
by

Sören Schwertfeger

A thesis submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Computer Science

Approved Dissertation Committee

Prof. Dr. Andreas Birk

Prof. Dr. Kaustubh Pathak

Prof. Dr.-Ing. Dietrich Paulus

Defended on July 6th, 2012

School of Engineering and Science
Abstract

Being able to generate maps is a significant capability for mobile robots. Measuring the performance of robotic systems in general, but also particularly of their mapping, is important in different aspects. Performance metrics help to assess the quality of developed solutions, thus driving the research towards more capable systems. During the procurement and safety testing of robots, performance metrics ensure comparability of different robots and allow for the definition of standards.

In this thesis, evaluation methods for the maps produced by robotic systems are developed. Those maps always contain errors, but measuring and classifying those errors is a non trivial task. The algorithm has to analyze and evaluate the maps in a systematic, repeatable and reproducible way. The problem is approached systematically: First the different terms and concepts are introduced and the state of the art in map evaluation is presented.

Then a special type of mapping using video data is introduced and a path-based evaluation of the performance of this mapping approach is made. This evaluation does not work on the produced map, but on the localization estimates of the mapping algorithm. The rest of the thesis then works on classical two-dimensional grid maps.

A number of algorithms to process those maps are presented. An Image Thresholder extracts informations about occupied and free cells, while a Nearest Neighbor Remover or an Alpha Shape Remover are used to filter out noise from the maps. This all is needed to automatically process the maps.

Then the first novel map evaluation method, the Fiducial algorithm, is developed. In this place-based method, artificial markers that are distributed in the environment are detected in the map. The errors of the positions of those markers with respect to the known ground truth positions are used to calculate a number of attributes of the map. Those attributes can then be weighted according to the needs of the application to generate a single number map metric.

The main contribution of this thesis is the second novel map evaluation algorithm, that uses a graph that is representing the environment topologically. This structure-based approach abstracts from all other information in the map and just uses the topological information about which areas are directly connected to assess the quality of the map. The different steps needed to generate this topological graph are extensively described. Then different ways to compare the similarity of two vertices from two graphs are presented and compared. This is needed to match two graphs to each other - the graph from the map to be evaluated and the graph of a known ground truth map. Using this match, the same map attributes as those from the Fiducial algorithm can be computed. Additionally, another interesting attribute, the brokenness value, can be determined. It counts the large broken parts in the map that internally contain few errors but that have, relative to the rest of the map, an error in the orientation due to a singular error during the mapping process.

Experiments made on many maps from different environments are then performed for both map metrics. Those experiments show the usefulness of said algorithms and compare their results among each other and against the human judgment of maps.
This work is dedicated to my daughter Mia.

You were the sparkling light that guided me out of even the darkest depths of this thesis.

❤️
Preface

This thesis is the scientific climax of my studies as part of the Jacobs Robotics Group at Jacobs University Bremen. I tremendously enjoyed the work and the colleagues here. Designing, building, improving, and repairing hard- and software for our different ground-, underwater-, and aerial-robots was lots of fun and I learned so much - and not only in my field of study (computer science) but also from all the other fields involved. Successfully participating in numerous challenges and competitions with our robots and software was a great experience. It emphasized the need to (at least once in a while) have actually functioning robots to play around with. This required lot of system integration efforts but also ensured, that our research was “grounded” to some real and useful applications from time to time.

My very active participation in all this had several effects: a) at times my work was less emphasized on computer science; b) very diverse publication topics; c) less focus on the work on the thesis; which in combination led to d) slight re-adjustment of the topic half way through; e) a late dedication of my time to just my thesis; and finally f) a slightly late submission of this work. So I am really grateful to everybody who helped in making everything work out fine in the end.

First and foremost, I would like to express my sincere gratitude to my advisor, Prof. Dr. Birk, for his confidence, support, patience, expert guidance and for giving me the opportunity to be part of his group. Towards Prof. Dr. Pathak I am also very grateful, for his efforts, suggestions and help in writing this thesis.

I would also like to thank all my other colleagues from the Jacobs Robotics Group for creating a friendly and supportive environment. This includes, in alphabetical order, Hamed Bastami, Heiko Bülow, Winai Chonnaramutt, Max Pfingsthorn, Jann Poppinga, Ravi Rathnam, and Narūnas Vaškevičius.

During my research visit at the National Institute of Standards and Technology (NIST) in Gaithersburg, Maryland, USA, I was welcomed warmly by Adam Jacoff, Ann Virst, Tony Downs, Tommy Chang and all the others. I enjoyed this eight month stay a lot and thank you all for this great time.

I also had lots of scientific input regarding the topic of this thesis during the different RoboCup competitions. I thus want to especially thank the not yet mentioned, Johannes Pellenz and Raymond Sheh, for the fruitful discussions and suggestions. Those events were also very important for this work, in that they triggered the creation of lots of maps. I have to thank all the teams that agreed to have their maps published here: CASualty: University of New South Wales, Australia; Freiburg: University of Freiburg, Germany; Hector: Technical University Darmstadt, Germany; IXNAMIKI: Universidad Panamericana, Aguascalientes, Mexico; Michigan: University of Michigan, Ann Arbor, USA; MRL: Azad University of Qazvin, Iran; PelicanUnited: Tohoku University, Sendai and Chiba Institute of Technology, Chiba, Japan; and Resko: University of Koblenz, Germany.

I am also grateful for the financial support and/ or work opportunities from the following organizations: Jacobs University, DFG, DAAD and three EU-FP7 projects.

