations as well as angles and orientation deviations are measured with respect to a robot-centric coordinate frame $\{R\}$. Here, $\tilde{\theta}$ corresponds to the robot's orientation towards, respectively, the goal position and the *look-ahead* or *carrot* position $\mathbf{g} = (g_x \;\; g_y)^T$. When applying the controller directly in the World reference frame $\{W\}$, the robot's current position $\begin{pmatrix} r_x & r_y \end{pmatrix}^T$ as well as the robot's current orientation r_{θ_z} need to be taken into account when computing the orientation to $\mathbf{g}$. That is, the coordinates of $\mathbf{g}$ need to be transformed into the robot's coordinate frame by applying the transformation ${}^{\{R\}}_{\{W\}}\mathbf{T} = {}^{\{W\}}_{\{R\}}\mathbf{T}^{-1}$ where ${}^{\{W\}}_{\{R\}}\mathbf{T}$ is the matrix representation of the robot's pose in the world coordinate frame $\{W\}$.

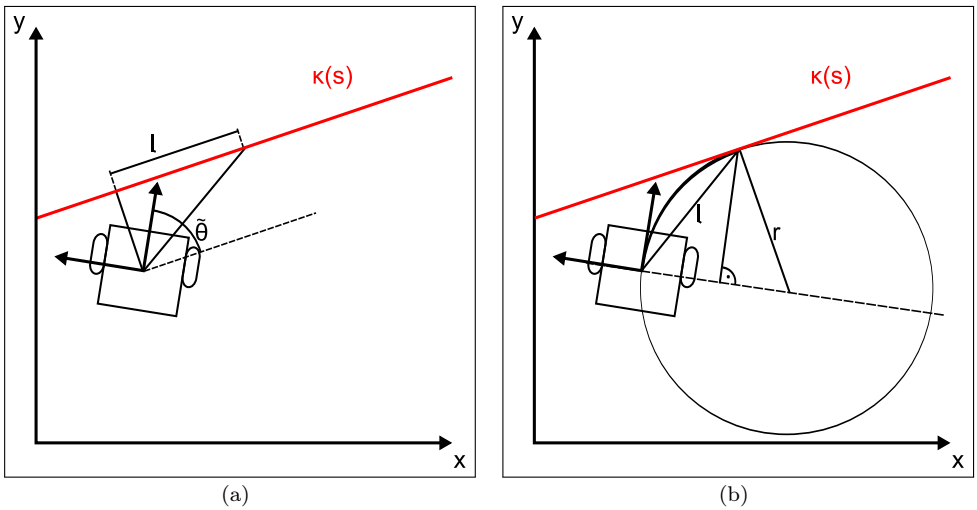


Figure 4.24: Follow-the-carrot (a) and pure pursuit (b).

Extensions to this simple path tracking controller are, amongst others, additionally using the derivative and integral of the heading error $\tilde{\theta}$ over time and corresponding gains (K_D and K_I respectively) in a PID-controller as, for example, in (Bruch et al., 2002). Travelling along a sequence of line segments using follow-the-carrot might result in not exactly tracking the line segments, as

the robot might eventually cut corners, when the carrot point is already located on the next line segment. This is due to the fact that the robot always directly steers towards the carrot point instead of orienting itself along the path. Furthermore, this simple proportional steering controller tends to oscillate when the robot is almost driving on the path. Despite these drawbacks, follow-the-carrot, or one of its derivatives, is a frequently and widely used path tracking controller, e.g. in (Hua et al., 2005). For a short overview on such extensions it is referred to (Mäkelä, 2001) and (Hellström and Ringdahl, 2006).

Instead of directly steering the robot towards a position on the path as in follow-the-carrot, the *pure pursuit* controller (Wallace et al., 1985) calculates a curvature. When traversing this curvature, the robot is brought back onto a given path segment. A more recent description of the algorithm together with some implementation details can be found in (Coulter, 1992). Here, the reference frame of the controller is robot-centric with the robot moving along the y-axis in a left-handed coordinate frame. As shown by Shin in (Shin, 1990) and (Shin and Singh, 1992) this setup shows some nice characteristics in the control of car-like vehicles since propulsion and steering are geometrically decoupled. Again, the controller only possesses a steering control law. Compared to follow-the-carrot, however, it does not control the angular velocity ω decoupled from the linear velocity v, but the curvature κ_r. The curvature κ_r, being the reciprocal of the radius r of the arc to traverse can be derived as (refer to Figure 4.24.b):

$$\kappa_r = \frac{1}{r} = \frac{2y}{l^2} \tag{4.34}$$

Regarding the actual movement command $u = (v, \omega)$, the angular velocity ω is the only controlled variable. Just like *follow-the-carrot*, the *pure pursuit* controller assumes, that the linear velocity v is either constant or set using another controller.

$$\omega = \kappa_r \cdot v = \frac{2y}{l^2} \cdot v \tag{4.35}$$

Although Coulter distinguishes the two path tracking problems, namely *retaining* and *maintaining* a given path, and states that the *look-ahead* distance depends on the actual control problem, he focuses on the latter situation and sets the *look-ahead* distance to a fixed value in advance.

For stability analyses of follow-the-carrot and pure pursuit it is referred to (Wit, 2000). Note that both follow-the-carrot and pure pursuit do not account for the orientation at the look-ahead point, but simply steer towards that point.

Wit (2000) proposed a controller called *vector pursuit* that takes this orientation into account. His controller is based on screw theory (Wit, 2000; Wit et al., 2004) and calculates, just like pure pursuit, the curvature of an arc that the robot needs to traverse in order to follow the path. Wit's calculation is, however, based on a complex system of screws taking into account and weighting the robot's current position and orientation as well as the position and orientation of a virtual target vehicle forming the look-ahead position. The results presented in Wit (2000) and (Wit et al., 2004) revealed that vector pursuit and pure pursuit are comparably equal stable but pure pursuit shows faster convergence

Hellström and Ringdahl (2006) proposed the *follow-the-past* controller for following a previously recorded trajectory originating from a human driver. They distinguish different situations and the controller is composed of three behaviors encoding particular reactions to these situations in the recorded trajectory. One behavior moves the robot towards the recorded path, a second behavior orients the robot according to the recorded orientation of the vehicle and a third behavior steers the robot to mimic the recorded steering angles. The outputs of the three behaviors are designated steering angles for the vehicle which are then fused using a weighted sum. The presented results show that the trajectory of the robot converges to the recorded trajectory and follows the recorded path without major oscillations. Recorded trajectories are, however, mostly not available in the context autonomous navigation.

Note that none of the aforementioned controllers take kinematic or dynamic constraints of the vehicle into account what is, however, important especially for the application on real physical systems. Koh and Cho proposed a Lyapunov-stable path tracking controller for controlling both

the linear velocity v and the angular velocity ω (Koh and Cho, 1999). The actual path tracking algorithm uses two controllers, one for each velocity component. That is, v and ω are controlled independently. Both controllers are based on a *bang-bang* solution[4] and take dynamic vehicle constraints into account, e.g. limits on curvature and accelerations. Traversing along a given path is obtained by tracking and following a *virtual target vehicle* that moves along the reference path as long as the robot's distance d to that path is less than some threshold δ_d, i.e. $d \leq \delta_d$. As soon as the distance exceeds the threshold, i.e. $d > \delta_d$, the virtual target vehicle stops and waits for the robot to regain the reference path.

4.5.3 Indiveri's Path Following Controller

Indiveri and Corradini presented a nonlinear path tracking controller with a gain switching strategy that explicitly takes the different problems stated in Chapter 4.5.1 into account. Designed for car-like vehicles, it guarantees (global) asymptotic converge to a given path and satisfies the vehicle's kinematic constraints during the complete traversal (Indiveri and Corradini, 2004). It is based on the nonlinear controller in (Canudas-de-Wit et al., 1993) and incorporates different controller gain tuning methods as well as a heuristic-based switching strategy that speeds up convergence by recomputing the controller gains over time.

The nonlinear controller by Canudas-de-Wit et al.

The path tracking controller of Canudas-de-Wit et al. can be applied to any kind of given paths and is, just like the aforementioned controllers, not restricted to linear path segments. It assumes a unicycle-like kinematic model with

$$\dot{x} = v\cos\theta \tag{4.36}$$
$$\dot{y} = v\sin\theta \tag{4.37}$$
$$\dot{\theta} = \omega \tag{4.38}$$

where (v, ω) are the controlled velocities and $\dot{\mathbf{x}} = (\dot{x}\ \dot{y}\ \dot{\theta})^T$ the resulting changes in the robot's position and orientation.

Given a path as the curvature $\kappa(s)$ along the curvilinear abscissa s along the path, the controller calculates the error signal $(l, \tilde{\theta})$ where l is the distance of the robot position $(x\ y)$ to its orthogonal projection point $\mathbf{p}$ on the path and $\tilde{\theta}$ the tanget at $\mathbf{p}$ corrected by the robot's orientation (refer to Figure 4.25):

$$l \equiv \operatorname{dist}\Big(\big(x(s) \quad y(s)\big)^T, \mathbf{p}(s)\Big) \tag{4.39}$$
$$\tilde{\theta} \equiv \theta(s) - \theta_{\mathbf{p}}(s) \tag{4.40}$$

Canudas-de-Wit et al. present a steering control law for both regaining and maintaining a path that guarantees asymptotic convergence of l and $\tilde{\theta}$ to zero.

$$\omega = \frac{v\kappa(s)\cos\tilde{\theta}}{1 - l\kappa(s)} - hvl\frac{\sin\tilde{\theta}}{\tilde{\theta}} - \gamma\tilde{\theta} \tag{4.41}$$

with $h, \gamma > 0$ being the controller gains. For the asymptotic convergence $(l, \tilde{\theta}) \to (0, 0)$ to hold, the linear velocity v has to be not asymptotically zero, i.e.

$$\lim_{t\to\infty} v(t) \neq 0. \tag{4.42}$$

With $v(t) = 0$, respectively, the position of the robot and the position deviation l can not be controlled. However, the actual choice of v is, just like the choice of the controller gains h and γ,

[4] A bang-bang controller is a controller that switches abruptly between two states like for instance a thermo-controller turning on the heating when the room temperature is below 18°C and turning it off again when the temperature exceeds 24°C. It is often used in cases where the controlled plant accepts only binary inputs.

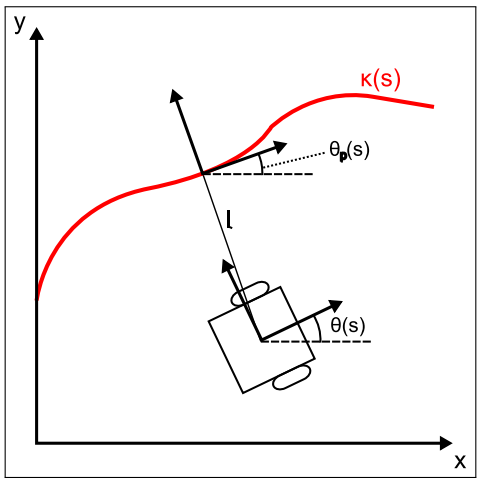


Figure 4.25: Nonlinear controller by Canudas-de-Wit et al..

free to the designer and can be used as additional (controllable) degrees of freedom (cf. Canudas-de-Wit et al., 1993).

Furthermore, the vehicle's initial conditions have to satisfy

$$l(0)^2 + \frac{1}{h}\,\tilde{\theta}^2(0) < \frac{1}{\limsup\left(\kappa\left(s\right)\right)} \tag{4.43}$$

where $l(0)$ and $\tilde{\theta}(0)$ are, respectively, distance and orientation deviation when starting to track the path, i.e. $s = 0$ denotes the first point on the path. The function lim sup is the limit supremum, that is defined as the supremum limit point of a sequence of real numbers. In this context, $\limsup\left(\kappa\left(s\right)\right)$ is the largest (finite) curvature on the path, i.e. the largest curvature despite a finite set of points $S = \{\mathbf{p}(s) \mid \kappa(s) = \infty\}$. Furthermore, satisfying Eq. (4.43) avoids that the robot might not be able regain the path. Here, the robot's initial distance $l(0)$ to its orthogonal projection on the path has to be less than the smallest radius $r_{\min} = 1/\limsup \kappa(s)$.

The steering control law in Eq. (4.41) solves both path tracking problems, as defined in Chapter 4.5.1, for the unicycle kinematic model and for a large class of paths. When using the control law to actually design a path tracking algorithm, the linear velocity v as well as the controller gains h and γ can be chosen arbitrarily by the designer as long as the above conditions are met.

An interesting property of the control law is that, when the robot has *regained* the path, i.e. $l = 0$ and $\tilde{\theta} = 0$, the second and third term vanish and the first term becomes the feedforward command $v\kappa(s)$ that *maintains* the path. Extensions to the steering control law as well as adaptations to robot platforms where the unicycle model can not be applied are described in (Canudas-de-Wit et al., 1997).

If the reference path is a straight line, i.e. $\kappa(s) = 0\ \forall s$, the feedforward term vanishes and the sum of the residual term determining the angular velocity ω is only bounded by the possibly but not necessarily bounded orientation deviation $\tilde{\theta}$ or not bounded at all and possibly going to infinity. That is, the domain of convergence extends to the whole state space (Indiveri and Corradini, 2004). The resulting control behavior can be compared to that of *follow-the-carrot*. Still, applying the controller of Canudas-de-Wit et al. to linear paths forms the basis of the controller extension proposed by Indiveri and Corradini.

Indiveri's Ansatz

The idea of the controller by Indiveri and Corradini is to apply the path tracking controller of Canudas-de-Wit et al. to linear paths, but to transform the problem into a goal-referenced frame similar to that in (Indiveri, 1999) and used in the simple motion controller in Chapter 4.2. In this frame, the linear path to be tracked forms the x-axis of the coordinate frame leading to $l = |y|$ as

well as $\theta_d = 0$ and hence $\tilde{\theta} \equiv \theta$. Thereby, the steering control law from Eq. (4.41) is simplified to:

$$\omega = -hvy\frac{\sin\theta}{\theta} - \gamma\theta \quad : \quad h, \gamma > 0 \tag{4.44}$$

Indiveri and Corradini then show that this steering control law, together with the assumption that the linear velocity v satisfies Eq. (4.42), makes $(y, \theta)^T = (0, 0)^T$ a globally asymptotic stable equilibrium for the subsystem $(\dot{y} = v\sin\theta, \dot{\theta} = \omega)$ as formed by the unicycle model (Equations 4.36 - 4.38). Furthermore, assuming that, according to Eq. (4.44), $h > 0$ and $\gamma > 0$ and assuming that $v(t)$ satisfies Eq. (4.42), the trajectory taken by the robot is uniquely determined by its initial conditions and the values of the controller gains h and γ. Whereas the initial conditions can not be changed, the controller gains h and γ can not only be arbitrarily chosen, but also recomputed during the controller's application. As already mentioned in Chapter 4.5.1 three situations can be distinguished:

Case i. The robot is not on the path, i.e. $d = |y| \geq \delta_y$, and needs to regain the currently tracked path segment. Here, small controller gains lead to an unwanted slow convergence. That the robot quickly regains the path can only be achieved by choosing larger gains.

Case ii. The robot is on the path, but is not oriented along it, i.e. $|y| < \delta_y$ and $|\theta| \geq \delta\theta$. Here, the controller gains need to be chosen in a way that $|\theta|$ is minimized while $|y|$ should remain small.

Case iii. The robot is on the path and correctly oriented, i.e. $|y| < \delta_y$ and $|\theta| < \delta_\theta$. Since larger controller gains might lead to oscillations in this situation, here smaller controller gains are needed to allow the robot to maintain the path through smooth movements.

Indiveri's idea is to re-compute the controller gains depending on the current situation and to additionally take the kimematic constraints of the vehicle into account.

Situation-dependent Computation of the Controller Gains

As derived by Indiveri and Corradini, the controller gains h and γ can be calculated as follows:

Case i: if $|y_j| \geq \delta_y, \forall\theta_j$

$$h_j^* = \frac{1}{2}\left(-\frac{\theta_j^2}{y_j^2} + \sqrt{\frac{\theta_j^4}{y_j^4} + \frac{4\kappa_r^{\dagger 2}}{y_j^2\left(1+2\alpha\right)^2}}\right) \tag{4.45}$$

Case ii: if $|y_j| < \delta_y, |\theta_j| \geq \delta_\theta$

$$h_j^* = \frac{\kappa_r^{\dagger 2}}{\theta_j^2\left(1+2\alpha\right)^2} \tag{4.46}$$

Case iii: if $|y_j| < \delta_y, |\theta_j| < \delta_\theta$

$$h_j^* = \frac{1}{2}\left(-\frac{\delta_\theta^2}{\delta_y^2} + \sqrt{\frac{\delta_\theta^4}{\delta_y^4} + \frac{4\kappa_r^{\dagger 2}}{\delta_y^2\left(1+2\alpha\right)^2}}\right) \tag{4.47}$$

In any case:

$$\gamma_j^* = 2\alpha v_0\sqrt{h_j^*} \quad : \quad \alpha > 1 \tag{4.48}$$

where $\kappa_r^\dagger$ is the upper bound on the maximum curvature that the robot can drive (Indiveri and Corradini, 2004), i.e. the reciprocal of the smallest radius. Here, δ_y plays the same role as the *look-ahead* distance in path tracking algorithms like for instance *follow-the-carrot*.

Calculating the controller gains according to Equations 4.45 to 4.48 was originally intended to guarantee stable behavior for certain initial conditions, i.e. the gains were only calculated at $t = 0$.

To avoid slow convergence resulting from the calculated controller gains, Indiveri and Corradini suggest to re-calculate the controller gains over time according to some switching strategy. They propose two gain switching strategies, one is to never update the gains, i.e. use Equations (4.45 - 4.48) according to their original intention, and the second strategy is to re-calculate the gains with every update of the controller. The latter strategy is also used here.

Representing and Selecting the Path Segment to track

The main drawback of the controller, as well as that of the original controller by Canudas-de-Wit et al., is that global convergence is only guaranteed for linear paths. In case of nonlinear paths, convergence is only guaranteed for certain initial conditions. This drawback is caused by the fact that the control law drives the robot towards the closest point on the path, instead of tracking and following a virtual target vehicle moving along the path as in *follow-the-carrot*, *pure pursuit* or the controller of Koh and Cho (1999). However, this drawback can be neglected here, since all paths planned on a path graph are composed of a sequence of connected linear path segments. The residual problem when following a planned path is to determine which path segment should be used as the current reference path for the controller. Hence, the problem of following a planned path is decomposed into following linear reference paths and a strategy to switch between path segments forming the new reference path.

In an early implementation[5] of the controller, adopted in (Zheng, 2007), Pluecker coordinates were used to represent the line segments of the path. For every controller update the closest path segment to the robot's current position was determined and, based on the Pluecker coordinates representing the closest line segment and the robot's current position, $(l, \tilde{\theta})$ was re-calculated.

Here, the path is represented as a chain of local coordinate frames, as proposed by Indiveri and Corradini. Given the path sequence $< \mathbf{x}_0, \ldots, \mathbf{x}_n >$, where $\mathbf{x}_0$ is the robot's initial position and $\mathbf{x}_n$ the goal position $\mathbf{x}_{\text{goal}}$, a local coordinate frame is attached to each $\mathbf{x}_i$ with $i = 0, \ldots, n-1$ by using the inverse of the matrix representation of the position in $\mathbf{x}_i$ and a rotation angle θ_z around the z-axis chosen in a way that the x-axis $\hat{X}_i$ of the frame points to the next position $\mathbf{x}_{i+1}$ on the path. An example for such a chain representation is shown in Figure 4.26.

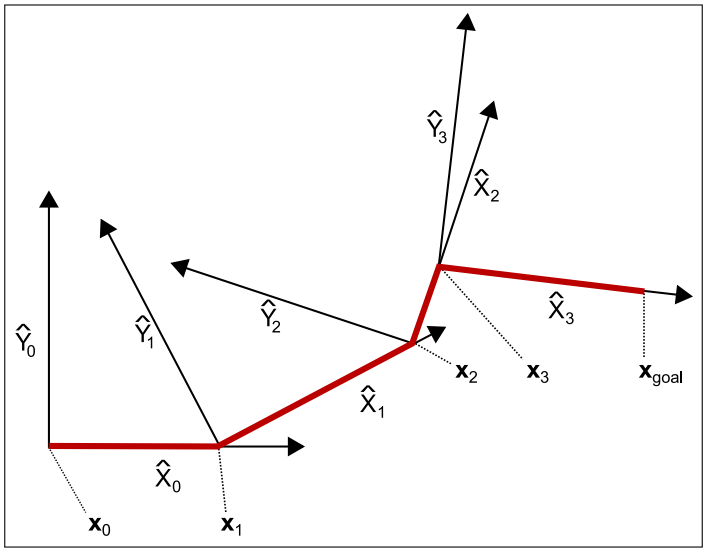


Figure 4.26: Example for representing a path by means of a chain of local coordinate frames.

The 4×4 homogeneous transformation ${}^{\{W\}}_{\{\mathbf{x}_i\}}\mathbf{T}$ mapping points from the local coordinate frame

[5] A documentation of this work or a publication is unfortunately not available.

of $\mathbf{x}_i$ into the world frame $\{W\}$ is thereby defined as being

$$
{}^{\{W\}}_{\{\mathbf{x_i}\}}\mathbf{T} = \begin{pmatrix} \cos\theta_z & -\sin\theta_z & 0 & \mathbf{x}_i^x \\ \sin\theta_z & \cos\theta_z & 0 & \mathbf{x}_i^y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.49}
$$

$$
\text{with} \quad \theta_z = tan^{-1}\left(\frac{\mathbf{x}_{i+1}^y - \mathbf{x}_i^y}{\mathbf{x}_{i+1}^x - \mathbf{x}_i^x}\right), \quad \theta_z \in [-\pi, \pi) \tag{4.50}
$$

The inverse of ${}^{\{W\}}_{\{\mathbf{x_i}\}}\mathbf{T}$ is then used in every controller update to map the robot's current pose into the local frame of the path segment that is currently tracked. The transformation matrix ${}^{\{W\}}_{\{\mathbf{x_i}\}}\mathbf{T}$ and its inverse are calculated only once when the robot starts tracking a new segment. In addition to the rotation angle and transformation matrix, the length L_i of the path segment $(\mathbf{x}_i, \mathbf{x}_{i+1})$ is calculated, when the robot starts tracking it.

The individual components of the robot pose $\mathbf{p}$ become a special meaning when expressing $\mathbf{p}$ in a local frame $\{\mathbf{x_i}\}$: $\left({}^{\{\mathbf{x}_i\}}\mathbf{p}^y,\ {}^{\{\mathbf{x}_i\}}\mathbf{p}^{\theta_z}\right)$ forms the error signal $\left(y, \tilde{\theta}\right)$ of the controller, whereas ${}^{\{\mathbf{x}_i\}}\mathbf{p}^x$ indicates the progress of traversing the currently tracked path segment. When ${}^{\{\mathbf{x}_i\}}\mathbf{p}^x$ is larger or equal to L_i, the robot has reached the end of the currently tracked path segment. Including an allowable deviation δ_L, the robot starts tracking the new segment if

$$
{}^{\{\mathbf{x}_i\}}\mathbf{p}^x \geq L_i - \delta_L.
$$

Similar to the experiments in (Indiveri, 1999) for the motion controller in Chapter 4.2, Indiveri and Corradini proposed a simple experiment showing the fast convergence when regaining the path and stability for mainting the path. The linear path to be tracked is the x-axis of an arbitrary coordinate frame. The robot is positioned in the origin of the frame such that $y = 0$, but oriented in the wrong direction, i.e. $\tilde{\theta} = \pm\pi$. The result of this experiment using a simulated ideal robot is shown in Figure 4.27. The resulting trajectory matches that of the results presented in (Indiveri and Corradini, 2004).

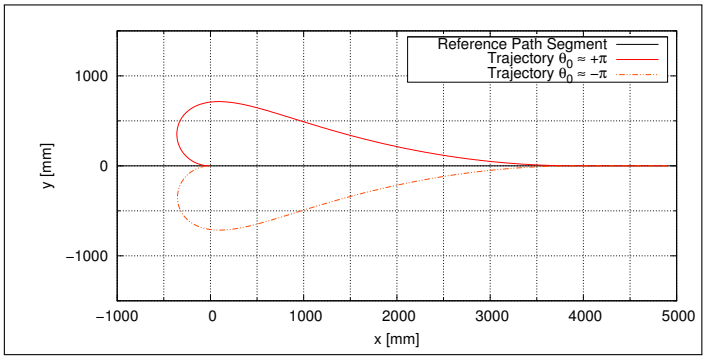


Figure 4.27: Simulated results for the U-turn maneuver as proposed in (Indiveri and Corradini, 2004). Depicted here is the x-axis forming the reference path segment as well as the paths taken by the robot with initial orientations of $\theta_0 \approx \pm\pi$ causing the robot to turn, respectively, clockwise and counter-clockwise.

4.6 Results and Open Problems

In this chapter two problems have been addressed, namely planning of shortest paths between two positions or poses and controlling the robot's motion to reach a certain goal. Two controllers have been implemented that allow the robot to approach a certain pose in its immediate vicinity and follow a planned path. For both types of environment representations, i.e. sparse point maps and

probabilistic reflection maps, different path planning algorithms have been implemented serving different purposes. All algorithms, as well as the SLAM procedures from Chapter 3, have been integrated into a simple multi-layered robot control architecture proposed in (Holz, 2007). By this means they can be applied both on a wide variety of physical robot platforms, including the mobile service robot Johnny Jackanapes, and in a selection of simulation environments containing the Player/Stage framework, the Carnagie Mellon Navigation Toolkit CARMEN and Microsoft Robotics Studio. The remainder of this chapter will briefly present a selection of experiments and achieved results with the focus on navigation in domestic environments. The final proof concept will be given with Chapter 5 where all the presented algorithms and mechanisms cooperate for the purpose of autonomously exploring and inspecting a robot's workspace.

Sparse Point Maps and Voronoi-based Path Planning in a Simulated Environment

For first tests of the algorithms in a simulation environment, Player/Stage world files have been constructed from sparse point maps obtained from real sensor data. Simulating robot and world in Stage allows not only for simulating experiments, but also to repeat (real-world) experiments and past performances.

In the first experiment presented here, the robot has been "put back" into the @Home arena of the RoboCup German Open 2008 in Hannover. The map constructed by the real physical robot by means of 2D laser scans in the real arena is shown in Figure 4.28. The sparse point map is augmented with a vector of named objects and locations. The figure also shows the constructed path graph formed by the pruned Voronoi diagram and a planned path resulting from the application of the path planning algorithm presented in Chapter 4.3. In this simple experiment, the robot was positioned in the doorway of "Front Door" and was told to move to the "Fridge". The robot accomplished this task by, first, constructing a path graph for the sparse point map, second, planning the shortest path on the path graph between the positions of "Front Door" and "Fridge", third, applying the path following controller presented in Chapter 4.5 for following the planned path until it reached the vicinity of "Fridge" and, finally, applying the motion controller presented in Chapter 4.2 to approach an appropriate pose in front of "Fridge". Along its way, the robot used simulated 2D laser scans to re-localize itself in the sparse point map by applying the SLAM algorithm presented in Chapter 3.5.

Sparse Point Maps and Voronoi-based Path Planning in a Real-world Environment

Final experiments of the presented algorithms in real-world environments have been carried out in the preparation for and at the actual event of the RoboCup World Championship 2008 in Suzhou, China. Participating in a wide variety of tests in the @Home league, the robot had to cope with several tasks, such as navigating and localizing in dynamic environments, exploring the arena by performing a SLAM algorithm while following a human guide, manipulating objects and recognizing face and speech of human users. Presented here is the final demonstration during the finals of the @Home World Championship traceable by means of a video accompanying this thesis.

Shown in the video is the autonomous service robot "Johnny Jackanapes". The robot is performing the following tasks (refer to Figure 4.29):

1. Manipulation of an object while standing in the kitchen. The robot "prepares a meal" by searching for an object (the "Chicken Noodles" as selected by the jury), grasps the object, "empties its content" into the pan and finally places the object back on the table.

2. Navigation to the kitchen table involving path planning and the presented controllers for following the planned path and orienting towards the table.

3. Manipulation of an object. The robot gets a drink from the table by grasping and lifting an object (the "Green Tea" as selected by the jury).

4. Navigation to the dinner table in the living room. The robot serves the drink by moving to the dinner table (as announced by its operator). Up to now, the robot does not know which person is "the guest".

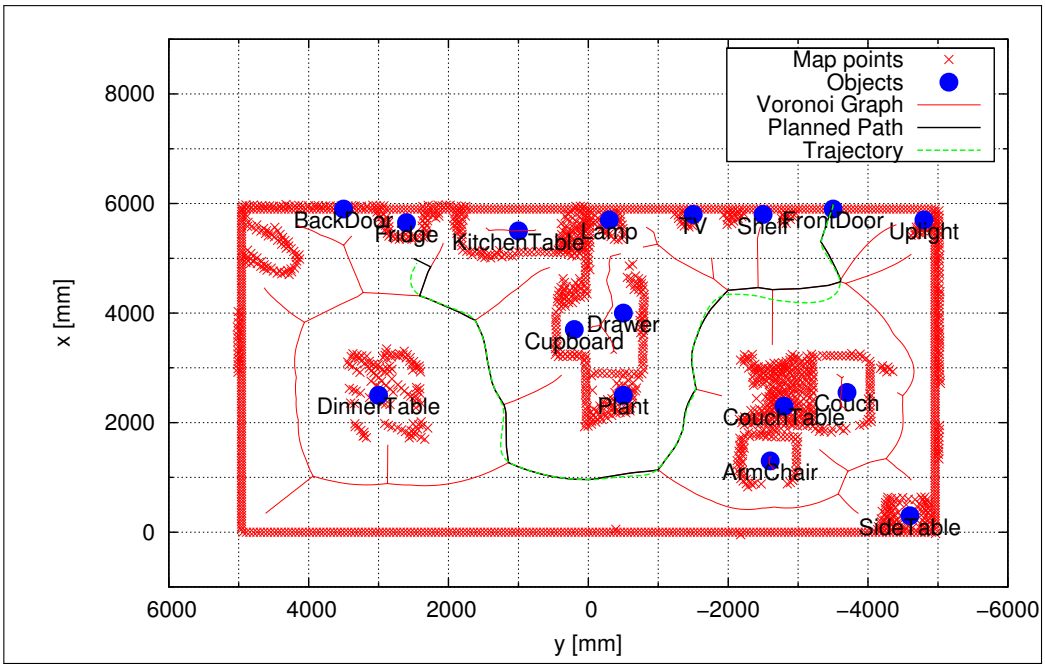


Figure 4.28: Path Planning and Following in a map of the RoboCup @Home arena at the German Open 2008. Shown are the Path Graph (thin red lines), the planned path (thicker black line) and the trajectory of the robot (dotted green line).

5. Face recognition. The robot is searching for faces, detects the face of the guest and recognizes the guest as being "Ronny" (selected by the jury).

6. Manipulation. The robot waits until the guest's hand is in the immediate vicinity of the object to release it and finally presses its own emergency stop button to finish the demonstration.

The video is available at `http://www.b-it-bots.de/media`.

The ICP algorithm together with the sparse point maps allowed for a robust and highly efficient localization of the robot and for constructing environment representations online. However, in this state of the system, probabilistic reflection maps have not yet been used to remove points from the map that correspond to environment dynamics or objects that are no longer at the place where they have been modeled. That is, when using a 3D sensor and the obstacle and structure maps (see Chapter 2.3.4), environment dynamics cause phantom effects in the map. This drastically affected the Voronoi-based path planning since these phantom effects can cause that the robot makes detours instead of approaching a goal on the shortest path. In most of the RoboCup@Home tests, this is not a problem since the arena is rather static and the robot does not need to update its internal map online. Hence, dynamics did not cause new measurements in the sparse point map. In the Walk&Talk test, however, where the robot has to construct a map of the preliminary unknown arena while following a human guide, some points not being classified as belonging to the guide can cause the aforementioned phantom effects. As a result, the robot makes detours and avoids non-existent obstacles in the subsequent navigation phase. With the introduction of probabilistic reflection maps, these points can be effectively removed from the sparse point map.

Another issue is that the motion controller for following paths was directly applied on the paths planned by in the pruned Voronoi diagram. Hence, the robot's path consisted of a large number of short line segments. Furthermore, due to the nature of the Voronoi diagram, the path swerved in order to maintain a maximum clearance to surrounding obstacles. That is, even a small object lying in front of a wall in a distance of 2 m caused the robot to swerve. Path smoothing or path subsampling has not been applied. Hence, the robot followed exactly the safest

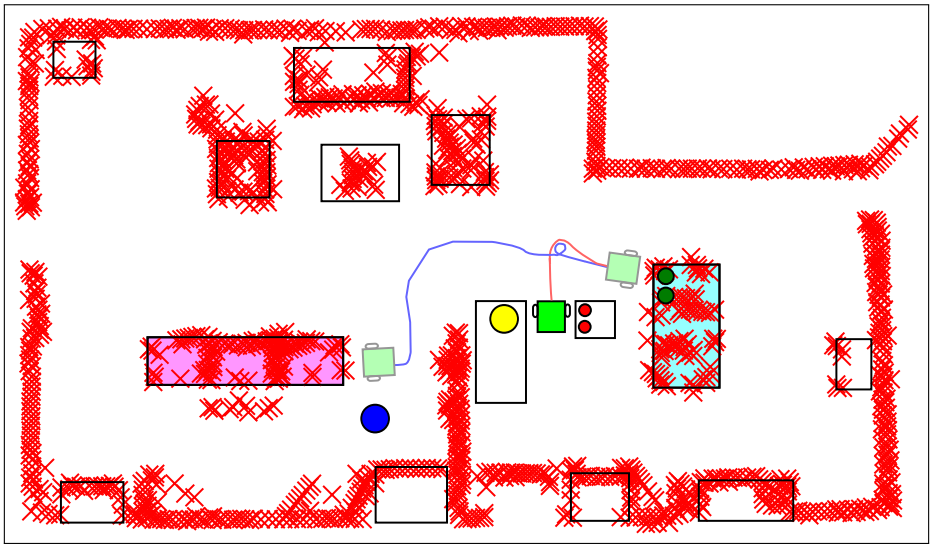


Figure 4.29: Illustration of the final experiment at GermanOpen 2008. Shown is the sparse point map (red crosses), the robot (green rectangle), the noodles (red circles) and the pan (yellow circle) as well as the drinks (green circles) on the kitchen table (rectangle in light cyan) and the guest (blue circle) at the dinner table (rectangle in light magenta). A visualization of the trajectories for tasks 2. and 4. are shown in light red and light blue.

path (in terms of its distance to surrounding obstacles) but instead of moving along a smooth curvature showed a wiggling behavior. This issue has been addressed by, first, adding alternative path planning algorithms and, second, smoothing planned paths. By planning directly on the probabilistic reflection maps, it is searched for shortest paths instead of instead of shortest paths in the graph. That is, instead of taking the safest path, the robot accepts moving close to obstacles when the overall travel distance can be significantly reduced. Resulting paths are, furthermore, smoothed by means of different subsampling techniques yielding less and longer line segments while reducing the number of abrupt changes in orientation. By this means the robot's movement is considerably smoother.

Reflection Maps and Grid-based Path Planning a Real-world Environment

For the 2009 RoboCup competitions as well as the exploration and inspection experiments presented in the next chapter, the aforementioned extensions in the form of reflection grid maps as well as grid-based path planning and subsequent smoothing of planned paths have been used. What is presented here as a final experiment is the robot's demonstration during the RoboCup@Home competition at GermanOpen in Hannover 2009. The scenario is, again, that the robot helps its human user in receiving guests and serving drinks or snacks. A video of this and other tests is available at `http://www.b-it-bots.de/media`. The overall demonstration is structured as follows (refer to Figure 4.30):

1. The robot waits near the entrance door besides the dinner table and detects incoming persons. Incoming persons are recognized by means of their faces whereas previously unknown persons are learned online for later recognitions. Once both the guest and the host are present, the robot offers drinks and snacks (the guest has chosen chips in *original* flavor).

2. The robot moves to the kitchen in order to grasp the wanted object.

3. The robot moves backwards away from the table thereby holding the grasped object. For this movement no path planning is necessary.

4. The robot moves back to the location where guest and host have been sitting before moving into the kitchen. In the meantime another person entered the scenario causing guest and host

to change places. The robot detects and recognizes (unfortunately with two false positives) the preliminary known host, the learned guest and the person. The chips are then handed over to the correctly recognized guest.

5. The robot moves backwards to leave the narrow passage
6. The robot moves towards the couch for the subsequent jury questions.

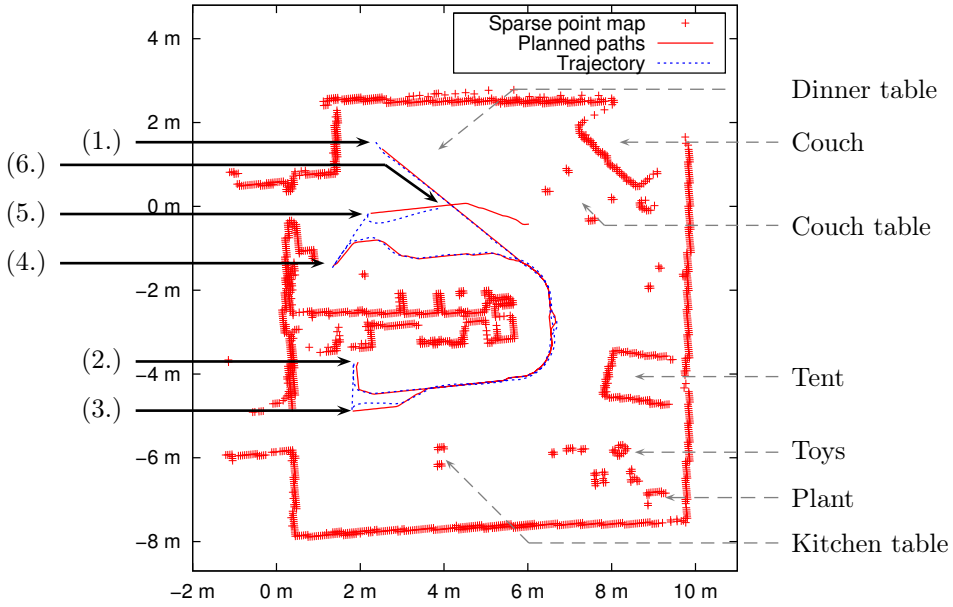


Figure 4.30: Illustration of final experiment at GermanOpen 2009. Shown are the sparse point map (red crosses), the planned paths (red lines) and the robot's trajectory for the complete demonstration at the GermanOpen finals.

The results described above as well as the underlying algorithms have been presented in (Holz, 2009) and (Holz et al., 2009).

Chapter 5

Exploration and Inspection

Up to now, two important capabilities of an autonomous robot have been addressed, namely *simultaneous localization and mapping (SLAM)* to construct internal environment models as well as *path planning* and *motion control*. With the algorithms presented in Chapter 3, an autonomous mobile robot is able to incrementally construct environment models given sensory information in an online fashion, i.e. while moving through the environment. Controlling the robot's motion as well as planning shortest paths to reach certain locations within the environment has been addressed in Chapter 4. Deciding where the robot has to move in order to model all relevant regions of its workspace is going to be addressed in this chapter.

After giving an introduction to robotic exploration and inspection as well as an overview on related work, three problems will be addressed. In human-guided exploration, a human shows the robot around, stops at relevant locations and assigns names to these locations. With these names the user can refer to certain objects or locations in later task assignments. Addressed problems are, amongst others, the detection of the human guide as well as motion control for following the guide.

The second topic in this chapter is the autonomous exploration of the environment. Instead of following a human guide, an exploration strategy decides where the robot has to move in order to construct a complete environment model. Different exploration strategies will be presented as well as various algorithms like for instance to segment a grid map into individual rooms.

The third topic is autonomous inspection. Addressed issues are, amongst others, the detection of changes in the environment, the application of greedy exploration strategies to approximately solve inspection tasks as well as fully autonomous strategies. The latter, first, determine a set of locations in the environment that allow for an almost complete coverage and, second, plan an optimal tour to visit all the locations.

5.1 Introduction and Related Work

Merriam-Webster defines *exploration* as an *"instance of exploring"*, that is *"traveling over (new territory) for adventure or discovery"* and *"making or conducting a systematic search"*. In fact, the task of *online-searching* with an autonomous robot (see e.g. Fekete et al., 2004) is quite similar to the task of robotic exploration since every part of the environment has to be taken into account for an investigation. Analogues to *"making or conducting a systematic search"*, the fundamental problem of robotic exploration is to decide where to perform a new sensing action for the purpose of acquiring new information about the environment to explore and, in particular, which sensing location, maximizes this information gain or other forms of utility. This problem is addressed by the robot's *exploration strategy*. However, merely maximizing the information gain in each loop of the exploration process, as defined above, will not necessarily lead to an optimal solution. What has to be taken into account in the decision process is the *cost* being incorporated with this decision, e.g. the time needed for travelling to this location and for performing the sensing action. According definitions of *exploration* and a 'good' *exploration strategy* are provided in (Yamauchi, 1997):

Definition 5.1 (Exploration)

Exploration is the "act of moving through an unknown environment while building a map that can be used for subsequent navigation."

Definition 5.2 (Exploration Strategy)

"A good exploration strategy is one that generates a complete or nearly complete map in a reasonable amount of time."

Randomly wandering around in the environment can, in this context, also be seen as an exploration strategy, although it will surely not yield optimal results. However, since it is the most simple strategy it will be taken into account in Chapter 5.3.1. As a side note, it is to remark that, in the same way as a random movement, also performing some other navigational task, searching for a particular object for example, leads to an exploration of the environment.

Before having a deeper look into different approaches to the task of robotic exploration and inspection as well as various exploration strategies that can be found in recent robotics literature, strongly related problems from the field of Computational Geometry – namely *Art Gallery and Illumination Problems* – will be presented since they form, together with according solutions, a theoretical basis for most of the later presented approaches.

5.1.1 Art Gallery and Illumination Problems

"How many guards are necessary, and how many are sufficient to patrol the paintings and works of art in an art gallery with n walls?". This simple question, asked by Victor Klee in 1973, forms the basis for the classic art gallery problem in combinatorial geometry. Being more precise, a polygon with n walls has to be visually covered by m guards or *watchmen*. One part of Klee's question has been answered by Chvátal. He showed that $n/3$ guards are always sufficient to visually cover a polygon with n walls (and without *holes*, see Chvátal, 1975). Named after the author this proof is referred to as *Chvátal's Art Gallery Theorem.*

Lots of research work has been investigated, especially in finding solutions for problems being deduced by adding different constraints to the classic art gallery problem and in answering related questions. Basically two problems can be distinguished:

- Illumination or visual coverage with **multiple stationary guards**: Here the number of guards being necessary or sufficient to survey an environment has to be determined as well as the positions and orientations of the guards.
- Illumination or visual coverage with **a single, mobile guard**: Here a path, the so-called *watchman route*, is to be determined which then allows a single mobile guard for seeing any point in its environment from at least one position and orientation on its way.

Polygons, Guards and Visibility

The art gallery problem is to determine the number of guards that are sufficient (or necessary) to visually cover or *see* every point in an art gallery. A common assumption is that the art gallery (or any other environment) can be represented by a polygon $\mathcal{P}$ with n vertices $v_1, \ldots, v_n$ and n edges $v_1v_2, \ldots, v_nv_1$. A guard is, thereby, represented by a point $\mathbf{p}$ in the polygon $\mathcal{P}$. If a guard is restricted to be positioned on a vertex or an edge of $\mathcal{P}$ it is referred to as, respectively, a *vertex guard* and an *edge guard*. If the guard's position is not restricted, it is referred to as a *point guard*. All three guards, however, are *stationary guards*, i.e. once positioned they cannot move inside the polygon and the environment. *Mobile guards*, on the other hand, can move through the polygon and patrol along a segment. An important issue in this context, regardless of whether the guards are stationary or mobile, is visibility. That is determining whether one point in the polygon $\mathcal{P}$ can *see* another point in $\mathcal{P}$.

The idea of visibility is, that the boundary of a polygon is solid and nontransparent. A point is visible from another point if the straight line connecting both points does not intersect the boundary of the polygon.

Definition 5.3 (Visibility)
Let $\mathbf{p}$ *be a point in a polygon* $\mathcal{P}$*. A point* $\mathbf{q}$ *is visible from* $\mathbf{p}$ *if the straight line segment* $\overline{\mathbf{pq}}$ *connecting* $\mathbf{p}$ *and* $\mathbf{q}$ *is completely contained in* $\mathcal{P}$*, i.e.* $\overline{\mathbf{pq}} \in \mathcal{P}$*.*

Two points in a polygon that are not visible from each other are shown in Figure 5.1.