Finally I owe my loving thanks to my family for their constant support and encouragement.
Together with my co-authors, I laid the basis for this thesis in the following publications (the order roughly corresponds to the order of their contribution of the thesis):


Other work related to mapping and applications of scan registration that the author of this thesis contributed to are published in:


• Pathak, K., Birk, A., Poppinga, J., and Schwertfeger, S. 3d forward sensor modeling and application to occupancy grid based sensor fusion. In IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, San Diego. 2007. [Pathak et al., 2007a]

Work related to intelligent and autonomous functions of robots has been published in:


## 3.3.4 Interpretation of the Experiment Results ............................... 63

### 4 Tools for Map Preprocessing and Human Quality Assessment 65

#### 4.1 Image Thresholder .................................................. 66

#### 4.2 Nearest Neighbor Removal ........................................ 68

#### 4.3 Alpha Shape Remover .............................................. 68

##### 4.3.1 Alpha Shape .................................................. 68

##### 4.3.2 Polygonization .............................................. 69

##### 4.3.3 Point Removal ............................................... 69

##### 4.3.4 Results ....................................................... 69

#### 4.4 Comparison between Nearest Neighbor Removal and Alpha Shape Remover ... 71

#### 4.5 Make Hollow ...................................................... 74

### 5 Place-Based Fiducial Approach 75

#### 5.1 Fiducials .......................................................... 75

##### 5.1.1 Identification of Fiducials in the map ...................... 77

##### 5.1.2 Automatic Assignment between Barrel Parts and Fiducials .... 77

##### 5.1.3 Global Accuracy ............................................ 77

##### 5.1.4 Relative Accuracy .......................................... 78

##### 5.1.5 Local Consistencies ........................................ 78

##### 5.1.6 Coverage .................................................... 79

##### 5.1.7 Resolution Quality ......................................... 79

##### 5.1.8 Gathering the Ground Truth Data ........................... 79

##### 5.1.9 Experiments .................................................. 83

### 6 Structure-Based Map Evaluation using Matched Topology Graphs 85

#### 6.1 Overview for Generating the Topology Graph .................... 86

##### 6.1.1 Graph Terms ................................................ 87

#### 6.2 Voronoi Graph ..................................................... 88

##### 6.2.1 Filter for Distance ......................................... 90

##### 6.2.2 Create Pruned Voronoi Graph .............................. 94

##### 6.2.3 Alpha Shape Boundary ..................................... 96

##### 6.2.4 Filter Dead Ends ........................................... 96

##### 6.2.5 Keep Biggest Connected Graph ............................ 100

##### 6.2.6 Join Adjacent Vertices ..................................... 100

#### 6.3 Topology Graph .................................................... 103

##### 6.3.1 Calculating the Angle of Exits .............................. 103

#### 6.4 Calculating the Similarity of two Vertices ........................ 105

##### 6.4.1 Propagation Rounds ........................................ 106

##### 6.4.2 Local Vertex Similarities .................................. 107

##### 6.4.3 Enhanced Vertex Similarities .............................. 108

##### 6.4.4 Range-sensor-based Vertex Similarities .................. 109

##### 6.4.5 ICP-based Vertex Similarity ................................ 110

##### 6.4.6 iFMI-based Vertex Similarity .............................. 111

##### 6.4.7 Vertex Similarity Experiments ............................. 111

#### 6.5 Matching of two Topology Graphs ................................ 116

##### 6.5.1 Graph matching ............................................ 116
6.5.2 Finding TG-Isomorphisms ........................................... 118
6.5.3 The TG-Isomorphism Example ..................................... 119
6.5.4 Grow Neighbors ..................................................... 127
6.5.5 Example Match of two Topology Graphs ......................... 128
6.5.6 Discussion of the Configuration Parameters ...................... 129
6.6 Map Evaluation using Matched Topology Graphs .................... 130
  6.6.1 Coverage ....................................................... 130
  6.6.2 Resolution Quality .............................................. 131
  6.6.3 Global Accuracy ............................................... 131
  6.6.4 Relative Accuracy .............................................. 131
  6.6.5 Local consistencies ............................................. 132
  6.6.6 Calculating the Distance Table .................................. 132
6.7 Computing the Brokenness using matched Topology Graphs .......... 133
  6.7.1 Finding Brokenness Groups ..................................... 133
  6.7.2 Brokenness Experiments ........................................ 136

7 Map Evaluation Experiments ........................................... 141
  7.1 Fiducials at MAGIC 2010 ............................................ 141
    7.1.1 Map evaluation during MAGIC 2010 .......................... 144
    7.1.2 Adjudication software for MAGIC 2010 ...................... 144
  7.2 Fiducials at Disaster City 2010 .................................. 151
    7.2.1 Coverage and Local Consistency Results ..................... 154
    7.2.2 Relative Accuracy Results .................................. 154
    7.2.3 Resolution Quality Results .................................. 154
    7.2.4 Result Discussion ............................................ 155
  7.3 Fiducials at RoboCup Rescue League 2010 .......................... 156
    7.3.1 Results ...................................................... 159
  7.4 Matched Topology Graph Evaluation of RoboCup Interleague Mapping Challenge 2010 162
    7.4.1 Results ...................................................... 163
    7.4.2 Comparison with the Fiducial Approach ..................... 167
  7.5 Topology Evaluation of Broken Maps ................................ 169

8 Conclusion .......................................................... 171

A Configuration Parameters for the Topology Graphs ................... 177

B Maps for Vertex Similarity Experiments ................................ 181
  B.1 Example Vertex Mapping from Map A to H ........................ 184
  B.2 Map Combinations used .......................................... 184

C NIST/ Jacobs Map Analysis Toolkit .................................. 185
  C.1 Implementation .................................................. 185
  C.2 Usage ........................................................... 187

Bibliography .......................................................... 188
Chapter 1

Introduction

The dream of mobile robotic systems that aid the human in a wide variety of tasks has been developed in science fiction literature since the early decades of the last century. Although it had often been predicted that those systems would be available in the not so distance future, it is only now that mobile robots are becoming available for the general public. Though still relatively simple (e.g. vacuum cleaners or lawn mowers), the research and development of such systems is constantly gaining pace.