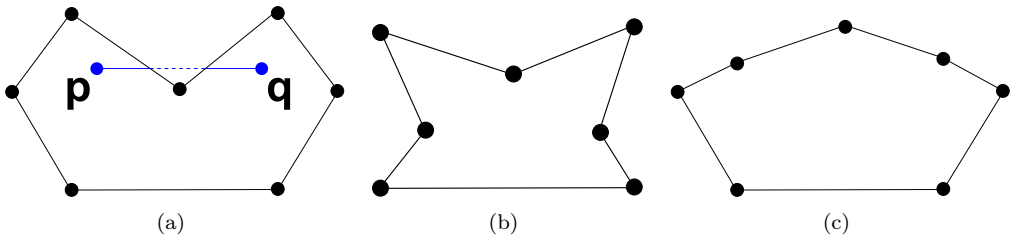


Figure 5.1: Polygons and Visibility. Point $\mathbf{p}$ is not visible from point $\mathbf{q}$. Both the polygon in (a) and the polygon in (b) are star-shaped. The polygon in (c) is convex, i.e. can be observed by a single stationary guard.

Based on the definition of visibility, *star-shaped* and *convex polygons* can be defined. A polygon $\mathcal{P}$ is star-shaped if it contains at least one point $\mathbf{p}$ from which all other points in the polygon are visible (Figure 5.1.b). A polygon is convex if from every point $\mathbf{p} \in \mathcal{P}$ all other points in $\mathcal{P}$ are visible (Figure 5.1.c).

Robotic Exploration and Art Gallery Problems

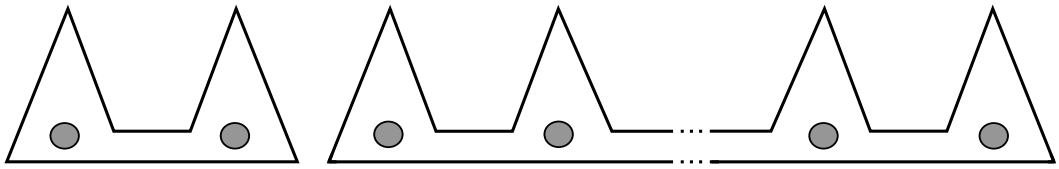
Both, the visual coverage of an environment with multiple fixed guards and with a single mobile guard, are directly related to the tasks of robotic exploration and inspection. Under the assumption that the robot's surrounding environment can be modeled using a simple geometric object, the polygon, an exploration strategy determines either a minimum set of robot poses (positions and orientations in the environment) from which the robot can construct a complete model by sensing the nearby environmental structures, or a path which when followed by the robot allows for sensing all structures and objects in the environment that is to be explored. Closed solutions, however, require that the environment in form of the polygon is already known, which is of course not the case if a mobile robot or a team of multiple robots has to explore an unknown environment. Here, respectively, incremental strategies and online algorithms are necessary which make use of heuristics or various techniques known e.g. from the area of decision theory. Such strategies allow for the determination of a next best view or a (short) path based on the partial knowledge about the environment acquired so far. In the case of inspection the environment is already known and the robot is left with a classic art gallery or illumination problem. However, the complexity of this combinatorial problem does not allow for computing solutions online. Instead, the problem of determining a minimum set of vehicle poses or a shortest paths for completely inspecting an environment is approximated e.g. by sampling candidates from the space of possible vehicle poses until the environment is fully covered.

Robotic exploration does not only deal with the construction of a map for a previously unknown environment, but also with the update of such a map if the environment has once been modeled and the robot already has an internal representation of its surrounding environmental structures respectively. The reason for such a *re-exploration* lies in the fact that an internal world representation for an autonomous mobile robot interacting with its environment should be always up-to-date. As a simple example, think of a service robot which is told to fetch a cup of coffee from the coffee machine. Now, if the position of the coffee machine has been changed after the robot has explored its working area the task of, respectively, fetching a cup of coffee and approaching the coffee machine gets unfeasible. Hence an autonomous robot interacting with its environment must be able

to construct an internal representation of the complete working area within a reasonable amount of time, able to update the internal representation while wandering around or while performing some other task and able to update the complete model, again within a reasonable amount of time. Such a re-exploration or inspection task is e.g. addressed in (Danner and Kavraki, 2000).

The Art Gallery Problem and Chvátal's Theorem

Chvátal (1975) has shown that $\frac{n}{3}$ stationary guards are always sufficient (and sometimes necessary) to visually cover a simple polygon $\mathcal{P}$. Regarding to the number of guards as $g(\mathcal{P})$, it is easy to show that for some polygons three guards are necessary for a full coverage, i.e. $g(\mathcal{P}) \geq \frac{n}{3}$. For a concrete example consider the polygons shown in Figure 5.2. Because of their form, they are also called *comb polygons*. Every tooth of the comb consists of three edges and needs one guard. Accordingly and provable by means of induction, a comb polygon with m teeth and $3m$ edges needs m guards, i.e. $g(\mathcal{P}) = \frac{n}{3}$.



(a) Comb with 2 teeth (b) Comb with m teeth

Figure 5.2: Comb polygons necessitate $n/3$ guards. The comb polygon $\mathcal{P}_2$ (a) needs $2 = {}^6/_3$ guards. The comb polygon $\mathcal{P}_m$ with $n = 3m$ edges needs $m = {}^n/_3$ guards (Figure adapted from Urrutia, 2004, Ch. 2).

In addition to Chvátal, Fisk (1978) has proven that ${}^n/_3$ stationary guards are allways sufficient to guard a simple polygon. His proof is based on triangulation and graph coloring. First, a triangulation $\mathcal{T}$ is computed for the polygon $\mathcal{P}$ (see Figure 5.3). A popular linear time algorithm for triangulating simple polygons has been presented by Chazelle (1991). Given $\mathcal{T}$ one can define a graph $G\mathcal{T}(\mathcal{P})$ such that the vertices of $G\mathcal{T}(\mathcal{P})$ are the vertices of $\mathcal{P}$ and two vertices of $G\mathcal{T}(\mathcal{P})$ are adjacent if they are connected by an edge of $\mathcal{T}$ (see Urrutia, 2000, 2004). *Coloring* now refers to assigning colors to the vertices of a graph such that adjacent vertices have different colors. An important characteristic of G, in this context is the *chromatic number*, i.e. the number of different colors being necessary for a valid graph coloring. For a triangulation $\mathcal{T}$ of a polygon $\mathcal{P}$, it can be easily proven that the chromatic number of $G\mathcal{T}(\mathcal{P})$ is 3 (Urrutia, 2004). That is, $G\mathcal{T}(\mathcal{P})$ for a simple polygon $\mathcal{P}$ can be colored with three different colors (C_1 (green), C_2 (red) and C_3 (blue) in Figure 5.3). The actual proof of Fisk (1978) is quite simple. By coloring $G\mathcal{T}(\mathcal{P})$, the set of vertices in $G\mathcal{T}(\mathcal{P})$ and $\mathcal{P}$ is partitioned into three classes C_1, C_2 and C_3. What can also be seen in Figure 5.3 is that at least on of these classes (C_1 and C_3 in Figure 5.3) contains at most $3 = \lfloor {}^{10}/_3 \rfloor$ vertices, where $\lfloor\rfloor$ is the *floor* operator. Furthermore, for all vertices in a triangle of $\mathcal{T}$ all other points in the same triangle are visible. Since the three vertices of a triangle receive different colors, placing guards at all vertices of one class allows for guarding the complete polygon. Hence, $\lfloor {}^n/_3 \rfloor$ stationary guards are always sufficient to guard a simple polygon with n edges.

Considering the example in Figure 5.3, placing the guards at the (green) vertices of C_1 or the (blue) vertices of C_3 allows for guarding the complete polygon. The depicted polygon has 10 edges and the classes C_1 and C_3 both contain $3 = \lfloor {}^n/_3 \rfloor$ vertices. The original proof of Chvátal (1975) is more complex. Brief summaries can be found in (Urrutia, 2000) and (Urrutia, 2004). An alternative proofs of Chvátal's Art Gallery Theorem can be found (O'Rourke, 1987) and (O'Rourke, 1994).

For moving guards it can be proven that $\lfloor {}^n/_4 \rfloor$ are sufficient to guard a simple polygon (O'Rourke, 1987).

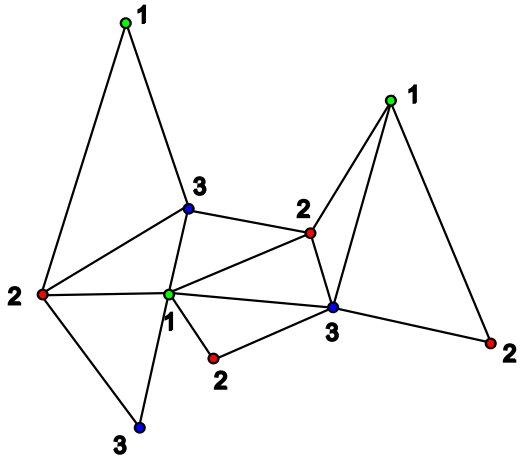


Figure 5.3: Triangulation and coloring of a simple polygon. All three vertex classes allow for completely guarding the polygon (Figure adapted from Urrutia, 2004, Ch. 2).

Variations of the Classic Art Gallery Problem

The classic art gallery problem incorporates various assumptions about the environment as well as for the guards. First of all, it is assumed that the environment, even if it is completely unknown, can be modeled using a polygon or multiple polygons in the two-dimensional plane. However, to deal with more complex structures that require for a three-dimensional model it is a common technique to apply the same assumption by means of a (two-dimensional) bird's eye view on the environment (Nüchter et al., 2003a). Whereas flat indoor environments can be modeled by the interior of a two-dimensional polygonal structure, unbounded outdoor environments can be modeled by the exterior of the same polygon. This problem is also referred to as the *Prisonyard Problem* (see e.g. Urrutia, 2004).

Another special form is the usage of *orthogonal polygons* that contain only right angles, compared to simple polygons, which may contain angles in the continuous range of values between 0° and 360°. That is, all edges of an orthogonal polygon are either horizontal or vertical. Kahn et al. (1983) have shown that $\lfloor n/4 \rfloor$ stationary guards are always sufficient to guard an orthogonal polygon $\mathcal{P}$. If the guards are restricted to be positioned on edges, $\lfloor \frac{3n+4}{16} \rfloor$ guards are sufficient to guard $\mathcal{P}$ (Bjorling-Sachs, 1998). The same upper bound has been proven for guarding an orthogonal polygon with mobile guards (Aggarwal, 1984).

Some polygons, being called *star-shaped polygons* (Figure 5.1), contain points from which, respectively, the complete boundary and interior of the polygon can be seen. This set of points is referred to as the kernel of the polygon (see e.g. Icking and Klein, 1995). In *convex polygons* (all internal angles are not larger than 180°) the interior as well as the boundary can be seen from any contained point and the kernel is thus equal to the polygon itself. Positioning a single stationary guard in the kernel allows for guarding the polygon.

Up to now another assumption is being made – namely that the boundaries of the environment are modeled by the boundaries of a polygon and that the polygon is explored if every part of its boundaries has once been seen. If you now think of pillars and abutments in a room, parts of the environment are hidden if the robot stands in front of such an obstacle. The formerly applied assumption as well as according exploration strategies get unfeasible, since these pillars are not taken into account. To conform to this new requirement *polygons with holes* are examined and used as a basis to find according algorithms. It has been proven, that an art gallery with n walls and h holes requires $\lceil \frac{n+h}{3} \rceil$ point guards (Hoffmann et al., 1991). For vertex guards a sufficient number of $\lfloor \frac{n+2h}{3} \rfloor$ has been proven (O'Rourke, 1987). If the polygon is orthogonal and contains h holes it needs $\lfloor \frac{3n+4h+4}{16} \rfloor$ mobile guards (Györi et al., 1996).

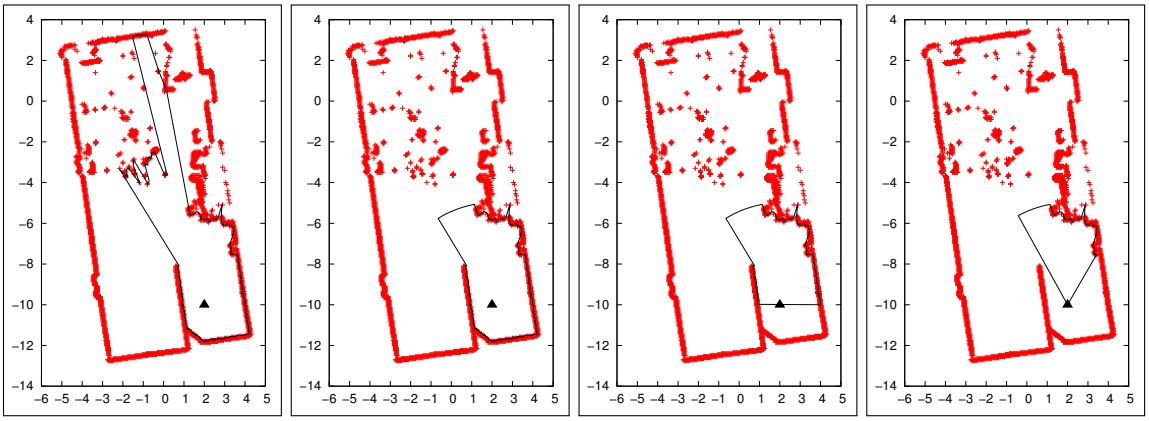
Another problem is that a selected next best view might not always be reachable, for instance it might be blocked by some static or dynamic obstacle or the effort being necessary to reach the

position in the desired orientation is no longer reasonable due to mobility constraints of the robot itself.

Visibility Constraints, Floodlight Illumination and Intruder Detection

A common assumption with respect to the guard is that its range of vision is not limited and thus 360°. Being placed at one position the guard is able to see in all directions, regardless of its orientation. However, sensors for distance measurements and to sense the geometric structure of a surrounding environment are normally limited in their range of vision (e.g. 180° for commonly used 2D laser range finders). By adding such a visibility constraint a strategy for finding a path or a minimum set of positions gets more complex since the orientation, respectively of the robot and the sensor, has to be taken into account in addition to its position. According problem definitions can be merged to *Floodlight Illumination Problems* where the guards are seen as floodlights standing at certain positions, having a certain orientation and a fixed field of view (FOV) (Csizmadia and Tóth, 1998).

Having a limited field of view is also of particular interest in intruder detection problems. In principal, intruder detection or *pursuit evasion* problems are highly related to inspection. The central question of pursuit evasion games is how to guide one or a group of pursuers so that one or more evaders or intruders are detected within a geometric environment (Isaacs, 1965). Parsons (1976) presented a different formulation where intruders and pursuers move along the edges of a graph. A pursuit evasion problem is comparable to an art gallery problem with multiple moving guards. Another possible formulation of such a problem has been introduced by Suzuki and Yamashita (1992). A pursuer or moving guards is able to freely move inside a polygonal environment and emit k beams called *flash lights*. An intruder is defined as a moving point in the interior of the polygon. It is defined as being caught when it is hit by a flashlight. Suzuki and Yamashita refer to these guards as k-searchers as they are able to emit k flashlights. Compared to that, stationary or moving guards without a limited field of view are referred to as ∞-searchers. A model that is inspired by the geometry of laser scanners actually used in mobile robotics has been presented by Gerkey et al. (2006). Their ϕ-searcher has a limited field of view with a size of ϕ without further restricting the angular resolution. That is, compared to k-searchers the ϕ-searcher still emits $k = \infty$ beams. Hence, it can detect all intruders in its field of view. Gerkey et al. primarily distinguish π- and 2π-searches, i.e. guards with a field of view of, respectively, $\phi = \pi$ and $\phi = 2\pi$.



(a) $k = \infty$, $D_{\max} = \infty$ (b) $k = \infty$, $D_{\max} = 5\,\text{m}$ (c) $\phi = 180°$, $D_{\max} = 5\,\text{m}$ (d) $\phi = 60°$, $D_{\max} = 5\,\text{m}$

Figure 5.4: Guards with limited field of view and measurement range. The guard in (a) has an unlimited field of view and an unlimited measurement range. The measurement range of the other guards (b-d) is limited to 5 m. The field of view of the guards in (c) and (d) is limited to, respectively, 180° and 60°.

The main difference between ϕ-searchers and actual range sensors is that the searcher's range of

vision is not limited. That is, only its field of view is limited but not the distance in which it can observe intruders or environmental structures (Kolling and Carpin, 2007). Several approaches, e.g. (Ge et al., 2007), have been proposed that consider both limiting the range of view and the field of view. For exploring and inspecting the workspace of mobile robots in later parts of this chapter, we assume π-searchers that have a limited range of view comparable to the maximum measurable distance $D_{\max}$ of a laser range finder. Different setups for a such guard are visualized in Figure 5.4. Shown is a point map of the RoboCup@Home lab at the campus of the Bonn-Rhein-Sieg University of Applied Sciences. The robot stands at a certain position (marked with a black triangle). Based on the robot's position and the point map, laser range scans have been simulated, where both the field of view ϕ and the maximum measurable distance $D_{\max}$ are limited. The primary used sensors in this thesis are 2D laser range finders with opening angles of $\phi = 180°$ and $\phi = 270°$. They are both limited in range with $D_{\max} = 30\,\text{m}$. To form an ∞-search with limited range of $D_{\max}$ a second laser range scanner can be mounted on the robot in opposite direction to the normally used one. The data of both the front and the rear scanner is then merged to form a single laser range scan with a field of view of 360°. For a comprehensive overview on pursuit evasion games it is referred to (Gerkey et al., 2006) and (Moors, 2008).

Complexity and Approximations of Art Gallery Problems

Finding the minimum number and position of guards is merely of theoretical interest for robotic exploration. Since the environment is initially unknown or only partially known, exploration is rather iterative. The primary problem is determining the next location for acquiring sensory information in order to extend the map. This decision together with the robot's movement to that location and the actual sensing action are repeated until the environment is completely modeled.

In robotic inspection, however, the environment is completely known and it can be addressed just like guarding an art gallery with a single mobile guard or multiple stationary guards. Hence, the problem of robotic inspection becomes the problem of determining a minimum number of vehicle positions where the robot has to sense environmental structures. For a concrete example, consider again the example in Figure 5.3. Instead of placing stationary guards at all vertices of class C_1, an inspecting robot has to approach all vertices of C_1 to inspect the polygonal environment. Herefore, the robot should preferably move along a shortest path what will be addressed later in this section.

Finding the minimum number of vehicle poses or stationary guards to see all points in the interior of a polygon is also referred to as the *minimum guard problem* (Urrutia, 2004). O'Rourke and Supowit (1983) has shown that determining the minimum number of point, vertex or edge guards in polygons with holes is NP-hard. That it is also NP-hard for polygons without holes has been shown by Lee and Lin (1986). However, the problem is not only to determine the number of necessary guards, but also their position. Based on Fisk's proof of Chvátal's Art Gallery Theorem, Avis and Toussaint (1981) proposed an $O(n \log n)$ algorithm for decomposing a polygon into $\lfloor n/3 \rfloor$ star-shaped polygons, to color the resulting triangulation and to determine the positions of the $\lfloor n/3 \rfloor$ necessary guards in the kernels of the star-shaped polygons. For orthogonal polygons, Edelsbrunner et al. (1984) have proposed an $O(n \log n)$ algorithm for locating the $\lfloor n/4 \rfloor$ guards. Their algorithm is based on the corresponding proof by Kahn et al. (1983) and a similar algorithm by Sack (1982). However, both algorithms are only applicable to polygons without holes. A proof that some art gallery problems for polygons with holes cannot even be approximated using polynomial time algorithms has been presented by Eidenbenz et al. (1998). By transforming the art gallery problem into a *set cover problem*, Ghosh (1987) presented an approximation algorithm of minimum edge guard and minimum vertex guard problems for both polygons with and without holes. For simple polygons, the algorithm runs in $O(n^4)$. The resulting solutions are at most $O(\log n)$ times the optimal solution. The same competitive ratio can be achieved for polygons with holes where, however, the algorithm runs in $O(n^5)$ (cf. Ghosh, 1987, 2007). A randomized version of this approximation algorithm has been proposed by González-Baños and Latombe (2001a) forming the basis for their exploration strategy. This strategy is commonly used, e.g. in (Nüchter et al., 2003a) and (Tovar et al., 2006), and is further described in Chapter 5.3.4.

Shortest Paths and Watchmen Routes

The search for a path allowing a single mobile guard for covering the complete environment from at least one position is, in equivalence to the art gallery problem, referred to as the *Watchman Route Problem*. This path can also be generated by finding a minimum set of vehicle poses and connecting them yielding a trajectory throughout the environment. For *Rectilinear Polygons* an efficient algorithm is presented in (Chin and Ntafos, 1986). In later work, the problem is further extended by given doors in the polygons boundary that should be passed by the watchman route (Chin and Ntafos, 1991). If such a door or another position is known as a start point, Chin and Ntafos present an $O(n^4)$ algorithm to calculate the shortest watchman route where n corresponds to number of line segments in the polygon's boundaries. However, that the time bound analysis of this approach and other algorithms in the field of shortest watchman routes are erroneous is shown in (Hammar and Nilsson, 1997).

The exploration of an already modeled environment, e.g. to update the robot's knowledge about its workspace or various kinds of inspection tasks, by means of shortest paths has been presented in (Danner and Kavraki, 2000). Since the environment is already known to the robot the determination of that path can be carried out by means of an (optimal) offline algorithm. In the same way a minimum set of vehicle poses can be computed offline once the environment and especially its structures and boundaries are known. The problem of connecting the vehicle poses is similar to the *Traveling Salesman Problem*. Here visibility and mobility constraints have also to be taken into account. An overview on recent algorithms to determine shortest obstacle free paths in different geometric structures and trees is given by Mitchell in (Mitchell, 1998).

Other questions arising when exploring unknown environments or when searching for a specific object are for instance *"How to look around a corner?"*, i.e. how to move the robot so that it is able to perceive objects occluded by a corner as soon as possible (covered e.g. by Hoffmann et al., 2002) or how to take localization issues into account in the process of planning positions and paths (see e.g. Yamauchi, 1997; Moorehead et al., 2001).

There are lots of further constraints and assumptions which can be applied that lead to many different problem definitions and approaches. For more detailed information on art gallery and related problems, it is referred to (O'Rourke, 1987), (Urrutia, 2000, 2004) and (Ghosh, 2007).

5.1.2 Planning the Next Best View

The selection of *Next Best Views (NBVs)* is a well-known problem in the field of Computer Vision and especially in digitalizing and modeling of real-world objects e.g. by means of range images (Allen et al., 1998; Sequeira et al., 1998). A first approach to NBV planning has been presented by Connolly in (Connolly, 1985). To build a volumetric model of an object by means of several range images the object is mounted at a fixed position forming the origin of the used coordinate frame. A range camera is then rotated around the object. The model is thereby represented by an oct-tree data structure – a three-dimensional representation similar to the concept of occupancy grid maps but organized in a tree-structure, i.e. an attribute is assigned to each cell in the discretized three-dimensional space encoding its occupancy. Possible attributes are *empty*, *occupied* and *unseen*. While *empty* and *occupied* are used as in common occupancy grid approaches, *unseen* is of particular interest and denotes that the corresponding spherical cube has not yet been explored and sensed respectively.

With the objective of modeling an unknown three-dimensional object, Connolly generates sample points yielding *view vectors* that are directed towards the origin. The sample positions are thereby uniformly distributed on a sphere that has been set up around the object. For each sample position, both the number and size of unseen cells seen along the view vector is determined. The sample position yielding the maximum *information gain* by eliminating the largest unseen volume together with the according orientation towards the origin is selected as the NBV.

Although this early work addresses all underlying problems of determining NBVs it requires several assumptions that can not be met when exploring with mobile robots e.g. that every NBV is also accessable by the robot. Furthermore, Connolly addresses only the selection mechanism but

not other incorporated tasks and problems e.g. the cost that is accompanied to position the sensor what is, after all, reasonable when digitalizing objects instead of larger environments.

A complete approach to NBV planning incorporating randomized generation of candidate positions, determination of the NBV by maximizing the expected information gain while, at the same time, minizing costs and planning collision free paths to each candidate has been presented by González-Baños and Latombe in (González-Baños and Latombe, 2002). The representation of costs being associated with a candidate position is thereby solely based on the distance that the robot has to travel along the planned collision free path in order to reach that candidate. To overcome the problem of selecting NBVs that are located in non-traversable areas González-Baños and Latombe introduce *safe regions* containing as few obstacles as possible – candidates are generated to primarily lie in these particular regions. Furthermore, their approach to robotic exploration forms a fundamental basis for recent exploration strategies and is followed e.g. in (Nüchter et al., 2003a) and (Tovar et al., 2006).

Although Nüchter et al. aim at the construction of 3D models, they use the same algorithms and methodologies for candidate sampling by extracting those points in the 3D map that approximately intersect a plane in a height of 1.2 m. To travel from one sensing location to another they apply a controller for approaching vehicle poses that has been designed especially for *non-holonomic robots* (Indiveri, 1999). Thereby, they overcome the aforementioned problem of taking vehicle constraints into account in the planning process.

Tovar et al. extent the approach by González-Baños and Latombe, amongst other improvements, to the application of a team of multiple mobile robots. Furthermore they increase the probability of a successfull localization of an individual robot by adding uncertainty to the cost function, that may arise e.g. by odometric pose estimations. Furthermore, their exploration strategy prefers those NBVs that allow for sensing corners or artificial landmarks and which are likely to provide sensor readings that have a larger overlap with the so far built map.

In the following other approaches to NBV planning and robotic exploration will be shortly presented. Although they appear some kind of grouped in the context of this thesis, the conducted classification is rather fuzzy. Whereas some approaches present a basic principle in its own right, e.g. the *Frontier Approach* by Yamauchi, other approaches share only a common idea with that principle. Furthermore all approaches try to maximize some information gain or the value of some particular function while minizing the value of some cost function. Since these operations carried out over the complete continuous range of vehicle poses is unfeasible for online processing, basically all presented exploration strategies make use of sampling and could thus be classified into a general group of sample-based, decision-theoretic approaches.

5.1.3 Occlusion-based Approaches to NBV Planning

Occlusion-based approaches to NBV planning can primarily be found in digitalization of art (Guidi et al., 2004). The key idea of these approaches is the detection and examination of occlusions in acquired range images in order to recognize exactly those areas that have not been sensed yet but have to be explored in order to construct a complete model of the scene.

One of the first approaches addressing this problem is the one of Maver and Bajcsy (1993). For the acquisition of range images they use a 2D active triangulation system – a range sensor composed of a laser beam emitting device and a camera. This sensor emits a spread laser beam which leads to an illuminated plane in the environment that is simultaneously sensed by a standard CCD camera. Thus, they distinguish two types of occlusions – namely *camera occlusions* and *light occlusions*. Whereas the former one arises if an object occludes the illuminated scene from the camera, the latter one arises if the environment does not get illuminated due to occlusions between the scene and the emitting laser device. Maver and Bajcsy follow a histogram over all single distance measurements in order to find the orientation of the sensor that illuminates a maximum number of edges in the environment. These edges are derived from, respectively, the detected occlusions and perceived obstacles in front of the sensor. As a side note, it is to remark that this exploration strategy is not only used for automated model construction but also to acquire three-dimensional data at all by rotating the sensor around its illumination plane – the same technique as that for taking 3D scans of the environment by means of 2D laser range finders (Wulf and Wagner, 2003).

5.1.4 Frontier-based Approaches

Being focused on grid maps, the basic idea behind this kind of approaches being introduced by Yamauchi (1997), is that the robot should navigate to the frontiers between already explored and unexplored environment to expand, respectively, the map of the environment and the world knowledge of the robot. Thereby, a frontier is defined as the transition of a region of cells estimated to be open space and cells with, respectively, unknown occupancy and initial prior probability of 0.5 (see Chapter 3.2.2).

The initial exploration strategy presented in (Yamauchi, 1997) solely grounds its decision for planning NBVs by the selection of the nearest frontier. Another source of information (Moorehead et al., 2001) has not been taken into account. This is obviously a fundamental drawback of that initial approach since it completely neglects the individual information gains of the surrounding frontiers. However, in a second revision (Yamauchi et al., 1998) other fundamental tasks like localization and path planning are not only performed on the same environment representation but are also taken into account when selecting the frontiers to explore, e.g. by prioritizing those where the probability of a successful re-localization is high.

Another interesting approach to frontier-based exploration is that of Simmons et al. (2000) who apply these techniques to distribute a team of multiple robots over a preliminary unknown environment. This has, of course, a strong advantage regarding the time needed to sense all environmental structures compared to an explorative behavior of a single mobile robot.

Moorehead et al. (2001) define exploration as a task- and application-dependent problem and, thereby, take multiple sources of information into account. For each location or cell (the world is modeled as a uniform grid, similar to Yamauchi, 1997) a vector of cell properties or attributes and a vector of expected information gains is, respectively, considered and stored in the world model. The size of these vectors depends on the number of information sources. To construct a traversability map for example, the cell attribute vector contains information about the reachability, height and traversability of that particular cell together with the certainty in these values. The expected information gain vector accordingly consists of the corresponding expected information gains together with the expected increase of the certainties. An additional frontier information, similar to the approaches in (Yamauchi, 1997) and (Simmons et al., 2000), "rewards" for viewing previously unexplored terrain. The cost of the exploration process, that should be minimized is defined as accumulated time for planning, moving and sensing for each viewpoint.

5.1.5 Decision-theoretic and Sampling-Based Approaches

Decision theory combines the principles of utility and probability theory (see Russell and Norvig, 1995, Chapter 16). In the context of exploration, the utility of a candidate sensing location is determined e.g. by the expected information gain and the distance to that location as a measure of cost. The preference of the exploration strategy for particular sensing locations is represented in the *utility function*. In addition, full decision-theoretic approaches combine the utility of a sensing location with a measure of probability, e.g. representing the certainty about successfully reaching a location or about the expected information gain. The candidate maximizing the value of that function is selected as the NBV. According approaches can thus be classified by the type of utility function and the contained factors.

The utility function of Makarenko et al. (2002), for example, is a summation of the different factors. Thereby a factor with a small value e.g. high cost or a large probability that the robot won't be able to localize itself, does not cancel out other factors with a high value, e.g. the expected information gain. Thus, a selected sensing location might be disadvantageous.

The utility function of Tovar et al. (2006) has a multiplicative form allowing to balance such opposite factors. That is, a candidate with a large expected information gain and a low probability of a successfull localization has a low utility to the robot and is thus unlikely to be selected as the NBV. Furthermore, Tovar et al. take additional costs into account like for instance uncertainty in the robot's pose due to errors in motion control. In comparison, Makarenko et al. (2002) and Newman et al. (2003) do not take these into account (cf. Tovar et al., 2006).

Simmons et al. (2000), here categorized under frontier-based approaches, also investigate not

only the size of surrounding frontiers but also the distance to that frontier as a cost measure and is thus based on two *sources of information* (Moorehead et al., 2001). In the same way the different factors in a utility function in decision-theoretic approaches can be seen as multiple sources of information, whereas approaches like for instance (Newman et al., 2003) and (Amigoni et al., 2004) use only one source of information – the expected information gain. Amigoni et al., however, extent their approach accordingly to a so-called multi-objective strategy in (Amigoni and Gallo, 2005).

Exploiting the nature of probability theory, e.g. by means of probabilistic sensor models and particle filters, is the focus of the work by Stachniss and Burgard (2003b). They use entropy to measure the uncertainty in modeled geometric structures and use this information for both the actual map construction and planning the next sensing locations (Stachniss and Burgard, 2003b; Stachniss, 2006).

5.2 Human-Guided Exploration

The idea of human-guided exploration is that a human user shows the robot around. By this means the robot acquires information about its workspace and constructs an internal environment representation on its own without the need of autonomously exploring the workspace. For a concrete example, consider a human that just bought a mobile service robot. For the robot to be able to accomplish tasks like retrieving a soft drink from the refrigerator in the kitchen, the human user has to show the robot where the kitchen and the refrigerator are located, as well as how the different soft drinks and other objects in refrigerator look like. That is, the robot does not need to be manually provided with all the information being necessary to cope with such a task but, instead, is taught how its workspace looks like using a simple user interface. Problems like this are, amongst others, addressed in the EU FP-6 projects *CoSy* (Cognitive Systems for Cognitive Assistants) and *robots@home* (An open Platform for Home Robotics). In short, capabilities for human-guided exploration provide the means for efficiently providing the robot with all the information being necessary to accomplish subsequent tasks. In the literature a procedure like that is also referred to as *human-augmented mapping* Topp and Christensen (2005).

An according problem formulation forms a particular test in the RoboCup@Home league, namely *Walk&Talk*. In the first part of this test, a mobile service robot has to follow a person through the arena. The person shows the robot around and annotates five locations with a specific name, like for instance the couch and the refrigerator. This annotation is carried out solely using natural speech, e.g. by guiding the robot into the kitchen and in front of the refrigerator. Once the robot has reached that location by following the person, the human guide announces that the object in front of the robot is the refrigerator and the room in which it is located is the kitchen (cf. Nardi et al., 2008). Human-guided exploration consists of several problems that need to be addressed: i) the robot needs to detect persons in its surroundings. Furthermore, it needs to ii) detect the human user that should be followed. As the user is moving, the robot has to iii) track position and velocity of the guide as well as iv) control its movements accordingly. The primary problem in the context of SLAM is that the human guide forms a dynamic obstacle right in front of the robot which, depending on distance and orientation, might occlude larger portions of the environment. That is, for being able to construct a consistent model of the visited regions of the environment, the robot needs to v) filter out resulting dynamics in its sensor readings and vi) robustly localize itself even if larger parts of the environment are occluded in acquired range images. After the teach-in phase is finished, i.e. after the robot has been taken to all five locations, a navigation phase is carried out in which the robot has to show that it has correctly learned all places and that it constructed a consistent environment model in which it can localize itself.

With the algorithms presented in the previous chapters, the robot is already able to navigate in constructed environment representation and to autonomously construct consistent environment representations on its own. Primary issues that need to be addressed here are detecting and tracking the human guide as well as controlling the robot's motion so that it follows the tracked guide.

5.2.1 Detecting and Tracking Human Guides

For detecting the human guide a rather simple approach has been chosen. Just like any other algorithm in this thesis, it is solely based on range sensors. Visual features are not considered altough they can provide the means for better distinguishing the guide from other persons moving around in the robot's vicinity. To address person tracking solely on the basis of 2D laser range scans is, however, not uncommon. Kluge et al. (2001), for example, employ a two-step segmentation of the laser range scan to extract approximately convex objects. They first segment every laser range scan into densely sampled parts. For these parts or clusters, the convex hull is computed. If a point in a cluster has a larger distance to the convex hull than some threshold, the cluster is split into two subsequences at that very point. The splitting of clusters and the resulting subsequences is recursively repeated until the laser range scan is completely segmented into almost convex objects. Using a graph-based matching algorithm, these clusters are associated to other clusters in preceding laser range scans. By this means, dynamic objects can be distinguished from static objects. Furthermore, moving objects can be tracked over a sequence range scans. Their approach is, however, only intended to track moving objects. Tracking and following moving humans is not considered. Although Kluge et al. can track moving people over a sequence of range scans, they do not use a motion model, i.e. they cannot predict future positions of the tracked moving objects.

Montemerlo et al. (2002b) integrate laser-based people detection into a particle filter that is used for both localizing the robot in a previously built map as well as detecting and tracking moving people. They show, amongst other results, that localization can be clearly improved when range measurements corresponding to moving people or dynamic objects are neglected. Their approach is based on a conditional particle filter using a probabilistic model for human motion. By this means, motions can be predicted and people can be tracked even when they are occluded in a sequence of range scans. Montemerlo et al. also present a simple reactive behavior for following tracked persons. The association of measurements to, respectively, a particular person and a particular filter is carried out using a modified nearest neighbor search. Another example of using nearest neighbor search for the data association problem is the work of Lindström and Eklundh (2001). Estimating the robot's pose shift since the last acquisition of a laser range scan, they apply a purely reactive detection of range measurements that were not present in the last scan. After associating these measurements to individual moving objects, each object is tracked using a Kalman Filter.

Schulz et al. (2003) combine the detection, association and tracking of moving people using sample-based Joint Probabilistic Data Association Filters (SJPDAFs). As the number of moving objects to be tracked has to be known for this purpose, Schulz et al. additionally estimate the posterior distribution over the number of objects. If the number of moving objects exceeds the amount of currently used particle filters, new particle filters are added to the SJPDAF. A similar approach also based on SJPDAFs is followed by Topp and Christensen (2005). Their data association is based on scan matching that is also used to determine the robot's ego motion.

Lee et al. (2006a, 2008) first segment the laser range scan into clusters according to the measured distances and merge clusters that are likely to belong to the same person. In this merging step, they assume a constant hip width of 50 cm. These clusters are then tracked using a Kalman filter and a biped walking model. By this means legs can be assigned to individual persons and the persons in the scene can be clearly distinguished. Their approach is, however, not intended to follow persons but to pre-plan the robot's motion in order to avoid collisions with moving people.

Compared to the aforementioned approaches, the detection mechanism use here is rather naive and simple. Instead of really detecting and tracking dynamic objects, the closest object in front of the robot is selected to be tracked. That is, the target to be followed is newly determined for every single range scan. However, instead of determining the closest object in the complete field of view of the range scanner, only an angular range is examined whose size depends on the distance to the last range measurement being followed. By this means, the robot can follow a guide in a distance of e.g. 50 cm while moving along a wall that is only 10 cm away from the robot (see Figure 5.5.a). The idea is to set and move the center of this angular range according to the orientation to the closest measurement in the previously examined angular range. By this means, the angular range moves with the tracked person. Furthermore, the size of the angular range is adapted to take into account that objects occlude a wider angular range when they are closer to the robot. An angle

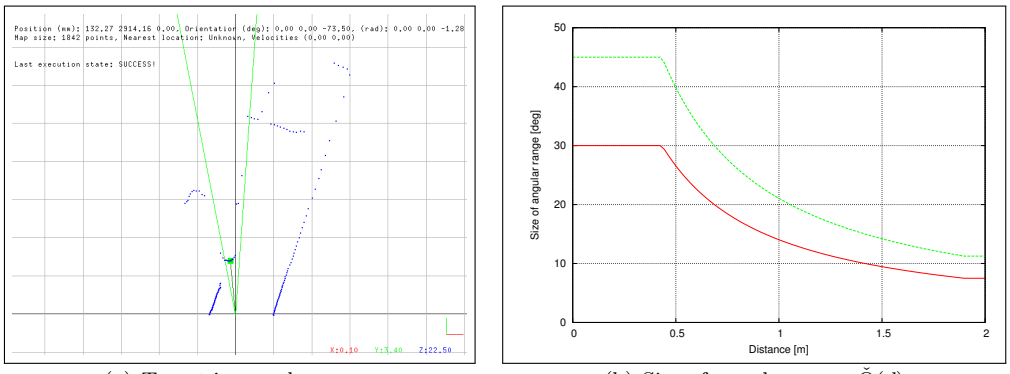
Θ is used to adapt the size of the angular range in both clockwise and counter-clockwise direction from the range center. It solely depends on the distance d to the last measurement being followed:

$$\Theta(d) = c_\Theta \cdot \text{atan2}(0.25\,\text{m}, d). \tag{5.1}$$

The function atan2 computes the arcus tangent taking the the signs of the coordinates into account to determine the angle in the right quadrant of the coordinate system. The constant of 0.25 m allows to detect objects with a width of 50 cm just like the constant hip width assumed by Lee et al.. The factor c_Θ is used to scale the function allowing to adapt the size of the angular range to a certain type of environment. Here, it is simply set to one, i.e. $\Theta(d) = \text{atan2}(0.25\,\text{m}, d)$. To avoid that the angular range gets too small or too large, and upper bound $\Theta_{\max}$ and a lower bound $\Theta_{\min}$ are used to used to limit the size of the angular range:

$$\check{\Theta}(d) = \begin{cases} \Theta_{\min}, & \text{if } \Theta(d) < \Theta_{\min} \\ \Theta_{\max}, & \text{if } \Theta(d) > \Theta_{\max} \\ \Theta(d), & \textit{otherwise} \end{cases} \tag{5.2}$$

Plots of the resulting function are shown in Figure 5.5.b. In the immediate vicinity of the robot the angular range is larger in order to detect objects in a larger range. With an increasing distance to the object, the size of the angular range is reduced to avoid that the tracked point jumps from the guide to a nearby object.



(a) Target in angular range (b) Size of angular range $\check{\Theta}(d)$

Figure 5.5: Size of the examined angular range. Shown are the target in the examined angular range (a) and the dependency of the angular range on the distance to the target (b). Using a simple scaling factor the angular range can be widened (green) and narrowed down (red).

The inherent disadvantage of this simple approach is that the detection is solely based on the latest laser range scan. Consider for example another person that passes by and occludes the human guide from the robot. As the guide is not tracked over a sequence of range scans but newly determined on every single range scan, the other person gets the focus. Hence, the robot is no longer following the guide but the person that passed by. However, by making the robot following fast, the distance to the guide can be limited to a reasonable amount, e.g. 20 cm so that no other person can move in between robot and guide. Of course, another simple alternative is to really track a person once it has been detected e.g. using a Kalman filter. However, experiments have shown that the above procedure is sufficient as long as no person moves between robot and guide and as long as the distance of the guide to surrounding static objects is large enough (e.g. 10 cm) to not cause a jump of the tracked point from the guide to the object.

However, it is a matter of future work to integrate human motion models in order to adequately distinguish the guide from other humans and objects. Interesting approaches in this context are, for example, the biped motion model of Lee et al. (2008) and approaches for learning human motion models especially for prediction (Bui et al., 2001; Bennewitz et al., 2002, 2005). Another idea is adapt the robot's motion, e.g. velocities, to that of surrounding people. An according approach

has been presented by Zender et al. (2007) who detect and track persons in the robot's vicinity in order to navigate in a reasonable and socially acceptable way. Lee et al. (2008), for example, use the information about moving people to plan the robot's motion in order to avoid future collisions and to keep a minimum clearance to expected trajectories of surrounding persons.

A nice property of the presented approach is that it does not require a map of the environment and can thus be applied in completely unknown environments. However, the same holds true for detection mechanisms that track the guide only over a sequence of laser range scans and do not necessitate a model of static objects to detect moving people. In the context of human-guided exploration it would also make no sense if the detection and tracking algorithm required a map of the environment. Furthermore, as this approach does not depend on a specific human motion model or the leg-like structures in the laser range scan, the robot can detect and follow arbitrary moving objects like, for instance, another robot.

5.2.2 Following Tracked Targets

Regardless of whether the guide is detected using a sophisticated tracking algorithm or using a simple approach as presented above, another central problem is how to control the robot's motion so that it adequately follows the guide. That is, the translational velocity v and the rotational velocity ω need to be controlled in way that the robot follows the guide on a smooth trajectory. Important issues are to keep the distance to the guide small to avoid that the guide gets out of the robot's field of view, e.g. in the vicinity of corners. However, the robot should also maintain a minimum clearance to the guide. Informally, these issues can be described as pursuing the guide without pestering it. An according behavior of the robot is also referred to as being *socially acceptable* (Zender et al., 2007).

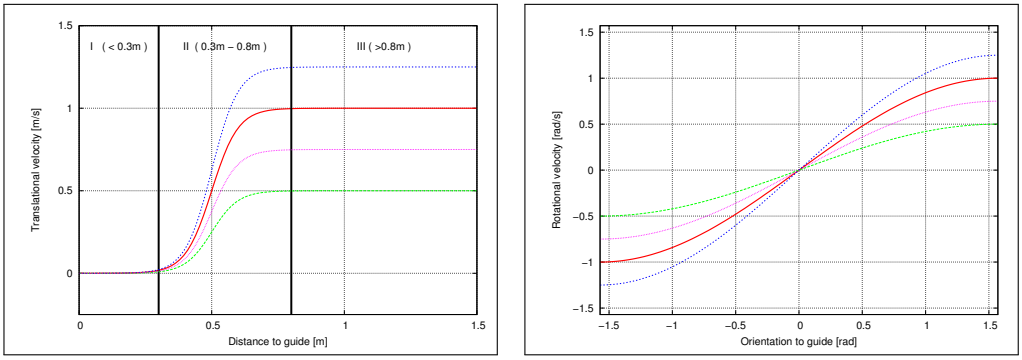
The issue that the guide can get out of the robot's field of view is addressed by Lavalle et al. (1997). They propose probabilistic algorithms for maximizing the expected visibility of the guide while taking into account both motion and visibility constraints. Others issues are that the guide might be occluded by obstacles and that the robot has to avoid collisions while following. González-Baños et al. (2002) proposed a motion controller that explicitly computes the geometric structure of surrounding obstacles. This information is used for avoiding both collisions and occlusions of the guide. Topp and Christensen (2005) use simple linear motion controllers to follow the guide. Translational and rotational velocities depend, respectively, on the distance and the orientation to the target to follow. They explicitly use these following capabilities in a human-guided exploration approach in (Topp et al., 2006). Shaker et al. (2008) use a motion controller based on fuzzy-inference to follow human guides. Due to the fuzzy nature of the controller, the robot's moves along a smooth trajectory even when the guide performs abrupt movements. Parker (1997) addresses the problem of controlling and coordinating multiple mobile robots to track one and multiple targets. Müller et al. (2008) presented a path planning algorithm that explicitly takes into account detected moving people. The robot reaches the target location by following people moving into the same direction.