There are still lots of open topics with regards to mobile robots (e.g. energy storage, miniaturization and durability of actuators and sensors, etc.), and one crucial area with regard to the performance of those systems is their artificial intelligence. In order to interact with humans and among each other and to perform non-trivial tasks, robots need many different advanced capabilities. Many of those capabilities require spatial information about the robot, the interaction partners, or of objects and places of interest. This information is stored in maps. Even if a-priori information about the robots environment is available, mapping is needed to deal with dynamic and changing environments.

The maps generated by robots are usually used to enable the robot to perform certain tasks like for example autonomous navigation using path planning. Maps also assist an operator of a remotely teleoperated robot in locating the robot in the environment. They do this by providing information of features of interest like corners, hallways, rooms, objects, voids, landmarks, etc. Robotic mapping is interesting for many areas of applications in various scenarios. This could be the classical service robotics in industry or office environments, robots participating in the normal traffic, or mining, wilderness or search and rescue robotics and of course military robotics. Map generation is important for ground robots as well as aerial, underwater and even space robots.

The quality of the mapping algorithms and the generated maps has to be ensured. In order to be able to specify and test the performance of mapping systems, the result - the maps - have to be analyzed and evaluated in a systematic, repeatable and reproducible way. The focus of this work is thus to give an overview of some of the important aspects of mapping and to develop and test algorithms to assess the quality of maps generated by mobile robots.
1.1 SLAM and Maps

Mapping is the process of creating maps. A mobile vehicle moves through the environment and collects data with its sensors. The sensors collect data in their own frame of reference, which is typically rigidly connected to the reference frame of the robot. In order to put the sensor readings in a common and global map, the robot location has to be known. Simple mapping algorithms estimate the movements of the robots by using the turn-speeds of their wheels or tracks and apply a dead reckoning algorithm. In those approaches, the errors in localization typically build up very rapidly, inducing errors to the maps. This is especially true if the robot traverses unstructured terrain, for example, in search and rescue scenarios. Other approaches use the global positioning system (GPS) to constantly update the global pose of the robot. This only works with good reception conditions to the satellites (outdoors) and has quite a big error in general.

A better solution to keep track of the robots location is to use the sensor data to localize yourself in the environment. Using sequential scan-matching of the sensor data, one can reduce the errors a lot compared to simple dead reckoning. An even better approach is to employ a Simultaneous Localization and Mapping (SLAM) algorithm. This is a chicken and egg problem where the tasks of localization and mapping are combined. Here the map or prior scans are matched against the current sensor data, which is referenced against the current robot frame. This way the robot can be localized using the map. This knowledge of the new robot position is then in turn used to update the map with the new sensor data.

There are a number of popular approaches to solve the SLAM problem [Thrun, 2002, Thrun et al., 2005, Frese, 2006]:

- Using Kalman filter [Gutmann et al., 1998, Dissanayake et al., 2001]
- Using Expectation Maximization [Thrun et al., 1998, Burgard et al., 1999]
- Using Particle filter [Hahnel et al., 2003, Grisetti et al., 2007a, Marks et al., 2008]

1.1.1 Map Types

The sensors used to perceive the environment determine the different research areas within the field of mapping. Obviously the choice of sensors used for mapping also influences what kind of information about the environment can be represented.

Using range sensors to generate a two-dimensional representation of the environment has a long tradition in robotics [Moravec and Elfes, 1985, Elfes, 1989, Thrun, 2002, Durrant-Whyte and Bailey, 2006]. Here, typically ultra sound rangers or laser range finders (LRF) are used to measure the distance to the nearest obstacles. Often a number of samples are gathered on the 2D plane. The maps are usually represented in a grid with probabilistic entries in the cells which represent whether or not there is an obstacle present. Here the robot is just sampling data in a 2D plane, although it is embedded in a 3D space. This works quite well in structured environments with straight walls and easy locomotion.
1.1 SLAM and Maps

conditions for the robot, for example in office scenarios or certain industrial contexts. But once the environment gets more and more unstructured and the robot cannot keep the sensors level and at a constant height all the time, just using 2D data will not suffice anymore.

Hence the field of 3D mapping has been tackled with a number of approaches [Fischer and Kohllepp, 2000, Surmann et al., 2003, Thrun et al., 2003, Weingarten and Siegwart, 2006, Magnusson et al., 2007, Nüchter et al., 2007, Pathak et al., 2009, Birk et al., 2010, Pathak et al., 2010b]. Different sensors are used for 3D mapping. The first work was probably done on stereo data where the 3D data was extracted from the disparity information of two synchronized video cameras [Barnard and Fischler, 1982]. Actuating normal 2D LRF to get 3D data is another popular approach [Surmann et al., 2003]. Besides that, there are also off the shelf 3D laser range finder in the market [Point of Beginning, 2006]. Those are usually either very fast but quite sparse in one dimension or very precise and slow. The last class of sensors is the 3D cameras which either use modulated infrared light and the phase shift of the reflected light to capture 3D depth images [Lange and Seitz, 2001, Weingarten et al., 2004] or structured (infrared) light approaches [Lai et al., 2011].