The approach used in the context of this thesis is, again, kept simple while satisfying the most important requirements. The motion controller is inspired by the simple *follow-the-carrot* controller (see Chapter 4.5) and consists of two control laws. The first one adapts the robot's translational velocity v (Eq. 5.3) according to the distance d to the tracked target. To avoid that the guide gets occluded by environmental structures or obstacles, the robot follows rather fast while still maintaining a minimum clearance of 20 cm. This also avoids that another person moves through the space between robot and guide thereby becoming the tracked target. The second control law steers the robot by adapting the rotational velocity ω (Eq. 5.4) according to the orientation ϕ to the guide. As long as the translational velocity is larger than zero, the robot moves along a smooth trajectory. The sinusoid guarantees small rotational velocities in cases where the orientation to the guide is almost zero. That is, larger oscillations are avoided when the guide is moving almost right in front of the robot. In cases where the guide is in the immediate vicinity of the robot, e.g. $d \leq 20\,\text{cm}$, but not standing in front of it, i.e. $\phi \neq 0$, the robot turns on the spot. That is, $v = 0$ and $\omega \neq 0$. This is, of course, only possible if the robot platform is able to turn on the spot.

$$v(d) = c_v \cdot \frac{1}{1 + \exp\left(-\frac{d-d_0}{\delta_d}\right)} \tag{5.3}$$

$$\omega(\phi) = c_\omega \cdot \sin\phi \tag{5.4}$$

The parameters c_v and c_ω can be used to scale the control laws. The control law for the translational velocity $v(d)$ can be further parameterized by setting the inflection point d_0 as well as δ_d determining the slope. A plot of $v(d)$ with different values for c_v is shown in Figure 5.6.a. What can be seen is that the possible outputs of the control law are split into three regions. If the guide is very near (region I), the translational velocity is (almost) zero. If the guide is further away (region III), the robot drives with the maximum translational velocity (here $1\,\mathrm{m\,s^{-1}}$). In between (region II), the exponential function guarantees smooth accelerations. The steering control law is nothing more but a scaled sinusoid. Having a direct linear dependency on the orientation to the guide as in follow-the-carrot turned out to produce less smooth trajectories and to stronger tend to oscillations especially when the tracked point jumped from one leg to another. What is not explicitly described in the control laws is that in the actual implementation, c_ω is adapted according to the region in $v(d)$. If the robot is moving faster because of a larger distance to the guide (region III), a larger c_ω is used in order to allow for moving along a larger curvature. That is, the robot cuts the corner. A larger c_ω is also used when the guide is in the immediate vicinity of the robot (region I) allowing for fast turning on the spot in order to regain a small orientation ϕ.



(a) Translational velocity $v(d)$ (b) Rotational velocity $\omega(\phi)$

Figure 5.6: Motion control laws for following tracked guides. Shown are plots of the control laws for (a) the translational velocity $v(d)$ and (b) the rotational velocity $\omega(\phi)$.

The motion controller as described above has been excessively tested, most notably, in the RoboCup@Home competitions. The robot followed the guide on smooth trajectories even through narrow passages. As the motion controller allows the robot to turn on the spot, the guide can also visit narrow corners, e.g. for teaching a certain location, without loosing the follower when leaving the corner. Problems occurred rarely and only when the tracked target jumped from the guide onto a static object, e.g. when the guide disappeared behind a table and distance and orientation to guide and table fell into the same polar region. Another open issue is to better adapt the robot's movements to surrounding obstacles.

5.2.3 Constructing Environment Models and Learning Locations

For actually learning the position of taught locations and objects, the robot possesses a semantic layer on top of the metric environment representations. This layer consists of a vector storing the name of objects and locations together with position and orientation in the metric environment representations. During the teach-in phase, the human guide can announce that the robot stands in front of some particular object or that it is in a particular room. These announcements are made by

means of natural speech and are detected and recognized by a speech recognition component. The name of the object or the location is then stored in the object vector (the semantic layer) together with the robot's current pose. When commanding the robot to move to one of these locations or objects in subsequent tasks, the robot approaches the same pose in the environment where the corresponding announcement and teaching of that very object has taken place. As information about locations and objects can also be manually provided to the robot, other features like for instance shape and color can be added to the description of objects in the vector. Really using this additional information is, however, a matter of future. And so is the autonomous acquisition of this additional information.

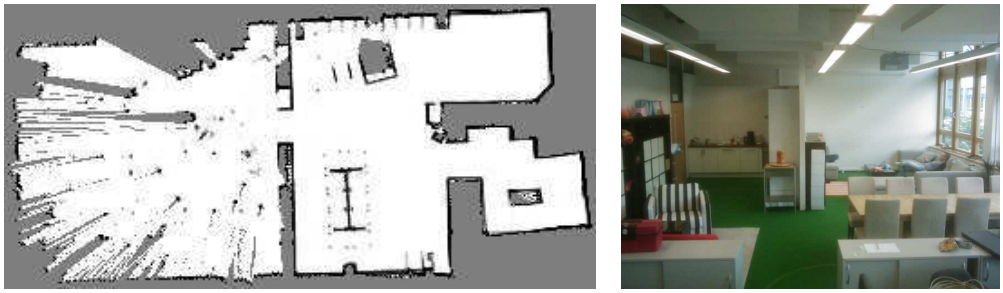
Another important issue is the actual construction of the environment map, especially that of a sparse point map. While following the human guide, larger parts of the environment are occluded for the robot's range sensors. Furthermore, as the guide forms a dynamic object in the immediate vicinity of the robot, phantom effects can be caused. That is, the environment model contains information about obstacles that are no longer present. These phantom obstacles can cause major problems in subsequent navigation tasks. Consider for example, the guide moves through a straight corridor and parts of the guide's legs are sporadically modeled in the map. If these phantom effects form passages that are too narrow for the robot to traverse, this corridor cannot be part of a solution when searching for a path from one room to another. Hence, the robot might take longer paths to reach the goal avoiding the corridor although no object is blocking it. In the worst case, the goal might not be reachable at all when there is no alternative path.

In an early stage of the work presented here, this issue has been addressed by neglecting all measurements in the angular range used for detecting the guide. Although this simple approach already allowed for successful navigation along the shortest path in many cases, it has two major drawbacks. First of all, by completely ignoring all measurements in the angular range, not only the guide but also larger parts of environmental structures are neglected. This might cause localization problems especially when the guide is right in front of the robot and the examined angular range is large respectively. Another problem is caused by the fact that the tracked target can jump from one leg to the other. If, for example, the robot follows the left leg, the measurements corresponding to the right leg might not be completely contained in the angular range. That is, measurements outside the angular range can still cause phantom effects. During the RoboCup world championship in Suzhou 2008, this approach allowed for constructing an almost consistent environment model and to successfully approach a learned location. However, during that navigation phase parts of the taken path necessitated larger changes in orientation due to various phantom effects and the trajectory of the robot was heavily wiggling.

To improve this approach, a simple clustering algorithm has been implemented. Operating in polar space, it segments the range scan into clusters of comparable distance and orientation to the robot. The idea was to neglect the cluster in which the tracked target lies together with those clusters whose distance to the tracked cluster is smaller than some threshold, e.g. 50 cm (the hip width assumed by Lee et al.). Using this extension constructed maps contained considerably less phantom effects. However, other clusters not belonging to the guide could not be modeled by this means. When, for example, the guide closely passed a table, clusters corresponding to table legs were not contained in the resulting map. In the subsequent navigation phase, these not modeled structures can cause that the robot plans direct paths through the table or cuts the corner. That is, in the worst case the robot collides. Having a robust obstacle avoidance can, of course, avoid this collision but the movement of the robot still looks awkward, as the robot is not taking a path that would have been taken by a human. Another possibility of filtering out dynamic objects in range scans is to construct virtual structure and obstacle maps over a sequence of range scans. Dynamic objects are inherently filtered out if enough range scans are used to construct the structure map. Here, the main problem is to determine the size of the sequence of scans that are used in this filtering step. A promising idea is to apply some kind of dynamic window approach that has, however, not been further considered in the work presented here.

In all of the aforementioned experiments, the first SLAM approach has been used, i.e. a sparse point map has been incrementally constructed using the ICP algorithm. Probabilistic reflection maps were not constructed, neither for filtering out phantom effects in sparse point maps nor as an

environment model in its own right. In fact, adequately dealing with dynamic objects for SLAM was one of the reasons for extending the overall system to additionally constructing probabilistic reflection maps in Chapter 3.6. Using the counting model to represent the reflection probability of regions in the environment, completely solves the problem of having obstacles in the map that correspond to the human guide. Measurements taken at the legs of the guide do not need to be explicitly filtered out, but the complete range scan is used to update the map. Assuming that the guide traverses free space, the corresponding regions in the probabilistic reflection map will have less range beams reflected by the guide than range beams passing through the cells without being reflected. That is, the reflection probability of the regions traversed by the guide is low, e.g. < 0.1. A probabilistic reflection map constructed in a human-guided exploration of the RoboCup@Home lab at the campus of the Bonn-Rhein-Sieg University of Applied Sciences is shown in Figure 5.7. The guide started to show the robot around at the doorway in the middle of the map. The robot was then taken through the apartment-like environment in the right half of the map. At the end of the experiment, the guide leaves the apartment and moves two meters into the students' working area (the left half of the map). The guide did not cause any phantom effects and the trajectory taken by guide and robot is not visible in the reflection probabilities of the according cells. Smaller static objects, however, that have been passed by the guide, like for instance legs of tables and chairs are adequately modeled. The resulting map can be used for navigational purposes just like a map constructed in a static environment.



(a) Constructed map (b) Photo of the scene

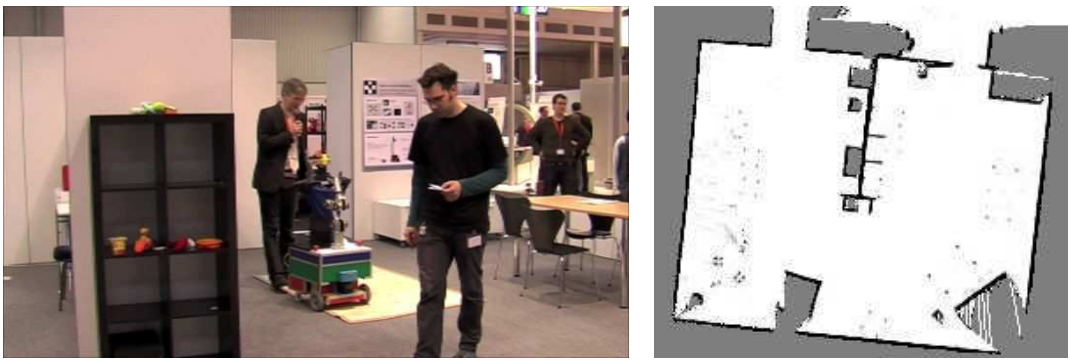
Figure 5.7: Example of a probabilistic reflection map (a) constructed while following a human guide. The guide did not cause phantom effects and actually traversable regions are traversable in the map. The right side of the map is an apartment-like test scenario (b).

5.2.4 Results and Open Problems

Using the simple guide detection mechanism together with the motion controller for following tracked targets and the incremental construction of probabilistic reflection maps, the robot is able to follow human guides and to construct consistent environment models that can be used for navigational purposes. That is, the robot possesses the necessary capabilities for a human-guided exploration of its workspace.

In the *Walk&Talk* test of the German Open 2009 in Hannover, the robot was able to successfully following to and learning all five locations as well as to move to all five learned locations in the navigation phase. Figure 5.8.a shows the robot following a human guide through the arena. At five locations, specified by the referees, the guide stops and tells the robot the name of the just approached location. The map constructed during this guide phase is shown in Figure 5.8.b. The complete video of the robot in this and other tests of the Germen Open 2009 and previous performances during the world championship 2008 are available at `http://www.b-it-bots.de/media`.

The SLAM approach using probabilistic reflection maps togeher with the path planning algorithms for grid maps and the motion controllers for following planned paths both the construction of the environment model during the guide-phase as well as the navigation phase can be regarded as being solved. Open issues are primarily the detection of the guide. Although the currently used simple detection approach suffices in the majority of situations, the robustness of detecting



(a) Robot following guide (b) Constructed map

Figure 5.8: RoboCup@Home Walk&Talk test. Shown are (a) the robot following the person in front as well as (b) the reflection map constructed during this guide phase and used in the subsequent navigation phase.

the guide can be clearly improved by using a more sophisticated approach. Furthermore, the guide is not tracked and the robot reactively follows the target detected in the latest single range scan. Really tracking the guide, e.g. by means of Kalman filters or particle filters, is expected to considerably increase the reliability of the robot's following behavior.

Currently, the approach is solely based on leg-detection in laser range scans. Other means to improve reliability and robustness of detecting and tracking human guides is fusing laser range data with visual information. Cui et al. (2008) proposed such a fusion algorithm using multiple laser range scanners and a camera. The laser range data is used to detect newly appearing persons and to initialize the mean-shift filters (Comaniciu et al., 2002) used for tracking. People are then tracked using laser-based leg tracking and visual body tracking.

Another possible extension is to base detection and tracking on three-dimensional data and to incorporate gestures in addition to speech. Koenig (2007) use a SwissRanger 3D time-of-flight camera to detect the silhouette of persons. Furthermore, they detect gestures like raising an arm. These gestures together with a set of spoken commands can then be used to directly control the robot's movement and to start and stop its following behavior.

5.3 Strategies for Autonomous Exploration

With the aforementioned capabilities for human-guided exploration, the human user possesses efficient means for providing the robot with all the information being necessary to cope with subsequent tasks. A specific exploration strategy is not needed as the user shows the robot around by moving to all relevant places and locations. Moreover, when *teaching* these locations to the robot, the leared places get directly named. That is, the user does not only specify that the current location is relevant and should be contained in the robot's environment model, but also how the location is named and will be referred to (by the user) in later task assignments.

The idea of autonomous exploration is that the robot decides, by itself, where it has to move in order to construct a complete model of its workspace. That is, it is tried to minimize the amount of information that needs to be provided to the robot manually, e.g. by the human user, and that the robot autonomously acquires as much information as possible. Compared to human-guided exploration, the robot will visit all relevant locations by itself as it is exploring the complete reachable workspace. However, these locations are not directly named. That is, the robot cannot guess how a human user might refer to the locations in later task assignments. Hence, even when the robot explores its workspace autonomously, there is still some interaction necessary in which one or multiple human users assign names or certain characteristics to all the places visited by the robot.

Another important difference to human-guided exploration, is that the robot can, in principal,

explore all rooms or regions that are reachable. A human guide can decide, simply by its movements, whether the robot should operate in a certain region or not. Not entering a region in the environment during the guidance phase will cause that this particular region does not get modeled in the robot's internal environment representations and, hence, is not traversed during regular operation later. When the robot explores the environment autonomously and the aforementioned region is reachable, i.e. can be entered by the robot, it will be visited and thus contained in the robot's environment representation. Of course, these regions can also be made unreachable for the robot before it starts exploring, e.g. by closing doors. In the following, different strategies and techniques are presented that allow a mobile robot to autonomously explore its workspace.

5.3.1 Random Walks and Random Exploration

As already mentioned in Chapter 2.4, simply commanding the robot to move forward together with reactive collision avoidance yields an emergent behavior of wandering around. That is, the robot moves randomly inside its workspace. When performing SLAM while moving around, the robot constructs an internal model of the thereby visited places. One could argue that randomly wandering around and constructing a map modeling the seen environmental structures is also a type of exploration although it is not goal-directed. That is, instead of actively moving towards so far unvisited regions, the conducted motion is random and the robot might visit the same place again and again. Still, random motion can be seen as an exploration strategy, although it is not an optimal strategy (Yamauchi et al., 1998). One could expect that wandering around endlessly will also yield a complete environment model after some time. Purely using reactive collision avoidance is, however, not purely random but the environment and objects contained therein determine how the robot has to react. As shown in Figure 5.9.a a static environment might cause that the robot follows the exact same trajectory again and again. That is, the robot is caught in an infinite loop. One possibility to avoid situations like this is to insert pseudo-random movements into the otherwise constant forward movement. Another possibility is to actively remember visited places e.g. by keeping track of the robot's trajectory. When moving into a direction where the robot has been just a few minutes ago, the forward movement and the reactive collision avoidance are interrupted to steer the robot into the opposite direction. The latter strategy has been used to search for an object in the robot's workspace at the RoboCup@Home world championship in Atlanta 2007.

However, it should be mentioned that the experiments visualized in Figure 5.9 were carried out in a simulated environment. That is, the static environment in Figure 5.9.a is really static with not a single change during the experiment. Furthermore, the robot's range readings and its movements are not distorted by noise. That is, a real robot robot will not exactly follow the same trajectory again and again and might, thus, also visit otherwise unvisited areas. This is shown in Figure 5.9.b. Here the environment is not static but a second robot is wandering around in the environment. This dynamic obstacle causes the robot to take another route in order to avoid a collision. By this means, the robot is able, by a fluke, to explore more regions of the environment and to construct an almost complete model of the environment. Compared to the static case in Figure 5.9.a less regions remain unvisited and unmodeled in the robot's internal environment representation.

As described in Chapter 2.4, several parameters determine the behavior of the robot when reacting to obstacles. Changing, for example, the slope and the inflection point of the exponential used in the determination of the freespace orientation allows the robot to sense and model are larger portion its workspace (Figure 5.9.c). The robot explores the area behind the kitchen table (top) by driving around and the enters the living room (bottom). Further decreasing the minimum distance to obstacles determining whether the robot is stopped and turned away from an obstacle, allows the robot to enter narrow passages. As shown in Figure 5.9.d, all parts of the environment are explored and the robot constructs a complete model of its workspace. For smaller environments like the one used in this example, a pseudo-random exploration by means of reactive collision avoidance can already meet the demands put on a good exploration strategy. For larger environments, however, random exploration is unfeasible. Consider for example an environment consisting of 10 copies of the depicted room connected by a corridor. Here the chance of constructing a complete model by means of random exploration is rather improbable. Only with a very small probability the robot

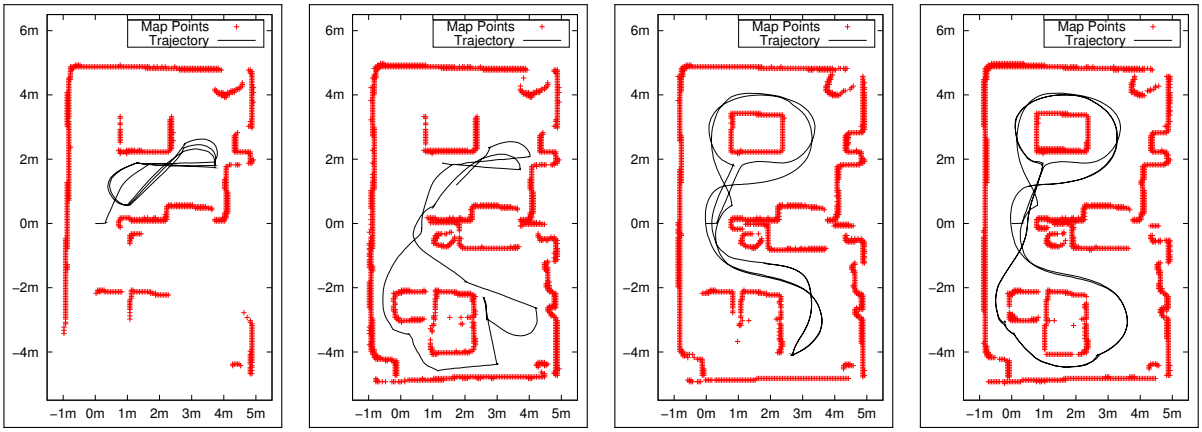


Figure 5.9: Example of a random-walk exploration. Visualized are the trajectories and the constructed maps of a robot randomly exploring a static environment (a) and a dynamic environment (b). Adapting the configuration of the underlying reactive collision avoidance allows to model more regions of the environment (c+d).

will enter every single room and if so, the time taken for exploration will considerably exceed that of a goal-directed exploration strategy.

Here it is referred to (Wagner et al., 1998) where random walks for exploration as well as other randomized approaches to robotic exploration and coverage of graph-like structures have been investigated.

5.3.2 Exploring Frontiers in Probabilistic Reflection Maps

A very intuitive exploration strategy is that of actively approaching previously unvisited regions in the robot's workspace. Frontier-based exploration refers to visiting the transition between modeled free space and unvisited regions in the environment (Yamauchi, 1997). In a frontier-based exploration the robot needs to follow three fundamental steps: i) the robot needs to determine the frontiers in the so far built environment model, ii) it has to select a frontier or a next best view in the vicinity of a frontier and iii) it has to approach the selected frontier. All three steps are described in the course of this section.

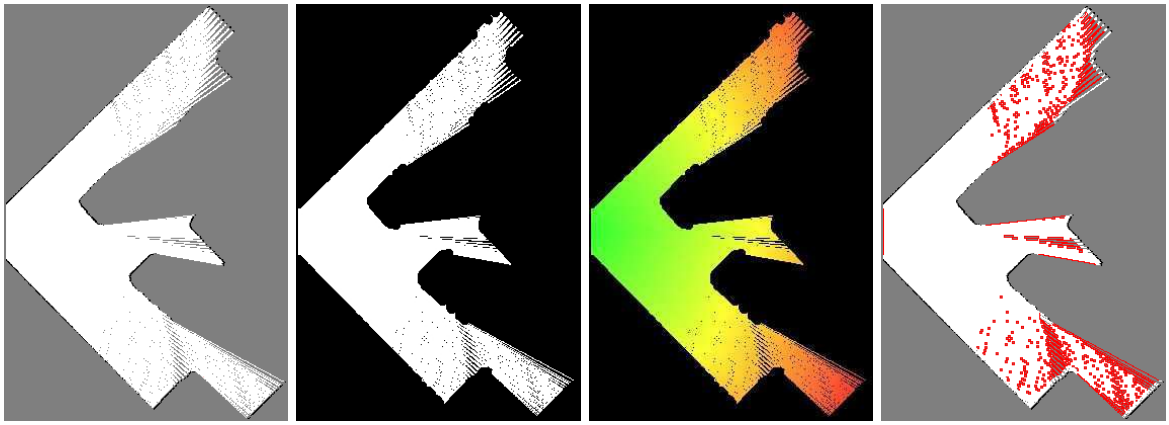
Determining Frontiers

Probabilistic reflection maps directly provide the means of determining frontiers to unexplored terrain since they distinguish free and unknown regions in the map. Here, we define a frontier as being a traversable and reachable cell in a probabilistic reflection map that has at least one neighboring cell whose reflection probability is unknown, i.e. exactly 0.5. Traversability in this context means that the frontier cell's distance to, respectively, the nearest obstale and the nearest cell with a large reflection probability, is larger than some threshold, e.g. 30 cm. Reachability thereby corresponds to the concept of reachability maps presented in Chapter 4.4.5. That is, a grid map encoding for every cell whether or not there is a pass from the robot's current position to the center of that cell. Furthermore, it contains the length of the shortest path to a cell. A procedure for computing a list of all frontier cells in a partial reflection map can be formulated as follows:

1. Determine the set T of traversable cells, i.e. compute a traversability map (Chapter 4.4.3).
2. Determine the set R of reachable cells, i.e. compute a reachability map by conducting a path search without goal specification (Chapter 4.4.5).
3. Determine the candidate set C of cells that are reachable and traversable, i.e.
$$C = \{c^{[xy]} \mid c^{[xy]} \in T, c^{[xy]} \in R\}.$$
4. Determine the set of frontier cells F by checking for every cell in the candidate set C if it is adjacent to a cell with unknown reflection probability:
$$F = \{c^{[xy]} \mid c^{[xy]} \in C, \exists c^{[(x+m)(y+n)]} : P(c^{[(x+m)(y+n)]}) = 0.5, m \in [-1,1], n \in [-1,1]\}$$

As described in Chapter 4.4, computing traversability and reachability maps is inherently carried out in the path planning algorithm for grid maps. One constraint in the determination of traversability was to keep a minimum distance to unknown cells. For determining frontier cells this constraint has to be neglected since the desired cells are those that are directly adjacent to unknown cells.

An example of determining frontier cells by means of the above procedure is depicted in Figure 5.10. What can be seen, besides the determined frontier cells, is that the reflection map contains artifacts due to the fact that artificially widening laser beams (see Chapter 3.2.2) is disabled. In the context of frontier-based exploration this leads to the nice effect that the robot approaches visible regions not directly in the robot's vicinity. With beam widening visible regions will not exhibit frontier cells and the robot might never move to the corresponding part of the environment. Moving there, however, might add additional information and detail to the so far built model. Hence, beam widening is disabled in the update procedure of reflection maps in order to show the artifacts that attract the robot in frontier-based exploration. Furthermore, it should be noted that in this particular case all traversable cells are also reachable which might not be the case in any environment.



(a) Reflection map (b) Traversability map (c) Reachability map (d) Frontier cells

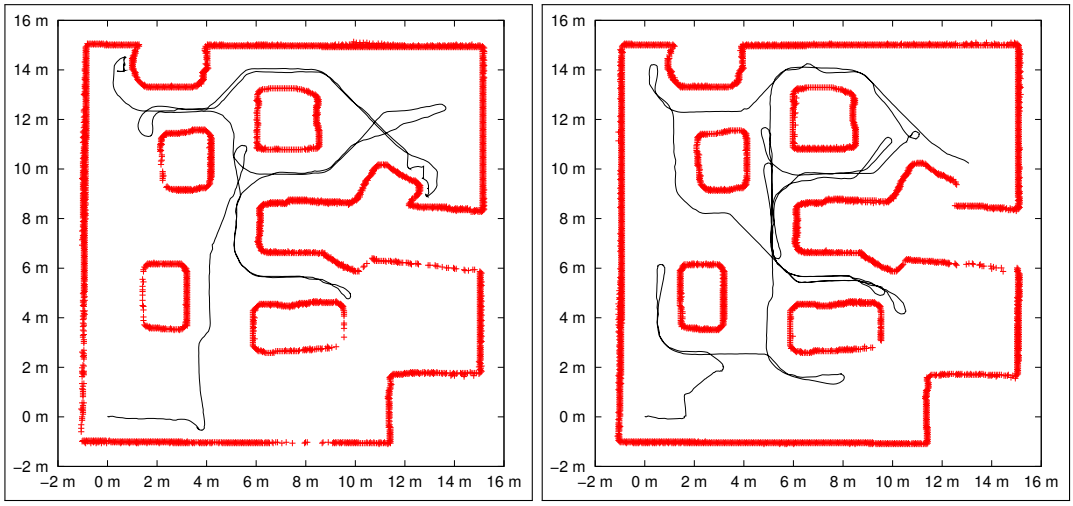
Figure 5.10: Determining frontiers in a probabilistic reflection map. Shown is a partial reflection map (a) together with the computed traversability and reflection maps (b+c) used for determining frontier cells (d). Frontier cells are marked red. The color coding in the reachability map corresponds to the length of the shortest path to that cell.

Randomly Selecting Frontiers as the Next Best View

After determining all frontier cells in the so far built model, the next question is which frontier cell should be approached for the next sensing action. That is, the robot needs to decide which frontier cell is the next best view. A comparably simple strategy is to pick frontiers at random. After determining all frontier cells in the probabilistic reflection map, the robot moves towards a random frontier cell. Naturally, the robot might perform zigzag-movements by this means, as it might approach a frontier cell in another region of the environment and return to the current region for the next frontier cell to approach. In applications where the robot does not continuously acquire information about environmental structures and update its internal environment representation, this strategy can lead to major map inconsistencies since the robot is travelling larger distances without acquiring new information to localize itself. In the context of this thesis, however, information is acquired continuously even when using 3D laser scanners. Nüchter et al. (2003a), for example, stop the robot at the next best view to acquire a 3D scan. The scan is then registered with the so far built map. Based on the updated map, the next best view is determined. During navigation, the pitching 3D laser scanner is kept in a horizontal position to track the robot's pose based on acquired 2D scans. The map is, however, not updated. When continuously acquiring new information and updating the map, the robot can travel longer distances, e.g. through large regions containing frontier cells. Furthermore, when moving through regions containing frontier cells, continuously updating the map will also add new information about the corresponding transitions between free and unknown regions in the map. Hence, previously determined frontier cells do no longer meet the conditions for being a frontier cell. That is, by moving to a random frontier, all regions being passed by the robot are updated and might no longer contain unknown space. This drastically decreases the number of poses that are approached by the robot and the number of next best views. The trajectory of a robot randomly exploring frontiers in a simulated environment is shown in Figure 5.11.a. The robot approached a total of 7 frontier cells until the exploration task has been terminated because the map did not contain any more frontier cells. The individual paths to the selected frontier cells are rather long and the robot explored larger regions of unknown reflection probability not belonging to the approached frontier cell. This is the reason why the number of view is so small, although the environment is rather large. However, it should be noted that solely acquiring information at the selected frontiers will not result in a complete environment model. That is, the robot's workspace is only completely covered since the robot was acquiring information and updating the map continuously.

A general issue in frontier-based exploration is that the currently approached frontier cell might be no longer adjacent to a cell of unknown reflection probability due to the continuous map updates. When approach a cell at the end of a corridor, for example, the robot will move to this end although the corresponding environmental structures have already been sensed and modeled in the robot's internal environment representation. This also increases the chance of getting stuck in a narrow passage or corner if the robot still wants to approach a cell contained therein. Especially when not conducting map updates, the robot might not recognize that the approached region is, for any reason, no longer traversable.

Whether or not the currently approached cell is still belonging to a frontier can, however, easily be checked. If none of the cell's neighbors has an unknown reflection probability, the cell is no longer a frontier cell and the robot needs to determine a new next best view. We will refer to this procedure as *repetitive (re-)checking* and to the overall strategy as *repetitive random frontiers*. However, as can be seen in Figure 5.11.b repetitively checking the status of the traversed frontier cell considerably increases the path taken by the robot until the environment is completely explored. Random frontier exploration (Figure 5.11.a) resulted in a path of 96.28 m length. With repetitive random frontier exploration (Figure 5.11.b), the robot travelled over a distance of 132.81 m until the constructed map was complete. This increase lies in the nature of random frontier exploration, as the robot might more often traverse the same region to approach frontier cells in other parts of the environment. The center part of this scenario, for example, is traversed seven times with repetitive rechecking and only four times without. Repetitive re-checking can, however, improve other frontier-based exploration strategies.



(a) Random frontiers (b) Repetitive random frontiers

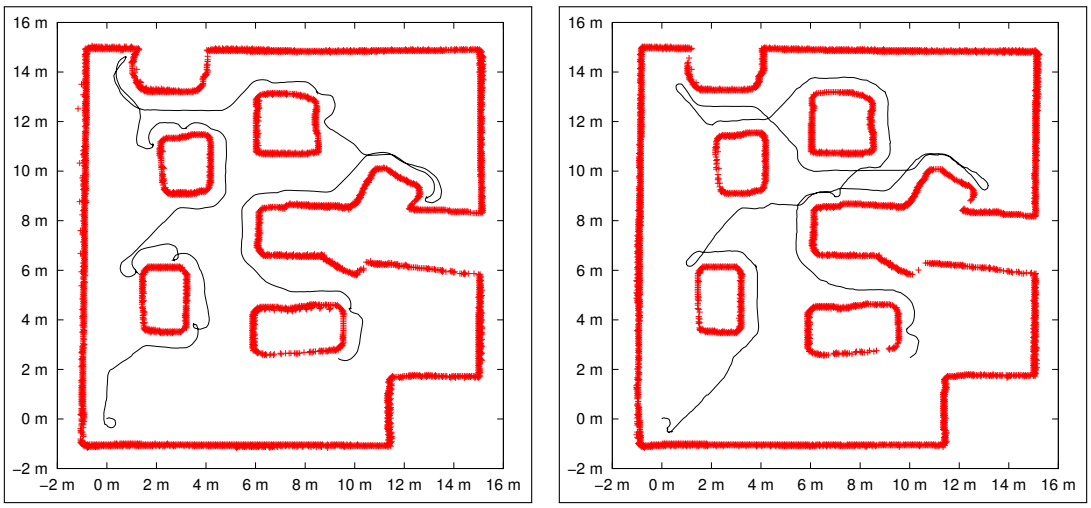
Figure 5.11: Exploring random frontiers. Shown are the trajectories of a mobile robot exploring a simulated environment. In (a) the robot randomly approaches frontier cells. In (b), the robot additionally checks whether the currently approached cell is still a frontier and picks another cell if not.

Exploring Closest Frontiers

More reasonable than randomly selecting frontier cells, is to approach that very frontier that lies closest to the robot and for which the shortest path is shorter than those to all other determined frontier cells respectively. Exploring closest frontiers is the strategy that has been proposed by Yamauchi (1997) when introducing frontier-based exploration. Here, we will select the closest frontier cell based on the reachability map that has been already used to determine whether or not a cell is reachable at all. A nice characteristic of the reachability map is that it directly contains the length of the shortest path to every reachable cell. This path map, constructed using Dijkstra's search algorithm without a goal specification, not only contains the length of the shortest path but also the preceding cell along the shortest path for every cell. That is, the contained path length can be used to determine which frontier cell is the closest and the tree-like structure allows for directly extracting the shortest path to the selected closest frontier cell. Besides the construction of the reachability map, no additional path planning is necessary.

When exploring closest frontiers, one and the same region in the environment is normally not traversed more than a reasonable amount of times and the trajectory of the robot follows a trend from one unknown region to the next. A trajectory of the robot exploring the simple example scenario, that has already been used for exploring random frontiers, is shown in Figure 5.12.a. In these experiments, the robot uses only a single laser range scanner mounted in front. The apex angle of the scanner is limited to 180°. This causes a characteristic behavior when starting the exploration task. All cells along the y-axis of the robot's coordinate frame form a transition from free to unknown regions and are, for this reason, determined as being frontier cells. This naturally holds also true for the two cells on the y-axis that are directly adjacent to the robot's center of rotation. When exploring closest frontiers, these two cells are approached first, causing that the robot almost turns on the spot. During this rotation, new information is acquired about the preliminary unknown region behind the robot. As soon as no frontier cells remain in the robot's vicinity, it starts leaving its initial position and exploring other regions in its workspace. In all experiments in this example scenario, the robot starts in the lower left of the environment. The traversed circle at the beginning of the robot's trajectory is clearly visible in this region.

What can also be seen in Figure 5.12.a is that all approached frontier cells lie in the direct vicinity of environmental structures and behind corners respectively. This leads to the undesired behavior



(a) Closest frontiers (b) Repetitive closest frontiers

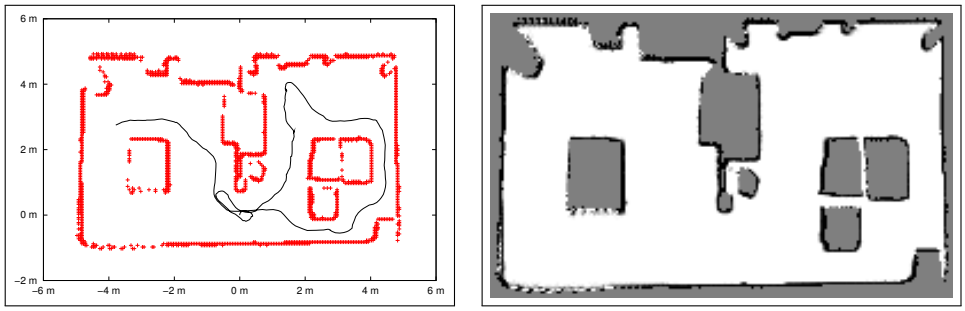
Figure 5.12: Exploring closest frontiers. Shown are the trajectories of a mobile robot exploring a simulated environment. In (a) the robot selects the closest frontier cell as the next best view. In (b), the robot additionally checks whether the currently approached cell is still a frontier and determines the next closest frontier cell if not.

that the robot moves directly in front of environmental structures when approaching a selected frontier. For moving to the next closest frontier, the robot needs to move backwards or turn on spot to regain a minimum clearance to surrounding objects. Although the robot's trajectory for covering all regions in the environment is rather short (73.71 m), the time for exploring seems to be rather long for an external spectator as the robot is slowing down several times for approaching cells in the vicinity of environmental structures as well as moving backwards and turning to reposition itself in free space. Here, repetitively re-checking whether the currently approached cell is still a frontier and determining the next frontier cell to be approached, not only shortens the traversed trajectory but also considerably decreases the time needed for exploring the complete environment. The trajectory resulting from exploring closest frontiers with repetitive re-checking is shown in Figure 5.12.b. It has a total length of 70.45 m. However, both trajectories show a reasonable exploration behavior and the robot explores one unknown region after the other.

That the trajectories of a mobile robot exploring closest frontiers are kept reasonably small has been shown by Koenig et al. (2001). They provide both a lower and upper bound on the distance travelled by the robot. However, the number of poses that need to be approached in a closest-frontier exploration is larger compared to that of other strategies. Stachniss and Burgard (2003a) have shown, in experiments, that the bounds by Koenig et al. hold, but that other strategies yield considerably less views. This is, however, quite natural since frontier-based exploration does not take into account the expected information gain at the currently traversed position. From another pose in the workspace a complete unknown region might be visible. Approaching this pose instead of the frontier to that region can drastically decrease the length of the traversed path. A larger number of sensing location is, however, only problematic when not continuously acquiring information and updating the map. Also shown by Stachniss and Burgard (2003a), is that the traversed path was the shortest for closest-frontier exploration in all conducted experiments.

Regarding the explorative behavior of the robot without repetitive re-checking, Mei et al. (2006) combine an exploration strategy, that is similar to frontier-based exploration, with a special motion planner. With this combination, Mei et al. try to not only minimize the path travelled by the robot but also the number of turns and stops. Such an approach can also be used to handle the problem of approaching cells in the immediate vicinity of environmental structures when not performing repetitive re-checking. As this re-checking is, however, nothing more but a constant time lookup in the reflection map, it is preferred in the context of this thesis. As a frontier cell is defined as being

adjacent to a cell whose reflection probability is exactly 0.5, only eight cells need to be examined. Looking up the reflection probability of a cell in the grid map is constant and done $O(1)$. If none of the neighboring cells has a reflection probability of 0.5, the currently approached cell is no longer a frontier cell and the robot will not further traverse towards it and a possibly near environmental structure or obstacle respectively.



(a) Trajectory and point map (b) Constructed reflection map

Figure 5.13: Exploring a simulated RoboCup@Home environment. Shown are the trajectory of the robot (a) as well as the reflection map constructed during exploration (b). After exploring the living room (left part), the robot enters the kitchen (right part). After *looking* behind the table the environment model is complete and the exploration task is terminated.

Another nice characteristic of the closest frontier exploration with repetitive re-checking is that the robot immediately turns into another direction once the currently entered region is fully explored. The resulting behavior of the robot e.g. when moving around a corner can be seen in Figure 5.13. In this example, the robot is exploring a simulated environment reconstructed from a data set recorded during the RoboCup GermanOpen in Hannover 2008. Starting at $(0, 0)$ the robot performs a small loop and then explores the living room on the right side of the map. After moving around couch, arm chair and couch table, the robot enters the kitchen on the left side of the map. With the first range scans providing information about the backside of the table, no more frontier cells can be determined and the exploration task is terminated. What is important in the context of service robotics is that the resulting behavior of the robot is more than reasonable for an external spectator or a human operator. However, this characteristic of repetitive re-checking can be further improved by applying techniques to reasonably move around corners. Fekete et al. (2004) present a strategy for looking around corners which guarantees that the driven curvature is not considerably longer than taking the direct pass while providing information about the region behind the corner as early as possible. Such a strategy could replace the motion controller described in Chapter 4.5 exactly in those situations where the robot approaches a frontier cell behind or adjacent to an already modeled environmental structure.

Non-Existence of Frontier Cells as a Termination Criterion

Besides the question of how to control the movement of a mobile robot to fully cover its workspace, another very important question in the context of autonomous exploration is when to stop exploring the workspace (Koveos et al., 2007). Since the environment is preliminary unknown the robot can never know that it has fully explored its workspace. A random exploration as described in the beginning of this chapter, for example, does not even take information about the current state of the constructed map into account. Instead, the robot is endlessly wandering around in the environment although after some time a visual inspection of the so far built model would suggest that the map is complete and the robot should stop exploration. Generally applicable termination criteria are, for example, based on time, distance travelled or leaving a previously defined region. All of the above are quite reasonable as long as it can be estimated how long the robot will take to explore the environment or how long the resulting path might be. That is, adequately configuring such a criterion e.g. by setting an appropriate threshold, requires at least a rough estimate of the

size of the environment. Furthermore, it requires knowledge about the strategy followed by the exporing robot.

Frontier-based exploration strategies allow for defining a clear termination criterion – namely the *non-existence of frontiers*. As soon as all known free space in the map is bounded by environmental structures or the boundary of objects, the robot cannot acquire any new information since the reachable workspace is fully explored. This suggests to take the number of determined frontier cell as a termination criterion for the exploration task. When no frontier cell is found in the so far built model, exploration is stopped and the robot can accept new orders. For all frontier-based exploration strategies in this thesis this termination criterion has been used to autonomously decide when to stop exploration. Human-guided and random exploration are, compared to that, run endlessly and only stopped on request by a human user.

Figure 5.14 shows a typical progress of the information contained in an environment model constructed during the exploration of closest frontiers. As the robot moves from frontier cell cluster to frontier cell cluster, unknown regions in the environment are explored one after another. The number of frontier cells during exploration can drastically change. That is, in the middle of the exploration procedure the robot might be left with only a few frontier cells e.g. being located near a doorway or a narrow passage. While approaching this region, new information added to the map causes that more and more frontier cells in the region behind that passage are determined. However, as long as the environment is not fully explored, the number of determined frontier cells will always be larger than zero. Still a little trick needs to be applied to terminate at all in any given environment, as especially highly cluttered environments might cause that single frontier cells remain in the map that lie in non-traversable regions or are where no information can be acquired. Consider for example a chair standing in the corner of a room. If the distance between its legs is large enough and 3D information cannot be used to perceive the complete chair a single unknown cell that is occluded by a chair might be classified as traversable. Hence, an adjacent cell with a low reflection probability might be classified as a frontier cell although the robot cannot acquire information about the unknown cell from any traversable and reachable pose in its workspace. For this reason, the path planning algorithms for computing the traversability and reachability maps are configured to constrain the robot's movements such that it maintains a minimum clearance of at least one cell to an occupied cell, i.e. a cell with a high reflection probability. This extension on its own will, however, not erase all of the aforementioned *non-explorable* cells as single unknown cells might still cause frontier cells in their vicinity. These *single frontier cells* also need be to neglected in the exploration strategy. Here, this addressed inherently in the determination of frontier cells by applying a two-steps approach. In the first step, it is checked for every cell whether or not it is adjacent to a cell with unknown reflection probability. In the second step, it is checked whether or not the unknown cell is adjacent to another unknown cell. Only if the free cell being examined is adjacent to an unknown cell in a larger block of unknown cells it is considered in the exploration strategy.

For safety reasons the robot is stopped when planning the next action and the frontier cell to approach next respectively. However, this stop is not considerable since all algorithms used up to now are computationally efficient and all calculations, e.g. that for determining the closest frontier, are carried out within the update interval of the motor controller. That is, the robot stops and starts moving again within one interval and the translational velocity of the robot is never zero. Some of the exploration strategies presented in the following are computationally more expensive and the robot might stop for a second before travelling to the next sensing location. Due to the very short time the robot is standing still, these short stops are, however, rather undisturbing compared to other systems for autonomous exploration where the robot is standing still for more than 20 s or even multiple minutes. Nüchter et al. (2003a), for example need to stop the robot for acquiring a 3D scan while standing. The scan is then registered with the so far built environment model used to decide where the robot has to move next. The overall procedure during which the robot is standing still, can take longer than a minute.