Most often 3D mapping generates maps consisting of 3D point clouds, where there are sets of coordinates of obstacle measurements. The most popular approach for 3D mapping is to register the scans using the iterative closest point (ICP) algorithm [Besl and McKay, 1992]. Recent developments, also made by our group, use dominant planes detected in the environment to perform 3D mapping [Pathak et al., 2009, Pathak et al., 2010c].

2D and 3D mapping can also be done in the area of underwater robotics [Fairfield et al., 2007, Williams and Mahon, 2004, Nicosevici et al., 2009, Saez et al., 2006, Buelow and Birk, 2011a, Pathak et al., 2010a]. Here the challenges are even greater for a number of reasons. The sensor data, which is usually acquired with sonar systems, is much more noisy than most other data (e.g. LRF). Also the motion estimates of the vehicles usually have a much bigger uncertainty, because of the dynamic nature of the movement and the environment (currents). Furthermore, there is no GPS available, the visual range is often quite low and the communication bandwidth available is usually very limited.

Mapping using camera images is called Mosaicking, Photo Mapping or visual SLAM (VSLAM) [Davison et al., 2007, Eustice et al., 2006, Angeli et al., 2006, Davison, 2003, Castellanos et al., 1998, Broida et al., 1990]. Usually VSLAM algorithms work best on information-rich images, because feature extraction and tracking algorithms are used. In the related area of computer vision this application scenario is called ”structure from motion” [Fitzgibbon and Zisserman, 1998, Azarbayejani and Pentland, 1995, Faugeras, 1993, Adiv, 1989]. Mosaicking is quite popular for aerial and underwater applications.

1.1.2 (Geo) Referencing of Maps

Mapping is about representing the locations of certain aspects and features of the environment. It is thus important that all those features can be represented in a common frame of reference. The robotic application has to define what this common frame of reference is. Sometimes it is sufficient to just use the frame of reference from the first sensor reading. This could be the case if, for example, a robot in a closed environment just has to ensure that it visited all accessible areas.

But usually the robot or the user of the map is supposed to spatially relate to objects or features in the map outside of the scope of the mapping algorithm. Then the frame of reference of the map has to be put into the relation of a common world frame. For example, this global frame could be
determined by the building the robot moves in.

For outdoor robotics, the global reference frame is usually one defined by geographers with respect to a model of the shape of the Earth. Here the map is anchored using a datum like the Universal Transverse Mercator (UTM) system, most often by utilizing the Global Positioning System (GPS).

1.2 Factors for Mapping

The quality of a map produced by a robot is influenced by many different factors:

- Environment: Sparse environments are difficult for most mapping approaches. Those can be wide open areas or hallways with minimal features.

- Robot path: The path that a robot took to gather the sensor data can contain different amounts of loops. Furthermore, the terrain can be even or difficult to traverse, causing the robot to roll and pitch.

- Robotic platform: Features like active sensing or suspended locomotion can increase map quality while parts of the robot in the field of view of the sensors are disadvantageous.

- Sensors: The range, field of view, structural errors, accuracy and the position of the sensor on the robot are factors influencing mapping algorithms.

- Computation power: SLAM algorithms can be computational very intensive. If the map has to be generated on-line to aid other tasks like path planning, processing time is often limited on mobile robots. Then less scans can be used, or the number of particles in a particle filter is reduced, or loop closing and graph optimization algorithms are executed less frequently.

- Algorithm: Of course the mapping algorithm itself influences the map quality to a great extend.

When comparing mapping approaches, one can try to control some of these parameters to concentrate on the effects of the others. Some scenarios in which different amounts of factors are being kept constant are listed below. Some of those are described in more detail in Chapter 7.

- VirtualRoboCupRescue [Carpin et al., 2006]: Control the environment, and to a great extend platform, sensors and computation power. Robot path (autonomy algorithms) and mapping algorithms determine result.

- RoboCup RealRescue [Tadokoro et al., 2000]: Control the environment. All other factors vary strongly from robot to robot.

- RoboCup Interleague Mapping Challenges 2009 and 2010 [Kleiner et al., 2009, Schwertfeger, 2010]: Control everything except the algorithm.
1.3 Usage of Maps / Applications

Maps generated by mobile robots are used for various purposes in many different areas of application. The two main customers of the maps are a) humans who are interested in the information represented in the maps [Thayer et al., 2000] and b) intelligent functions on the robot that need a map or the localization information in order to perform their task.

Humans play different roles with regard to mobile robots and mapping. The *operator* can be using maps for orientation and navigation. He might also annotate points of interest or other information in the map or set way-points for an autonomous robot. For the *mission control*, maps usually are an important tool to assess the current situation and (re-)define and execute mission goals. This is especially true if there is more than one robot involved. The *end user* is typically using the map as one of his tools to perform a given task. This could be a map on an electronic device - maybe even as part of a Geo Information System (GIS) - or the map can be part of a report printed on paper.