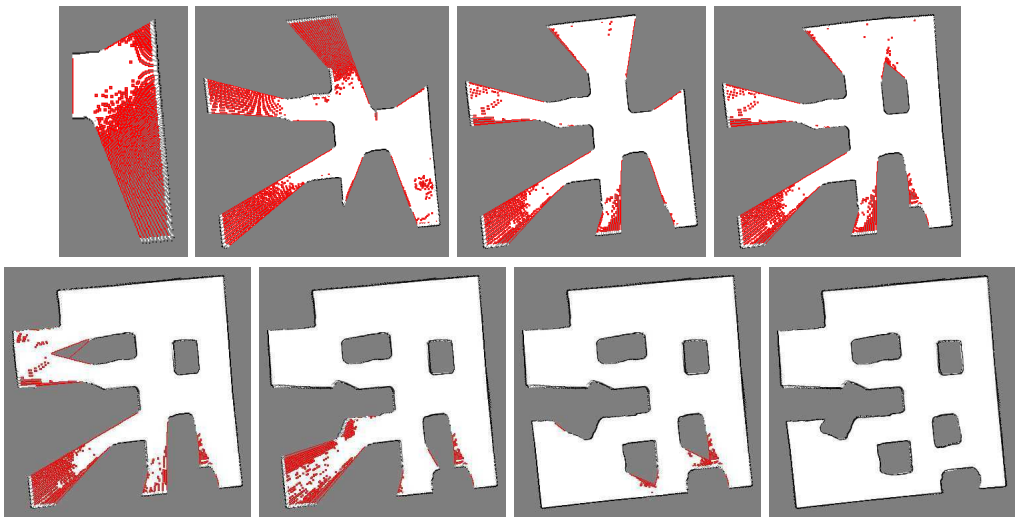


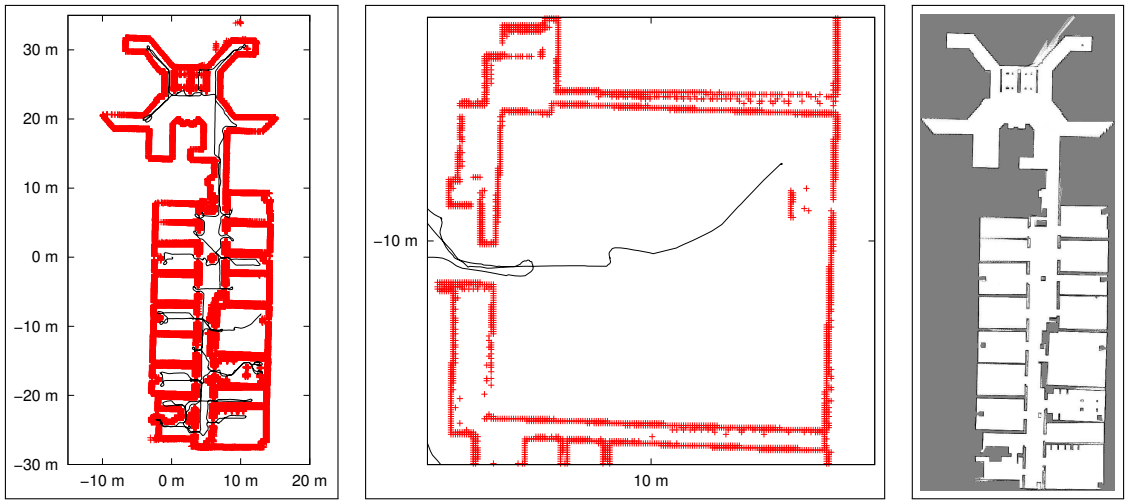
Figure 5.14: Frontier cells in the progress of map construction. Shown are the different states of a probabilistic reflection maps during closest frontier exploration with the currently determined frontier cells marked red.

5.3.3 Exporing Closest Frontiers in Segmented Reflection Maps

Up to now, the explored environments were rather small and did not contain multiple rooms as it is normally the case in domestic or office-like environments. In environments that contain multiple rooms possibly connected by a longer corridor, using an exploration strategy solely based on exploring closest frontiers can cause that the robot enters a room and leaves again before the room is completely explored. Consider, for example, that the currently approached frontier cell lies directly behind the doorway to a room. When the robot has reached this position or when the cell is no longer a frontier cell, it again determines all frontier cells and selects the closest as the one being approached next. Especially if the room is larger, contained frontier cells might be farther away from the robot than frontier cells in other rooms. That the robot continuous its exploration in another region is not a problem by itself, but it might get farther and farther away from that room in the process of exploration. That is, when all other rooms are explored, the robot has to return to the previously neglected room in order to explore the remaining frontiers.

It should be noted that the aforementioned behavior is not always undesirable. If, for example, the frontier cell, for which the robot left the room, is lying on a corridor and the robot continues exploring the corridor until it is completely modeled before it re-enters the room, this explorative behavior is reasonable. If, however, the robot hops from one room to the other or if it explores a chain of multiple connected rooms until it returns, the explorative behavior is no longer reasonable and the path travelled by the robot gets unnecessary long. Such a situation is shown in Figure 5.15. In this example the robot explored a simulated environment reconstructed from a floorplan modeling the 5-th floor of the AVZ building at the University of Osnabrück. This environment contains an entry-way with staircases and elevators as well as several offices and seminar rooms connected by a longer corridor. As the explored environment has itself been reconstructed from a floor plan it is empty and contains only environmental structures such as walls and doors. When entering a room the robot has in most of the cases already fully explored the room by taking a single sensor reading and updating the world model. Some rooms, however, contain pillars whose occlusions in a range scan taken at the door cause, respectively, unknown regions and frontier cells. As these regions might be farther away from the robot than other frontier cells, the robot leaves the room and continues exploration in other regions (Figure 5.15.a). At the end of the exploration task when all other regions of the environment are fully explored, the robot needs to move back to an already entered room in order to explore the remaining frontier cells. This room as well as the loop when entering and leaving for the first time and the trajectory for entering and exploring the

remaining frontier cells are shown, in detail, in Figure 5.15.b.



(a) (b) (c)

Figure 5.15: Closest frontier exploration in an office-like environment. Shown is the path of the robot taken for exploring an environment with multiple rooms (a) as well as a room that needs to be entered twice in order to explore remaining frontiers. The constructed reflection map is shown in (c).

What is desirable in situations like the one just described is that the robot completely explores a room once it has entered it. Edlinger and von Puttkamer (1994) have proposed a hierarchical exploration strategy that explicitly addresses this issue. Their strategy considers a room as an environment on its own and the robot only leaves for exploring other regions once it has fully explored the room. Doorways are used to mark the transition between one room and another. Edlinger and von Puttkamer divide their exploration approach into two exploration strategies, a local exploration strategy for exploring a room and a global exploration strategy for exploring the complete reachable workspace. The local exploration strategy is based on a *virtual bubble* that is bounded in radius by the maximum measurable distance of the used range sensor – a so-called *virtual border*. Sensed environmental structures further limit the boundary of the bubble in the form of *real borders*. The idea of the two exploration strategies is that the bubble can not extend through holes in the real boundary being more narrow than a certain threshold. The local exploration strategy solely moves the robot towards virtual borders, until the bubble does no longer contain virtual borders except for the doorway determined by means of simple length thresholding. Once the bubble is only constrained by real borders or doorways, the room is explored, and the global exploration strategy is used to construct a topological map of connected rooms.

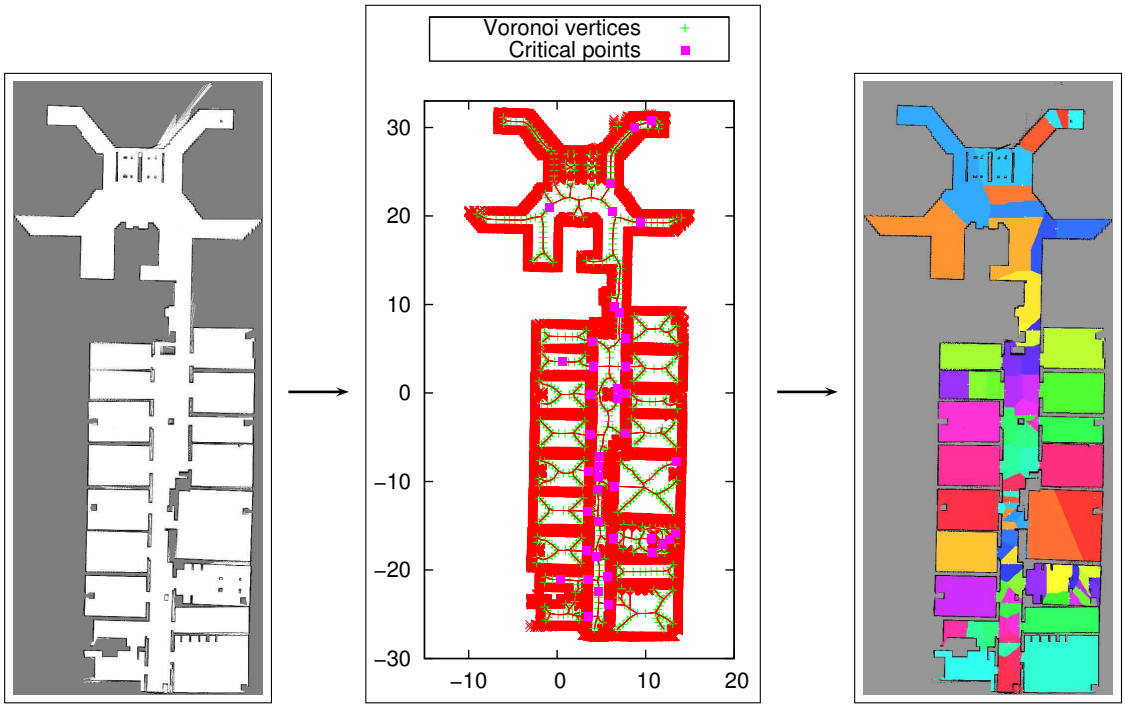
Initial Map Segmentation

This idea can be simply adapted to frontier-based exploration by preferring frontier cells that are contained in the same room as the last cell that has been approached. What remains are the questions on how to segment the map into rooms and how to assign frontier cells to segmented map regions. Thrun (1998) addresses the segmentation of grid maps for constructing a topological map that explicitly describes the connectivity of rooms and segments in the grid map. As path-planning in large scale environments is more efficient than on an explicit metric representation, Thrun tries to combine the strengths of both approaches. Global path planning, i.e. for moving from one segment to another, is carried out on the topological representation whereas local path planning, e.g. moving through a door or approaching a certain position within a segment, is based on the more detailled information in the grid map. Thrun's segmentation to construct the topological map is based on the Voronoi diagram (see Chapter 4.3.3). Wurm et al. (2008) construct the Voronoi diagram for a grid map by means of a skeletonization of the distance transform (Chapter 4.4.3).

Another possibility is to extract the set of cells that are likely to be occupied by an obstacle and to compute the Voronoi diagram for the resulting point set as it was done in Chapter 4.4.1. Once the Voronoi diagram is constructed, Thrun searches for so-called *critical points*. These critical points are nodes in the Voronoi diagram and local minima w.r.t. to the distance to surrounding objects. That is a critical point is closer to surrounding objects than all the nodes being directly adjacent to it. After the set of critical points has been determined, Thrun constructs *critical edges* by connecting critical points to the closest objects in their vicinity. Consider for example a critical point lying in doorway. The critical edges than connect the critical point to the left and right wall besides the doorway. The resulting critical line thereby splits the modeled region and the Voronoi diagram into two parts, one being in front and one being behind the critical point. Thrun also proposes a pruning procedure for the resulting graph that merges adjacent segments. However, as shown by Wurm et al. (2008), this segmentation can still cause a large number of segments especially in cluttered environments. Applying the additional pruning step, however, does not merge all smaller segments. That is, some parts of the environment are still *over-segmented*. Furthermore, the pruning step can merge multiple rather large segments, e.g. in a corridor, to a single very large segment. Such a segmentation is not desirable for assisting an exploration strategy as it might cause the robot to fully explore a very large segment until it returns for exploring smaller adjacent segments. Especially in corridors this can be problematic for the SLAM algorithm. When exporing a corridor, all environmental structures are sensed from one and the same direction. As corridors are by itself rather featureless, distinct environmental structures being important for localization, like for instance corners, primarily lie inside rooms. When returning, i.e. moving along the corridor in the opposite direction, already modeled distinct structures are not perceivable and misregistrations can occur.

To avoid that cluttered environments cause many small segments, Wurm et al. (2008) further constraint critical points to be nodes of degree 2, i.e. nodes that are adjacent to only two other nodes. Furthermore, they define critical points to be adjacent to at least one node of degree 3, i.e. a junction node. However, in experiments carried out in the context of the work presented here, this constraint is too restrictive and especially doorways are not adequately segmented. Especially in the vicinity of wider walls, the node of the Voronoi diagram that directly lies in the doorway may not be connected to a junction node. For this reason the constraint is loosened here. If directly adjacent nodes are not junction nodes, it is sufficient that an adjacent node is adjacent to a junction node. By applying the loosened constraint the segmentation procedure normally found all doorways reliably.

Another issue, as already mentioned, is that Thrun's pruning step might produce segments that are too large for a reasonable exploration. Furthermore, we are not directly interested in obtaining the topological structure of the environment, but in assigning each free cell to an individual segment. For this reason, the segmentation algorithm is adapted to assign free cells to segments based on the reachability map. Determined critical points turn into split points causing the creation of two segments. Both segments get assigned the corresponding adjacent node of the critical point. These *split neighbors* are then used to determine which segment is the closest for a free cell. This segmentation causes the construction of double the amount of segments compared the number of critical points. This causes two types of transitions between segments: transitions being caused by split points and those transitions caused from the reachability-based assignment. The latter arises due to the fact that every split point causes two segments. Hence, two neighboring split points cause two adjacent segments that do not have a split point in between but where the transition comes from the distance to the nearest split point neighbor. The transitions without a split point are those that will be used for merging segments as presented in the following. Assigning cells to segments based on the distance according to the reachability map instead of the e.g. the Euclidean distance reliably assigns all cells in a room to segments in the same room and not to closer segments in an adjacent room. This is of particular interest in regions where the Voronoi diagram cannot adequately model the topology of rooms. This primarily happens due to the fact that the robot's knowledge about rooms and environmental structures contained therein can be only partial or imperfect. The result of the so far described segmentation procedure is shown in Figure 5.16. As can be seen, larger parts of the environment have been cleanly segmented into distinct rooms.



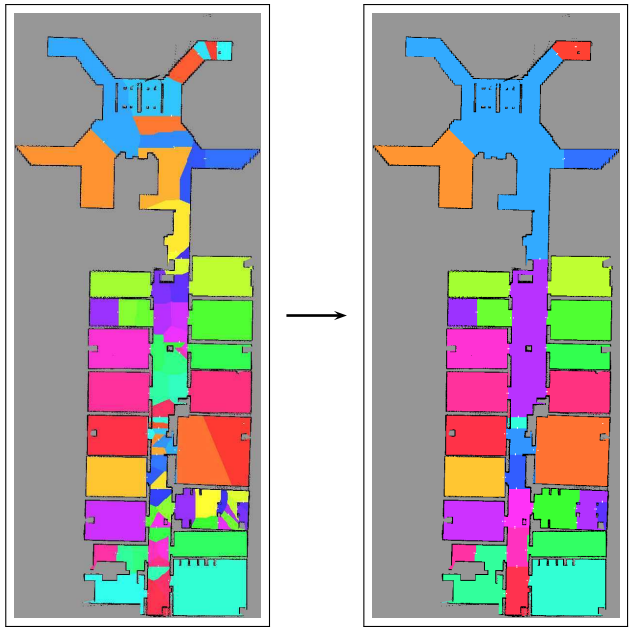
(a) Map (b) Critical points (c) Segmentation

Figure 5.16: Segmentation of a reflection map. Shown is a probabilistic reflection map modeling the 5-th floor of the AVZ building at the University of Osnabrück (a). The constructed Voronoi diagram and the critical points are shown in (b). The resulting segmentation of free space into differently colored segments is shown in (c).

Furthermore, the amount of small cells in cluttered regions is considerably small. How to remove a bigger part of these regions, is described in the following.

Segmentation Refinement by Merging Adjacent Segments

Several parts of the environment in Figure 5.16, like for instance, the corridor and rooms containing multiple pillars, consist of multiple smaller segments. However, by distinguishing segment transitions according to whether or not they are caused by a split point, adjacent segments can be easily merged as long as no split point transition is breached. By means of a single loop over the segmented cells, all transitions can be determined and stored in an adjacency list. This list is then repetitively scanned. Adjacent segments are merged if the transition between them is not caused by a split point. Furthermore, two segments whose transition is caused by a split point, can be merged if there is a third segment being adjacent to both with transitions not caused by split points. By this means, the according split point is removed.



(a) Segmentation (b) Refined segmentation

Figure 5.17: Refining the segmentation of reflection maps. Shown are the original segmentation (a) as well as the refined segmentation (b) resulting from merging segments.

In every scan through the adjacency list, segments are merged with neighboring segments. The procedure is repeated until no more segments can be merged since the remaining split points can not be removed because of a third segment being adjacent to the two segments being examined. The result of this simple merging step for refining the segmentation is shown in Figure 5.17. In many cases segments could be merged so that the resulting segments better reflect the structure of the environment. Besides a smaller number of rooms, the majority of regions in the environment is reliably segmented. In this context it should also be noted that that the coloring of the segments does not follow any reasonable coloring scheme but is pseudo-random by assigning values for hue in the HSV color space. Furthermore, the segmentation has been applied to all free cells. In the actual implementation, only traversable cells that can be frontier cells are considered in the segmentation.

The Overall Segmentation Algorithm

The overall segmentation algorithm can be formulated as follows:

1. Construct the Voronoi diagram, e.g. by means of Fortune's Sweep Line Algorithm (Fortune, 1987, 1998) on extracted occupied cells or by means of the distance transform (Wurm et al., 2008):
$$G = (V, E)$$
2. Extract the set of critical points in the Voronoi diagram
$$C = \{v_c \mid v_c \in V\}$$
where v_c is a local minimum, has degree 2 and is adjacent to a junction node of degree 3 or a node of another degree that is adjacent to a junction node.
3. Create for the two neighboring nodes v_{n_1} and v_{n_2} to segments s_1 and s_2 and add them to the list of segments S.
4. Determine for all free cells c in the map the closest segment $s \in S$. Assign s to c.
5. Determine all transitions (s_1, s_2) in the segmented map and add them the adjacency list A.
6. Merge two segments s_1 and s_2 with $(s_1, s_2) \in A$ if s_1 and s_2 do not originate from the same split point or terminate if no such (s_1, s_2) exists. Repeat step 6.

Important in the above procedure is that split points need to be stored for every segment. When merging segments, the resulting segment has to store the split points of both segments. When a third segment is merged to the resulting segment but its split point is already contained in the list of split points of the other segment, the split point is removed. This happens, for example, in the upper part of the map in Figure 5.17 where multiple segments are merged into one large segment. Merging the segments is done in multiple steps during which several split points are removed since separated segments are adjacent to one and same segment not being caused by a common split point.

Another approach for causing the robot to completely explore specific regions before visiting other regions has been proposed by Stachniss and Burgard (2003a), described in detail in (Stachniss, 2006). The restrict their exploration strategy to a local window covering the region that needs to be completely explored first. The strategy can, thereby, only take those possible exploration targets into account that lie in the local window in the respective region in the environment. Stachniss and Burgard (2003a) initially needed this extension to overcome the problem that their exploration strategy took not into account the distance that need to be travelled by the robot. The local window, however, provided an upper bound for the travel distance. The specific situation that the robot does not completely explore a room and needs to return later can not avoided with this extension, since environmental structures are not segmented and the local window might not fully cover individual rooms. In simulated experiments, carried out in the context of this thesis, it turned out that applying local windows normally decreases the performance of closest-frontier exploration, if it affects the explorative behavior it all. In only a few cases, rooms causing the aforementioned problems fell, by chance, completely into a local window leading to a shorter path than that of exploring closest frontiers.

Application to Multi-Robot Exploration

Segmenting the so far built environment model is of particular interest in the context of multi-robot exploration. Here, the central question is how to assign robots to different parts of the environment. One of the first approaches to coordinate a team of multiple cooperating robots based on exploring closest frontier has been presented by Yamauchi (1998). Here the robots communicate sensor readings so that they are able to use information acquired by other robots. However, each robot constructs an individual map of the environment and a central mapping component merges maps constructed by individual robots. Decisions where to move are, however, not made in cooperation but by every robot on its own. It can thus happen in this implicit form of coordination, that

multiple robots explore the same region in the environment. A segmentation of a common global environment map does not only allow for distinguishing regions in the environment but also to segment determined frontier cells into clusters belonging to the same region. Information like this can be used in an explicit coordination where a central planner decides where the robots have to move in order to optimize the performance of the overall team.

Wurm et al. (2008) use almost the same segmentation technique and then assign multiple robots so that every robot explores a different segment. For this purpose they further constraint critical points so that they lead from an explored region into an unexplored region. The frontiers behind these split points then form possible targets for exploration. The actual assignment of robots to targets is then based on the work of Ko et al. (2003). Ko et al. use the *Hungarian method* (Kuhn, 1955) for this assignment problem. The Hungarian method encodes, formulated in the context of exploration, the assignment of robots to targets in form of a matrix. Possible targets make up the columns of the matrix whereas the robots form the rows. The individual cells then represent the cost for a robot to move to a target. Wurm et al. (2008) assign costs based on the path length comparable to the cost assignment by means of reachability maps. They further apply a discount if a robot is already located in a segment being one of the targets. The Hungarian method then tries to find the least-cost assignment of robots to targets. The runtime complexity of the original algorithm is $O(n^4)$ but was later improved to be $O(n^3)$ (cf. Schrijver, 2003, Ch. 17). As it assumes a $n \times n$ matrix, the cost matrix needs to be augmented with dummy targets or dummy robots. If there are less robots than targets, dummy robots cause that some targets are not explored in this step. Less targets than robots, on the other hand, are handled by duplicating targets. The respective segments are then explored by one or more robots.

An assignment like the one just described is carried out by a central planning component. That is, in addition to the team of robots, there is an additional machine like the central mapping component of Yamauchi (1998) that receives information from all robots, determines targets for exploration and assigns targets to robots. Fundamental problems in this kind of coordination are, for example, how to handle the loss of a team member, failures in communication or the addition of a robot to the team. Another type of coordination approaches that distributes knowledge and computational load between the members of a team are based on market-economy and trading (Dias and Stentz, 2000). The idea of this coordination is that individual robots offer jobs to the team and other robots bid for these jobs. Consider for example, that of team of robots has to enter different rooms. A larger robot, standing in front of small doorway, might not be able to enter the room because of its size, so it offers the job to the team. Another robot that can accomplish this task with minimum cost (selected in the bidding procedure) enters the room and the overall task of the team is accomplished. Zlot et al. (2002) apply this coordination scheme to robotic exploration. The idea is that individual robots determine possible targets in their vicinity e.g. by means of determining closest frontiers or conducting any other exploration strategy. The possible targets are then offered to all other team members with which the robot can currently communicate. In an auction, the other robots can bid for exploring a particular exploration target. The robot with the lowest involved costs (including the robot that started the auction), explores the target. The emergent team strategy is thus robust against individual failures, the loss or addition of team members and interruptions as well as failures in communication. Xiong et al. (2007) follow a similar approach and segment the so far built environment models by means of quadtrees partitioning unknown cells. The cells of the quadtree leaves are then used as possible exploration targets and traded among the team members. Simmons et al. (2000) also follow a market-economy approach but do not segment the environment. Instead they prefer frontiers in the visibility regions of the individual robots.

Gossage et al. (2006) combine a local frontier-based exploration strategy with a global graph-based exploration strategy. The graph is used to minimize the number of multiple traversals of one and the same room and to assign regions to individual robots of a cooperating team. Such a graph-structure can be obtained by applying the Voronoi-based segmentation as initially proposed by Thrun (1998).

For a brief survey on exploration with a team of multiple mobile robots as well as evaluations of different coordination techniques it is referred to (Burgard et al., 2005) and (Stachniss, 2009,

Ch. 4). In the context of this thesis the segmentation is, however, solely used for preferring frontier cells in the currently explored regions and multi-robot exploration is not further considered. Efficiently using all algorithms presented throughout this thesis in a team of multiple homogeneous or heterogeneous robots is a matter of future work.

Integrating Map Segmentation into Closest-Frontier Exploration

The aforementioned segmentation algorithm as well as the information gained by its application can be easily integrated into the frontier-based exploration strategy. Once the robot has determined all frontier cells, it performs a segmentation of the so far built reflection map. The segmented map is represented in terms of a grid storing an integer-value for each cell. A value larger than zero thereby corresponds to the number of the segment to which a cell belongs. Obstacles and environmental structures, i.e. cells with a high reflection probability, are assigned -1 whereas unexplored terrain, i.e. cells with the prior reflection probability of 0.5, are assigned -2. Transitions between regions with positive values and regions with value -2 indicate which segments are not fully explored yet. Cells that could not be assigned to a segment have a value of 0. This only happens when, for any reason, no segment is reachable from that cell. Furthermore, this weakness can be neglected since determined frontier cells are always reachable from the robot's current position and are, hence, assigned to some segment.

Given the set of frontier cells and the segmented map, the region in which a frontier cell lies can be looked up in constant time in the segmented map so as the current position of the robot. The idea is now to prefer frontier cells in the same segment and to choose the closest frontier upon the cells in the same segment. The exploration strategy and the selection mechanism for determining the frontier to approach, respectively, are changed accordingly. If there are frontier cells in the same segment as the robot, only these cells are considered and the exploration strategy chooses the closest among them. Only if the segment in which the robot is located, does not contain any frontier cells and is fully explored, all frontier cells are considered as possible targets. That is, the explorative behavior of the robot is only changed, compared to classic closest-frontier exploration, in those situations where the currently explored segment is not yet fully covered.

An according exploration strategy can be formulated as follows:

1. Integrate latest sensor measurement and perform map updates
2. Determine the set of frontier cells F by checking for every cell in the candidate set C if it is adjacent to a cell with unknown reflection probability (Chapter 5.3.2):

   $$F = \{c^{[xy]} \mid c^{[xy]} \in C, \exists c^{[(x+m)(y+n)]} : P(c^{[(x+m)(y+n)]}) = 0.5, m \in [-1,1], n \in [-1,1]\}$$

3. Compute map segmentation to obtain the set of segments S
4. Look up for each frontier cell $c^{[xy]} \in F$ the corresponding segment s_i
5. Determine the segment s_r containing the robot's pose $\mathcal{P}$
6. Build the subset F_{s_r} or frontiers being contained in the same segment as $\mathcal{P}$
7. If $F_{s_r} \neq \emptyset$, select the closest $c^{[xy]} \in F_{s_r}$ and the closest $c^{[xy]} \in F$ otherwise.

Results and Open Problems

The above exploration strategy using segmentation of the map for determining the next action has been excessively tested in different simulated environments containing a larger number of rooms. A typical result is shown in Figure 5.18. Here the robot explored more than 30 rooms and a longer corridor. The path travelled by the robot in this experiment is 253.40 m. The robot entered all rooms and the constructed map is complete and fully covers all environmental structures. What can also be seen is that several rooms at the top of the map have been explored solely by passing by as the doorway was large enough to perceive all environmental structures behind. This is

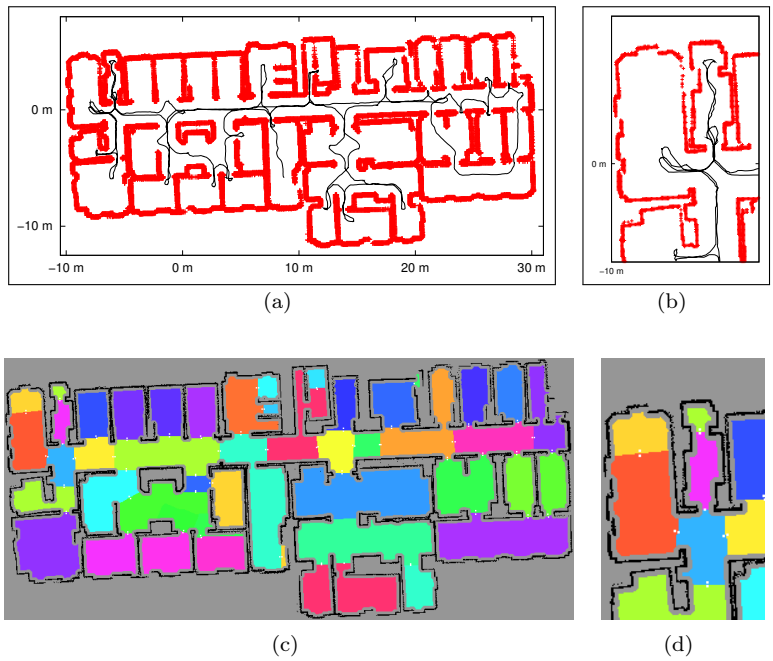


Figure 5.18: Closest frontier exploration using map segmentation. Shown is the trajectory of the robot exploring a larger indoor environment (a). By using the segmentation algorithm and preferring frontiers in the currently explored segment, only two rooms are traversed twice (b). These two are caused by false positives in the determination of the critical points for constructing the segmented map (c). The wrong segmentation of the two rooms is shown in detail in (d).

characteristic for the exploration using map segmentation as the robot is forced to first explore the segment in the corridor. In experiments without map segmentation, situations like these did occur considerably less often.

Unlike the simulated experiments in Chapter 3 and Chapter 4, here the robot performs ideal movements without disturbances, errors or noise in pose shift estimates and conducted movements. Furthermore, simulated range scans did not contain any noise or erroneous measurements. That is, starting at the same pose and using the same strategy deterministically results in the same trajectory and constructed map. This has been done for evaluating the strengths and weaknesses of different exploration strategies without external disturbances. Several experiments have been conducted where the robot was exploring the simulated environment, with and without map segmentation, and starting from different initial poses as these primarily determine the outcome of the experiment and the path travelled by the robot. In most of the experiments, the exploration strategy making use of segmenting the so far built model resulted in shorter trajectories. In some experiments, the segmentation has even been ideal and all rooms were cleanly separated from each other. In these experiments the shortest paths that could be measured were lying around approximately 200 m. In most of the experiments, however, the segmentation was not ideal and a smaller number of rooms has not been segmented correctly as in the case of the experiment visualized in Figure 5.18. In the shown maps, more or less five rooms could have been segmented better. Only two of the rooms had to be entered two times in order to fully explore them. Without map segmentation, especially in the depicted environment, the robot had to re-enter considerably more rooms for a full coverage. However, it should also be mentioned that in a few experiments, the trajectory obtained without map segmentation was shorter than that of using map segmentation. The primary reason for that is that the segmentation itself is not reliable and perfect in many cases and that a robot exploring without map segmentation takes, by chance, a shorter path than a robot exploring segment-wise.

Not only that the used segmentation approach does not always yield perfect segmentations, but

it is also quite inefficient in the current implementation. As described earlier, the algorithm itself is rather naive and computationally complex but also the implementation can be clearly improved. In the context of this thesis, map segmentation has been added to closest-frontier exploration merely because of theoretical interest and to see whether or not it can improve the overall performance. It is a matter of future work to improve the algorithm itself as well as to make the implementation more efficient. As described in Chapter 5.3.2, the robot is stopped when deciding which frontier should be approached next. The strategy for exploring closest frontiers without map segmentation is quite efficient and the robot started moving to the selected frontier immediately. Segmenting a map as the one shown in Figure 5.18 takes up to one to two seconds in the current implementation. That is, the robot really stops and stands still for this amount of time.

5.3.4 Sampling and Decision-Theoretic Exploration

In frontier-based exploration the robot moves from one unknown region in the so far built environment model to the next one. When exploring closest frontiers, the robot chooses to approach the nearest unexplored region regardless of its size. Consider for example that there are two frontiers in the vicinity of the robot, one in a distance of 1 m and the other in a distance of 2 m. Even when the unexplored region corresponding to the latter frontier is larger than that of the first one, the robot will select the first frontier. Another issue not being considered in frontier-based exploration strategies is that there might be a vehicle pose from which both unexplored regions are visible. Regarding the last issue, frontier-based exploration is rather reactive as the robot is attracted by and moves to the closest unexplored region regardless of the expected outcome of this action. However, even when taking the expected outcome into account, exploration strategies are normally greedy and their look-ahead distance is limited to one sensing action (Thrun et al., 2005, Ch. 17.2).

The idea of exploration strategies being based on utility or decision theory, is that a certain number of candidate configurations is drawn from the free space in the robot's internal environment model. The exploration strategy then determines the utility of each candidate for the robot and selects the candidate with the largest utility. This simple principal allows for expressing complex exploration strategies e.g. by integrating multiple factors or *attributes* in the function for evaluating a candidate's utility. In this context, the exploration of closest frontiers can be described as taking the set of determined frontier cells as the set of candidate poses and assigning zero utility for all candidates except for the one being closest to the robot. A common idea in decision-theoretic exploration strategies is to explicitly take into account uncertainties, e.g. in the robot's pose, observations and the so far built model, when selecting an action (Ko et al., 2003). The difference between individual decision-theoretic exporation strategies is the definition of the utility of candidates and especially which attributes are used in the utility function. A common definition is expressing exploration in terms of maximizing some information gain while while minizing some cost (Stachniss and Burgard, 2003a).

$$\text{Utility} = \text{information gain} - \text{cost} \tag{5.5}$$

The utility function can also be regarded as a reward function being maximized when the robot chooses the desired action. This particular utility function has two attributes and is maximized when the robot moves to a pose in its vicinity that is expected to provide the largest information gain. Moorehead et al. (2001) refer to the different attributes as *multiple sources of information*. Besides the choice of attributes, the way in which these attributes are integrated into the utility function influences the explorative behavior of the robot. Whereas Eq. (5.5) is a summation of attributes, utility functions often have a multiplicative form allowing to completely cancel out other attributes and yield low utility when the outcome for a single attribute is undesirable (Tovar et al., 2006).

Drawing Samples from Grid Maps

The first step in decision-theoretic exploration strategies is to generate the set of candidate vehicle poses. There are, in principal, two possibilities for generating candidates. The first possibility is

to draw samples from some random distribution preferably well distributed over the so far built environment model. The second possibility is to generate samples based on predefined classes of actions, e.g. actions for exploring unknown terrain near frontiers and actions for visiting already modeled parts of the environment to close loops. Here we will consider drawing all samples from the same distribution. As long as the number of samples is large enough and all samples are well distributed over the map, all possible actions should be represented in one or more samples.

Let us first consider the generation of candidate positions regardless of the orientation of the robot. A probabilistic reflection map represents for every so far visited region in the environment the probability of reflecting a range beam and, thus, whether it is occupied by an object or not. As described in Chapter 4.4 we want to restrict the movements of the robot so that it does not travel through unknown regions or close to obstacles. Hence, a straightforward way is to draw samples directly from traversability of reachability maps and from a two-dimensional uniform distribution covering the size of the maps respectively. Consider a grid map is organized as a matrix and has a size of $n_x \times n_y$ cells. Selecting a random cell $c^{[xy]}$ from this matrix is decomposed into drawing integers x and y from the intervals $[0, n_x]$ and $[0, n_y]$ respectively. This gives an initial set of candidates C_0:

$$C_0 = \{c^{[xy]} \mid x \in [0, n_x], y \in [0, n_y]\}. \tag{5.6}$$

To actually sample from a traversability map T or reachability map R, we simply need to remove those cells from the initial sampling set that are not traversable and reachable respectively. Every removed sample needs, of course, to be replaced by a new valid sample in order to always have a constant number of samples. From this simple procedure two sample sets can be derived C_T being the set of traversable candidate positions and C_R the set of reachable candidate positions:

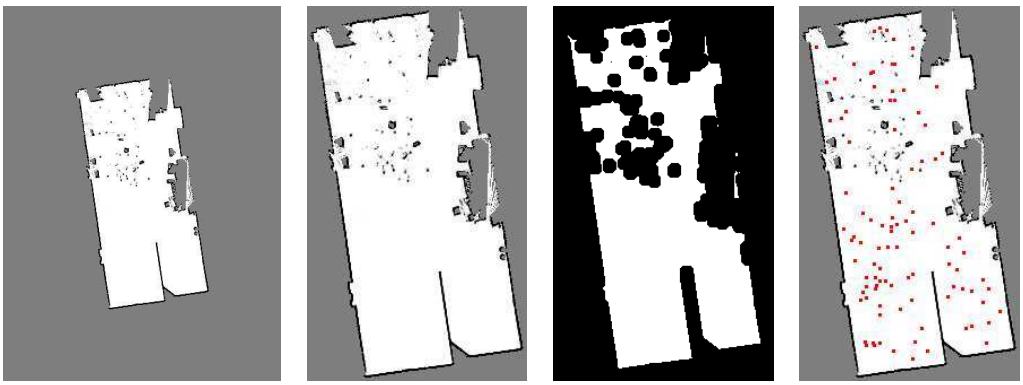
$$C_T = \{c^{[xy]} \mid x \in [0, n_x], y \in [0, n_y], t^{[xy]}\} \quad \text{and} \tag{5.7}$$

$$C_R = \{c^{[xy]} \mid x \in [0, n_x], y \in [0, n_y], r^{[xy]}_{\text{cost}} \neq \infty\}. \tag{5.8}$$

where $t^{[xy]}$ is only true if $c^{[xy]}$ is traversable in T. Accordingly, $\left(r^{[xy]}_{\text{cost}} \neq \infty\right)$ represents that there is a finite length path to $c^{[xy]}$ in R. The drawback of sampling from the traversability map is that not all candidates might also be reachable. Furthermore, the reachability map needs to be computed anyway as it will be used to evaluate the costs involved when moving to a candidate.

A general issue in this sampling procedure is that its runtime is affected by the number of rejected samples. Consider for example the reflection map in Figure 5.19.a. Large portions of the map are unknown. Samples drawn from these unknown regions need to be neglected as there are neither traversable nor reachable. During SLAM and the integration of new information into the map, the grid map is dynamically re-sized. That is, the grid map will never be smaller than the size of modeled environmental structures. However, depending on the map's initial size, it might be larger. In the dynamic reallocation the map is only enlarged, but never downsized, by simply extending it into a particular direction if the robot moves outside of the modeled region or when it perceives information about environmental structures lying outside. To avoid that too many samples need to be rejected and in order to reduce the runtime of the sampling procedure, we first crop the map so that it's size is limited to the area spanned by free space and environmental structures. That is, the size of the map is reduced to correspond to the axis-aligned bounding box around cells with a reflection probability of $p(c^{[xy]}) \neq 0.5$. By this means the amount of unknown cells is minized and, hence, the number of samples that need to be rejected. A cropped copy of the map in Figure 5.19.a is shown in Figure 5.19.b. The traversability map computed for the cropped copy is shown in Figure 5.19.c. Using the aforementioned procedure 100 candidate positions have been generated that are more or less uniformly distributed over the free space in the cropped reflection map (see Figure 5.19.d). Note that the same cropping procedure is applied before determining frontier cells in the frontier-based exploration strategies in Chapter 5.3.2.

Up to now, the candidates are positions in either traversable or reachable space in the so far modeled environment. Orientations of the robot at these candidate positions are, however, not considered. If the robot's field of view is limited and does not provide a full panoramic view, i.e. the robot is not a 2π-observer, the orientation has to be taken into account. As we already have valid candidate positions, it is sufficient to draw orientations from a uniform distribution



(a) Reflection map (b) Cropped map (c) Traversability map (d) Drawn samples

Figure 5.19: Cropping maps and drawing samples. Shown here is a probabilistic reflection map (a) together with a cropped copy (b). Extracting traversable cells from the cropped copy and drawing from the resulting traversability map (c) results in the samples (d).

over the interval $[-\pi, \pi)$. The resulting candidate poses are visualized in Figure 5.20.c. Here the coordinates x and y are also transformed from discrete matrix indices into the continuous range of values corresponding to the world coordinate frame $\{W\}$.

The overall procedure for generating n candidate poses in the reachable workspace, i.e. drawn from the reachability map R, can be formulated as follows. The set of sampled candidate positions (in discrete map indices) is C_R. The set of actually generated candidate poses in the form of poses $\mathcal{P}$ is P.

$C_R = \emptyset$, $P = \emptyset$
while $|C_R| \neq n$

1. Draw sample (x, y) such that $x \in [0, n_x]$ and $y \in [0, n_y]$.
2. Determine reachability $r_{\text{cost}}^{[xy]}$ of cell $c^{[xy]}$.
   If there is no valid path, i.e. $r_{\text{cost}}^{[xy]} = \infty$, reject $c^{[xy]}$ and go to step 1.
3. Add $c^{[xy]}$ to the set of candidates C_R

$$C_R = \{c^{[xy]} \mid x \in [0, n_x], y \in [0, n_y], r_{\text{cost}}^{[xy]} \neq \infty\}$$

4. Determine continuous coordinates $\mathcal{P}^x$ and $\mathcal{P}^y$ in $\{W\}$
   (according to map size and resolution)
5. Sample $\mathcal{P}^\theta$ from a uniform distribution over $[-\pi, \pi)$
6. Add pose $\mathcal{P} = \begin{pmatrix}\mathcal{P}^x & \mathcal{P}^y & \mathcal{P}^\theta\end{pmatrix}^T$ to the set of candidate poses P.

$$P = \{\mathcal{P} \mid \begin{pmatrix}\mathcal{P}^x & \mathcal{P}^y\end{pmatrix}^T \text{ is reachable}, \mathcal{P}^\theta \in [-\pi, \pi)\}$$

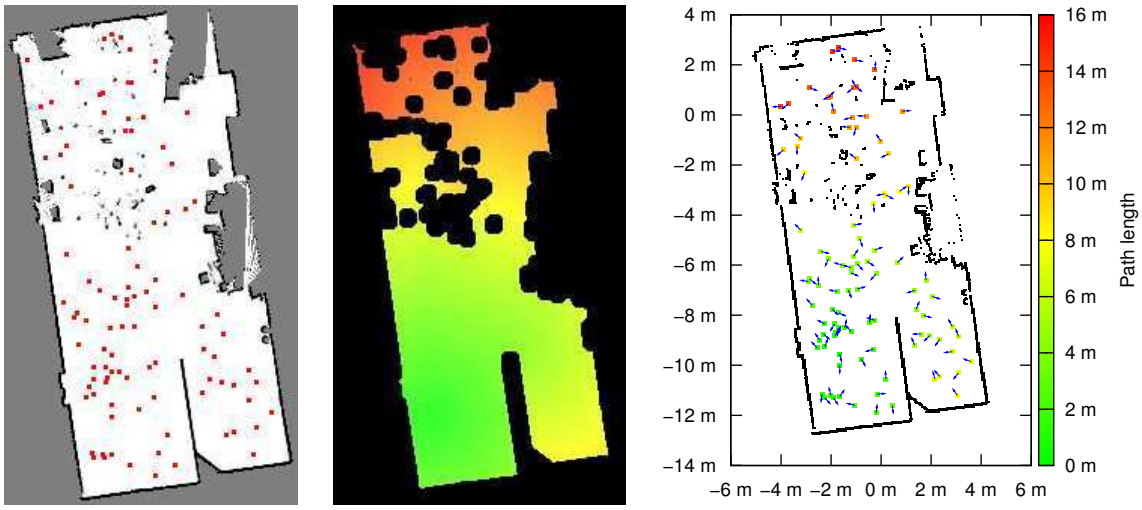
end

An important issue in this sampling procedure is choosing the number of candidates being appropriate for a reasonable explorative behavior. Evaluating a larger number of samples in a small environment is, for example, too expensive. In larger environments, the number of candidates need to be larger to provide an appropriate distribution. Generating, for example, only 10 candidates for an environment with a size of $100\,\text{m} \times 100\,\text{m}$ will necessarily neglect larger regions of the workspace. If the number of candidates does not need to be constant, e.g. for having an approximately constant runtime, it is suggestive to make the number of candidates dependend on the size of the so far

built model. Here, this is implemented as an optional extension and parametrized by specifying a desired candidate density, e.g. one candidate per square meter.

Determining Expected Costs

Once the candidate set has been generated, the second step of decision-theoretic exploration strategies is to evaluate the utility of each candidate for the robot and the currently performed task respectively. As exploration aims at constructing a complete environment in a reasonable amount of time or by traveling a preferably short path, cost is an important attribute in the utility function used for evaluation. An intuitive way of representing cost is to estimate the time taken by the robot to approach a candidate plus the time for acquiring sensory information and updating the internal environment representation. A simpler and commonly used way is to express cost in terms of the shortest path to a candidate, i.e. the path length encoded in the reachability map. As already mentioned, candidate poses are preferably constraint to be reachable poses and, hence, the reachability map is inherently constructed in the above sampling procedure. The length of the shortest path to each candidate can thus be directly looked up in the reachability map without additionally planning paths just like for determining the closest frontier in Chapter 5.3.2. The resulting cost in terms of path lengths for the set of candidate poses generated above, is visualized in Figure 5.20. As a side not it is to remark that the path length is not directly used in the utility function but instead a cost function depending on the path length.



(a) Drawn samples (b) Reachability map (c) Candidates

Figure 5.20: Generated candidate poses and cost. Shown are generated candidate positions in a cropped reflection map (a) as well as the corresponding reachability map (b). The candidate poses with associated costs in the form of path length are shown in (c). Positions are visualized using filled rectangles. The color of the rectangles encodes the length of the shortest path to a position. The orientation of the poses are visualized using small blue arrows.

Determing Information Gain in Grid Maps

Whereas measuring the cost is normally straightforward by taking the length of the path that needs to be travelled by the robot, measuring the information gain is more complex. A common way of expressing information gain in probabilistic grid maps is the reduction of the robot's uncertainty about the so far built model. A general way to measure the uncertainty of a posterior is the entropy $H_p(x)$ (Thrun et al., 2005, Ch. 17.2):

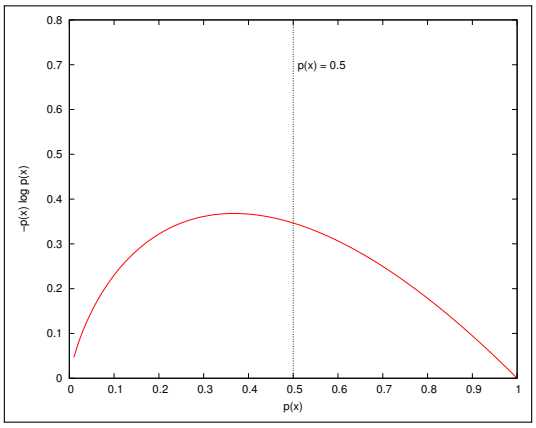
$$H_p(x) = -\int p(x)\log p(x)\mathrm{d}x \quad \text{or} \quad -\sum_x p(x)\log p(x). \tag{5.9}$$

The term $-p(x)\log p(x)$ is minimal if we are certain about x, i.e. $p(x) = 0$ or $p(x) = 1$, and maximal if we are uncertain about the state of x (see Figure 5.21.a). That is, the bigger the value of $H_p(x)$ is, the more uncertain is the robot about the state of the world. When $H_p(x)$ approaches zero, the robot is highly certain about the constructed map and the state of the cells contained therein.

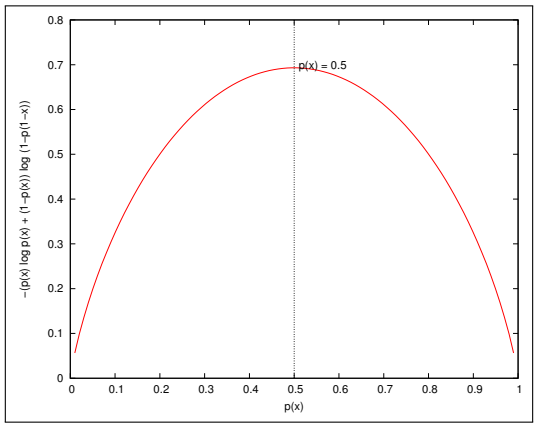
Both occupancy and reflection grid maps, in principal, represent two probabilities: the probability of being occupied $p(occupied)$ and the probability of being free $p(free)$. To account for this, the entropy measure (for the discrete case) is split into:

$$H_p(x) = -\sum_{c^{[xy]}} \left[\underbrace{p\left(c^{[xy]}\right) \log p\left(c^{[xy]}\right)}_{\widehat{=} H_p(occupied)} + \underbrace{\left(1 - p\left(c^{[xy]}\right)\right) \log\left(1 - p\left(c^{[xy]}\right)\right)}_{\widehat{=} H_p(free)} \right] \tag{5.10}$$

Regarding a probability of $p\left(c^{[xy]}\right) = 0.5$, this function is symmetric (see Figure 5.21.b) and, again, maximal if the robot is uncertain about whether cell $c^{[xy]}$ is occupied or not. Of course, the same holds true for probabilistic reflection maps. For both functions it should be noted that $\log(0)$ is not defined. The multiplication with 0 will, however, results in 0. In the implementation this is handled by explicitly setting an entropy of 0 for cells whose reflection probability is, for any reason, exactly 0 or 1. The result of calculating the entropy for every cell in a probabilistic reflection map is shown in Figure 5.22.a.



(a) $-p(x)\log p(x)$



(b) $-(p(x)\log p(x) + (1 - p(x))\log(1 - p(x)))$

Figure 5.21: Entropy and cell posteriors. Shown are a plot of the entropy $H_p(x)$ (a) and the entropy for a cell posterior in a probabilistic reflection map (b). The latter is maximal for unknown cells.

Now the idea is to express the information gain $I(t+1, t)$, expected after choosing an action at time step t, as the reduction in uncertainty

$$I(t+1, t) = H_p(x_t) - H_p(x_{t+1}) \tag{5.11}$$

where, in the context of the exploration problem, $H_p(x_{t+1})$ is the entropy after integrating sensory information acquired at pose $\mathcal{P}_{t+1}$ into the map. Hence, we can derive a simple procedure for estimating the expected information gain. For every candidate we simulate a range scan at the respective pose, integrate the simulated range into the so far built map and calculate the entropy for the resulting map. This entropy will be smaller than the entropy of the map before integrating the range scan if any new information can be acquired and equal if not. Hence, the information gain is either positive or zero. Note that, depending on the integrated sensory information and the procedure for simulating sensory information, the entropy of the updated could also be larger causing a negative information gain. This is not the case for the procedure presented in the following.

A range scan can be simply simulated by performing ray casting at the candidate poses. If the ray hits a free cell it is not reflected. When hitting an occupied cell or a cell with a high reflection probability, the beam is reflected causing the simulated range measurement in the corresponding direction. However, the question remains how to handle situations in which the ray extends into unknown regions. For estimating the average length of measurements extending into unknown regions, one can apply simple statistics. Here, exploring closest frontiers has been used in different environments resulting in large sequences of partial maps as well as complete maps of the respective environments. Laser scans where then simulated in the partial maps at the poses approached by the robot. As soon as a ray intersects frontiers to unknown terrain, the true distance has been determined by means of the complete environment map. The difference between the simulated measurement (true distance to obstacle) and the distance to the frontier provides the true length of the beam in the unknown region and to which extend the beam continued over unknown cells. These differences have been collected for all beams intersecting unknown regions at all poses approached be the robot in ten explorative runs each carried out in a different environment. The ten simulated environments differed in overall size, the number of rooms and the size of rooms, ranging from the small RoboCup@Home arena at GermanOpen 2008 to large-scale environments like the Player/Stage Hospital world and the AVZ building at the University of Osnabrück. For all differences the mean value has been calculated resulting in an average extend over unknown regions of approximately 4.5 m. Note that the determined standard deviation for this value is quite large as measured differences range over minimal measurements, e.g. 5 cm to maximum range measurements when larger rooms are hidden behind the frontier. When simulating a laser scan, a range beam is cut and an artificial measurement is added to the scan, when a beam is extending over unknown terrain for a distance larger than 4.5 m. Applying this procedure yields more realistic estimates of the expected information gain than simulating laser range scans with max. range readings in the vicinity of frontiers to unknown regions. The overall procedure for estimating the information gain can be summarized as follows:

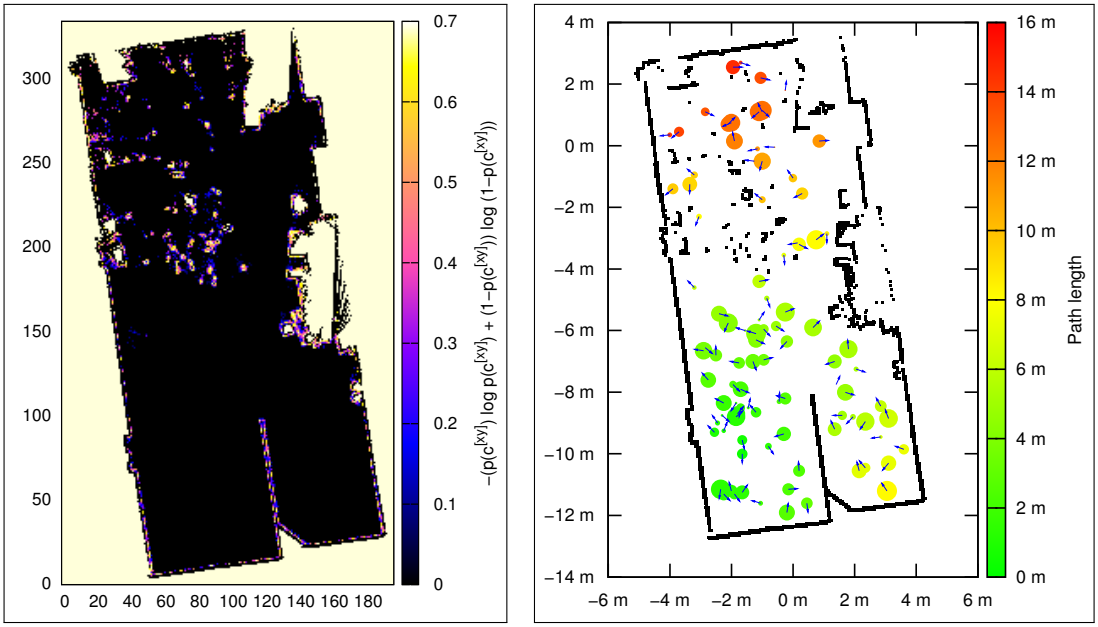
Compute entropy $H_p(m_t)$ for the current reflection map m_t according to Eq. (5.10)
for all $\mathcal{P} \in P$

1. Simulate range scan at $\mathcal{P}$
2. Update a local copy of the reflection map m_t using the simulated scan yielding the updated map m_{t+1} (see Chapter 3.2.2).
3. Estimate entropy $H_p(m_{t+1})$ for the updated copy m_{t+1}.
4. Compute information gain: $I(t+1,t) = H_p(m_t) - H_p(m_{t+1})$.

end

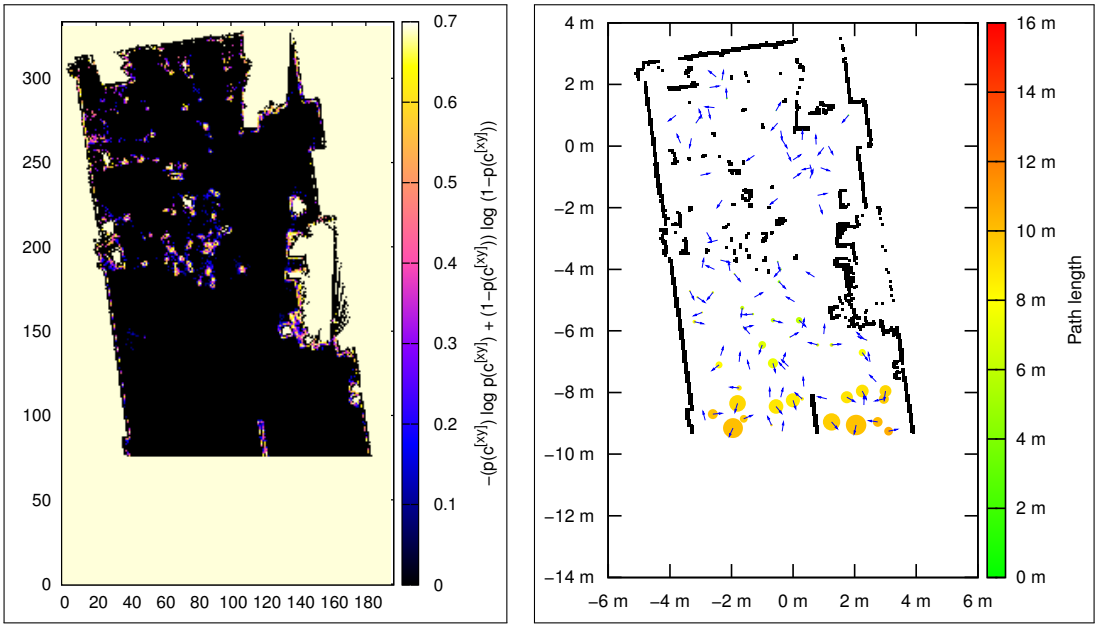
For the integration of the expected information gain into the utility function, the resulting gains are normalized so that the largest expected information gain is scaled to 1 and the other gains are given relative to the largest and in the interval $[0, 1]$. The expected information gain as well as the determined path lengths for a set of candidate poses generated in the laboratory environment is visualized in Figure 5.22.b. Here, the environment is already fully explored, and the individually expected information gains are small.

In Figure 5.23 the constructed map is cut and the bottom part is replaced by unknown cells. This inserts an artificial frontier causing large expected information gains for those candidates that have a direct line of sight to that border. Candidate poses in the already modeled part of the environment have, compared to that, a small relative information gain. In Figure 5.22, the robot's current position is $(-2\,\text{m}, -10\,\text{m})$. The (color-coded) lengths of the shortest paths to the candidates are, hence, small (green) in the respective region, and large (red) for candidates at the opposite side of the environment. In Figure 5.23, the robot's current position is $(0, 0)$. Candidates in the vicinity of the artificial border lie in the range of 8 m to 11 m away from the robot (along the shortest path). These candidates should be preferred in the utility function as they have a considerably larger information gain than candidates in the rest of the environment.



(a) Map entropy (b) Utility of candidates

Figure 5.22: Map entropy and utility of candidates in a completely explored environment. Shown is the color-coded entropy for each cell in the reflection map (a) as well as the generated candidates (filled circles) with expectend information gain and path length. The color of the circles encode the path length whereas their size corresponds to the normalized information gain.



(a) Map entropy (b) Utility of candidates

Figure 5.23: Map entropy and utility of candidates in a partially explored environment. Shown is the color-coded entropy for each cell in the reflection map (a) as well as the generated candidates (filled circles) with expectend information gain and path length. The color of the circles encode the path length whereas their size corresponds to the normalized information gain. Candidates, for which the artificial frontier is visible, have the largest expected information gain.

A commonly found approximation of the expected information gain is simply counting the number of unknown cells visible in a simulated range scan (Thrun et al., 2005, Ch. 17.2). As already mentioned, the entropy of cells with very low or very high reflection probabilities is almost zero. Their influence to the overall map entropy is thus small. Hence, when integrating a laser range scan into an empty map, the uncertainty reduction is proportional to the number of cells being touched by the individual beams. This approximation can considerably reduce the runtime of the exploration strategy as it avoids creating and updating local copies as well as computing the entropy for the resulting maps. The actual implementation allows for using both, the original information gain formulation from Eq. (5.11) as well as the approximation by means of counting unknown cells visible in a simulated range scan.

It should, furthermore, be noted that even the original formulation is only a rough estimate. Not only that simulated laser range scans can only be roughly estimated by means of the aforementioned statistics, information acquired during the movement to a candidate is completely ignored. This can be neglected in approaches where sensory information is only acquired and integrated in the map when standing at a selected candidate pose. Nüchter et al. (2003a), for example, only integrate 3D laser scans taken at the determined next best views into the map. Sensory information acquired during movement is not integrated in the map. Here, the environmental structures are continuously sensed during the robot's motion and acquired sensory information is always integrated into the map. As a result, the information gain metric can change drastically after only small motions (Simmons et al., 2000). A better estimation would be to subsample the shortest paths to the candidates and to simulate multiple range scans along the way travelled by the robot. By this means information acquired during the motion to a candidate is better taken into account when estimating the expected information gain. Such a procedure will, however, considerably increase the runtime of the exploration strategy. Finding and integrating a way of better estimating the information gain when continuously acquiring information about the environment is matter of future work. Another issue is, that dynamically resizing the constructed map can drastically change the evolution of the map's entropy. Increasing the map size also inserts new unknown cells causing an increase of the entropy. The expected information gain is, however, not affected since it is measured on the same map before and after updating it with a simulated range scan.

Determining Information Gain in Continuous Metric Maps

Up to now, all exploration strategies inherently require that a probabilistic reflection map is constructed. In the case of continuous metric maps, like the sparse point maps described in Chapter 3.5, evaluating information gain is not as straightforward as in the case of probabilistic grid maps. Continuous metric maps do not distinguish between known free and unknown regions in the map since they only model the surface of environmental structures.

An exploration strategy being applicable on continuous metric maps has been proposed by González-Baños and Latombe (2001a). They assume that a polygonal environment representation is constructed and, furthermore, that this environment representation is sufficient to represent the real environment, what is normally the case for indoor environments. In addition, they assume that this indoor environment is structured what does not necessarily hold for domestic environments. A domestic environment is not only cluttered but also dynamic. However, by means of the obstacle and structure maps presented in Chapter 2.3.4, the robot is able to perceive rather structured and static information about its surrounding environment.

A fundamental problem is that sparse point maps are an unordered set of points instead of a polygonal representation. González-Baños and Latombe (2001b) compute so-called *safe regions* from the first range scan. These safe regions, bounded by a polygon, are extracted by means of representing raw range scans in the form of polylines. The raw scan points are clustered and a line is fitted to every cluster. The lines are then ordered according to the orientation to the robot's pose. By connecting subsequent line segments the bounding polygon of the safe region is determined. Those lines that need to be added to complete the polygon's boundaries yield the so-called *free lines* (see Figure 5.24) that are further examined in the approach of González-Baños and Latombe.

Nüchter et al. (2003a) follow a similar approach. They take 3D scans and extract a slice in

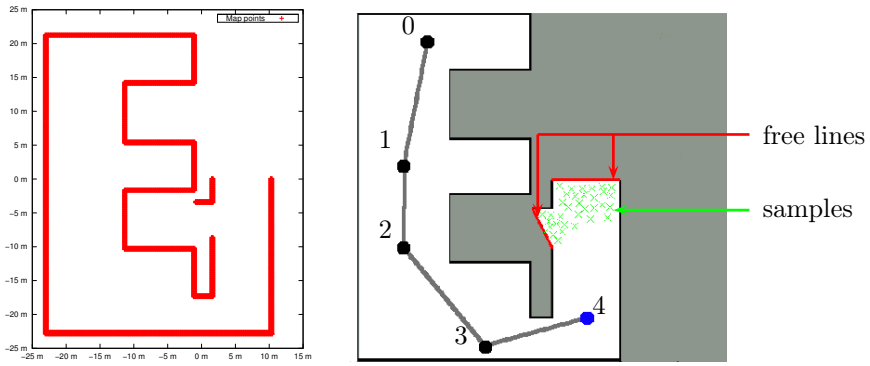


Figure 5.24: Safe regions and free lines. The visualization shows a safe region obtained from merging five local safe regions. Candidates (green crosses) are generated in the safe region and in the vicinity of free lines (red lines). The figure is adapted from (González-Baños and Latombe, 2001b).

a height of 1.2 m. Lines are then extracted from all points within the slices using a custom variant of the well-known Hough-transform. The extracted lines for each 3d scan are then ordered and connected to determine the free lines. The local polygons computed for every 3D scan are then merged by means of a polygon clipping algorithm (Nüchter et al., 2003a). The remaining exploration strategy is that of González-Baños and Latombe (2001b). Just like in the approach of Nüchter et al. (2003a), the polygonal representation can be generated by extracting lines from, respectively, sparse point maps and the laser range scans used to generate them.

González-Baños and Latombe (2001b) address planning of next best views and the art gallery problem, in general, by means of a randomized approximation algorithm (González-Baños and Latombe, 2001a). In order the determine the next best sensing location, i.e. the very location providing as much new information as possible and guaranteeing a sufficient overlap for the matching process while minimizing the according motion cost, candidates are uniformly distributed in those areas inside the polyon from that the *free lines* are visible. They proceed in the following four steps (cf. González-Baños and Latombe, 2002):

1. **Candidate Generation:** A fixed number of m candidate locations $\mathcal{P}_i$ is generated with $i = 1 \ldots m$ in the visibility range (determined by the maximal measurable distance of the sensor) of each free line (in the context of González-Baños and Latombe (2002) referred to as *free curves*).

2. **Overlap determination:** To guarantee a sufficient overlap of the expected sensor reading for each candidate, the summed length of the line segments resulting from intersection of the sensor's visibility polygon and the already modeled lines is computed as an estimation of the individual overlap. If this overlap is below a certain threshold, that very candidate is discarded and a new candidate is generated. This primarily happens when generating candidates near to static obstacles or when a larger part of the visibility polygon intersects free lines.

3. **Cost Evaluation:** The *cost* of acquiring information at a sensing location is defined as the length of the shortest obstacle-free path $L(\mathcal{P})$ to that candidate. If no such path exists the candidate is discarded and a new candidate is generated.

4. **Information Gain Estimation:** For each candidate the individual information gain $I(\mathcal{P})$ is estimated defined as the area outside of the known boundaries and visible through free lines from the position and orientation of the candidate.

If no free lines are found in the polygon, the exploration task is complete and the robot's workspace has been completely modeled. As a side note, it is to remark that, in contrast to

other approaches presented in this chapter, this particular exploration strategy has not been implemented and evaluated since the only difference to comparable approaches is the evaluation of the information gain. It is expected that the explorative behavior of a robot following the strategy of González-Baños and Latombe does not differ from the behavior when exploring candidates in the vicinity of frontiers in probabilistic reflection maps.

However, it should be mentioned that especially the randomized approximation algorithm from (González-Baños and Latombe, 2001a) forms the basis for a wide variety of exploration strategies (Tovar et al., 2006). Besides the consideration of multiple robots in the exploration strategy, Tovar et al. extend the original approach by not only taking the overlap of new measurements into account but the number and characteristics of features. The basic idea is to position the robot in a way that it perceives environmental structures with the new sensor reading that have already been modeled and that provide, in terms of matching, a characteristic in the environmental structure that supports the alignment of data. This especially applies to corners. In the original approach by González-Baños and Latombe, a long wall in a corridor is interpreted as a structure with enough overlap and the according candidates are taken into account. However, such a *featureless* range scan is normally hard to register since the problem of determining the robot's pose is underconstrained in these situations. In the approach of Tovar et al. candidates, where the simulated range scan does not provide a sufficient number of distinctive features in environmental structures are neglected.

Defining and Evaluating the Utility Function

Another important issue in the context of decision-theoretic exploration strategies is the definition of the utility function. The utility function exemplarily presented in Eq. (5.5) is a summation of attributes. In particular, some cost value is subtracted from some measure of information again. The drawback of utility functions summing over attributes is, in general, that a single attribute with low utility cannot cancel out other attributes with high utility. Furthermore, the attributes need to be scaled or normalized so that their range of values is comparable. The more attributes have to be integrated, the more complex gets the definition of an adequate utility function (Moorehead et al., 2001; Makarenko et al., 2002).

In the context of this thesis we only wish to integrate the expected information gain as well as the cost for the involved movement. A simple but promising approach has been presented by González-Baños and Latombe (2002). For determining the utility of a candidate $\mathcal{P}$, they evaluate a score function $g(\mathcal{P})$. This function is similar to a utility function and has a multiplicative form thereby weighting the individual information gain $I(\mathcal{P})$ of a candidate by the cost $L(\mathcal{P})$ involved in travelling there.

$$g(\mathcal{P}) = I(\mathcal{P})\ \mathrm{e}^{-\lambda L(\mathcal{P})} \tag{5.12}$$

The constant $\lambda > 0$ is used to weight the impact of the path length on the information gain. As described earlier we normalize expected information gains so that they are relative to the largest expected information gain in one update of the exploration strategy. The cost has, furthermore, been defined as being the length of the shortest path to a candidate. It is measured in meters. Experiments carried out in simulated environments suggested that $\lambda = 0.2$ is an adequate value for the weighting factor. A plot of the resulting utility function is shown in Figure 5.25.

Exploring a Simulated Environment

By means of the exponential decay in the cost term, the robot prefers candidates in its immediate vicinity. As soon as these candidates do no longer provide large information gains, the robots starts selecting candidates in other regions of the environment. As the largest expected information gain is often measured in the vicinity of frontiers, the resulting explorative behavior is quite similar to that of exploring closest frontiers. Figure 5.26 shows the trajectories of the robot exploring a simulated environment by means of the aforementioned procedures for generating samples in free space, determining cost based on the reachability map, estimating the expected information gain by simulating laser range scans and computing map entropy as well as for evaluating candidates based on the utility function in Eq. (5.25).

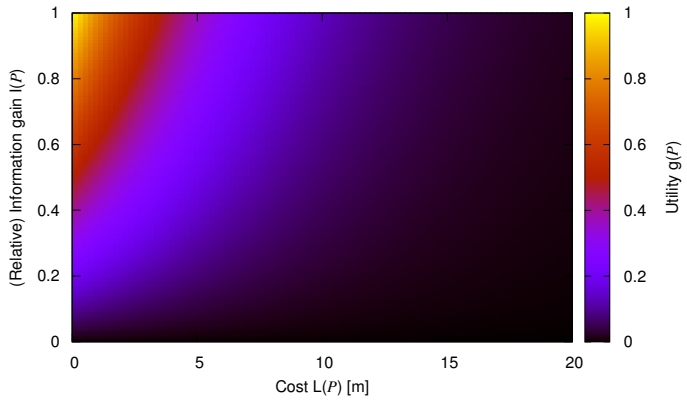
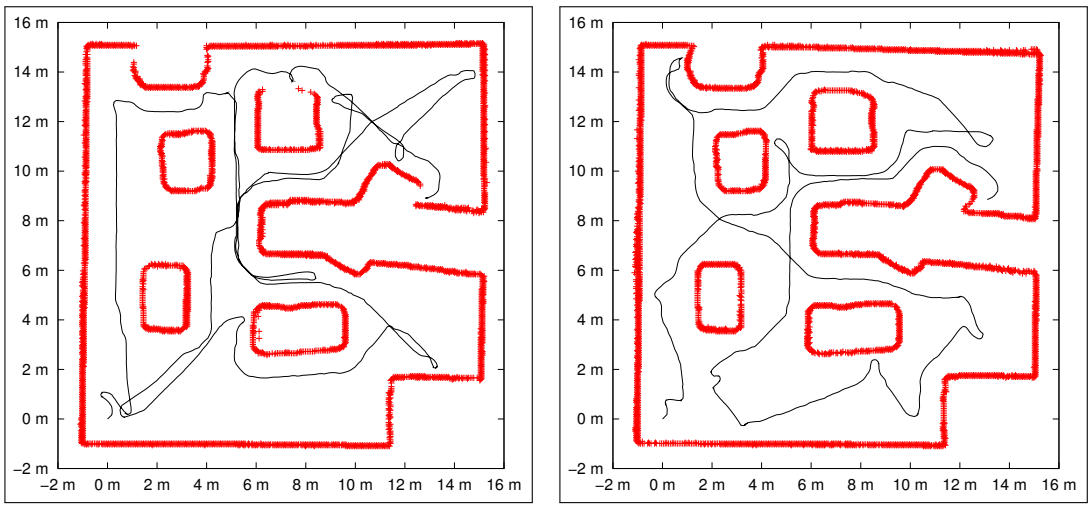


Figure 5.25: Utility function for candidate evaluation: $g(\mathcal{P}) = I(\mathcal{P})\ \mathrm{e}^{-\lambda L(\mathcal{P})}$ with $\lambda = 0.2$.

As expected the explorative behavior of the robot is comparable to that of exploring closest frontiers. However, if the number of candidates is low, it might happen that not a single candidate has a large expected information gain, e.g. when frontiers are, if at all, only partially visible. Using a constant number of 10 candidates results in a rather awkward trajectory. Several regions are traversed multiple times and the robot performs zigzag-movements between different regions of the environment (see Figure 5.26.a). The overall length of the robot's trajectory is 125.86 m. With a constant number of 100 candidates, the generated candidates are better distributed over the so far built model and the aforementioned problems do not occur. Except for central parts of the map, all regions are traversed only once (see Figure 5.26.b). The overall length of the resulting trajectory is 100.88 m. For comaprison, closest frontier exploration with repetitive re-checking resulted in a trajectory of 70.45 m. Since the utility function prefers candidates in the robot's vicinity, the trajectories for both $n = 10$ and $n = 100$ candidates show NBV-selections that are comparable to the characteristic behavior of closest frontier exploration in the sense that the robot first examines environmental structures behind its initial position. A characteristic difference lies in the deterministic nature of exploring closest frontiers. In simulated completely static environments, the robot's explorative behavior is always the same when starting at the same initial pose. By randomly generating candidates, decision-theoretic approaches are not deterministic except for cases where the same set of candidates is used throughout all experiments.

The explorative behavior improves with an increasing number of candidates. Using more candidates, however, also increases the runtime of the exploration strategy. A promising approach is to make the number of candidates dependend on the map size. This, however, causes the problem that the approach is no longer scalable as the number of candidates might exceed dimensions where it no longer feasible to evaluate all candidates. A more reasonable approach is to generate candidates only in the vicinity of frontiers as it is done, for example, by González-Baños and Latombe (2002). This will, however, turn the exploration strategy into exploring closest frontiers.

In the approach used here, the information gain solely depends on the uncertainty in the map. That is, the robot selects, respectively, actions and next best views that provide new information about the environment but neglect, for example, the robot's uncertainty in its own pose and trajectory. The decision-theoretic exploration strategies presented in (Stachniss, 2006) do not only focus on map entropy. Here, the information gain is derived from the underlying Rao-Blackwellized particle filter used for simultaneous localization and mapping. By this means the entropy used for estimating the information gain does not only represent the uncertainty in a map of a particle but also in the robot's pose estimated by a particle. This drastically changes the explorative behavior of the robot as there are now two preferred types of candidates: those in the vicinity of frontiers that provide new information and those in already modeled parts of the environments where a large reduction in uncertainty can be expected due to loop closures. The latter allows for increasing the probability of constructing globally consistent maps (Stachniss et al., 2004).



(a) $n = 10$ candidates (b) $n = 100$ candidates

Figure 5.26: Decision-theoretic exploration. Shown are the trajectories of a mobile robot exploring a simulated environment. In (a) $n = 10$ candidates are generated and evaluated to determine the next best view. The resulting path length is 125.86 m. With $n = 100$ candidates (b) the resulting path length is 100.88 m.

5.3.5 Results and Open Problems

Different exploration strategies have been presented in this chapter. Conducted experiments have shown that frontier-based exploration strategies resulted in the shortest trajectories for covering all environmental structures. As the strategy of exploring closest frontiers is also the computationally most effective procedure, compared to the decision-theoretic approach and the frontier-strategy with additional segmentation, it is the preferred exploration strategy for the real robot. However, before presenting a final experiment carried out during the RoboCup@Home competitions at the GermanOpen in Hannover 2009, some results obtained from simulation are discussed to ground the preference for exploring closest frontiers.

Evaluating Exploration Strategies in Simulation

As previously described, throughout the work on this thesis, a vast variety of experiments have been conducted primarily in simulated environments, as these experiments provide equal preconditions and static environments. Different environments have been explored ranging from small apartment-like environments to large scale office-like environments. Besides exploration solely using random walks, all exploration strategies allowed the mobile robot to fully cover environmental structures with its sensors and to construct a complete environment model. Here the outcome of three such simulation experiments is presented. The robot explored three different environments, the “Cave” and “Hospital” environments provided with Player/Stage as well as a custom map modeling the AVZ building at the University of Osnabrück. For each environment an initial pose has been defined. The robot then explored the environment multiple times from this initial pose.

For the “Cave” environment 10 runs have been carried out for all presented exploration strategies. Mean values and standard deviations for the measured trajectory lengths are visualized in Figure 5.27.a. Not surprisingly, trajectories resulting from exploring random frontiers together with repetitive re-checking (RR) yielded the longest trajectories and also the largest standard deviations. This is due to the fact that repetitive re-checking forces the robot to perform even more zigzag-movements from one environment region to another than with random-frontier exploration without re-checking. Exploring closest frontiers with and without re-checking as well as with and without map segmentation yielded the shortest trajectories. Standard deviations for the three strategies are considerably smaller than that of the other strategies since exploring closest

frontiers in simulated static environments is almost deterministic. Here, it is to remark that map segmentation and preferring frontiers in the robot's segment resulted in even longer trajectories than without segmentation. This is due to the fact that some of the frontiers falling into the same segment caused the robot to move along an additional curve in some of the experiments. Without segmentation, the robot has chosen to approach another frontier and information about the aforementioned frontiers has been acquired en route. For the decision-theoretic strategy, different constant numbers of candidates have been used. Since the performance of this strategy highly depends on the distribution of candidates, measured trajectory lengths are decreasing with increasing number of candidates. However, it should be mentioned that evaluating a larger number of candidates also drastically increases the runtime of the strategy. Since only map uncertainty is considered in this strategy, it would be more reasonable to generate candidates only in the vicinity of frontiers. Uniformly distributing candidates only allows for a wider class of actions if, for example, also the robot's uncertainty about its pose is taken into account (Stachniss et al., 2004). Candidates in already modeled regions can then approached for actively closing loops. However, restricting candidates to lie in the vicinity of frontiers is not expected to show an explorative behavior that considerably differs from exploring closest frontiers.

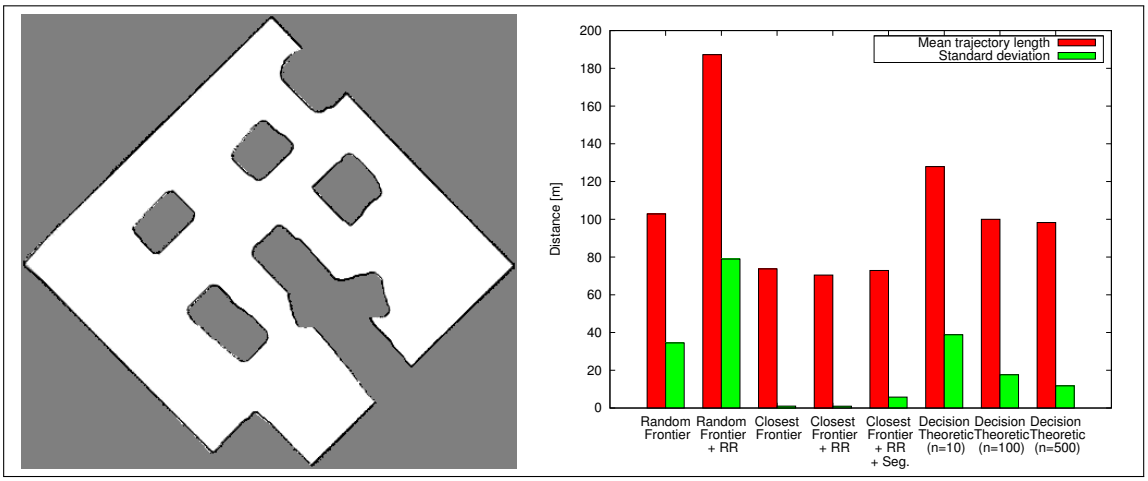
For exploring the AVZ building five explorative runs have been carried out for each strategy. Three strategies have, however, not been considered here including exporing random frontiers with repetitive re-checking as well as the decision-theoretic strategies with smaller numbers of candidates. Furthermore, the 500 candidates used have been generated in regions that contain frontiers. Again, exploring closest frontiers yielded the shortest trajectories (see Figure 5.27.b). Here, map segmentation and preferring frontiers in the robot's segment yielded, in average, the shortest trajectories. However, the difference to exploring closest frontiers without segmentation is only very small, since the segmentation algorithm did not properly segment the map into single rooms. That is, even with segmentation the robot had to travel back to a single room in order to acquire information about the last frontier after exploring the rest of the environment.

In the "Hospital" environment the same strategies has been used and five explorative runs have been carried out. Here, the difference between exploring closest frontiers with and without segmentation is noticeable. Map segmentation resulted, in average, in shorter trajectories. This might be due to the fact that in this environment rooms are multiply connected and the interiors are not visible from the same corridor for all rooms. Although, the segmentation is, again, not perfect, the robot had to approach considerably less frontiers remaining in already explored regions. However, this small improvement of the travel distance seems to not justify the increased runtimes due to the segmentation and the increasing complexity of the overall exploration strategy.

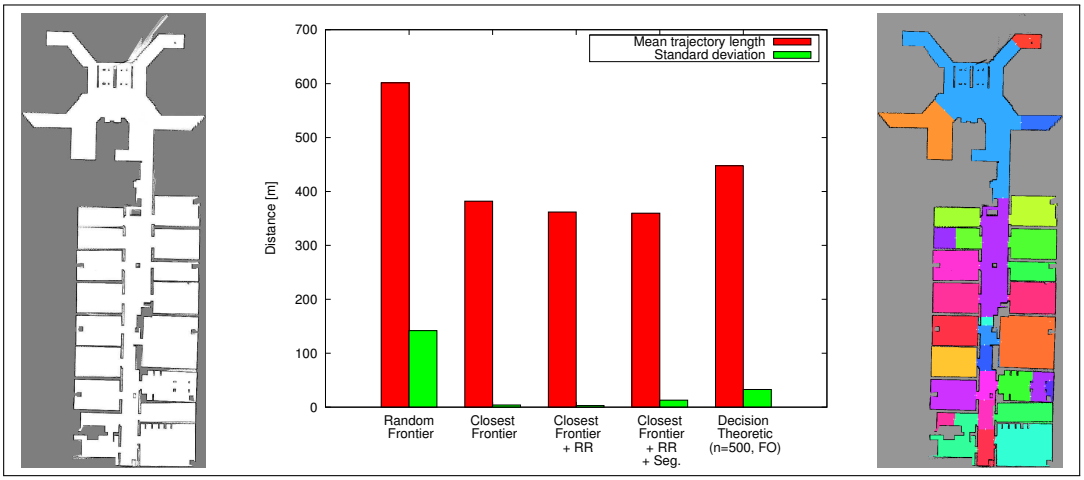
Regarding the decision-theoretic approach, both the estimation of the expected information gain as well as the utility function used for evaluating candidates might need to be improved. Even when restricting candidates to lie in the vicinity of frontiers, measured trajectories are still considerably longer than trajectories resulting from exploring closest frontiers. Furthermore, the integration of the robot's uncertainty about its pose is expected to improve, in general, the quality and consistency of constructed environment representations (Stachniss et al., 2004; Stachniss, 2006). Throughout all the experiments carried out here, the resulting maps were, however, correct and consistent.

Exploring the RoboCup@Home arena

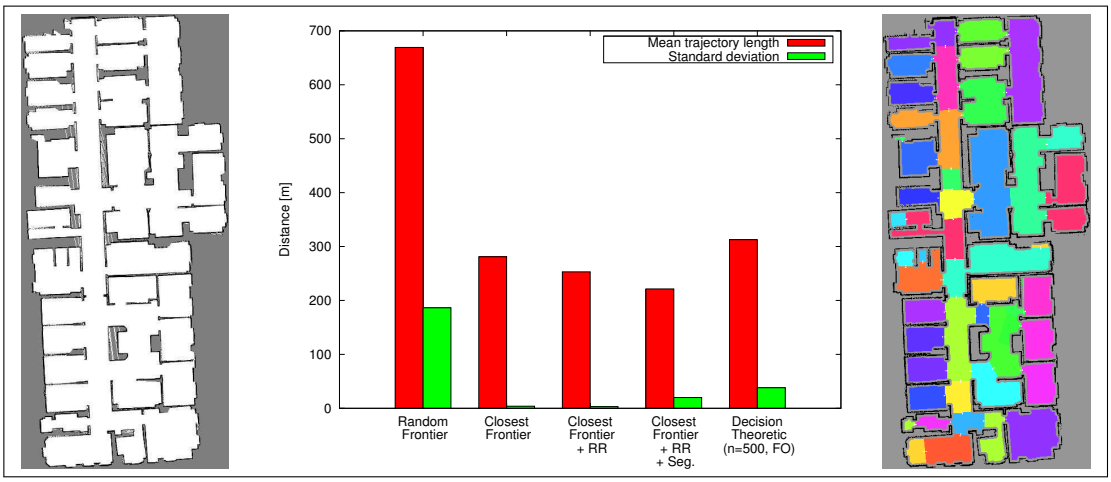
The goal of this thesis was to develop a complete set of algorithms for exploring and inspecting the workspace of a real service robot. All involved algorithms should not only be robust against e.g. noise, internal failures or environment dynamics, but also applicable in real-time. That is, all algorithms are run online and the robot actively explores a real environment while constructing an internal environment representation. Regarding the real-time applicability, especially the algorithms for performing simultaneous localization and mapping as well as path planning and motion control can be run within the measurement frequency of the laser range finder (75 Hz) or at least do not exceed the update frequency of the motor controller (10 Hz). What seems to be more important is that the robot is acting naturally for an external spectator. The robot should, for example, move continuously and along smooth trajectories without the necessity to stop in order to carry out expensive computations. It should, furthermore, not collide with any obstacles



(a) Player/Stage "Cave" environment



(b) AVZ building at University of Osnabrück

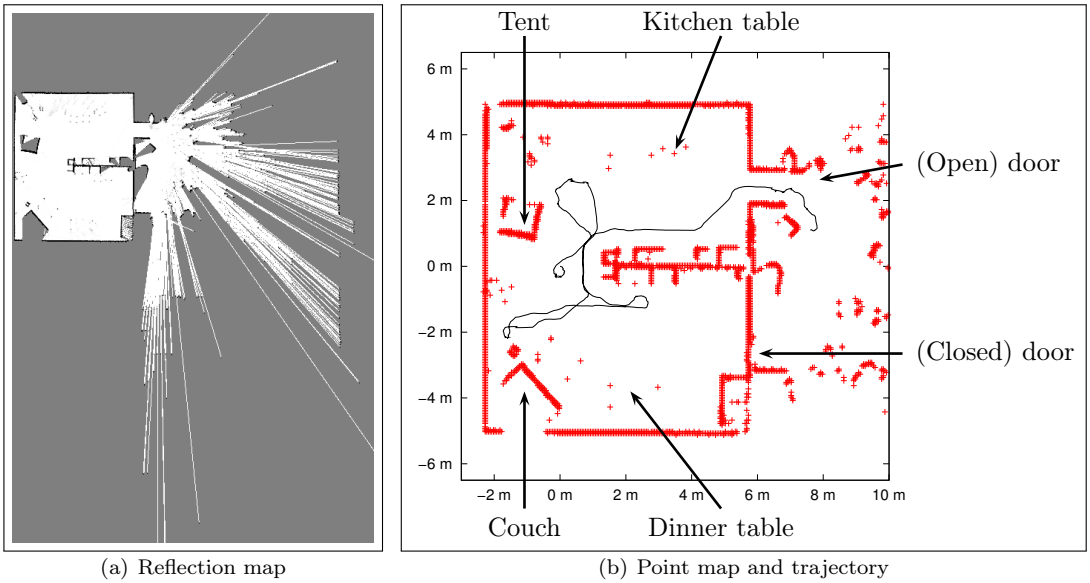


(c) Player/Stage "Hospital" environment

Figure 5.27: Evaluation results in simulated environments. Shown are measured mean values and standard deviations for the paths travelled by the robot in three different environments (used abbreviations: RR = repetitive re-checking, Seg. = map segmentation, FO = frontiers only).

and move along the shortest path to a goal instead of making considerable detours. Naturally, the environment model constructed during the explorative run should be consistent and accurate so that it is applicable in subsequence navigation tasks.

To provide a proof of concept, the service robot "Johnny Jackanapes" has been used to explore the RoboCup@Home arena at the GermanOpen in Hannover 2009. Starting at the center of the arena, the robot completely explored all environmental structures finally leaving through one of the doors. In the team area behind the arena, the exploration task was manually terminated to avoid that the robot continuous exploring the entire exhibition hall. The probabilistic reflection map and the point map constructed during the explorative run as well as the robot's trajectory are shown in Figure 5.28. The path travelled by the robot has a total length of 32.37 m. After conducting the characteristic loop for exploring environmental structures behind the robot's initial pose, it moves along a first loop through the kitchen exploring the unknown region behind the tent. As the legs of the kitchen table and the surrounding chairs caused a very small number of reflections, almost the entire kitchen is already explored after this first loop. The robot then enters the living room sensing environmental structures at the separating wall in a second loop. The only remaining unknown region is then to the left of the couch. In a third loop the robot explores this region and then travels to the doorway in the kitchen. This last frontier also forms the transition between arena and team area. After travelling a few meters through the team area, exploration was stopped. The constructed maps accurately model environmental structures and objects in the arena. By means of the probabilistic reflection map even regions with low reflectivity, especially at the two tables, are accurately modeled and not considered when planning paths through the arena. It should also be noted that the environment was not static but populated by humans during the explorative run. These, however, did not affect the robot's trajectory and the robot did not need to actively make detours in order to avoid collisions. The overall exploration run took less than 5 min. In principal, the robot could have directly started to accomplish necessary tasks defined in the RoboCup@Home competition such as navigating to a certain location or searching the workspace for a particular object.