Figure 1.1 shows the three users in a scenario where the application is Search and Rescue [Birk et al., 2009b]. The end user is the responder rescuing the child that was found by the robots. In this figure, a mapping scheme is shown where the robots do distributed mapping. They share their sensor data and the estimates of their location and orientation and incorporate this with their own sensor data into a

![Figure 1.1: The data-flow of a group of robots in a Search and Rescue Scenario. The robots exchange Laser Range Finder (LRF) readings and position/orientation (pose) data for distributed mapping (from [Birk et al., 2009b]).](image)
Given the current state of the art, fully autonomous operation as the only means of control is neither feasible nor desirable for many mobile robot application areas. For example, in the Search and Rescue domain, human operators will still, for quite some time, be in charge of missions [Murphy, 2004]. But robots become increasingly useful with any bit of intelligent function added [Murphy et al., 2001]. Robots that employ semi- or fully-autonomous behaviors make very intense use of Simultaneous Localization and Mapping (SLAM). The localization part of SLAM is very important for nearly all levels of autonomy. But also the current map is needed for a number of algorithms. Figure 1.2 shows the three stages of teleoperation that can be identified for most applications involving mobile robots. For all of those the robot can make use of SLAM.

Even on the motion level, when teleoperating, an obstacle avoidance algorithm can help an operator to prevent bumping into the environment. Although such algorithms can work without SLAM, the superior localization that SLAM provides can improve the obstacle avoidance performance. On the behavior level, autonomous navigation is the main tool. Here the robot is driving from its current location to a specified goal point, avoiding obstacles along the way. The path is typically quite short and not very complicated. The map can be used to find the best way around an obstacle. Other semi-autonomous assistive functions like autonomous stair climbing, opening doors and passing through them or following other robots or persons often benefit from SLAM techniques.

On the mission level, path planning is a very important function. Here, the map is used intensively to find the best path (according to some cost function) for a robot to a goal point that can be far away. If there are multiple possible ways to reach the goal point, one has to be chosen. Also additional factors such as avoiding or preferring unexplored areas along the way or staying within radio distance to other robots or the operator station might affect the path planning. More application specific autonomous tasks are building upon path planning. The exploration task is to map a certain area. That usually means that the robot has to visit (or come close to) all possible locations at least once, such that they can be mapped. In multi-robot coordination, the map is a critical tool used to achieve the coordination. This could be cooperative exploration, maintaining communication, keeping a formation etc. For example, in patrolling scenarios the different robots have to revisit certain interest points, keeping
constraints such as visiting interesting points at least every x minutes, distributing the robots nicely or being unpredictable regarding the time a certain location is visited.

Figure 1.1 shows some of the autonomous functions. One can see that Robot A is being tele-operated, possibly with the help of obstacle avoidance. Robot C has been given a way-point by the operator. So it is using path-planning and navigation to drive to that point. Meanwhile robot B is in full autonomy mode. It is possibly using an exploration algorithm to determine the current goal by itself. This exploration algorithm might take the multi-robot coordination constraints into account. It will try to stay in radio contact such that the distributed mapping task can work uninterruptedly.

Most of the research presented in this paper is done in the context of Search and Rescue, because rescue robotics is an important domain that can serve as an important milestone on the road to truly autonomous mobile robots [Birk and Carpin, 2006b]. This application features very unstructured terrain, such that mapping in this environment can be considered one of the most difficult challenges for SLAM algorithms. Often the maps contain large errors. Thus map metrics that can handle those maps will be applicable for all other application scenarios.

The area of Search and Rescue can be divided into several sub-fields. There is, for example, Urban Search And Rescue (USAR), where peoples’ lives should be saved in an urban environment after events such as earthquakes, accidents involving nuclear, biological or chemical (NBC) materials or other natural catastrophes. The robotic mission could be on a large scale, for example with unmanned aerial vehicles (UAV) providing an overview of a wide area, on a medium scale where mobile robots enter houses in danger of collapsing to search for possible victims, or on a small scale where small robots enter into rubble fields. Also the inspection of structures after such disasters, especially under water, is a task that can be done by robots. In Wilderness Search and Rescue multiple robots cover a large area to search for one or a few missing persons.

But of course SLAM and maps are also used in other application areas of mobile robots. There are military applications like camp security and patrolling, autonomous convoy driving and reconnaissance tasks. Also for the wide area of service robotics, maps play an important part for the autonomous functions - and not only for the obvious path planning. Locations of objects of interest (e.g. fridge, table, the human) have to be stored and updated, such that they can be interacted with later-on. Even if an a priori map is provided, SLAM is still important if objects can change their location (like chairs).

For automation in factory settings, like autonomous trucks, forklifts or transport systems that don’t use a special infrastructure, being able to map the environment and localize yourself is an essential capability.

So it can be seen that SLAM and maps are a very important part for many applications in mobile robotics. Thus a way is needed to measure the quality of the results of the SLAM algorithms - the locations and the map. This thesis will look at SLAM and mapping using different types of robots, sensors and algorithms and introduce novel methods to assess the quality of maps.

1.4 Overview

This work is structured as follows: Following this Introduction, Chapter 2 gives an overview of the process of mapping, the maps and the state of the art of map evaluation. For that, first the sensors that robots typically use to perceive their environment are introduced. Then different algorithms that use this data to estimate the movement of the robot between two successive sets of sensor data are
presented. This process is called scan registration. SLAM and how it differs from mapping just using scan registration is introduced next. In Section 2.3 the most popular ways to internally represent the map are presented. Then algorithms using maps are shown, using examples from the Jacobs Robotics Group. In Section 2.5 different aspects of map quality are explored and map attributes are introduced. Last but not least the state of the art in map evaluation is presented.