(a) Reflection map (b) Point map and trajectory

Figure 5.28: Exploring the RoboCup@Home arena (GermanOpen 2009). Shown is the constructed reflection map (a) as well as the robot's trajectory and an annotated point map (b). Exploration started at $(0, 0)$ and was terminated outside the arena.

Concluding Remarks and Future Work

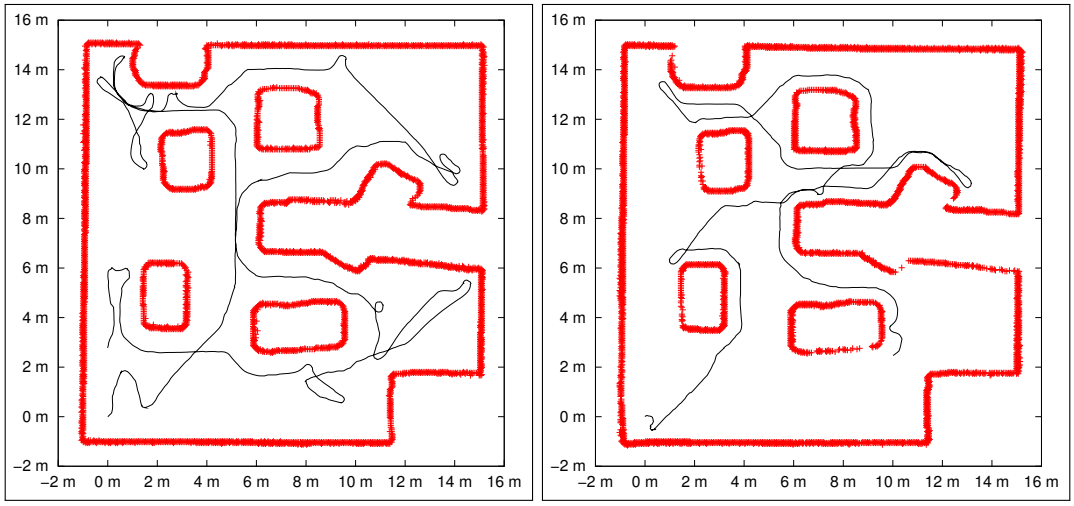
All sampling-based approaches to NBV planning that balance expected information gain and cost of candidates bear one particular problem. Once the expected information gain of a candidate being far away from the robot cancells out the involved costs, e.g. when candidates in the immediate surrounding environment let not expect a large information gain, the robot moves to that other area, although a small door or opening placed at a disadvantageous position in the immediate surrounding environment would have led to a large unexplored area. Furthermore, even if there is no larger unexplored area, the above problem can force the robot move out and back, travelling again and again from one area to another. A comparable problem has been described for exploring closest frontiers. Here a frontier in another room that is closer to the robot as frontiers in the same room can cause that the robot leaves regions that are not yet fully explored. Both problems lead to the situation that the robot has to come back to the partially explored regions for a complete coverage. The segmentation of the map as described in this thesis as well as comparable approaches in recent literature, seem to provide promising means to address this issue and to decompose the exploration problem into a local and global exploration strategy. The local strategy forces the robot to completely explore the currently visited room, whereas the global strategy picks the next unexplored room once the the current room is completely modeled. Gossage et al. (2006), for example, combine a local frontier-based exploration strategy with a global graph-based exploration strategy. An integration of such a concept into the set of algorithms presented in the context of this thesis is a matter of future work.

Another interesting extension would be to classify determined segments, e.g. into rooms, corridors and doorways (Martínez Mozos et al., 2007). Although NBV planning is a quite complex problem of deliberative nature in the context of robot control architectures, the actual idea behind most approaches is rather reactive. Oversimplified, the robot is attracted by unexplored areas and repulsed from already explored areas as in a potential field approach. Integrating semantic information obtained from segmenting and classifying rooms and corridors can be used to make the robot's explorative behavior more goal-directed. If, for example, the robot detects that it is currently exploring a corridor (Martínez Mozos et al., 2007) it could neglect other unexplored areas and first model the complete corridor and than one room after another. By concentrating on one particular area the number of samples in a decision-theoretic approach can be kept small while still finding good locations in the candidates.

Occlusion-based approaches to NBV planning examine which structures in a generated model can be expected to occlude larger volumes behind them in order to position, respectively, the sensor and the robot in order to sense that very occluded volume. In the same way obstacle and structure maps can be used to determine the size of the area that is occluded by those structures in the obstacle map that are not present in the structure map. The size of that area could then be taken into account in the exploration strategy. This can be efficiently done by updating a probabilistic reflection map using the obstacle map. Approaches that extract planar slices in a certain height out of a 3D model and use this information for NBV planning, as done for example in (Nüchter et al., 2003a), neglect that objects below or above that plane can occlude larger regions of the environment. As soon, as the boundaries of the environment are completely modeled, the approach of Nüchter et al. assumes that also the corresponding region in the environment is completely explored. Here it would be interesting to examine to which degree obstacle and structure maps can improve the explorative behavior of a mobile robot by actively *looking behind* closest obstacles that can be expected to occlude larger areas or volumes.

A general issue in exploration strategies is the question of when to terminate. For frontier-based exploration strategies exploration is simply stopped when the robot cannot determine any more frontier cells. For decision-theoretic exploration strategies this is more complex. In this thesis we expressed information gain in terms of uncertainty reduction. Accordingly exploration can be stopped if the map entropy converges or when the largest expected information gain falls below some threshold. During exploration, the map entropy might drastically change to the dynamic re-allocation and resizing of the map. When the environment is almost completely explored, the map is no longer re-sized and due to the fact that no more information can be acquired the map entropy converges. Here, it is important to choose an adequate value for thresholding the expected

information gains and to decide when the map entropy has converged. Here, a value of 100 has been used to threshold the largest expected information gain corresponding to removing the uncertainty of approximately 70 unknown cells. The value is rather high since inaccuracies in the range sensor and the discretization of the map cause that boundaries never have a reflection probability of 1. That is, even in a completely explored environment, information gains up to 100 can be measured. Note that this value highly depends on the grid resolution and the used range sensor as well as the representation of the grid map and its update procedure. What can also be observed is that a robot using the decision-theoretic approach more often visits places in already modeled terrain to reduce the uncertainty in boundary cells. With the frontier-based approach, these regions are no longer considered since the reflection probability of the corresponding cells is already larger than 0.5. Consider for example the maps shown in Figure 5.29. The entropy for the map resulting from decision-theoretic exploration (Figure 5.29.a) is 77 393.86. The map entropy for the frontier-based exploration strategy (Figure 5.29.b) is 77 613.87. Both values correspond to an average cell entropy of approximately 0.4. That is, most of the cells have a reflection probability of more than 0.9 or less than 0.1. The slight difference in entropy is, however, visible when extracting points at the centers of cells whose reflection probability is larger than 0.75. Especially in narrow corners the point density in Figure 5.29.b is lower than in the map of Figure 5.29.a. The decision-theoretic approach better explores this corners by approaching multiple poses in their vicinity whereas the frontier-based exploration strategy moves the robot into another region of the environment once the corner does no longer contain unknown regions. The same holds true for all environmental structures. With the decision-theoretic exploration strategy the robot approached more poses in the vicinity of already modeled structures in order to reduce the uncertainty about boundary cells in the robot's vicinity before exploring other regions. This is also the primary reason why the resulting paths are longer than the paths resulting from exploring closest frontiers.



(a) Decision-theoretic (b) Repetitive closest frontiers

Figure 5.29: Completeness in decision-theoretic and frontier-based exploration. Shown are maps and trajectories resulting from decision-theoretic exploration (a) and frontier-based exploration (b). The points correspond to cells with a reflection probability $p(c^{[xy]}) > 0.75$. The robot is more certain about the map in (a).

5.4 Strategies for Autonomous Inspection

Robotic exploration addresses the problem of covering all regions in a preliminary unknown environment with the robot's sensors. Here it is quite natural to apply greedy strategies. Until the environment is completely explored the robot only has partial knowledge about its workspace. Furthermore, those regions that need to be visited for a complete coverage are those where no

information is available. Hence, it is quite natural to address exploration by means of greedy strategies instead of trying to solve an art gallery problem. These greedy strategies plan only one action ahead. When exploring closest frontiers, for example, these actions lead the robot to the closest region where no information is available. By this means the robot travels from one unknown region to the next in a greedy fashion.

In contrast to exploration, robotic inspection addresses investigating already modeled environments. That is, the robot has already completely explored its workspace. The constructed environment representation models all environmental structures and the robot has complete knowledge about its workspace at the time of exploring it. With this knowledge the robot could, in principal, determine the optimal shortest route through the complete environment allowing to sense all environmental structures afresh. For determining this shortest route, the robot has to address the *watchmen route* problem. That is, given a (polygonal) workspace, find a tour in form of a continuous curve so that all environmental structures are visible from at least on point on the curve (Chin and Ntafos, 1991).

Chin and Ntafos presentend an algorithm that finds the shortest watchmen route passing a given point on the polygon's boundaries for a simple polygon with n vertices in $O(n^4)$ time (cf. Chin and Ntafos, 1991; Urrutia, 2004). Tan et al. adapted this original formulation reducing its runtime complexity to $O(n^3)$ (Tan et al., 1993) and $O(n^2)$ (Tan and Hirata, 1993). Hammar and Nilsson (1997) revealed that an error in all three algorithms can lead to exponential runtimes in some special cases. Tan et al. (1999) finally combined the techniques of the original algorithm with principals of dynamic programming and presented a correct $O(n^4)$ algorithm. The restriction that the route has to go through one specific point can be neglected when, for example, the robot has to enter the workspace through a door before being able to explore it. The algorithms can, however, not be applied when there is no such point through that the watchman route has to go. That is, if we cannot determine a starting point for the algorithm beforehand, we cannot find an optimal watchmen route. An approximation of the shortest watchman tour for polygons with holes has been presented by Arkin et al. (2003).

In principal, the problem of finding an optimal watchman route can be decomposed into two parts (cf. Danner and Kavraki, 2000):

1. Determine a minimum set of sensing locations allowing for a complete coverage of the workspace, i.e. solve an *art gallery problem.*

2. Connect the sensing locations by determining a shortest path through all locations in the minimum set. This problem can be re-casted as a *travelling salesman problem (TSP).*

Both problems have been shown to be NP-hard. Both can, however, be approximated as described in the beginning of this chapter. In fact, many approaches related to robotic inspection found in the literature apply greedy strategies. That is, the robot plans its actions solely on what it has seen before instead of considering the complete environment model.

The problem of the travelling salesman is to find the shortest route through n cities that visits every city only one. Related to finding shortest watchman routes and to the travelling salesman problem are pursuit evasion and vehicle routing problems. Vehicle routing problems from a generalization of the TSP to multiple vehicles. These vehicles can have, for example, a limited payload and need to transport cargo to a finite set of customers (Clarke and Wright, 1964). In pursuit evasion and intruder detection problems, an agent inspects a workspace and actively searches for intruders, i.e. other agents moving around in the workspace (Moors, 2008). An inspected area is considered save until the agent looses sight of any possible entry to that region. As soon as another agent could possibly enter the region, it is considered contaminated again. These problems are normally addressed by teams of agents and the actions are planned in way to iteratively minimize the size of the contaminated region. In constract to that, inspection aims at visiting all regions in a robot's workspace in order to notice all relevant changes and update internal environment representations accordingly. That is, we neglect that changes can happen in already inspected regions when the robot is visiting another region. Instead, we assume that either this change gets noticed when inspecting the corresponding region again or is only temporal and no longer perceivable when re-inspecting the region. In the latter case, the change is not important and the environment model

does not need to be updated. Compared to classic watchman route problems, we are not interested in solely inspecting the boundaries of the robot's workspace but all traversable regions. The vertices of a polygon can be inspected by moving along its boundaries. Taking, however, visibility constraints like for instance maximum measurable distances and limited fields of view into account might cause that the interior of the polygon is not completely inspected.

The remainder of this chapter addresses the problems of detecting changes in already modeled regions in the robot's workspace, the application of greedy exploration strategies for robotic inspection as well as inspecting manually and autonomously selected places considering, amongst other issues, the travelling salesman problem.

5.4.1 Detecting Novelties and Changes

An important issue in the context of inspection is the detection of changes in addition to inherently considering them in the internal environment representation. Consider, for examples, that a piece of furniture was moved from the place where it has been modeled to another location in the robot's workspace. In the probabilistic reflection maps this change causes that the reflection probabilities in one region decrease while increasing in another region. Accordingly, new measurements are added to a sparse point map at the new location, whereas points measured at the old location are deleted once they are lying in cells with low reflection probabilities in the validity check (see Chapter 3.6.2). That is, the environment model is updated without actually detecting that a change has happened. However, reflection maps represent for each region in the environment the probability of reflecting laser range beams. Hence, they provide all the means being necessary to determine whether a perceived range reading could be expected or not. Repeatedly acquiring unexpected measurements in the same region suggests that a change has taken place. Based on this idea, a simple novelty detection procedure can be defined.

Coming back to the above example, there are two types of changes perceivable in the robot's workspace: an object modeled in the internal representation is no longer present in the real environment and an object being present in the real environment is not yet modeled in the internal representation. Hence, we can define two types of novelties:

Unexpected Hits occur when a range beam is reflected by an object in a cell with low reflection probability; in other words, *sensing something unexpected*.

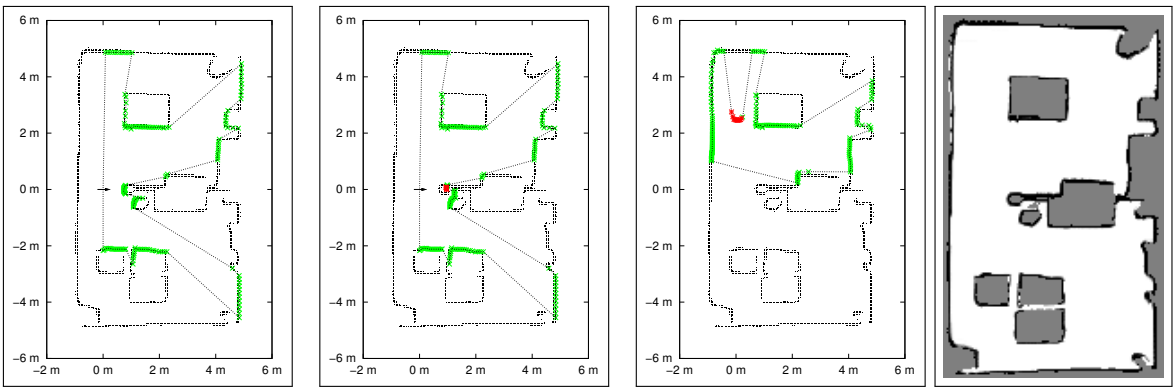
Unexpected Misses occur when a range beam passes through a cell with high reflection probability without being reflected; in other words, *not sensing something expected*.

Based on these definitions we can classify cells being touched in a reflection map update as well as laser range beams causing the update. Updated cells can be classified in four categories, unexpected hits and misses as described above as well as expected hits and misses. That is, cells with low reflection probability are not expected to reflect a range beam. If such a cell reflects a laser beam it causes an unexpected hit. Accordingly, a cell with high reflection probability is expected to reflect a range beam. Not reflecting the beam causes an unexpected miss. A simple procedure for detecting these unexpected events can be simply integrated into the update procedure of a probabilistic reflection map since it performs ray casting anyway in order to determine which cells need to be updated.

In the actual implementation, the probabilistic reflection map is extended to maintain arrays for counting unexpected hits and misses for each cell in addition to the regular hits and misses used to determine the cell's reflection probability. To minimize the amount of false positives in the detection of novelties unexpected hits and misses are counted. Once, the number of unexpected events in a cell exceeds a certain a threshold the according event is considered to be something novel. By this means, only temporal dynamics do not cause the detection of novelties since they do not cause a sufficient number of unexpected events. In contrast, rather static changes like for instance moved furniture is reliably detected.

Keeping track of unexpected events allows not only for storing information about things that happened during an inspection run, but also to classify single range measurements. Figure 5.30

shows examples of laser range scans with expected and unexpected hits and misses. In this experiment, the robot explored the simulated RoboCup@Home arena of the GermanOpen 2008. In the beginning of the experiment all modeled objects are at their place. One object is then moved through the environment and positioned at a different place. This causes unexpected misses at the original location of the object as well as unexpected hits at the new locations.



(a) Not sensing unexpected (b) Unexpected misses (c) Unexpected hits (d) Reflection map

Figure 5.30: Examples for unexpected misses and hits. Shown are scans during an inspection run in a simulated environment. One object has been moved around during the experiment causing no novelties before moving (a), unexpected misses after removing (b) and unexpected hits after putting it in the kitchen (c). The reflection map is shown in (d).

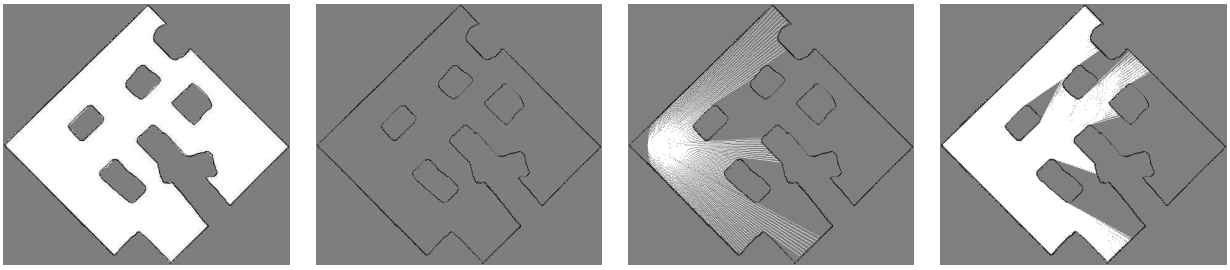
A possible application of this simple novelty detection procedure is to make the detection of human guides (see Chapter 5.2.1) more robust since the guide is not modeled in the map and will surely cause unexpected hits while moving through the environment. For this application, however, the threshold for detecting a novelty needs to be decreased since the human guide can be considered as a temporal change in the robot's surroundings. It should also be mentioned that the classification of measurements not yet modeled in the map, is inherently given in the registration procedure since these points will not find corresponding points in the model set.

5.4.2 Inspection using Greedy Exploration Techniques

As already mentioned, a common approximation of art gallery and watchman route problems is to iteratively plan actions one after another in a greedy fashion. For deciding which action should be carried out next only the so far acquired information is used. In the context of inspection this means, that the available complete environment representation is not used to determine a set of sensing location and a shortest watchman route. Instead, the robot re-explores the complete environment.

Using the greedy exploration strategies presented in Chapter 5.3.2 and Chapter 5.3.4 a simple inspection procedure can be formulated. Since we want to primarily inspect the interior of the workspace, i.e. all traversable and reachable cells in the reflection map, the idea is to make exactly these cells unknown again while maintaining the environmental structures. With such a map the robot can still localize itself in the complete workspace while the regions that need to be inspected are unknown and, thus, approached by a greedy exploration strategy. We will refer to this modified map as a *local inspection map*. Once the robot is commanded to inspect the environment, it creates a copy of the current state of the probabilistic reflection map. This copy then makes up the local inspection map. After determining all traversable cells in the map, they are marked as being unknown in the local inspection map. To account for the robot's current visibility, the latest laser range scan used for registration is integrated into the local copy. An example of a local inspection map as well as the just described procedure for generating it is visualized in Figure 5.31.

The actual inspection of the workspace is then carried out using an arbitrary greedy exploration strategy. As the exploration of closest frontiers has yielded the shortest trajectories, it is used in



(a) (b) (c) (d)

Figure 5.31: Local reflection map copies for inspection tasks. Shown is the initial reflection map (a), the local copy with all traversable cells being marked as unknown (b) and after updating it with the latest range scan (c). The greedy exploration strategy than inspects the workspace by exploring unknown regions in the local inspection copy (d).

this example to explore the unknown regions in the inspection map and, by this means, inspect the traversable regions in the reflection map. Instead of using the regular reflection map for determining frontier cells and planning paths, the local inspection map is used. Here, it should be noted that it is actually not needed to maintain two maps. The idea of the local copy is to maintain the original reflection probabilities in order to apply the aforementioned novelty detection procedure. It is also to remark that the detected novelties can also cause differences in the original map and the local inspection map. If the original reflection map is solely used for localizing the robot without performing map updates, changes in the environmental structures are reflected in differences in the individual cell reflection probabilities of the two maps.

An interesting property of this approach is that the same exploration strategy can be used for both exploring a preliminary unknown environment to construct a model and inspecting the constructed model. Figure 5.32 shows the trajectories from a experiment where the root was put into a preliminary unknown environment. The robot immediately started exploring its workspace. As soon as no more frontier cells could be determined in the thereby constructed reflection map, the robot stopped exploration and switched to inspecting the constructed map. The overall path length of the robot in this experiment is 120.69 m. The first part of the trajectory where the robot explored the environment has a length of 66.05 m obtained using closest frontier exploration with repetitive re-checking and map segmentation. In the second part of the experiment, the robot explored the local inspection map. The resulting path has a length of 54.64 m. This difference is caused by the fact that especially those cells that lie in narrow corners and cause detours in the exploration phase, are not traversable due to safety distance constraints. Hence, the corresponding cells do not cause frontiers in the local inspection map, the robot does not make detours for exploring them and the overall inspection path is shorter than that for exploration.

5.4.3 Inspection of Manually Selected Places and the Travelling Salesman Problem

Determining a shortest route and selecting the order in which locations are inspected are not only an important issue in completely autonomous inspection but also when inspecting a given set of locations. An application example is to assign a task to the robot that lets it inspect multiple entry halls in a larger building to welcome arriving guests. Given a set of manually selected locations, the task of inspection turns into i) selecting an order of locations or determining a shortest tour through all given locations and ii) approaching all location along the tour. Once, the robot has finished one inspection round, it starts again from the beginning. An important issue in this formulation of an inspection task is determining the order in which locations are approached. Picking some random order may cause that the robot moves from one region to another without really showing a reasonable inspection behavior. The problem of determining the order resulting in the shortest path is what makes the travelling salesman problem. Given a set of n cities, a salesman needs to

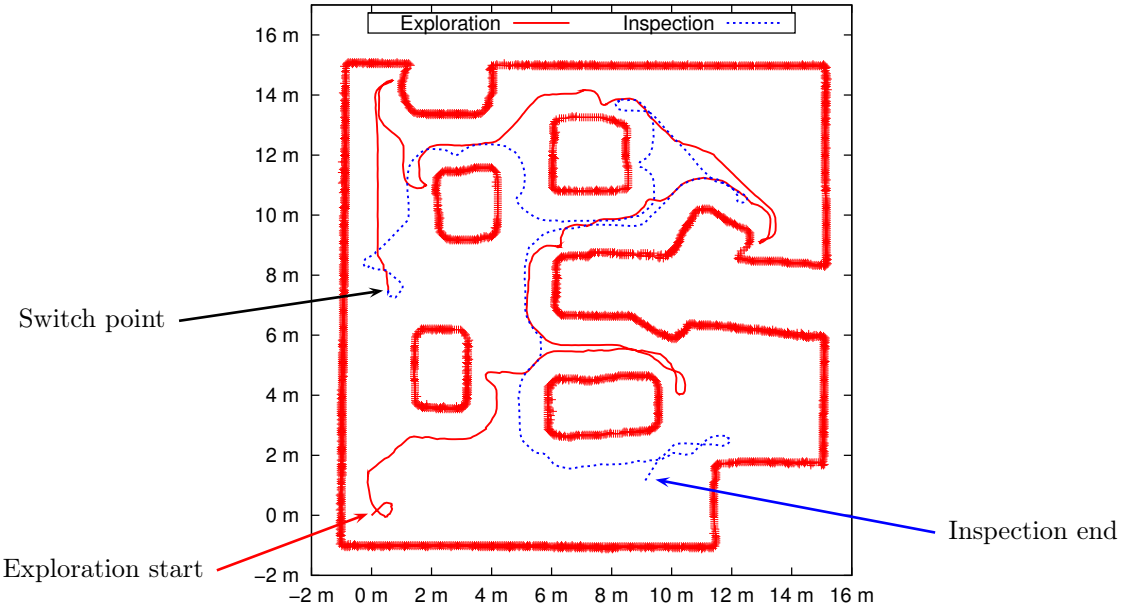


Figure 5.32: Successive exploration and inspection. The robot first explored the preliminary unknown environment. The robot stopped exploration when the map did no longer contain frontiers and started inspection (switch point). The robot stopped inspection when the local inspection copy was fully explored.

travel along a shortest path through all cities visiting each city only once. The latter restriction can be removed in the context of inspection, as it not problematic to inspect one location more often than others. However, removing this restriction does not change the NP-hardness of the travelling salesman problem (see Applegate et al., 2007, Ch. 1). For a comprehensive overview one the complexity classes of the TSP and derived subproblems as well as a survey on different means and algorithms for solving TSPs it is referred to (Applegate et al., 2007).

Approximating TSPs using Self-Organizing Maps

A computationally efficient way of addressing the travelling salesman problem is by using self-organizing maps (Angéniol et al., 1988). As shown by Favata and Walker (1991), self-organizing maps adapted to TSPs yield slightly longer tours than, for example, simulated annealing (Kirkpatrick et al., 1983) but require less runtime (cf. Choy and Siu, 1995).

Self-organizing maps form a special form of artificial neural networks and are trained by means of unsupervised learning. Associated with each neuron is a weight vector whose dimension is that of the input data (Kohonen, 1982). In the context of the travelling salesman problem, the input vector is formed by the x- and y-coordinates of the positions of the cities. The neurons form a map and are organized in a topological structure. Compared to other neural network structures, the input is connected with each neuron. Self-organizing maps are trained by means of competitive learning. That is, when giving a training example to the network, its Euclidean distance to the weights of all neurons is computed. The neuron being closest to the input example wins the competition. The weight vector of the winning neuron as well as for those neurons that lie in a certain distance to the winning neuron is adapted. The learning rule for updating the weight vector $\mathbf{w}_i$ of neuron n_i is, for example,

$$\mathbf{w}_i(t+1) = \mathbf{w}_i(t) + \Theta(\mathbf{w}_i(t), \mathbf{w}_j^\star(t), t)\ \eta\,(\mathbf{w}_i(t) - d(t)) \tag{5.13}$$

where $\mathbf{w}_i^\star(t)$ is the weight vector of the winning neuron $n_j^\star$, $\Theta(\mathbf{w}_i(t), \mathbf{w}_j^\star(t), t)$ a neighborhood function and η the learning rate. In its simplest form, $\Theta(\mathbf{w}_i(t), \mathbf{w}_j^\star(t), t)$ is 1 if the Euclidean distance between $\mathbf{w}_i(t)$ and $\mathbf{w}_j^\star(t)$ falls below some threshold and 0 otherwise. This learning rule together with the neuron topology cause that neurons in the same region of the map respond similarly to

certain input patterns. This is actually not surprising since the concept of self-organizing maps is partially inspired by the human brain that processes different sensory inputs in different parts of the cerebral cortex (Kohonen, 1982; Haykin, 1998, Ch. 9).

For applying self-organizing maps to the travelling salesman problem, the topology of the neurons is chosen so that their connections make up a circle. This circle forms the route for solving the traveling salesman problem. That is, we do not determine the order in which we approach locations since the order is given initially by the order of neurons. Instead we move the neurons by updating their weight vectors according to Eq. (5.13) so that the path determined by the tour goes through the given set of locations. To consider the circular topology in the update rule, we need to adapt the distance function. Whereas the distance between weight vectors and input data is still expressed by means of Euclidean distances, the distance function Θ is changed to no longer measure distances in Euclidean distance but in the discrete one-dimensional index-space. That is, neighbors are determined with respect to their difference in indices instead of their Euclidean point-to-point distance. That is,

$$\Theta(\mathbf{w}_i(t), \mathbf{w}_j^\star(t), t) = \begin{cases} 1 & \text{, if } (i-j) < r(t) \\ 0 & \text{, otherwise} \end{cases} \tag{5.14}$$

where $r(t)$ is a neighborhood radius that decreases over the course of training.

A nice property of this approach is that it is easy to understand and can be implemented with just a few lines of C-code. Common configurations for solving a TSP problem are to set the number of neurons in the self-organizing map to an integer multiple of the number of input cities (cf. Favata and Walker, 1991). Here we will use $m \geq 4n$ nodes for n cities. Furthermore the map is trained in $T_{\max} \geq 100m$ times for every city. Both the learning rate η and the neighboor radius r are adapted during training using the following functions:

$$\eta(T) = \left(\frac{1}{20}\right)^{T/T_{max}} \tag{5.15}$$

$$r(T) = \frac{N_{nodes}}{2} \cdot \left(\frac{1}{N_{nodes}}\right)^{T/T_{max}} \tag{5.16}$$

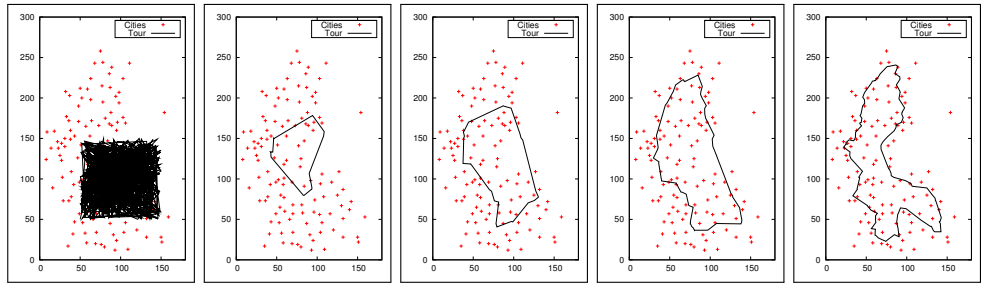
where N_{nodes} is the number of neurons.

For providing a first proof of concept the implemented SOM-TSP algorithm has been used to find the shortest route through 120 cities in (former) Western Germany, a TSP problem presented and solved by Grötschel (1977). Even this small example cannot be solved using e.g. brute force search through all permutations. Starting with an arbitrary city we have $n-1$ cities that can be used as the second city, $n-2$ choices for the third city and so on. Hence, we have a total of $(n-1)!$ different routes through n cities. Since the travel direction does not play a role, pairs of two routes correspond to the same tour and we get a total $(n-1)!/2$ tours. That is, for the 120 German cities there are $2.787\,292\,88 \times 10^{196}$ possible tours. The data for this TSP problem has been obtained from TSPLIB, a comprehensive collection of TSP problems including optimal solutions and recent problems (Reinelt, 1991).

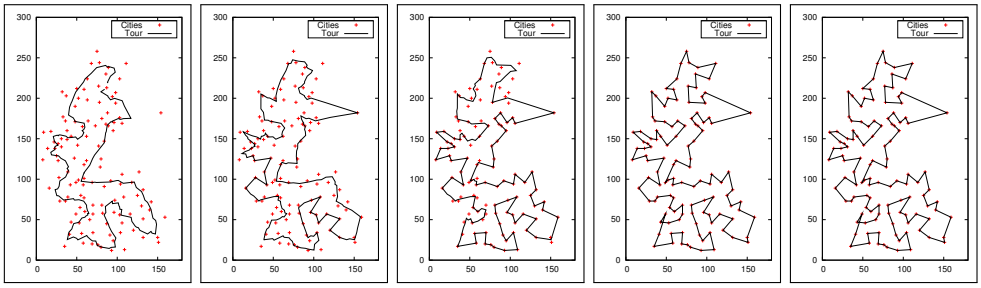
For the $n = 120$ cities, $m = 500$ neurons have been used. The neurons were trained in $T = 50\,000$ iterations. The weight vectors of the neurons are initialized to be uniformly distributed over a square near the centroid of the city positions. Another possible initialization is to set all weight vectors to $(0, 0)$. The random initialization, however, showed a faster convergence towards the final solution in the experiments carried out. Schabauer et al. (2005) have shown that initializing the weight vectors so that they lie on a circle converges even faster. The evolution of the weight vectors during training with random initialization is shown in Figure 5.33.

The obtained solution (after roughly 40 s), shown in Figure 5.34.a, is not optimal but at least provides a sufficient approximation. For obtaining better approximations and speeding up the overall procedure a vast variety of extensions to this simple approach can be found. Choy and Siu (1995), for example, proposed an alternative winner selection mechanism that allows to considerably decrease the number of neurons while still obtaining a good approximation in less training cycles. Schabauer et al. (2005) present a parallelization of the SOM approach to distribute computational

load over several machines and thereby drastically decrease the overall runtime. Another interesting approach is that of Takahashi et al. (2002) who apply SOMs for solving three-dimensional travelling salesman problems.



(a) $T = 0$ (b) $T = 5000$ (c) $T = 10\,000$ (d) $T = 15\,000$ (e) $T = 20\,000$



(f) $T = 25\,000$ (g) $T = 30\,000$ (h) $T = 35\,000$ (i) $T = 40\,000$ (j) $T = 45\,000$

Figure 5.33: Neuron weights during the training of a 120-city TSP.

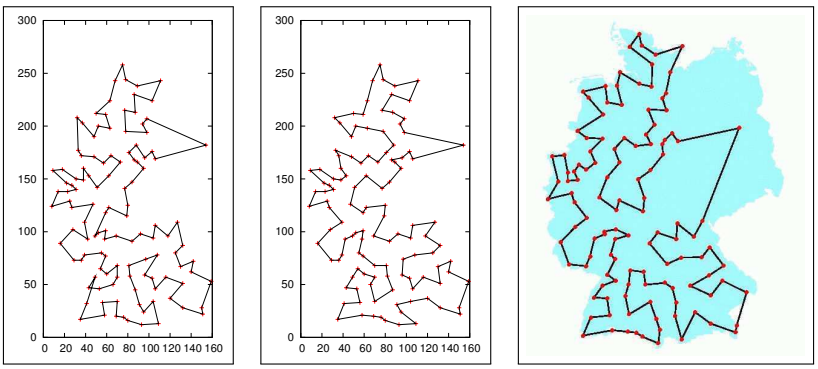
TSP Solvers using *Branch-and-Cut*

Since the SOM approach presented above provides only rough approximations to the TSP, the sophisticated *TSP/Concorde* solver has been used as an alternative algorithm and a simple wrapper interface has been implemented to use the algorithm for solving custom travelling salesman problems. Concorde has set several records including optimal tours for 107 of the 110 TSP problems in TSPLIB (Applegate et al., 2007). It implements a cutting-plane algorithm within a branch-and-bound search. It is based on creating subproblems for subsets of the original n cities and establishing a search tree. For details on the algorithm it is referred to (Appletgate et al., 1999). Very recently, TSP/Concorde has been used to verify an optimal tour through 85 900 cities (Applegate et al., 2009).

Obtained Solutions for the 120-City TSP

The solutions obtained for the aforementioned 120-city problem of (Grötschel, 1977) are shown in Figure 5.34. Training the self-organizing map with 500 neurons over 50 000 iterations took approximately 40 s. The obtained solution has a length of 7256 km. The optimal tour determined by Grötschel (1977) has a length of 6942 km. The tour resulting from the SOM approach is shown in Figure 5.34.a. Using the Concorde solver a tour with a length of 6942 km has been obtained. Although the tour is not equal to the tour of Grötschel (1977), both have the minimum length and form optimal solutions. Solving this 120-city TSP with Concorde took 1.2 s. The resulting tour is shown in Figure 5.34.b. The tour determined by Grötschel is shown in Figure 5.34.c.

Note that all found tours are, although only partly optimal, valid solutions to the given TSP. Since the tours form circles, the salesman can start at an arbitrary city and visit all other 119 cities in one of the two possible directions. That is, the salesman can travel clockwise and counterclockwise.



(a) SOM (b) TSP/Concorde (c) Grötschel (1980)

Figure 5.34: Obtained solutions for the 120-city TSP. Shown are the routes resulting for the SOM-TSP approach (a) with a length of 7256 km and TSP/Concorde (b) with length of 6942 km. The optimal solution (Grötschel, 1980) is shown in (c) (Visualization taken from `http://www.tsp.gatech.edu`).

Inspecting Manually Selected Places

Given an ordered set of locations, inspection is straightforward. The robot first determines the closest location on the route and approaches it. Once the robot has reached the currently approached location, it selects the next according to the order. Here we distinguish two types of inspection tasks, single-loop inspection and continuous inspection. In the single-loop inspection the robot has accomplished the task when it has reached the last location on the route. That is, when the next location is the first location that has been approached. In the continuous inspection task the robot follows the tour again and again. That is, when the robot has reached the last pose of the tour it approaches the first location again and starts a new loop. In both modes, the robot stops inspecting the selected locations when another task is assigned. Furthermore, the actual implementation allows for selecting whether the robot shall find a route through the given places on its own or simply approach the places in the given order. When the robot should find a route on its own, it applies either the SOM approach or calls TSP/Concorde by means of the implemented wrapper and starts inspecting once it has found a solution.

A visualization of an example for the application of this inspection strategy is shown in Figure 5.35.a. Here, five places have been manually selected. Four places form a loop around the kitchen table. The fifth position is the origin of the coordinate frame and the robot's start location. The five locations were given to the robot in random order and the robot was told to find the optimal route by itself. Both, the SOM approach and TSP/Concorde found the optimal solution with a theoretical travel length of 11.66 m. Here, the SOM approach with only 20 neurons for the 5 locations and 2000 training iterations was faster than TSP/Concorde. The runtimes of both were, however, lying in the range of only a few milliseconds. Since the robot was initially standing at one of the selected locations, it directly started to approach the next location on the tour. After reaching the fifth location, it stopped inspecting the workspace and returned to its initial position. The overall path travelled by the robot has a length of 13.53 m.

A similar experiment has been carried out in the simulated AVZ building at the University of Osnabrück. 18 locations have been manually selected, so that every room needs to be traversed when inspecting the locations. Again, both the SOM approach and TSP/Concorde found the optimal solution through the 18 locations. The resulting tour has a length of 106.60 m. The SOM approach took approximately 30 ms to find the tour in 7200 training iterations. Using the TSP/Concorde solver, it took roughly 25 ms to find the optimal solutions. Both runtimes have been measured over 100 runs. Figure 5.35.b shows a map of the environment, the manually selected places as well as the route found by both TSP approaches. For moving to all locations in the order given in the resulting route, the robot travelled a path of 240.17 m length. In this experiment the robot simply stopped inspection after reaching all selected places as described above and did not

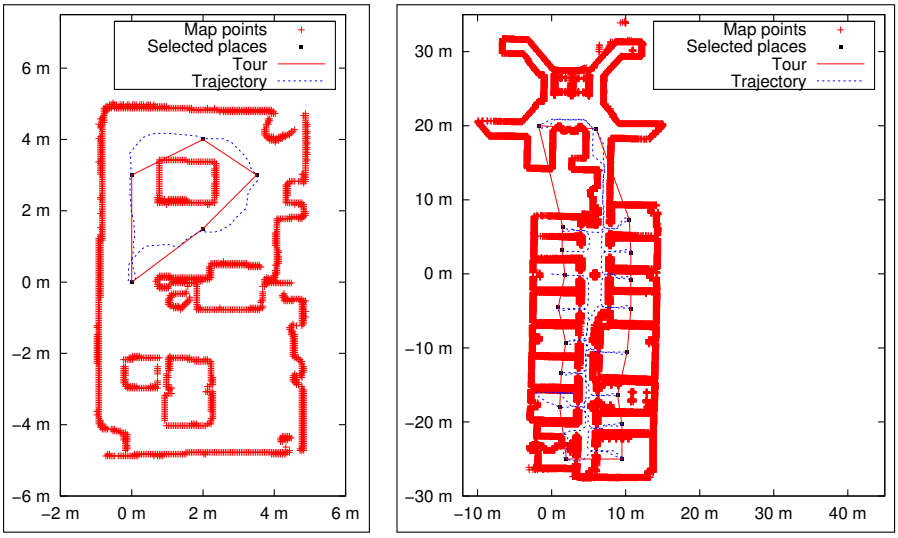


Figure 5.35: Inspecting manually selected places in two example scenarios. Shown are the points maps (red crosses), the selected places (black rectangles), the optimal tours (continuous red curves) and the robot's trajectories (dashed blue curves).

return to its initial position.

As a side note it is to remark, that the TSP tour is only computed to determine the order of approaching locations. As can be seen in Figure 5.35 the connections in the tour are along the direct sight between two locations. Hence, some of the connections intersect walls and cannot be traversed by the robot. The determined tour, however, suggests an order in which locations should be approached. For actually navigating to a location, the robot plans the shortest path within its environment representation. That is, environmental structures such as walls are only considered when navigating to a location but not in the definition of the TSP problem instance. Nevertheless, the resulting trajectories in both experiments provide a rather short watchman route through all manually selected locations. What can also be seen in the figure is that the robot always tries to maintain a maximum clearance to surrounding objects during navigation.

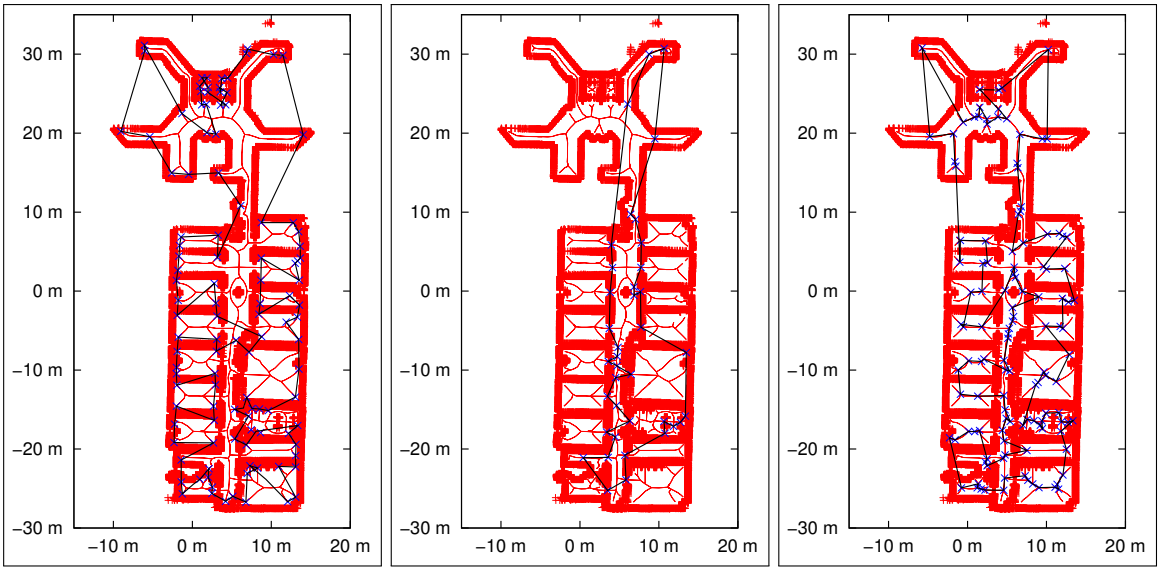
5.4.4 Autonomously Selecting Places for Inspection Routes

With the procedures and algorithms presented so far, the robot is able to inspect manually specified locations, detect novelties in the form of changes in the environment and to update its internal environment representation to take these changes into account. For a completely autonomous inspection, however, the robot needs to decide on its own which locations need to be inspected in order to cover all environmental structures. Consider, for example, that the robot gets the task of inspecting a preliminary unknown environment. That is, the robot has to explore its workspace and start inspecting it once the environment is completely explored. In such a scenario places forming inspection routes cannot be manually specified beforehand. In principal, the robot needs to i) solve an art gallery problem in order to find the minimum set of locations allowing for a complete coverage and ii) determine the shortest tour through these locations.

As already mentioned, we are primarily interested in inspecting the free space. Hence, chosen locations should be distributed over all traversable and reachable regions in the robot's workspace. To determine such a set of poses and, thus, approximate the art gallery problem, several algorithms have been proposed. The actual selection of places is thereby based, for example, on Delaunay graphs (Chen et al., 2008), visibility graphs (Lee and Lin, 1986; Kuipers and Byun, 1991) or Voronoi diagrams (Zelinsky, 1992; Wu et al., 2007). Efficient algorithms for computing visibility and visibility graphs in polygons have been presented, for example, by Sudarshan and Rangan

(1990) and Ben-Moshe et al. (2004). Pocchiola and Vegter (1995) proposed a greedy triangulation algorithm for computing visibility graphs. However, the goal of this thesis was to establish a minimum set of algorithms. So the approach that is followed here will be using the Voronoi diagram. However, for an overview on different applications of visibility graphs and efficient means for computing them, it is referred to (Alt and Welzl, 1988).

As shown in Figure 5.36, the edges of the Voronoi diagram span over all traversable regions in the robot's workspace. In particular, using the nodes of the diagram as possible target locations in an inspection run will force the robot to visit every room at least once. As described in Chapter 4.3, the shown graph has been constructed by i) computing the Voronoi diagram, ii) pruning all edges and nodes outside of the convex hull (or lying in unknown regions) and iii) pruning all edges that cannot be traversed by the robot due to their distance to surrounding obstacles. The selection of locations being used to find the shortest watchman tour determines the inspection behavior of the robot. In Figure 5.36 different possibilities are shown to select locations out of the nodes in the pruned Voronoi diagram. Choosing only end points, i.e. nodes of degree 1, forces the robot to inspect especially corners and dead ends. Since the robot continuously acquires information and updates its internal environment representation during navigation, it can be expected that the robot moves senses and detects changes in all traversable regions (see Figure 5.36.a). However, as shown in the context of frontier-based exploration explicitly moving into narrow corners does not provide more information than simply approaching a pose in its vicinity. Solely selecting the critical points that have been used to segment the map causes that the selected locations lie primarily in doorways. Inspecting these locations forces the robot to have at least a look into every room while navigating primarily in the corridor (see Figure 5.36.b). That is, solely computing the pruned Voronoi diagram allows for different inspection behaviors.



(a) End points (b) Critical points (c) Junction nodes

Figure 5.36: Inspecting nodes in a Voronoi diagram. Shown is the Voronoi diagram for the map of the AVZ building at the University of Osnabrück together with the found tours through the diagram's endpoints (a), the critical points used for segmentation (b) and the junction nodes (c).

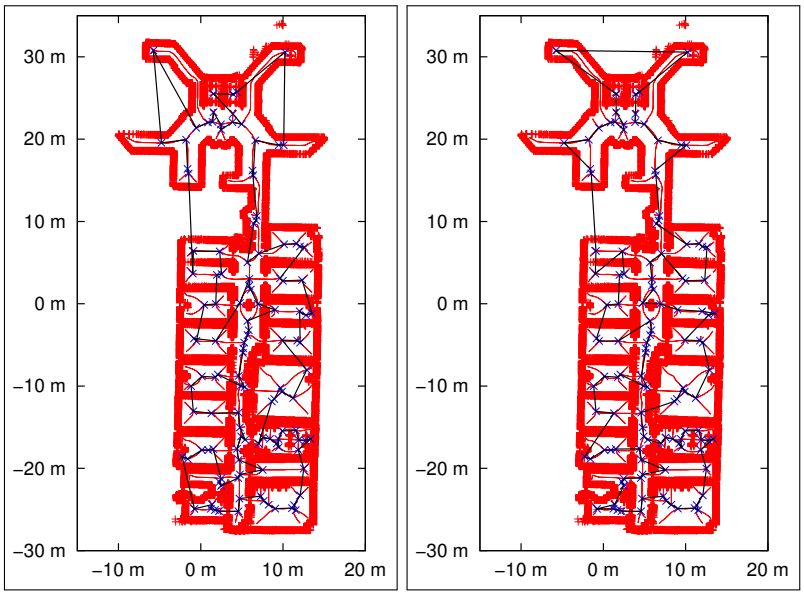
Especially the junctions nodes of the graph, i.e. nodes of degree 3 or higher, show a similar distribution as that of the manually selected places in Figure 5.35.a. They are distributed over all rooms and a larger portion lies in the center of the rooms just like the manually selected places (see Figure 5.36.c). As was shown in the previous section, a route through locations like these forces the robot to move into every room without the necessity to explicitly approach narrow corners. In other words, the robot will primarily follow reasonable walkways while having a deeper look

into every room. Here it would be interesting to compare the resulting tour with that of a real human watchman. For all three possibilities in Figure 5.36, the optimal route found with the SOM approach is also visualized. In most parts of the environment, the routes suggest a reasonable order for approaching the selected locations. However, it need to be mentioned that the routes can only provide an order for inspecting the selected places. The path to the currently inspected location needs to be planned on the robot's internal map taking environmental structures into account.

5.4.5 Results and Open Problems

In a final experiment one completely autonomous inspection strategy is used to make a single-loop inspection through the AVZ building at University of Osnabrück. The actual inspection strategy is composed of two phases: an initialization phase carried out when the inspection is started and an update phase checking the state of the robot's current traversal and specifying the location to be inspected.

In the initialization phase, the robot first constructs the pruned Voronoi diagram and extracts those nodes that have three or more connections, i.e. junction nodes. The junction nodes form the locations that will be inspected by the robot in the update phase. In the presented experiment, 123 nodes have been extracted. The robot than uses one of the two presented algorithms for finding a shortest tour through the extracted junction nodes by addressing a corresponding travelling salesman problem. The tours found in this phase of the conducted experiment are shown in Figure 5.37. The tour found with the approach based on self-organizing maps has a length of 270.95 m. For finding this route, 500 nodes have been trained in 50 000 iterations taking in total roughly 30 s. With 255.48 m the tour found by TSP/Concorde is considerably shorter and has been found in less than 850 ms. Besides a smaller number of zigzag-courses between rooms, both tours provide a sufficiently reasonable order for approaching locations.



(a) SOM (270.95 m) (b) TSP/Concorde (255.48 m)

Figure 5.37: Voronoi diagrams and junction routes. Shown is the Voronoi diagram for the map of the AVZ building at the University of Osnabrück together with the found tours through the diagram's junction nodes.

After computing the tour, the robot determines the location being closest to its current position and starts traversing the planned route by i) planning a path to the next location and ii) following the planned path while reactively avoiding obstacles. Once the robot has reached this location, the next location is selected in the update phase of the inspection strategy and according to the order determined by the TSP solver. During navigation between the inspection locations, the robot

furthermore applied the simple novelty detection procedure described in Chapter 5.4.1 in order to detect changes in the thereby sensed environmental structures. To provide a proof of concept, three objects have positioned in three different rooms of the simulated environment. When the robot has reached the last location on its single-loop tour, it segments the initial probabilistic reflection map and examines the number of unexpected hits and misses for every cell. Cells that contain more than 10 unexpected hits or 10 unexpected misses are assumed to have changed and are considered as a novelty. The segments containing these cells are marked e.g. to inspect them again.

The trajectory of the robot during the simulated experiment is shown in Figure 5.38. The order in which the selected locations are approached is that of TSP/Concorde and has a length of 255.48 m. Due to zigzag-tours between rooms, some regions and rooms are traversed several times leading to an overall trajectory length of 450.81 m. However, the robot approached all selected locations and entered every room in the simulated environment. The not yet modeled objects that have been put into the scenario cause unexpected hits and the corresponding cells as well as the corresponding segments are marked as having changed (see Figure 5.39). Small inaccuracies in the initial model constructed during exploration as well as in the registration of laser scans during inspection led to a false positive, i.e. a segment where novelties have been detected although they have not been changed.

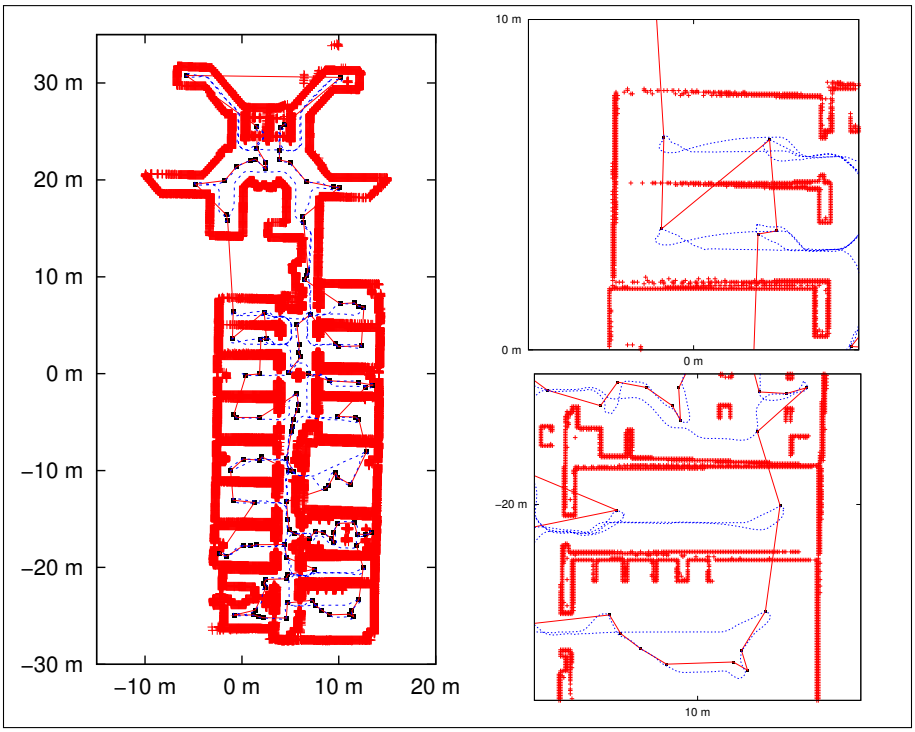
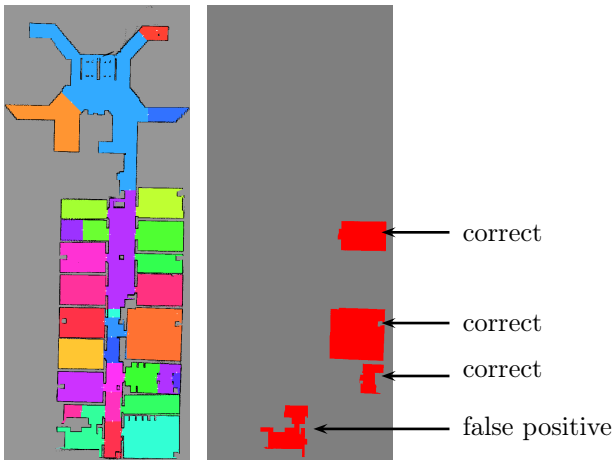


Figure 5.38: Autonomous inspection in a simulated environment. The robot inspects the AVZ building at the University of Osnabrück. Shown are the selected places, the determined order and the trajectory of the robot. Two regions that need to be traversed multiple times due to the not ideal order are shown in detail on the right.

What remains an issue for future work is to test the resulting system for autonomous inspection in real-world experiments. Furthermore, the means for detecting novelties as well as for selecting locations and planning a shortest tour can clearly be improved or replaced by more sophisticated algorithm. Here, the primary goal was to show that the presented set of algorithms can be used to autonomously inspect the workspace of a mobile service robot and update internal environment representations according to the detected changes.



(a) Segmentation (b) Novelties

Figure 5.39: Detecting novelties in a simulated environment. Shown is the segmented probabilistic reflection map (a) in the state before the inspection and the segments where more than 10 unexpected hits have been measured (b). Unexpected misses are not shown since not a single segment contained cells with more than 10 unexpected misses. The robot detected the changes in all three modified rooms and had one false positive in an unchanged room.

Chapter 6

Conclusion and Future Work

The goal of this thesis was to develop and implement a complete system for robotic exploration and inspection. Three major problems have been addressed: i) simultaneous localization and mapping (SLAM) to construct environment models while moving through the robot's workspace, ii) path planning and motion control to reach designated locations as well as iii) exploration and inspection to select the locations where the robot has to move to cover all environmental structures with its sensors. That is, instead of focussing on specific strategies, robotic exploration and inspection have been addressed as the integrated problems that they are. Primary design goals and requirements for the involved algorithms and the overall system were that they have to be robust especially against environment dynamics and that they have to be applicable online on a real robot operating in cluttered, dynamic and populated environments. The RoboCup@Home league, a contest for mobile service robots within the RoboCup competitions, has been used as the primary test bed.

SLAM has been addressed in terms of incremental range image registration. In incremental range image registration a newly acquired two-dimensional or three-dimensional range image is matched against a so for built model and then integrated into that to account for new information. For the actual matching or registration of range images, two algorithms have been presented. The Iterative Closest Point (ICP) algorithm determines the necessary transformation for mapping a new range image onto the so far built model by means of nearest neighbor search to determine correspondences and minimizing the distances between the correspondences. Different extensions to this algorithm, like for instance decaying thresholds for the distance between corresponding points and a reciprocal rejection of correspondence pairs, have been used to minimize the number of false correspondences and inaccuracies in individual registrations. As an alternative to the ICP, a matching algorithm based on normal distribution transforms (NDT) has been used. This algorithm avoids the computationally expensive nearest neighbor search in the ICP by transforming the map into a grid structure where each cell models the distribution of contained points. Both algorithms have been used to incrementally built point maps of the environment. To reduce the potentially large number of points in such a map, the concept of sparse point maps has been presented. In these maps new points are only added if there is no corresponding point in the model. The correspondences are determined just like in the ICP algorithm. Experiments have shown that, especially in the 2D case, the ICP-based approach for incrementally constructing sparse point maps outperforms the NDT-based registration in terms of accuracy and consistency of the resulting map. Furthermore, the expected lower runtimes of NDT-based matching could not be achieved due the fact that extensions to the algorithm, like alternative subdivision methods and linked cell structures, have been shown to be necessary in order to obtain correct registration results. Hence, it has been decided to base the final SLAM approach used on the real mobile robots on ICP-based registration and sparse point maps. As reported in (Holz et al., 2009), an issue remains in the concept of sparse point maps, namely that points are only added to the model but never removed. However, especially in dynamic environments this can lead to phantom effects – points in the map that do no longer correspond to objects in the real environment. This issue has been addressed by implementing probabilistic reflection maps as a second environment representation. These maps can be updated quite efficiently in addition to the sparse point maps used in the SLAM approach. It has been shown that removing points from the sparse point maps that fall in cells with low reflection probabilities efficiently removes (and avoids) phantom effects in the map.

Compared to e.g. Rao-Blackwellized particle filters (RBPFs), the resulting SLAM approach only carries a single hypothesis about the robot's trajectory and, thus, only a single map of the environment. Not surprisingly, the average processing time per scan of the ICP-based approach is considerably smaller than that of RBPFs. The ICP approach can process scans as they arrive in fast cycle times (75 Hz for the primary used SICK LMS 200), whereas recent RBPF implementations (Eliazar and Parr, 2004; Grisetti et al., 2007a) require, depending on the number of particles, several hundred milliseconds. A larger number of experiments in normally sized indoor environments have shown that maps resulting from the ICP-based SLAM approach do not rank behind those resulting from RBPFs in terms of accuracy and consistency.

The fundamental drawback of this SLAM approach is, however, that for localizing the robot a rough initial estimate is needed. During operation, this can be neglected once the robot has localized itself once. The pose determined in a previous registration together with odometric pose shift estimations forms an accurate estimation of the pose. Starting at an unknown position or with a wrong estimate (*kidnapped robot* problem) can, however, not be solved. That is, when starting the robot a rough initial estimate of its pose (position $\pm 2\,\mathrm{m}$, orientation $\pm 45°$) has to be provided. It is a matter of future work to address this issue e.g. by integrating concepts of adaptive Monte-Carlo localization (Fox, 2003; Pfaff et al., 2006; Ali and Mertsching, 2009). Another interesting entry point for future work is to completely integrate the ICP-based SLAM approach into a Rao-Blackwellized particle filter (Grisetti et al., 2007a). Possible extensions to the ICP-based SLAM approach for live-long mapping, e.g. explicitly modeling changes over time, are addressed in a follow-up project.

The second problem that has been addressed is that of navigating to designated positions and poses in the robot's workspace. This is not only a major precondition for robotic exploration and inspection but also a necessary capability of mobile service robots. Here two problems have distinguished: approaching a target pose in the robot's vicinity and navigation to a target location somewhere in the robot's workspace. For the first problem a simple motion controller has been presented that approaches a target position on a smooth curvature and under a designated target orientation. The second problem involves planning of shortest obstacle-free paths from the robot's current position to the target location as well as following planned paths. For following a planned path, a second motion controller has been presented that shows fast convergence for regaining a position on the path and high stability when following it. Both motion controllers are non-linear, derived from a Lyapunov-like synthesis and applicable to a wide range of non-holonomic robot platforms.

Path planning has been addressed for both types of environment representations constructed in the SLAM approach. To search for shortest paths in sparse point maps, a pruned Voronoi diagram is constructed using a highly efficient sweepline algorithm. The actual search in the resulting graph is carried out using $A^\star$ search with the Euclidian distance to the goal as an admissible heuristic function. As shown, the resulting paths are not optimal regarding the travelled distance but the safest by maintaining a maximum clearance to surrounding objects. This, however, causes that the robot moves a little bit shaky as it swerves to keep its position on the pruned Voronoi diagram. To address this issue different subsampling techniques have been applied to smooth the path and minimize abrupt changes in the robot's position and orientation.

In the case of probabilistic reflection maps, the grid structure can directly be used for planning shortest paths. Planning is decomposed into two phases. In the first phase all traversable cells are extracted from the reflection map by means of an efficient distance transform. In the second phase one of the presented path planning algorithms is used to search for the shortest path. A path search without a goal specification thereby allows to extract all reachable cells and to construct a reachability map encoding for every cell the shortest path to the robot's current position. The resulting paths are considerably shorter that those planned on the pruned Voronoi diagram. By regularizing the cost function of the search algorithm to incorporate additional costs when moving close to obstacles the robot still maintains a minimum clearance to surrounding obstacles. Especially the techniques for smoothing the planned path have to be seen as initial. Applying more sophisticated smoothing algorithms is a matter of future work. Furthermore, following a planned path and avoiding collisions is currently decoupled and handled by different reactive behaviors

implementing the presented algorithms. A possible extension to the overall system is to use a motion controller that integrates collision avoidance into path following (Lapierre et al., 2007). An according behavior could be simply achieved by applying a dynamic window approach and additionally plan the robot's velocities (Brock and Khatib, 1999; Maček et al., 2003; Berti et al., 2007).

Both, the SLAM approach as well as the different means for planning paths, have been excessively used by the mobile service robot "Johnny Jackanapes" participating in the RoboCup@Home league. The robot is able to navigate in dynamic environments and avoid collisions with suddenly appearing obstacles as well as to construct and update internal environment representations during operation. Videos showing all the performances of the robot in different RoboCup@Home tests during the world championship in Suzhou 2008 and the GermanOpen in Hannover 2009 are available at `http://www.b-it-bots.de/media`.

The third part of this thesis focused on the actual exploration and inspection strategies that guide the robot to exactly those places that need to be approached and examined to accomplish the respective task. These strategies make use of the SLAM component to construct and update an internal environment representation and the navigation abilities of the robot to reach selected locations. Three problems have been addressed: human-guided exploration, autonomous exploration and autonomous inspection.

In human-guided exploration a human guide shows the robot around and takes over the responsibility of the exploration strategy. Fundamental problems in this kind of exploration are detecting, tracking and following the human guide. Here, a very simple approach has been used. The robot examines an angular range in a 2D laser range or a virtual obstacle map (Holz et al., 2008) and assumes that the legs of the guide form the closest measurements in the examined range. Both the size of the angular range as well as its position depend on the last known pose of the guide. The robot is following the guide using a simple nonlinear motion controller adapting its translational and rotational velocities according to the distance and orientation to the guide. Although this approach is rather simple it has been shown that, besides some special situations, a human guide can be reliably followed. Here it should also be noted that developing a robust person detection and tracking mechanism would have been out of scope. Instead, the integration of more sophisticated algorithms for clustering range scans (Kluge et al., 2001) and detecting legs (Xavier et al., 2005) as well as for tracking the moving guide (Cui et al., 2008; Lee et al., 2008) is a matter of future work. Lee et al. (2008) consider, for example, geometric characteristics of human legs and hips to detect legs in a laser range scan and merge the detected legs to individual humans (Lee et al., 2006a). Humans in the scene are tracked using a human biped walking model. Extending the current approach with ideas like this is expected to make the people following behavior of the robot more robust especially in the presence of multiple moving people.

For the autonomous exploration of preliminary unknown environments, different strategies have been presented. These strategies are primarily operating on the probabilistic reflection maps since these distinguish between modeled free space and so far unmodeled space. Exploring closest frontiers directly guides the robot to the transitions between free and unknown regions in the map thereby causing that the robot successively visits and explores on unknown region after the other. In contrast to that, the decision-theoretic strategy draws a number of samples from the free space and evaluates the utility of each candidate for the robot. This utility is composed of the expected information gain and the cost involved in moving to a candidate. The robot then approaches the candidate with the highest utility. Both strategies have been shown to allow for completely exploring preliminary unknown regions in a reasonable amount of time and a reasonably small travelled distance. Experiments have further shown that exploring closest frontiers yielded the shortest trajectories thereby corresponding to and substantiating the results of Koenig et al. (2001) and Stachniss (2006, 2009). One issue in closest frontier exploration is that the robot might leave a room before it is fully explored causing that it has to return at a later stage in order to fully cover all environmental structures. This issue has been addressed by integrating a simple segmentation algorithm that segments the traversable space in the so far built model into individual rooms. Frontiers lying in the same segment as the robot are then preferred amongst others regardless of the involved travel distance. However, this segmentation algorithm does not yield an ideal

segmentation of the map. That is, the distances travelled by the robot can be shortened by using the segmentation but there is still place for improvements. It is also a matter of future work to classify segmented rooms e.g. into rooms, corridors and doorways (Martínez Mozos et al., 2007). Explicitly integrating segmentation and classification into an exploration strategy is expected to provide further improvements.

Autonomous inspection has been addressed in two phases. In the first phase, carried out when the inspection starts, the robot extracts traversable regions and determines a set of locations that will guide the robot through the entire workspace. The robot then determines a tour through all locations by solving a corresponding travelling salesman problem (TSP). In the second phase, the robot approaches one location after the other while continuously acquiring information about surrounding environmental structures and updating the map. For determining the locations a rather simple approach has been used. Instead of approximating a corresponding art gallery problem, the junction nodes from the Voronoi diagram are used as locations. These are distributed over the complete traversable space and all rooms in the environment. Moving to all these locations causes that the robot enters every room at least once. However, since the Voronoi diagram is pruned and only junction nodes are approached, small corners may not be examined during inspection. This is neglected in the context of this thesis and a matter of future work.

For determining the optimal order through the set of location two approaches have been used. The approach based on self-organizing maps (SOMs) approximates the travelling salesman problem and yields reasonably short, but not optimal tours. However, its advantage is that it is quite simple and can be easily implemented. Since a rather large number of training iterations is needed to achieve an approximate result, the SOM approach is rather ineffective compared to state of the art TSP solvers. As an alternative, the freely available TSP/Concorde solver (Applegate et al., 2007) has been used that determines a short tour in less computation time. In many cases this tour is optimal. The actual inspection strategy than simply approaches one selected location after another in the order given by the TSP solver. To detect changes in addition to inherently updating the map in the SLAM approach, a simple procedure has been presented that compares acquired range scans with the reflection probabilities of the sensed regions. Unexpected reflections of a laser beam as well as beams passing through cells with high reflection probability without being reflected are counted in a separate structure. Once these counts exceed some threshold, the corresponding map segment is marked as having changed. Experiments have shown that the resulting inspection approach allows a mobile robot to traverse and sense all regions in its workspace as well as to detect all changes that have taken place since the last update of the map.

Altogether, the presented algorithms form the desired complete system for robotic exploration and inspection. Although the involved approaches can be clearly improved, they provide a sufficient level of reliability and robustness as proven in many real-world experiments. Furthermore, all algorithms are applicable online and can be run in fast cycle times. The ICP-based SLAM approach for example can be used for pose tracking even without odometric estimates by registering every single range scan (at 75 Hz for the SICK LMS 200). All algorithms have been implemented to be self-sufficient. Interfaces as well as the underlying implementations of the algorithms use only standard C++ data types allowing for an easy integration of the algorithms into other robot control architectures.

Another possible improvement of the overall system is to extend it to full 3D exploration. Whereas the SLAM approach can already be used to construct 3D models and to localize the robot with six degrees of freedom, path planning as well as the exploration and inspection strategies do not work on 3D environment models. This has been neglected here since the virtual obstacle and structure maps provide efficient means for extracting relevant information from 3D range images. Hence, path planning as well as the exploration and inspection strategies make use of this additional information when a 3D sensor is used. Explicitly exploring a 3D model is, however, a matter of future work. Another issue being a matter of future work is to extend and apply the resulting system to exploration and inspection with a team of multiple mobile robots.

References

Aggarwal, A. (1984). *The art gallery theorem: its variations, applications and algorithmic aspects.* Ph. D. Thesis, John Hopkins University.

Ahmed, M. M., Yamani, S. M., and Farag, A. A. (1997). Fast Algorithm for Registration of Free-Form Curves and Surfaces. In *Proceedings of the IEEE International Conference on Image Processing (ICIP).*

Aicardi, M., Cannata, G., Casalino, G., and Indiveri, G. (2000). On the stabilization of the unicycle model projecting a holonomic solution. In *Proceedings of the International Symposium on Robotics and Applications (ISORA).*

Ali, I. and Mertsching, B. (2009). Surveillance System Using a Mobile Robot Embedded in a Wireless Sensor Network. In *Proceedings of the International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, Milan, Italy.

Allen, P., Reed, M., and Stamos, I. (1998). View Planning for Site Modeling. In *Proceedings of the DARPA Image Understanding Workshop (IUW).*

Alt, H. and Welzl, E. (1988). Visibility Graphs and Obstacle-Avoiding Shortest Paths. *Mathematical Methods of Operations Research*, 32(3):145–164.

Amigoni, F., Caglioti, V., and Galtarossa, U. (2004). A Mobile Robot Mapping System with an Information-Based Exploration Strategy. In *Proceedings of the First International Conference on Informatics in Control, Automation and Robotics (ICINCO).*

Amigoni, F. and Gallo, A. (2005). A Multi-Objective Exploration Strategy for Mobile Robots. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA).*

Andreasson, H. (2008). *Local Visual Feature based Localisation and Mapping by Mobile Robots.* Ph.d. thesis, Örebro University.

Angéniol, B., La, D., and Texier, J. Y. L. (1988). Self-organizing feature maps and the travelling salesman problem. *Neural Networks*, 1:289–293.

Applegate, D. L., Bixby, R. E., Chvátal, V., Cook, W., Espinoza, D., Goycoolea, M., and Helsgaun, K. (2009). Certification of an optimal TSP tour through 85900 cities. *Operations Research Letters*, 37(1):11–15.

Applegate, D. L., Bixby, R. E., Chvátal, V., and Cook, W. J. (2007). *The Traveling Salesman Problem: A Computational Study.* Princeton Univisity Press. ISBN 0-691-12993-2.

Appletgate, D., Bixby, R., Chvátal, V., and Cook, W. (1999). Finding tours in the TSP. Technical Report Report No. 99885, Institute for Discrete Mathematics, University of Bonn.

Arkin, E. M., Mitchell, J. S. B., and Piatko, C. D. (2003). Minimum-link watchman tours. *Information Processing Letters*, 86(4):203 – 207.

Arun, K. S., Huang, T. S., and Blostein, S. D. (1987). Least-squares fitting of two 3-d point sets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(5):698–700.

Avis, D. and Toussaint, G. (1981). An efficient algorithm for decomposing a polygon into star-shaped polygons. *Pattern Recognition*, 13:395–398.

Bailey, D. G. (2004). An efficient euclidean distance transform. In *Proceedings of the International Workshop on Combinatorial Image Analysis (IWCIA)*.

Bailey, T. and Durrant-Whyte, H. (2006). Simultaneous Localisation and Mapping (SLAM): Part II State of the Art. *Robotics and Automation Magazine*, September.

Balch, T. and Hybinette, M. (2000). Social potentials for scalable multi-robot formations. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Barber, C. B., Dobkin, D. P., and Huhdanpaa, H. (1996). The Quickhull Algorithm for Convex Hulls. *ACM Transactions on Mathematical Software*, 22:469–483.

Bay, H., Ess, A., Tuytelaars, T., and Gool, L. V. (2008). SURF: Speeded Up Robust Features. *Computer Vision and Image Understanding*, 110(3):346–359.

Bay, H., Tuytelaars, T., and Van Gool, L. (2006). SURF: Speeded-Up Robust Features. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

Beeson, P., MacMahon, M., Modayil, J., Provost, J., Savelli, F., and Kuipers, B. (2003). Exploiting Local Perceptual Models for Topological Map-Building. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

Beetz, M. (2003). A Roadmap for Research in Robot Planning. In *Technological Roadmap on AI Planning and Scheduling. PLANET: European Network of Excellence in AI Planning.*

Beetz, M., Arbuckle, T., Belker, T., Cremers, A. B., Schulz, D., Bennewitz, M., Burgard, W., Hähnel, D., Fox, D., and Grosskreutz, H. (2001). Integrated Plan-based Control of Autonomous Service Robots in Human Environments. *IEEE Intelligent Systems*, 16(5):56–65.

Bell, G. and Weir, M. (2004). Forward chaining for robot and agent navigation using potential fields. In *Proceedings of the Australasian conference on Computer science (ASC)*, pages 265–274.

Ben-Moshe, B., Hall-Holt, O., Katz, M. J., and Mitchell, J. S. B. (2004). Computing the visibility graph of points within a polygon. In *Proceedings of the Annual Symposium on Computational Geometry (SCG)*.

Bennewitz, M. (2004). *Mobile Robot Navigation in Dynamic Environments*. Ph. D. Thesis, University of Freiburg.

Bennewitz, M., Burgard, W., Cielniak, G., and Thrun, S. (2005). Learning motion patterns of people for compliant robot motion. *International Journal of Robotics Research*, 24(1):31–48.

Bennewitz, M., Burgard, W., and Thrun, S. (2002). Learning motion patterns of persons for mobile service robots. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Bennewitz, M., Stachniss, C., Behnke, S., and Burgard, W. (2009). Utilizing Reflection Properties of Surfaces to Improve Mobile Robot Localization. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Berti, H., Sappa, A., and Agamennoni, O. (2007). Autonomous robot navigation with a global and asymptotic convergence. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Besl, P. J. and McKay, N. D. (1992). A Method for Registration of 3-D Shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256.

Bhattacharya, P. and Gavrilova, M. L. (2007). Voronoi diagram in optimal path planning. In *Proceedings of the International Symposium on Voronoi Diagrams in Science and Engineering (ISVD)*.

Bhattacharya, P. and Gavrilova, M. L. (2008). Roadmap-Based Path Planning - Using the Voronoi Diagram for a Clearance-Based Shortest Path. *IEEE Robotics & Automation Magazine*, 15(2):58–66.

Biber, P. and Strasser, W. (2003). The normal distributions transform: a new approach to laser scan matching. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Biswas, R., Limketkai, B., Sanner, S., and Thrun, S. (2002). Towards Object Mapping in Dynamic Environments With Mobile Robots. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Bjorling-Sachs, I. (1998). Edge guards in rectilinear polygons. *Computational Geometry: Theory and Applications*, 11(2):111–123.

Blais, G. and Levine, M. D. (1995). Registering Multiview Range Data to Create 3D Computer Objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(8):820–824.

Borenstein, J. and Feng, L. (1996). Gyrodometry: A new method for combining data from gyros and odometry in mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Borenstein, J. and Koren, Y. (1991). The vector field histogram-fast obstacle avoidance for mobile robots. *IEEE Transaction on Robotics and Automation*, 7(3):278–288.

Borrmann, D., Elseberg, J., Lingemann, K., Nüchter, A., and Hertzberg, J. (2008). Globally Consistent 3D Mapping with Scan Matching. *Journal of Robotics and Autonomous Systems (JRAS)*, 56(2):130–142.

Bowyer, A. (1981). Computing Dirichlet Tessellations. *The Computer Journal by the British Computer Society*, 24(2):162–166.

Brock, O. and Khatib, O. (1999). High-speed navigation using the global dynamic window approach. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Brockett, R. W. (1983). Asymptotic Stability and Feedback Stabilization. In R. W. Brockett, R. S. M. and Sussmann, H. J., editors, *Differential Geometric Control Theory*, pages 181–191. Birkhauser, Boston.

Bruch, M. H., Gilbreath, G., Muelhauser, J., and Lum, J. (2002). Accurate waypoint navigation using non-differential GPS. In *Proceedings of the AUVSI Symposium on Unmanned Systems*.

Bui, H. H., Venkatesh, S., and West, G. (2001). Tracking and Surveillance in Wide-Area Spatial Environments Using the Abstract Hidden Markov Model. *International Journal of Pattern Recognition and Artificial Intelligence (IJPRAI)*, 15(1):177–196.

Burgard, W., Moors, M., Stachniss, C., and Schneider, F. (2005). Coordinated Multi-Robot Exploration. *IEEE Transactions on Robotics*, 21(3):376–386.

Canny, J. (1989). On the "Piano Movers" series by Schwartz, Sharir, and Ariel-Sheffi. *The robotics review 1*, pages 33–40.

Canudas-de-Wit, C., Khennouf, H., Samson, C., and Sordalen, O. (1993). Nonlinear Control Design of Mobile Robots. In Zheng, Y. F., editor, *Recent Trends in Mobile Robots*, pages 121–156. World Scientific Series in Robotics and Automated Systems.

Canudas-de-Wit, C., Ndoudi-Likoho, A. D., and Micaelli, A. (1997). Nonlinear Control for a Train-like Vehicle. *International Journal of Robotics Research*, 16(3):300–319.

Chazelle, B. (1991). Triangulating a simple polygon in linear time. *Discrete and Compututational Geometry*, 6(5):485–524.

Chekhlov, D., Pupilli, M., Mayol-Cuevas, W., and Calway, A. (2006). Real-Time and Robust Monocular SLAM Using Predictive Multi-resolution Descriptors. In *Proceedings of the 2nd International Symposium on Visual Computing*.

Chen, C., ping Hung, Y., and bo Cheng, J. (1998). A Fast Automatic Method for Registration of Partially-Overlapping Range Images. In *Proceedings of the International Conference on Computer Vision (CCV)*.

Chen, X., Pach, J., Szegedy, M., and Tardos, G. (2008). Delaunay graphs of point sets in the plane with respect to axis-parallel rectangles. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*.

Chen, Y. and Medioni, G. (1992). Object modelling by registration of multiple range images. *Image and Vision Computing*, 10(3):145–155.

Chetverikov, D., Stepanov, D., and Krsek, P. (2005). Robust Euclidean alignment of 3D point sets: the Trimmed Iterative Closest Point algorithm. *Image and Vision Computing*, 23(3):299–309.

Chetverikov, D., Svirko, D., and Stepanov, D. (2002). The Trimmed Iterative Closest Point Algorithm. In *Proceedings of the International Conference on Pattern Recognition*.

Chin, W. and Ntafos, S. (1986). Optimum Watchman Routes. In *Proceedings of the second annual symposium on Computational Geometry (SCG)*.

Chin, W.-P. and Ntafos, S. (1991). Shortest Watchman Routes in Simple Polygons. *Discrete Computational Geometry*, 6(1):9–31.

Choset, H., Lynch, K. M., Hutchinson, S., Kantor, G., Burgard, W., Kavraki, L., and Thrun, S. (2004). *Principles of Robot Motion*. MIT Press. ISBN: 0-262-03327-5.

Choset, H., Mirtich, B., and Burdick, J. (1997). Sensor Based Planning for a Planar Rod Robot: Incremental Construction of the Planar Rod-HGVG. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Choy, C. S.-T. and Siu, W.-C. (1995). New approach for solving the travelling salesman problem using self-organizing learning. In *Proceedings of the IEEE International Conference on Neural Networks*.

Chvátal, V. (1975). A Combinatorial Theorem in Plane Geometry. *Journal of Combinatorial Theory*, 18:39–41.

Clarke, G. and Wright, J. W. (1964). Scheduling of Vehicles from a Central Depot to a Number of Delivery Points. *Operations Research*, 12(4):568–581.

Comaniciu, D., Meer, P., and Member, S. (2002). Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5):603–619.

Connolly, C. (1985). The Determination of Next Best Views. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Coulter, R. C. (1992). Implementation of the Pure Pursuit Path Tracking Algorithm. Technical Report CMU-RI-TR-92-01, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

Craig, J. J. (1989). *Introduction to Robotics: Mechanics and Control*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. ISBN: 978-0201543612.

Csizmadia, G. and Tóth, G. (1998). Note on an Art Gallery Problem. *Computational Geometry. Theory and Applications*, 10(1):47–55.

Cui, J., Zha, H., Zhao, H., and Shibasaki, R. (2008). Multi-modal tracking of people using laser scanners and video camera. *Image and Vision Computing*, 26(2):240–252.

Danner, T. and Kavraki, L. E. (2000). Randomized Planning for Short Inspection Paths. In *Proceddings of the IEEE International Conference on Robotics and Automation (ICRA)*.

de Berg, M., Cheong, O., van Kreveld, M., and Overmars, M. (2008). *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin. 3rd rev. ed. 2008. ISBN: 978-3-540-77973-5.

Dias, M. B. and Stentz, A. (2000). A free market architecture for distributed control of a multirobot system. In *Proceedings of the 6th International Conference on Intelligent Autonomous Systems (IAS)*.

Diestel, R. (2005). *Graph Theory*. Springer-Verlag, Heidelberg. ISBN 3-540-26182-6.

Dijkstra, E. W. (1959). A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, 1(1):269–271.

Dold, C., Ripperda, N., and Brenner, C. (2007). Vergleich verschiedener Methoden zur automatischen Registrierung von terrestrischen Laserscandaten. In *"Photogrammetrie – Laserscanning – Optische 3D-Messtechnik", Proceedings of the Oldenburger 3D-Tage*.

Durrant-Whyte, H. and Bailey, T. (2006). Simultaneous Localisation and Mapping (SLAM): Part I The Essential Algorithms. *Robotics and Automation Magazine*, June.

Edelsbrunner, H., O'Rourke, J., and Welzl, E. (1984). Stationing guards in rectilinear art galleries. *Computer Vision Graphics and Image Processing*, 27(2):167–176.

Edlinger, T. and von Puttkamer, E. (1994). Exploration of an indoor-environment by an autonomous mobile robot. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Eidenbenz, S., Stamm, C., and Widmayer, P. (1998). Inapproximability of some Art Gallery Problems. In *Proceedings of the 10th Canadian Conference on Computational Geometry*.

Elfes, A. (1989). Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57.

Eliazar, A. and Parr, R. (2003). DP-SLAM: Fast, Robust Simulataneous Localization and Mapping without Predetermined Landmarks,. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI)*.

Eliazar, A. and Parr, R. (2004). DP-SLAM 2.0. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Everett, H. R. (1995). *Sensors for mobile robots: theory and application*. A. K. Peters, Ltd., Natick, MA, USA.

Fares, C. and Hamam, Y. (2005). Collision Detection for Rigid Bodies: A State of the Art Review. In *Proceedings of International Conference on Computer Graphics and Applications (GraphiCon)*.

Farris, J. S. (2003). *The Human-Web Interaction Cycle: A Proposed and Tested Framewor of Perception, Cognition, and Action on the Web*. Ph. D. Thesis, Kansas State University.

Favata, F. and Walker, R. (1991). A study of the application of kohonen-type neural networks to the travelling salesman problem. *Biological Cybernetics*, 64(6):463–468.

Fekete, S., Klein, R., and Nüchter, A. (2004). Searching with an autononmous robot. In *Proceedings of the 20th ACM Annual Symposium on Computational Geometry (SoCG '04)*.

Feldmar, J. and Ayache, N. (1994). Rigid, Affine and Locally Affine Registration of Free-Form Surfaces. Technical Report 2220, Institut National de Recherche end Informatique et Automatique (INRIA).

Feldmar, J. and Ayache, N. (1996). Rigid, Affine and Locally Affine Registration of Free-Form Surfaces. *International Journal of Computer Vision*, 18(2):99–119.

Felzenszwalb, P. F. and Huttenlocher, D. P. (2004). Distance transforms of sampled functions. Technical Report TR2004-1963, Cornell Computing and Information Science.

Fisk, S. (1978). A short proof of Chvatal's watchman theorem. *Journal of Combinatorial Theory*, 24:374.

Fortune, S. J. (1987). A Sweepline Algorithm for Voronoi Diagrams. *Algorithmica*, pages 153–174.

Fortune, S. J. (1998). Voronoi Diagrams and Delaunay Triangulations. pages 377–388. CRC Press, New York.

Fox, D. (2003). Adapting the Sample Size in Particle Filters Through KLD-Sampling. *International Journal of Robotics Research*, 22.

Fox, D., Burgard, W., and Thrun, S. (1997). The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1):23–33.

Frank, B., Stachniss, C., Schmedding, R., Burgard, W., and Teschner, M. (2009). Real-world Robot Navigation amongst Deformable Obstacles. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Freda, L., Loiudice, F., and Oriolo, G. (2006). A Randomized Method for Integrated Exploration. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Frese, U. (2004). *An $O(\log n)$ Algorithm for Simultaneous Localization and Mapping of Mobile Robots in Indoor Environments*. Ph. D. Thesis, University Erlangen-Nürnberg.

Frese, U. (2007). Efficient 6-DOF SLAM with Treemap as a Generic Backend. In *Proceedings of the Internation Conference on Robotics and Automation (ICRA)*.

Friedman, J. H., Bentley, J. L., and Finkel, R. A. (1977). An Algorithm for Finding Best Matches in Logarithmic Expected Time. *ACM Transactions on Mathematical Software*, 3(3):209–226.

Fuchs, S. and Hirzinger, G. (2008). Extrinsic and Depth Calibration of ToF-Cameras. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*.

Fuchs, S. and May, S. (2007). Calibration and Registration for Precise Surface Reconstruction. In *Proceedings of the DAGM Dynamic 3D Imaging Workshop (Dyn3D)*.

Furcy, D. and Koenig, S. (2000). Speeding up the convergence of real-time search. In *Proceedings of the National Conference on Artificial Intelligence and Conference on Innovative Applications of Artificial Intelligence*.

Fusiello, A., Castellani, U., Ronchetti, L., and Murino, V. (2002). Model acquisition by registration of multiple acoustic range views. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

Garrido, S., Moreno, L., Abderrahim, M., and Martin, F. (2006). Path Planning for Mobile Robot Navigation using Voronoi Diagram and Fast Marching. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Gavrila, D. and Philomin, V. (1999). Real-time object detection for "smart" vehicles. In *Proceedings of the IEEE International Conference on Computer Vision*.

Ge, S., Ma, H., and Lum, K.-Y. (2007). Detectability in games of pursuit evasion with antagonizing players. In *Proceedings of the IEEE International Conference on Decision and Control*.

Geraerts, R. and Overmars, M. H. (2006). Sampling and Node Adding in Probabilistic Roadmap Planners. *Journal of Robotics and Autonomous Systems*, 54:165–173.

Geraerts, R. and Overmars, M. H. (2007). Creating High-quality Paths for Motion Planning. *International Journal of Robotics Research*, 26(8):845–863.

Gerkey, B. P., Thrun, S., and Gordon, G. (2006). Visibility-based pursuit-evasion with limited field of view. *International Journal of Robotics Research*, 25(4):299–315.

Ghosh, S. (1987). Approximation algorithms for art gallery problems. In *Proceedings of the Canadian Information Processing Society Congress*.

Ghosh, S. K. (2007). *Visibility Algorithms in the Plane*. Cambridge University Press. ISBN 978-0521875745.

Godin, G., Rioux, M., and Baribeau, R. (1994). Three-dimensional registration using range and intensity information. *Proceedings of SPIE, Videometrics III*, 2350(1):279–290.

González-Baños, H. H. and Latombe, J. (2001a). A Randomized Art-Gallery Algorithm for Sensor Placement. In *Proceedings of the seventeenth annual symposium on Computational geometry*.

González-Baños, H. H. and Latombe, J. (2001b). Robot Navigation for Automatic Model Construction Using Safe Regions. In *International Symposium on Experimental Robotics VII (ISER)*, pages 405–415, London, UK. Springer-Verlag. ISBN 3-540-42104-1.

González-Baños, H. H. and Latombe, J. (2002). Navigation Strategies for Exploring Indoor Environments. *International Journal of Robotics Research*, 21:829–848.