Chapter 3 is then about the path-based evaluation of the iFMI Mosaicking algorithm. First a registration algorithm developed at the Jacobs Robotics Group, the iFMI, is presented. Then it is shown how this algorithm is used for mapping and SLAM. Finally, the performance of the iFMI mapping algorithm with respect to different parameters and options is examined, using a path-based evaluation approach.

The next chapter (4) deals with algorithms to pre-process maps before they can be applied to our map evaluation algorithms. Different algorithms are presented and, if applicable, compared.

Chapter 5 then presents a place-based map evaluation algorithm that has been developed in cooperation with the National Institute of Standards and Technology (NIST). This Fiducial Map Metric is developed in detail and ways on how to gather ground truth data for map evaluation are described.

The main contribution of this thesis, the structure-based Map Evaluation using Matched Topology Graphs, is presented in Chapter 6. After providing an overview of the whole process, it is first explained how a Topology Graph can be extracted from 2D grid maps using a Voronoi Diagram. In Section 6.4 similarity metrics between two vertices are developed. Then it is shown how two Topology Graphs can be matched, if they represent the same environment. The last two sections of this chapter then present ways of how to use matched graphs to evaluate the quality of maps.

In Chapter 7 several experiments using the two map evaluation methods, i.e. the Fiducial Approach and the map evaluation using Matched Topology Graphs, are performed and analyzed. Finally, in Chapter 8, a summary and conclusion is given.
Chapter 2

Mapping, Maps, and Map Evaluation

This chapter provides a small overview on maps and how they are created and evaluated. A number of scan registration and mapping approaches are presented and the different types of maps are explained. Some robotic applications that make use of maps are described. The evaluation of maps is analyzed and the state of the research in this field is presented. But first there is an introduction to the sensors that can be used for robotic mapping.

2.1 Perception - Sensors

Robots are often solely used as mobile platforms for the sensors - their main task is to bring the sensor where it needs to go. Understanding the differences in the capabilities, constraints and performance of the various sensors is thus important for the robotics application. This is also true for mapping. The most common type of sensor used for robotic mapping delivers some kind of range information of the distance between the sensor and the objects in the environment. In the special case of mosaicking, images obtained by a camera are the input for the mapping algorithm.

2.1.1 Range Sensors

The range information delivered by the sensor can be a single (1D) value, a set of values of distances in a plane, typically in a polar coordinate system (2D) or a set of range values from a volume with a certain horizontal and vertical opening angle. The latter sensors are often called 3D sensors. But most of them can only deliver one distance per direction, since objects (obstacles) behind the first object are occluded by it. One could then represent this data in a 2D grid with range information in the cells. This so called depth-image is said to be only 2.5D data since it was taken from just one view point and because of the occlusion from the closest objects.

Active sensors emit some sort of signal and use the reflection of this signal at the object to determine the sensor-object distance. Passive sensors, on the other hand, register signals from objects without emitting anything. All range sensors presented here are active sensors, with the exception of the stereo camera, which uses images of the objects in visible light.

Sound waves in air or water are used by sonar sensors. The time the ultrasound signal takes to reach an object and travel back to the sensor as a reflection is used together with the knowledge of
the speed of sound in air or water. In the air sonar sensors are just 1D, since far superior solutions exist for more complex sensing tasks. In the underwater domain sonars are often the only sensor that can deliver data. 2D sonars with a rotating head suffer from the slow speed of sound (compared to the speed of light for lasers), such that those sensors have a very slow update rate (often far less than 0.5 Hz for a 360° scan). Phased array sonars use multiple sensors to calculate the location of different obstacles along different angles from just one sound emission. Those sensors can deliver 2D and even 3D data. Since sonars deliver multiple reflectance values along each beam, the 3D sonars can, under ideal conditions, actually deliver real 3D data, since the sound might pass through the first obstacle and also deliver a value for an occluded, second obstacle. Figure 2.1 shows an example of a 3D underwater sonar scan that we took at the river flood gate (Sperrwerk) of the river Lesum. Also a map generated using 18 of such scans is depicted.

Infrared light is the most popular physical signal used to measure distances in robotic mapping. Due to the high speed of the light, it allows high update rates. Since air is transparent to this wave length, great distances can be achieved. Compared to sonar sensors the precision is also very good.

Classical 1D infrared distance sensors use triangulation and a series of photo cells to determine the distance of obstacles. In comparison, laser range finders (LRF), that use the time of flight principle, have higher precision. By using a rotating mirror, LRF sensors can deliver a 2D scan in a plane [Surmann et al., 2003]. This is called LIDAR (Light Detection And Ranging), although the term LRF is often also used for those devices. There are also devices that use lasers to gather 2.5D range images (3D LRF, 3D scanning) [Poppinga et al., 2010]. In order to achieve even higher update rates, some sensors use several lasers in one device. LRF sensors usually have a quite high field of view (minimum 180°, often more than 240° and up to 360°).

Depth cameras have a smaller field of view (around 30°), but can deliver 2.5D depth images with a very high update rate (25 Hz and more). Phase-shift approaches emit modulated infrared light and measure the phase-shift of the reflected signal per pixel [Lange and Seitz, 2001, Weingarten et al., 2004]. The accuracy of the data delivered by these sensors is not as high as 3D LRF data since there are a number of error sources for this approach.