González-Baños, H. H., Lee, C.-Y., and Latombe, J.-C. (2002). Real-time Combinatorial Tracking of a Target Moving Unpredictably among Obstacles. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Gossage, M., New, A. P., and Cheng, C. K. (2006). *Distributed Autonomous Robotic Systems 7*, chapter Frontier-Graph Exploration for Multi-robot Systems in an Unknown Indoor Environment, pages 51–60. Springer Japan.

Graham, R. L. (1972). An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set. *Information Processing Letters*, 1(4):132–133.

Greenspan, M. and Godin, G. (2001). A Nearest Neighbor Method for Efficient ICP. In *Proceedings of the International Conference on 3-D Digital Imaging and Modeling (3DIM)*.

Greenspan, M. and Yurick, M. (2003). An Approximate K-D Tree Search for Efficient ICP. In *Proceedings of the International Conference on 3-D Digital Imaging and Modeling (3DIM)*.

Grisetti, G., Stachniss, C., and Burgard, W. (2005). Improving Grid-based SLAM with Rao-Blackwellized Particle Filters by Adaptive Proposals and Selective Resampling. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Grisetti, G., Stachniss, C., and Burgard, W. (2007a). Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters. *IEEE Transactions on Robotics*, 23(1):34–46.

Grisetti, G., Stachniss, C., Grzonka, S., and Burgard, W. (2007b). A Tree Parameterization for Efficiently Computing Maximum Likelihood Maps using Gradient Descent. In *Proceedings of Robotics: Science and Systems*.

Grötschel, M. (1977). *Polyedrische Charakterisierungen kombinatorischer Optimierungsprobleme*, volume 36 of *Mathematical Systems in Economics*. Verlag Anton Hain, Meisenheim am Glan.

Grötschel, M. (1980). On the symmetric travelling salesman problem: solution of a 120-city problem. *Mathematical Programming Study*, 12:61–77.

Guibas, L. J., Knuth, D. E., and Sharir, M. (1990). Randomized incremental construction of delaunay and voronoi diagrams. In *Proceedings of the International Colloquium on Automata, Languages and Programming*.

Guidi, G., Beraldin, J. A., and Atzeni, C. (2004). High-accuracy 3D modeling of cultural heritage: the digitizing of Donatelloapos's "Maddalena". *IEEE Transactions on Image Processing*, 13(3):370–280.

Györi, E., Hoffmann, F., Kriegel, K., and Shermer, T. (1996). Generalized guarding and partitioning for rectilinear polygons. *Computational Geometry: Theory and Applications*, 6(1):21–44.

Haegele, M., Neugebauer, J., and Schraft, R. (2001). From Robots to Robot Assistants. In *Proceedings of the 32nd International Symposium on Robotics (ISR)*.

Hähnel, D. (2005). *Mapping with Mobile Robots*. Ph. D. Thesis, University of Freiburg.

Hammar, M. and Nilsson, B. J. (1997). Concerning the Time Bounds of Existing Shortest Watchman Route Algorithms. In *Fundamentals of Computation Theory*, pages 210–221.

Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107.

Haykin, S. (1998). *Neural Networks. A Comprehensive Foundation*. Prentice Hall. ISBN 978-0139083853.

Hellström, T. and Ringdahl, O. (2006). Follow the Past - a Path Tracking Algorithm for Autonomous Vehicles. *International Journal of Vehicle Autonomous Systems*, 4(2-4):216–224.

Hoffmann, F., Icking, C., Klein, R., and Kriegel, K. (2002). The Polygon Exploration Problem. *SIAM Journal on Computing*, 31(2):577–600.

Hoffmann, F., Kaufmann, M., and Kriegel, K. (1991). The Art Gallery Theorem for Polygons with Holes. In *Proceedings of the 32nd annual symposium on Foundations of computer science*, pages 39–48.

Holz, D. (2006). Kontinuierliche Umgebungskartographie mittels 3D-Laserscanner auf autonomen mobilen Robotern. Diploma thesis, University of Applied Sciences Cologne, Cologne, Germany.

Holz, D. (2007). Competing in RoboCup@Home: Prerequisites for Acting in the real Real-World. Technical Report R&D1, Bonn-Rhein-Sieg University of Applied Sciences.

Holz, D. (2009). Effiziente 2D-Navigation für Mobile Service Roboter. In *Proceedings of the 11. Fachwissenschaftlicher Informatik-Kongress, Lecture Notes in Informatics (LNI)*, Series of the Gesellschaft für Informatik (GI).

Holz, D., Kraetzschmar, G. K., and Rome, E. (2009). Robust and Computationally Efficient Navigation in Domestic Environments. In *Proceedings of the 13th RoboCup International Symposium*. To appear.

Holz, D., Lörken, C., and Surmann, H. (2008). Continuous 3D Sensing for Navigation and SLAM in Cluttered and Dynamic Environments. In *Proceedings of the International Conference on Information Fusion (FUSION)*.

Horn, B. K. P. (1987). Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America A*, 4(4):629–642.

Hsu, D., Latombe, J., Motwani, R., and Kavraki, L. (1999). Capturing the connectivity of high-dimensional geometric spaces by parallelizable random sampling techniques. In Pardalos, P. and Rajasekaran, S., editors, *Advances in randomized parallel computing*, pages 159–182. Kluwer Academic Publishers, Boston, MA.

Hsu, D., Latombe, J.-C., and Kurniawati, H. (2006). On the Probabilistic Foundations of Probabilistic Roadmap Planning. *International Journal of Robotics Research*, 25(7):627–643.

Hua, K., Foroosh, H., Leonessa, A., Qu, Z., Harper, D., Pillat, R., Norvell, D., Santiago, S., Collins, T., Stein, G., Stickler, S., Decker, G., Andres, R., Shen, Y., Chen, H., and Xie, F. (2005). Team Description Paper University of Central Florida:Technical Paper DARPA Grand Challenge 2005. In *Proceedings of the 2005 DARPA Grand Challenge*.

Huhle, B., Magnusson, M., er, W. S., and Lilienhal, A. J. (2008). Registration of Colored 3D Point Clouds with a Kernel-Based Extension to the Normal Distributions Transform. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Icking, C. and Klein, R. (1995). Searching for the kernel of a polygon - a competitive strategy. In *Symposium on Computational Geometry*, pages 258–266.

Indiveri, G. (1999). Kinematic Time-invariant Control of a 2*d* Nonholonomic Vehicle. In *Proceedings of the 38th Conference on Decision and Control (CDC'99)*.

Indiveri, G. and Corradini, M. L. (2004). Switching linear path following for bounded curvature car-like vehicles. In *Proceedings of the IFAC Symposium on Intelligent Autonomous Vehicles (IAV04)*.

Isaacs, R. (1965). *Differential Games: A Mathematical Theory with Applications to Warfare and Pursuit, Control and Optimization*. John Wiley & Sons, New York.

Jankó, Z., Chetverikov, D., and Ekárt, A. (2007). Using Genetic Algorithms in Computer Vision: Registering Images to 3D Surface Model. *Acta Cybernetica*, 18(2):193–212.

Jensen, B., Weingarten, J., Kolski, S., and Siegwart, R. (2005). Laser Range Imaging Using Mobile Robots: From Pose Estimation to 3d-Models. In *Proceedings of the 1st Range Imaging Research Day*.

Kahn, J., Klawe, M., and Kleitman, D. (1983). Traditional Galleries Require Fewer Watchmen. *SIAM Journal on Matrix Analysis and Applications*, 4(2):194–206.

Kanehara, M., Kagami, S., Kuffner, J., Thompson, S., and Mizoguhi, H. (2007). Path shortening and smoothing of grid-based path planning with consideration of obstacles. In *Proceedings of the IEEE International Conference on Systems, Manand Cybernetics (SMC)*.

Katevas, N. I., Tzafestas, S. G., and c. G. Pnevmatikatos (1998). The Approximate Cell Decomposition with Local Node Refinement Global Path Planning Method: Path Nodes Refinement and Curve Parametric Interpolation. *Journal of Intelligent and Robotic Systems*, 22(3-4):289–314.

Kavraki, L. E., Latombe, J.-C., Motwani, R., and Raghavan, P. (1995). Randomized query processing in robot path planning. In *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*.

Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotic Research*, 5(1):90–98.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by Simulated Annealing. *Science*, 220(4598):671–680.

Klatzky, R. L. (1998). Allocentric and egocentric spatial representations: Definitions, distinctions, and interconnections. In *Spatial Cognition, An Interdisciplinary Approach to Representing and Processing Spatial Knowledge, Lecture Notes in Computer Sciences*.

Kluge, B., Köhler, C., and Prassler, E. (2001). Robust Tracking of Multiple Moving Objects with a Laser Range Finder. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Ko, J., Stewart, B., Fox, D., Konolige, K., and Limketkai, B. (2003). A practical, decision-theoretic approach to multi-robot mapping and exploration. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Ko, N. Y. and Simmons, R. (1998). The lane-curvature method for local obstacle avoidance. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Koenig, N. (2007). Toward Real-time Human Detection and Tracking in Diverse Environments. In *Proceedings of the IEEE International Conference on Development and Learning (ICDL)*.

Koenig, S. and Likhachev, M. (2002). Improved fast replanning for robot navigation in unknown terrain. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Koenig, S., Tovey, C., and Halliburton, W. (2001). Greedy Mapping of Terrain. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Koh, K. C. and Cho, H. S. (1999). A smooth path tracking algorithm for wheeled mobile robots with dynamic constraints. *Journal of Intelligent and Robotic Systems*, 24(4):367–385.

Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43:59–69.

Kolling, A. and Carpin, S. (2007). Detecting intruders in complex environments with limited range mobile sensors. In *Robot Motion and Control 2007. K. Kozlowski (Ed.), Lecture Notes in Control and Information Sciences (LNCIS) Vol. 360*, pages 417–246. Springer.

Koveos, Y., Panousopoulou, A., Kolyvas, E., Reppa, V., Koutroumpas, K., Tsoukalas, A., and Tzes, A. (2007). An integrated power aware system for robotic-based lunar exploration. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Kuffner, J. J. and LaValle, S. M. (2000). RRT-connect: An efficient approach to single-query path planning. In *Proceedings of the IEEE Internationall Conference on Robotics and Automation (ICRA)*.

Kuhn, H. W. (1955). The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(2):83–97.

Kuipers, B. and Byun, Y.-T. (1991). A Robot Exploration and Mapping Strategy based on a Semantic Hierarchy of Spatial Representations. *Journal of Robotics and Autonomous Systems*, 8:47–63. Reprinted in Walter Van de Velde (ed.), Towards Learning Robots, Bradford/MIT Press, 1993.

Kuipers, B., Modayil, J., Beeson, P., MacMahon, M., and Savelli, F. (2004). Local metrical and global topological maps in the hybrid spatial semantic hierarchy. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Kunze, L., Lingemann, K., Nüchter, A., and Hertzberg, J. (2007). Salient Visual Features to Help Close the Loop in 6D SLAM. In *Proceedings of the ICVS Workshop on Computational Attention and Applications (WCAA)*.

Lapierre, L., Zapata, R., and Lepinay, P. (2007). Combined path-following and obstacle avoidance control of a wheeled robot. *The International Journal of Robotics Research*, 26(4):361–375.

Latombe, J.-C. (1991). *Robot Motion Planning*. Kluwer Academic Publishers, Norwell, MA, USA. ISBN: 978-0792392064.

LaValle, S. M. (2006). *Planning Algorithms*. Cambridge University Press, Cambridge, U.K. Available at http://planning.cs.uiuc.edu/.

Lavalle, S. M., Gonz'alez-Baños, H. H., Becker, C., and claude Latombe, J. (1997). Motion Strategies for Maintaining Visibility of a Moving Target. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Lee, D. T. and Lin, A. K. (1986). Computational complexity of art gallery problems. *IEEE Transactions on Information Theory*, 32(2):276–282.

Lee, J. H., Abe, K., Tsubouchi, T., Ichinose, R., Hosoda, Y., and Ohba, K. (2008). Collision-free navigation based on people tracking algorithm with biped walking model. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Lee, J. H., Tsubouchi, T., Yamamoto, K., and Egawa, S. (2006a). People Tracking Using a Robot in Motion with Laser Range Finder. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Lee, Y.-S., Koo, H.-S., and Jeong, C.-S. (2006b). A straight line detection using principal component analysis. *Pattern Recognition Letters*, 27(14):1744–1754.

Leonard, J. and Feder, H. (1999). A computationally efficient method for large-scale concurrent mapping and localization. In J. Hollerbach, D. K., editor, *International Symposium on Robotics Research*.

Lindström, M. and Eklundh, J.-O. (2001). Detecting and tracking moving objects from a mobile platform using a laser range scanner. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Lingemann, K., Nüchter, A., Hertzberg, J., and Surmann, H. (2005a). About the Control of High Speed Mobile Indoor Robots. In *Proceedings of the Second European Conference in Mobile Robotics*.

Lingemann, K., Nüchter, A., Hertzberg, J., and Surmann, H. (2005b). High-Speed Laser Localization for Mobile Robots. *Robotics and Autonomous Systems*, 51(4):275–296.

Lörken, C. (2007). Introducing Affordances into Robot Task Execution. In Kühnberger, K.-U., König, P., and Ludewig, P., editors, *Publications of the Institute of Cognitive Science (PICS)*, volume 2-2007. University of Osnabrück, Osnabrück, Germany. ISSN 1610-5389.

Lorusso, A., Eggert, D. W., and Fisher, R. B. (1995). A Comparison of Four Algorithms for estimating 3-D Rigid Transformations. In *Proceedings of the British conference on Machine vision BMVC*, pages 237–246.

Lowe, D. G. (1999). Object recognition from local scale-invariant features. In *Proceedings of the IEEE International Conference on Computer Vision*.

Lowe, D. G. (2004). Distinctive Image Features from Scale-Invariant Keypoints. *Distinctive Image Features from Scale-Invariant Keypoints*, 60(2):91–110.

Lu, F. and Milios, E. (1997a). Globally consistent range scan alignment for environment mapping. *Autonomous Robots*, 4:333–349.

Lu, F. and Milios, E. (1997b). Robot Pose Estimation in Unknown Environments by matching 2D Range Scans. *Journal of Intelligent and Robotic Systems*, 18:249–275.

Mabrouk, M. H. and McInnes, C. R. (2008). Solving the potential field local minimum problem using internal agent states. *Robotics and Autonomous Systems*, 56(12):1050–1060.

Maček, K., Petrović, I., and Ivanjko, E. (2003). An approach to motion planning of indoor mobile robots. In *Proceedings of the IEEE International Conference on Industrial Technology*.

Magnusson, M. (2006). *3D Scan Matching for Mobile Robots with Application to Mine Mapping.* Licentiate Thesis, Örebro University.

Magnusson, M., Andreasson, H., Nüchter, A., and Lilienthal, A. J. (2009a). Appearance-Based Loop Detection from 3D Laser Data Using the Normal Distributions Transform. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA).*

Magnusson, M. and Duckett, T. (2005). A Comparison of 3D Registration Algorithms for Autonomous Underground Mining Vehicles. In *Proceedings of the European Conference on Mobile Robotics (ECMR).*

Magnusson, M., Lilienthal, A., and Duckett, T. (2007). Scan Registration for Autonomous Mining Vehicles Using 3D-NDT. *Journal of Field Robotics*, 24(10):803–827.

Magnusson, M., Nüchter, A., Lörken, C., Lilienthal, A. J., and Hertzberg, J. (2009b). Evaluation of 3d registration reliability and speed — a comparison of icp and ndt. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA).*

Makarenko, A., Williams, S., Bourgault, F., and Durrant-Whyte, H. (2002). An Experiment in Integrated Exploration. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Lausanne, Switzerland.

Mäkelä, H. (2001). *Outdoor Navigation of Mobile Robots.* Ph. D. Thesis, Helsinki University of Technology.

Marchand, J. (1988). The algorithm by Schwartz, Sharir and Collins on the piano mover's problem. In *Geometry and Robotics*, pages 49–66.

Martin, M. C. and Moravec, H. (1996). Robot evidence grids. Technical Report CMU-RI-TR-96-06, Robotics Institute, Carnegie Mellon University.

Martínez Mozos, O., Triebel, R., Jensfelt, P., Rottmann, A., and Burgard, W. (2007). Supervised semantic labeling of places using information extracted from sensor data. *Robotics and Autonomous Systems*, 55(5):391–402.

Masehian, E. and Amin-Naseri, M. R. (2004). A voronoi diagram-visibility graph-potential field compound algorithm for robot path planning. *Journal of Robotic Systems*, 21(6):275–300.

Masuda, T., Sakaue, K., and Yokoya, N. (1996). Registration and Integration of Multiple Range Images for 3-D Model Construction. In *Proceedings of the International Conference on Pattern Recognition.*

Maver, J. and Bajcsy, R. (1993). Occlusions as a Guide for Planning the Next View. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(5):417–433.

May, S., Droeschel, D., Holz, D., Fuchs, S., and Nüchter, A. (2009). Robust 3D-Mapping with Time-of-Flight Cameras. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).* To appear.

May, S., Werner, B., Surmann, H., and Pervölz., K. (2006). 3D time-of-flight cameras for mobile robotics. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).*

Mei, Y., Lu, Y.-H., Lee, C., and Hu, Y. (2006). Energy-efficient mobile robot exploration. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA).*

Meijster, A., Roerdink, J., and Hesselink, W. (2000). *Mathematical Morphology and its Applications to Image and Signal Processing*, chapter A general algorithm for computing distance transforms in linear time, pages 331–340. Kluwer.

Merriam-Webster (2009). Online dictionary. `http://www.merriam-webster.com`.

Meza, J. C., Oliva, R. A., Hough, P. D., and Williams, P. J. (2007). OPT++: An object-oriented toolkit for nonlinear optimization. *ACM Transactions on Mathematical Software (TOMS)*, 33(2):12. Article 12, 27 pages.

Minguez, J. (2005). Metric-Based Scan Matching Algorithms for Mobile Robot Displacement Estimation. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Mitchell, J. S. B. (1998). *Handbook of Computational Geometry*, chapter Geometric shortest paths and network optimization. Elsevier Science, Amsterdam.

Montemerlo, M., Thrun, S., Koller, D., and Wegbreit, B. (2002a). FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *Proceedings of the AAAI National Conference on Artificial Intelligence*.

Montemerlo, M., Thrun, S., Koller, D., and Wegbreit, B. (2003). FastSLAM 2.0: An Improved Particle Filtering Algorithm for Simultaneous Localization and Mapping that Provably Converges. In *Proceedings of the International Conference on Artificial Intelligence (IJCAI)*.

Montemerlo, M., Thrun, S., and Whittaker, W. (2002b). Conditional particle filters for simultaneous mobile robot localization and people-tracking. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Montesano, L., Minguez, J., and Montano, L. (2005). Probabilistic scan matching for motion estimation in unstructured environments. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Moorehead, S., Simmons, R., and Whittaker, W. R. L. (2001). Autonomous Exploration Using Multiple Sources of Information. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Moors, M. (2008). *Multi Robot Intruder Search*. Ph. D. Thesis, University of Bonn.

Moratz, R. and Wallgrün, J. O. (2003). Propagation of Distance and Orientation Intervals. In *Proceedings of 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Moravec, H. and Elfes, A. E. (1985). High Resolution Maps from Wide Angle Sonar. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Moravec, H. P. (1989). Sensor fusion in certainty grids for mobile robots. *Sensor Devices and Systems for Robotics*, pages 253–276.

Mount, D. and Arya, S. (1997). ANN: A library for approximate nearest neighbor searching. In *Proceedings of the 2nd Annual Fall Workshop on Computational Geometry*.

Muja, M. and Lowe, D. G. (2009). Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration. In *Proceedings of the International Conference on Computer Vision Theory and Applications (VISAPP)*.

Müller, J., Stachniss, C., Arras, K. O., and Burgard, W. (2008). Socially Inspired Motion Planning for Mobile Robots in Populated Environments. In *Proceedings of the International Conference on Cognitive Systems (CogSys)*.

Musser, D. R. (1997). Introspective sorting and selection algorithms. *Software Practice and Experience*, 27(8):983–993.

Nardi, D., Dessimoz, J.-D., Dominey, P. F., Iocchi, L., Rybski, P. E., Savage, J., Schiffer, S., Wisspeintner, T., van der Zant, T., and Yazdani, A. (2008). RoboCup@Home – Rules and Regulations. Available online at http://www.ai.rug.nl/robocupathome/documents/rulebook2008.pdf.

Nardi, D., Dessimoz, J.-D., Savage, J., Iocchi, L., Rybski, P., Dominey, P. F., Schiffer, S., Wisspeintner, T., and van der Zant, T. (2007). Robocup@home – rules and regulations. Available online at http://www.ai.rug.nl/robocupathome/documents/rulebook.pdf.

Neisser, U. (1976). *Cognition and Reality: Principles and Implications of Cognitive Psychology*. W. H. Freeman & Co (Sd). ISBN: 978-0716704775.

Newman, P., Bosse, M., and Leonard, J. (2003). Autonomous feature-based exploration. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Newman, P. and Ho, K. (2005). SLAM Loop Closing with Visually Salient Features. In *Proceedings of the IEEE nternational Conference on Robotics and Automation (ICRA)*.

Nguyen, V., A., Martinelli, Tomatis, N., and Siegwart, R. (2005). A Comparison of Line Extraction Algorithms using 2D Laser Rangefinder for Indoor Mobile Robotics. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Nilsson, N. J. (1969). Research on intelligent automata. Technical report, Stanford Research Institute.

Nüchter, A., Lingemann, K., and Hertzberg, J. (2007a). Cached k-d Tree Search for ICP Algorithms. In *Proceedings of the IEEE International Conference on Recent Advances in 3D Digital Imaging and Modeling (3DIM)*.

Nüchter, A., Lingemann, K., Hertzberg, J., and Surmann, H. (2007b). 6D SLAM – 3D Mapping Outdoor Environments. *Journal of Field Robotics (JFR), Special Issue on Quantitative Performance Evaluation of Robotic and Intelligent Systems*, 24(8–9):699–722.

Nüchter, A., Lingemann, K., Hertzberg, J., Surmann, H., Pervölz, K., Hennig, M., Tiruchinapalli, K. R., Worst, R., and Christaller, T. (2005a). Mapping of Rescue Environments with Kurt3D. In *Proceedings of the International Workshop on Safty, Security and Rescue Robotics (SSRR)*.

Nüchter, A., Lingemann, K., Hertzberg, J., Wulf, O., Wagner, B., and Surmann, H. (2005b). 3D Mapping with Semantic Knowledge. In *RoboCup International Symposium 2005: Robot Soccer World Cup IX*, pages 335–346, Osaka, Japan.

Nüchter, A., Surmann, H., and Hertzberg, J. (2003a). Planning Robot Motion for 3D Digitalization of Indoor Environments. In *Proceedings of the 11th International Conference on Advanced Robotics*, pages 222–227, Coimbra, Portugal.

Nüchter, A., Surmann, H., Lingemann, K., and Hertzberg, J. (2003b). Consistent 3D Model Construction with Autonomous Mobile Robots. In *KI 2003: Advances in Artificial Intelligence, Proceedings of the 26th Annual German Conference on AI*.

Olson, E., Leonard, J., and Teller, S. (2006). Fast Iterative Alignment of Pose Graphs with Poor Estimates. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Olson, E., Leonard, J., and Teller, S. (2007). Spatially-Adaptive Learning Rates for Online Incremental SLAM. In *Proceedings of Robotics: Science and Systems*.

O'Rourke, J. (1987). *Art Gallery Theorems and Algorithms*. Oxford University Press, Inc., New York, NY, USA.

O'Rourke, J. (1994). *Computational Geometry in C*. Cambridge University Press.

O'Rourke, J. and Supowit, K. (1983). Some NP-hard polygon decomposition problems. *IEEE Transactions on Information Theory*, 29(2):181–190.

Pajdla, T. and Gool, L. V. (1995). Matching of 3-D Curves using Semi-differential Invariants. In *Proceedings of the IEEE International Conference on Computer Vision*.

Pandey, S., Krishna, M., and Hexmoor, H. (2007). Feature Chain Based Occupancy Grid SLAM for Robots Equipped with Sonar Sensors. In *Proceedings of IEEE Knowledge Intensive Multiagent Systems (KIMAS)*, pages 283–288.

Parker, L. (1997). Cooperative Motion Control for Multi-Target Observation. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Parsons, T. D. (1976). Pursuit-evasion in a graph. In *Theory and Applications of Graphs, Y. Alavi and D. Lick (Eds), Lecture Notes in Mathematics*, pages 426–441. Springer.

Pathak, K., Birk, A., and Poppinga, J. (2008). Sub-pixel depth accuracy with a time of flight sensor using multimodal gaussian analysis. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Pauly, M., Surmann, H., Finke, M., and Liang, N. (1998). Real-time object detection for autonomous robots. *Informatik Aktuell. Autonome Mobile Systeme, 14. Fachgesprch*, pages 57–64.

Pébay, P. (2008). Formulas for Robust, One-Pass Parallel Computation of Covariances and Arbitrary-Order Statistical Moments. Technical Report SAND2008-6212, SANDIA National Laboratories.

Peikert, R. and Sigg, C. (2006). Optimized Bounding Polyhedra For GPU-Based Distance Transform. In Bonneau, G.-P., Ertl, T., and Nielson, G., editors, *Scientific Visualization: The Visual Extraction of Knowledge from Data*, pages 65–77. Springer.

Pfaff, P., Burgard, W., and Fox, D. (2006). Robust Monte-Carlo Localization using Adaptive Likelihood Models. In *Proceedings of the European Robotics Symposium (EUROS-06)*.

Pfaff, P., Triebel, R., and Burgard, W. (2007). An Efficient Extension to Elevation Maps for Outdoor Terrain Mapping and Loop Closing. *International Journal of Robotics Research*, 26(2):217–230.

Pocchiola, M. and Vegter, G. (1995). Computing the visibility graph via pseudo-triangulations. In *Proceedings of the Annual Symposium on Computational Geometry (SCG)*.

Pollack, M. E., Engberg, S., Matthews, J. T., Thrun, S., Brown, L., Colbry, D., Orosz, C., Peintner, B., Ramakrishnan, S., Dunbar-Jacob, J., McCarthy, C., M. Montemerlo, J. P., and Roy, N. (2002). Pearl: A Mobile Robotic Assistant for the Elderly. In *Proceedings of the AAAI Workshop on Automation as Eldercare*.

Powell, M. J. D. (2004). The NEWUOA software for unconstrained optimization without derivatives. Technical Report DAMTP 2004/NA08, Cambridge Centre for Mathematical Sciences (CMS).

Pulli, K. (1999). Multiview Registration for Large Data Sets. In *Proceedings of the Second International Conference on 3D Digital Imaging and Modeling (3DIM'99)*.

Ratering, S. and Gini, M. (1993). Robot navigation in a known environment with unknown moving obstacles. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Reinelt, G. (1991). Tsplib - a traveling salesman problem library. *ORSA Journal on Computing*, 3:376–384.

Ripperda, N. and Brenner, C. (2005). Marker-Free Registration of Terrestrial Laser Scans Using the Normal Distribution Transform. In *Proceedings of the ISPRS Working Group V/4 Workshop 3D-ARCH 2005: "Virtual Reconstruction and Visualization of Complex Architectures"*.

Robertson, C. and Fisher, R. B. (2002). Parallel evolutionary registration of range data. *Computer Vision and Image Understanding*, 87(1-3):39–50.

Rosenfeld, A. and Pfaltz, J. L. (1968). Distance functions on digital pictures. *Pattern Recognition*, 1(1):33–61.

Rousseeuw, P. J. and LeRoy, A. M. (2003). *Robust Regression and Outlier Detection*. Wiley & Sons.

Rusinkiewicz, S. and Levoy, M. (2001). Efficient Variants of the ICP Algorithm. In *Proceedings of the Third International Conference on 3D Digital Imaging and Modeling*.

Russell, S. and Norvig, P. (1995). *Artificial Intelligence: A Modern Approach*. Prentice Hall. ISBN: 978-0137903955.

Sack, J. R. (1982). An $o(n \log n)$ algorithm for decomposing simple rectilinear polygons into convex quadrilaterals. In *In Proceedings of the 20th International Conference on Communications, Control, and Computing*.

Saha, M., Latombe, J.-C., Chang, Y.-C., and Prinz, F. (2005). Finding Narrow Passages with Probabilistic Roadmaps: The Small-Step Retraction Method. *Autonomous Robots*, 19(3):301–319.

Schabauer, H., Schikuta, E., and Weishaupl, T. (2005). Solving Very Large Traveling Salesman Problems by SOM Parallelization on Cluster Architectures. In *Proceedings of the International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*.

Schneider, J., Kraus, M., and Westermann, R. (2009). GPU-based real-time discrete euclidean distance transforms with precise error bounds. In *International Conference on Computer Vision Theory and Applications (VISAPP)*. to appear.

Schrijver, A. (2003). *Combinatorial Optimization: Polyhedra and Efficiency*. Springer, Berlin.

Schulz, D., Burgard, W., Fox, D., and Cremers, A. B. (2003). People tracking with mobile robots using sample-based joint probabilistic data association filters. *International Journal of Robotics Research*, 22(2):99–116.

Schwartz, J. T. and Sharir (1983a). On the Piano Mover's Problem II: General Techniques for Computing Topological Properties of Algebraic Manifolds. *Advances in Applied Mathematics*, 4:298–351.

Schwartz, J. T. and Sharir, M. (1983b). On the Piano Movers' Problem: I. The Case of a Rigid Polygonal Body Moving Amidst Polygonal Barriers. *Communications on pure and applied mathematics*, 36:345–398.

Segal, A., Haehnel, D., and Thrun, S. (2009). Generalized-icp. In *Proceedings of Robotics: Science and Systems*, Seattle, USA.

Sequeira, V., C.Ng, K., Wolfart, E., Goncalves, J. G., and Hogg, D. C. (1998). Automated 3D Reconstruction of Interiors with Multiple Scan Views. In *Proceedings of the Society of Photo-Optical Instrumentation Engineers (SPIE) Conference*, volume 3641, pages 106–117.

Shaker, S., Saade, J. J., and Asmar, D. (2008). Person-following using fuzzy inference. In *Proceedings of the WSEAS International Conference on Applied Computer and Applied Computational Science*.

Sharp, G. C., Lee, S. W., and Wehe, D. K. (2002). Icp registration using invariant features. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(1):90–102.

Shi, J. and Tomasi, C. (1994). Good features to track. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition*.

Shin, D. H. (1990). *High Performance Tracking of Eplicit Paths by Roadworthy Mobile Robots*. Ph. D. Thesis, Department of Civil Engineering, Carnagie Mellon University.

Shin, D. H. and Singh, S. (1992). Explicit Path Tracking by Autonomous Vehicles. *Robotica*, 10:539 – 554.

Shneier, M., Chang, T., Hong, T., Shackleford, W., Bostelman, R., and Albus, J. S. (2008). Learning traversability models for autonomous mobile vehicles. *Autonomous Robots*, 24(1):69–86.

Siegwart, R., Arras, K. O., Bouabdallah, S., Burnier, D., Froidevaux, G., Greppin, X., Jensen, B., Lorotte, A., Mayor, L., Meisser, M., Philippsen, R., Piguet, R., Ramel, G., Terrien, G., and Tomatis, N. (2003). Robox at Expo.02: A large-scale installation of personal robots. *Robotics and Autonomous Systems*, 42(3-4):203–222.

Siméon, T., Laumond, J.-P., and Nissoux, C. (2000). Visibility-based probabilistic roadmaps for motion planning. *Advanced Robotics Journal*, 14(6).

Simhon, S. and Dudek, G. (1998). A Global Topological Map formed by Local Metric Maps. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Simmons, R. (1996). The curvature-velocity method for local obstacle avoidance. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Simmons, R. G., Apfelbaum, D., Burgard, W., Fox, D., Moors, M., Thrun, S., and Younes, H. (2000). Coordination for Multi-Robot Exploration and Mapping. In *AAAI/IAAI*, pages 852–858.

Simon, D. A. (1996). *Fast and Accurate Shape-Based Registration*. Ph. D. Thesis, Carnagie Mellon University.

Sproull, R. F. (1991). Refinements to nearest-neighbor searching in k-dimensional trees. *Algorithmica*, 6:579–589.

Stachniss, C. (2002). Zielgerichtete Kollisionsvermeidung für mobile Roboter in dynamischen Umgebungen. Master's thesis (in german), Univerisity of Freiburg.

Stachniss, C. (2006). *Exploration and Mapping with Mobile Robots*. Ph. D. Thesis, University of Freiburg.

Stachniss, C. (2009). *Robotic Mapping and Exploration*. Springer Tracts in Advanced Robotics, Vol. 55. ISBN: 978-3-642-01096-5.

Stachniss, C. and Burgard, W. (2002). An Integrated Approach to Goal-directed Obstacle Avoidance under Dynamic Constraints for Dynamic Environments. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Stachniss, C. and Burgard, W. (2003a). Exploring Unknown Environments with Mobile Robots using Coverage Maps. In *Proceedings of the International Conference on Artificial Intelligence (IJCAI)*.

Stachniss, C. and Burgard, W. (2003b). Mapping and Exploration with Mobile Robots using Coverage Maps. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Stachniss, C. and Burgard, W. (2003c). Using Coverage Maps to Represent the Environment of Mobile Robots. In *Proceedings of the European Conference on Mobile Robots (ECMR)*.

Stachniss, C., Haehnel, D., and Burgard, W. (2004). Exploration with Active Loop-Closing for FastSLAM. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Stentz, A. (1994). Optimal and Efficient Path Planning for Partially-Known Environments. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Stentz, A. (1995). The Focussed $A^\star$ Algorithm for Real-Time Replanning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

Sudarshan, S. and Rangan, C. P. (1990). A Fast Algorithm for Computing Sparse Visibility Graphs. *Algorithmica*, 5(2):210–214.

Surmann, H., Lingemann, K., Nüchter, A., and Hertzberg, J. (2001a). Aufbau eines 3D-Laserscanners für autonome mobile Roboter. Technical Report GMD-Report 126, ISBN 3-88457-974-6, Gesellschaft für Mathematik und Datenverarbeitung mbH (GMD).

Surmann, H., Lingemann, K., Nüchter, A., and Hertzberg, J. (2001b). Fast acquiring and analysis of three dimensional laser range data. In *Proceedings of Vision, Modeling, and Visualization (VMV)*.

Surmann, H., Nüchter, A., and Hertzberg, J. (2003). An autonomous mobile robot with a 3D laser range finder for 3D exploration and digitalization of indoor environments. *Journal Robotics and Autonomous Systems (JRAS)*, 45(3-4):181–198.

Surmann, H., Pervölz, K., Nüchter, A., Lingemann, K., Hertzberg, J., and Hennig, M. (2005). Simultaneous Mapping and Localization of Rescue Environments. *it - Information Technology, Schwerpunktheft Autonome Fussball-Roboter*, 5(47):282–291.

Surmann, H. and Peters, L. (2001). *Fuzzy Logic Techniques for Autonomous Vehicle Navigation*, chapter MORIA - A Robot with Fuzzy Controlled Behaviour, pages 343–365. Physica-Verlag.

Suzuki, I. and Yamashita, M. (1992). Searching for a mobile intruder in a polygonal region. *SIAM Journal on Computing*, 21(5):863–888.

Szeliski, R. and Lavallée, S. (1996). Matching 3-d anatomical surfaces with non-rigid deformations using octree-splines. *International Journal of Computer Vision*, 18(2):171–186.

Takahashi, S., Fujimura, K., and Tokutaka, H. (2002). The SOM-TSP method for the three-dimension city location problem. In *Proceedings of the International Conference on Neural Information Processing*.

Tan, X. and Hirata, T. (1993). Constructing shortest watchman routes by divide-and-conquer. In *Proceedings of the 4th International Symposium on Algorithms and Computation (ISAAC)*.

Tan, X., Hirata, T., and Inagaki, Y. (1993). An incremental algorithm for constructing shortest watchman routes. *International Journal of Computational Geometry and Applications*, 3(4):351–365.

Tan, X., Hirata, T., and Inagaki, Y. (1999). Corrigendum to "an incremental algorithm for constructing shortest watchman routes". *International Journal of Computational Geometry and Applications*, 9(3):319–323.

Thrun, S. (1998). Learning Metric-Topological Maps for Indoor Mobile Robot Navigation. *Artificial Intelligence*, 99(1):21–71.

Thrun, S. (2002). Robotic Mapping: A Survey. In Lakemeyer, G. and Nebel, B., editors, *Exploring Artificial Intelligence in the New Millenium*, pages 1–35. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. ISBN 1-55860-811-7.

Thrun, S. (2003). Learning Occupancy Grid Maps With Forward Sensor Models. *Autonomous Robots*, 15:111–127.

Thrun, S. and Bücken, A. (1996). Integrating Grid-Based and Topological Maps for Mobile Robot Navigation. In *Proceedings of the AAAI Thirteenth National Conference on Artificial Intelligence*.

Thrun, S., Burgard, W., and Fox, D. (2005). *Probabilistic Robotics*. MIT Press, Cambridge, MA. ISBN: 0-262-20162-3.

Thrun, S., Liu, Y., Koller, D., Ng, A., Ghahramani, Z., and Durrant-Whyte, H. (2004). Simultaneous Localization and Mapping with Sparse Extended Information Filters. *International Journal of Robotics Research*.

Topp, E. A. and Christensen, H. I. (2005). Tracking for following and passing persons. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Topp, E. A., Huettenrauch, H., Christensen, H. I., and Eklundh, K. S. (2006). Bringing Together Human and Robotic Environment Representations – a pilot study. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Tovar, B., Muñoz-Gomez, L., Murrieta-Cid, R., Alencastre-Miranda, M., Monroy, R., and Hutchinson, S. (2006). Planning exploration strategies for simultaneous localization and mapping. *Journal on Robotics and Autonomous Systems*, 54(4):314–331.

Turk, G. and Levoy, M. (1994). Zippered Polygon Meshes from Range Images. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*.

Ulrich, I. and Borenstein, J. (1998). VFH+: Reliable Obstacle Avoidance for Fast Mobile Robots. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Ulrich, I. and Borenstein, J. (2000). VFH*: Local Obstacle Avoidance with Look-Ahead Verification. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Urrutia, J. (2000). *Handbook of Computational Geometry*, chapter Art Gallery and Illumination Problems, pages 973–1027. Elsevier Science Publishers.

Urrutia, J. (2004). *Art Gallerey and Illumination Problems*. on-line published DRAFT VERSION 2004-04-28 – available at http://www.matem.unam.mx/∼urrutia/ArtBook.html/Completo.pdf.

Vlassis, N. A., Papakonstantinou, G., and Tsanakas, P. (1997). Learning the Voronoi Centers of a Mobile Robot's Configuration Space. In *Proceedings of the ECPD International Conference on Advanced Robotics, Intelligent Automation and Active Systems*.

Wagner, I. A., Lindenbaum, M., and Bruckstein, A. M. (1998). Robotic exploration, brownian motion and electrical resistance. In *Proceedings of the Second International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM)*.

Wallace, R., Stentz, A., Thorpe, C., Moravec, H., Whittaker, W. R. L., and Kanade, T. (1985). First Results in Robot Road-Following. In *Proceedings of the International Joint Conference on Artificial Intelligence*.

Wang, C.-C. (2004). Simultaneous localization, mapping and moving object tracking. PhD Thesis CMU-RI-TR-04-23, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

Waringo, M. and Henrich, D. (2006). Efficient Smoothing of Piecewise Linear Paths with Minimal Deviation. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3867–3872.

Watson, D. F. (1981). Computing the n-dimensional delaunay tessellation with application to voronoi polytopes. *The Computer Journal by the British Computer Society*, 24(2):167–172.

Weingarten, J. W., Gruener, G., and Siegwart, R. (2004). Probabilistic plane fitting in 3d and an application to robotic mapping. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Wisspeintner, T. and Bose, A. (2005). The VolksBot Concept – Rapid Prototyping for real-life Applications in Mobile Robotics. *it – Information Technology*, 47(5):274–281.

Wit, J., Carl D. Crane, I., and Armstrong, D. (2004). Autonomous ground vehicle path tracking. *Journal of Robotic Systems*, 21(8):439–449.

Wit, J. S. (2000). *Vector Pursuit Path Tracking for Autonomous Ground Vehicles*. Ph. D. Thesis, University of Florida.

Wooden, D. and Egerstedt, M. (2006). Oriented Visibility Graphs: Low-Complexity Planning in Real-Time Environments. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Worst, R. (2003). KURT2 – a mobile platform for research in robotics. In Rückert, U. and Sitte, J., editors, *Proceedings of the 2nd International Symposium on Autonomous Minirobots for Research and Edutainment*, pages 3–12, Brisbane, Australia. Queensland University of Technology.

Wu, L., García, M. A., Puig, D., and Solé, A. (2007). Voronoi-Based Space Partitioning for Coordinated Multi-Robot Exploration. *ournal of Pysical Agents*, 1(1):37–44.

Wulf, O., Nüchter, A., Hertzberg, J., and Wagner, B. (2008). Benchmarking Urban Six-Degree-of-Freedom Simultaneous Localization and Mapping. *Journal of Field Robotics (JFR)*, 25(3):148–163.

Wulf, O. and Wagner, B. (2003). Fast 3D-Scanning Methods for Laser Measurement Systems. In *International Conference on Control Systems and Computer Science (CSCS14)*.

Wurm, K. M., Stachniss, C., and Burgard, W. (2008). Coordinated Multi-Robot Exploration using a Segmentation of the Environment. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Xavier, J., Pacheco, M., Castro, D., Ruano, A., and Nunes, U. (2005). Fast Line, Arc/Circle and Leg Detection from Laser Scan Data in a Player Driver. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Xiong, G., Gong, J., Chen, H., and Su, Z. (2007). Multi-Robot Exploration Based on Market Approach and Immune Optimizing Strategy. In *Proceedings of the International Conference on Autonomic and Autonomous Systems (ICAS)*.

Yamany, S. M., Ahmed, M. N., Hemayed, E. E., and Farag, A. A. (1998). Novel Surface Registration Using the Grid Closest Point (GCP) Transform. In *Proceedings of the IEEE International Conference on Image Processing (ICIP)*.

Yamauchi, B. (1997). A Frontier Based Approach for Autonomous Exploration. In *Proceedings of the IEEE International Symposium on Computational Intelligence in Robotics and Automation*, Monterey, CA.

Yamauchi, B. (1998). Frontier-based exploration using multiple robots. In *Proceedings of the International Conference on Autonomous Agents (AGENTS)*.

Yamauchi, B., Schultz, A., Adams, W., and Graves, K. (1998). Integrating Map Learning, Localization and Planning in a Mobile Robot. In *Proceedings of the IEEE International Symposium on Intelligent Control (ISIC)*.

Yan, P. and Bowyer, K. W. (2007). A fast algorithm for icp-based 3d shape biometrics. *Computer Vision and Image Understanding*, 107(3):195–202.

Zavlangas, P. G. and Tzafestas, S. G. (2002). Integration of Topological and Metric Maps for Indoor Mobile Robot Path Planning and Navigation. In *Lecture Notes In Computer Science; Vol. 2308, Proceedings of the Second Hellenic Conference on AI*, pages 121–130, London, UK. Springer-Verlag.

Zelinsky, A. (1992). A mobile robot exploration algorithm. *IEEE Transactions on Robotics and Automation*, 8(6):707–717.

Zender, H., Jensfelt, P., and Kruijff, G. (2007). Human- and Situation-Aware People Following. In *Proceedings of the IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*.

Zhang, Z. (1992). Iterative Point Matching for Registration of Free-Form Curves. Technical Report 1658, Institut National de Recherche en Informatique et en Automatique (INRIA), Valbonne Cedex, France.

Zhang, Z. (1994). Iterative point matching for registration of free-form curves and surfaces. *International Journal of Computer Vision*, 13(2):119–152.

Zheng, Y. (2007). The Round-Trip Control for Autonomous Robot. Master Thesis, University of Applied Sciences Bonn-Rhein-Sieg.

Zilberstein, S. (1996). Using anytime algorithms in intelligent systems. *AI Magazine*, 17(3):73–83.

Zinßer, T., Schmidt, J., and Niemann, H. (2003). A Refined ICP Algorithm for Robust 3-D Correspondence Estimation. In *Proceedings of the International Conference on Image Processing (ICIP)*.

Zivkovic, Z., Booij, O., and Kröse, B. (2007). From images to rooms. *Robotics and Autonomous Systems*, 55(5):411–418.

Zlot, R. M., Stentz, A., Dias, M. B., and Thayer, S. (2002). Multi-robot exploration controlled by a market economy. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.