Another depth-camera approach is to use structured (infrared) light to compute the 3D information of the pixels [Lai et al., 2011], which works quite well.
2.1 Perception - Sensors

The passive stereo vision uses triangulation to determine the distance of a common feature detected in two displaced cameras. It is challenging for 3D mapping because it has been considered inadequate for reconstructing the dense and accurate 3D data required for many tasks [Sumi et al., 2002, Barnard and Fischler, 1982].

2.1.2 Example Sensor Suite of a Search and Rescue Robot

A number of sensors typically found on mobile robots is presented here. Figure 2.2 shows a picture of a search and rescue robot from the Jacobs Robotics Group, where a few important sensors are pointed out.

As can be seen from the figure, it has a wide array of sensors, whose output can be seen in Figure 2.3:

- a 3D time-of-flight (TOF) camera (SR: SwissRanger SR-3000 from CSEM),
- a stereo camera (STOC from Videre),
- a video camera,
- an infrared camera (to detect heat sources such as humans),
- a pan-tilt-zoom-camera (from Panasonic), and
- two laser range finders (LRF) (both URG from Hokuyo).
2.2 Mapping

In the simplest form of mapping, sensor data, that has been gathered from the robot in the sensor reference frame, is transformed to the global frame of reference by using the known location of the robot. This data is then added to a data structure (“the map”) in this global reference frame. But in most robotic applications the robot location is not readily known and Simultaneous Localization and Mapping (SLAM) techniques have to be used to build the map and localize the robot within it at the same time. In this thesis the term mapping usually refers to the full SLAM problem.

It is common to all mapping algorithms that at some point in the computation, the sensor data has to be matched against previously collected data or the map. This matching process assumes that both sets of data represent overlapping readings from the same environment. A successful matching returns the transformation (rotation, translation, sometimes also scaling) that one of the data sets has to undergo such that the data representing the same real world objects is at the same coordinates. This process is called Registration. From the resulting transformation between the two (sensor) reference frames, the transformation of the robot reference frame can be easily computed, thus providing localization information.

2.2.1 Registration Methods

Iterative Closest Point (ICP) is a classical scan registration method between two point clouds [Besl and McKay, 1992]. ICP starts with an initial guess (usually the odometry from the robot) and assumes that nearest neighbors correspond. It then transforms one of the point clouds such that the nearest neighbors move towards each other, such that a least mean square (LMS) of errors is achieved, using Horns algorithm [Horn, 1987]. This process is iterated with the found transformation as new initial transformation until either a maximum of iterations or a threshold for the mean squared error or its change is reached. ICP is based on a heuristic - the nearest neighbor criterion. That means that the
2.2 Mapping

Figure 2.4: An overview of the different steps in 3D Plane SLAM using high resolution 3D laser range data (from [Birk et al., 2010]).

initial guess for the transformation has to be rather good, otherwise the process might be caught in a local minimum which does not correspond to the actual transformation.

ICP is the most popular registration method for the 2D and 3D registration of point clouds. But in the Jacobs Robotics Group we have been successfully exploring two other registration approaches. The Plane SLAM approach (see Figure 2.4) does not work with point clouds but with dominant planes extracted from the point clouds [Birk et al., 2010]. The fact that planes, unlike points, have an orientation (defined by their normal vector) helps to significantly speed up the search for the transformation to which a maximum of planes from both plane sets agree. Also, there are orders of magnitude fewer planes in a plane set than points in a point cloud, such that the plane registration is very fast.

The Improved Fourier Mellin (iFMI) is another registration technique which is described in more detail in Chapter 3 [Buelow et al., 2009, Pfingsthorn et al., 2010b]. It is a spectral method that uses the signal shift in the phase differences in the spectra to determine the transformation (rotation, scaling, translation) between two sets of data. Variants of iFMI can be used to register video data as well as 2D LRF scans. Our group has also developed a spectral method to register 3D data [Buelow and Birk, 2011b].

Using feature extraction and matching is another way to register sensor data, which is most popular for visual camera data. Often the Scale-Invariant Feature Transform (SIFT) [Lowe, 2004] or the Speeded Up Robust Feature (SURF) [Bay et al., 2008] are used. Other innovative registration approaches use planar features [Weingarten and Siegwart, 2006] or normal distributions on an occupancy-grid like structure (3D-NDT) [Magnusson et al., 2007], to name just a few.
2.2.2 SLAM

The basic application of registration is to successively match the current sensor reading against the previous one. In this form of open loop localization, errors in the registration process quickly build up [Elfes, 1989, Moravec and Elfes, 1985]. That is why this approach to mapping is not considered to be a sufficient solution for most applications.

To mitigate this build up of errors, SLAM algorithms (additionally) try to register scans against older data - may it be in the form of the map (which contains the aggregated sensor data), older scans or already observed land marks.

In the Kalman filter approach to SLAM, the robot observes and revisits land marks (features). Using an Extended Kalman filter (EKF), the estimates of the position of the landmarks and the robot are kept and updated if a loop is closed (a land mark is re-visited) [Gutmann et al., 1998, Dissanayake et al., 2001].

Expectation Maximization is an optimization algorithm [Thrun et al., 1998, Burgard et al., 1999] based on landmarks, in which the robot motion and the perception are statistically processed. In this maximum likelihood estimation problem the location of landmarks and the robots position are estimated using two steps. The expectation step keeps the current map constant and calculates the probability distribution of past and current robot locations. Then the most likely map is computed in the maximization step. This is based on the estimation result of the expectation step. Through the alternation of these two steps simultaneous localization and mapping (SLAM) is achieved, that results in a local maximum in the likelihood space. According to [Aulinas et al., 2008], it is computationally very extensive and cannot incrementally build maps.

Particle filters are recursive Bayesian filters using Monte Carlo simulations to keep track of the robots location [Hahnel et al., 2003, Grisetti et al., 2007a, Marks et al., 2008]. A set of possible robot locations and maps (the particles) is kept and updated using the sensor input. A weight is associated with each particle, indicating how well the observations correspond to each other. At the end of the iteration a new set of particles is sampled using the calculated weights.

Graph based SLAM approaches save sensor data in nodes and the robot movement in the edges between the nodes [Lu and Milios, 1997, Frese, 2005, Frese et al., 2005, Olson et al., 2006, Grisetti et al., 2007b, Takeuchi and Tsubouchi, 2008, Pfingsthorn and Birk, 2008, Pfingsthorn et al., 2009]. At first, the common sequential mapping using one of the registration techniques described above is applied to compute the transformation between the nodes, which are saved in the edges. A loop detection algorithm is then responsible for finding overlapping sensor data from the different nodes. After applying registrations between those new found overlaps, new edges can be added between non sequential nodes. An optimization algorithm is then used to minimize the global error of the graph, thus improving the localization of the nodes. Uncertainty estimates from the registration step are usually used to improve the graph optimization. Figure 2.4 shows the different steps of a graph based SLAM algorithm using plane registration.
2.3 Map Representations

Maps can be represented in many different ways. In robotics, the most popular representations depict occupied space using grids or point clouds, but geometric representations like lines [Garulli et al., 2005] or large surface patches [Birk et al., 2010] can be considered. Also the dimensionality of the map is a very important factor. 2-D maps, saving occupied space in a horizontal plane through the sensor, were the earliest representation. 3-D representations demand much more computation power and memory, but are becoming increasingly popular. A couple of representations, such as elevation maps or data from a single 3-D range scan, should be placed between 2-D and 3-D maps.

Grid representations store information about the environment in a matrix [Elfes, 1989]. Each cell in this matrix represents a certain (square) area in the real world - the length of this square is called the resolution of this grid (e.g. 5cm x 5cm). There are different approaches as to what is stored in the cells. At least the information if a cell is occupied is saved, but usually there is also a differentiation between free and unobserved cells. In a probabilistic representation, each cell has a certain probability of being occupied or free. Instead of the probability, sometimes the odds of a cell being occupied or not are stored. In the probability map, one value can be used for each cell - either a real number of a value between 0 and 255 for using just one byte of memory. The map is initialized with the unknown/unobserved value between the minimum and maximum (0.5 or 127). Subsequent observations then update this value towards the maxima: either occupied (0) or free (1.0 or 255). Other approaches could use the output of a terrain classification algorithm as probability values [Poppinga, 2010] of the cells. Figure 2.5(a) shows a typical grid map in which additionally the path that the robot took is depicted. Three dimensional grids are sometimes also called regular mesh.

In the cells of elevation (grid) maps, the height of the obstacle (respectively ground) relative to a horizontal reference plane is stored [Kleiner and Dornhege, 2007]. Elevation maps are said to be 2.5D because they use a 2D grid but represent, to a limited extend, 3D information. They are not full 3D representations, since they can only store one height information. Figure 2.5(b) is an example for an elevation map.
Mapping, Maps, and Map Evaluation

(a) A map represented as 2D point cloud.
(b) A map represented as 2D line segments.

Figure 2.6: 2D Map representations (both from [Garulli et al., 2005]).

More sophisticated implementations can use trees to store the grid. There, adjacent cells containing the same information are joined together, thus saving space. There are also advanced algorithms that perform certain tasks faster on trees than on plain grids. For 2D grid maps, quad-trees are used, while 3D grids can be compressed using oct-trees [Jackins and Tanimoto, 1980, Meagher, 1982]. Section 2.3.1 describes an oct-tree implementation that was developed in the context of this thesis. Figure 2.8 shows a path (in red) in a 3D grid which is internally represented as an oct-tree.

Point clouds are saving the observed locations of obstacles in a set. The coordinates of the points are saved as 2D or 3D values, typically in a Cartesian coordinate system relative to some global frame of reference. Extended versions might save additional information per point, such as a reflectance value (also known as intensity) or the color of the obstacle. Point clouds might also be stored in a tree (e.g. kd-tree [Bentley, 1975]), which can significantly speed up certain algorithms. Figure 2.6(a) shows a 2D point cloud while in 2.7(a) a 3D point cloud of the city center of Bremen can be seen.

Graph-based representations (also known as pose graph) save sensor observations in nodes representing the frame of reference of the sensor (or robot) at the time the observation was made [Pfingsthorn et al., 2007, Pfingsthorn and Birk, 2008, Pfingsthorn et al., 2009]. Those nodes are connected by edges representing the movements the robot has made between two observations. The sensor observations are typically saved as some kind of point cloud since this representation is closer to the raw sensor data than for example a grid based representation. A mapping algorithm might correct (or optimize) the transformation (robot movement) in the edges thus improving the map. Since the sensor observations are most often stored as points, a graph based representation can be seen as a special form of point cloud representation.

One main difference between grid and cloud representations is, that free space can be distinguished from unobserved space in grids, while this is typically not possible for clouds. Clouds do not suffer from the quantization of space by the resolution of grid cells. On the other hand, grids automatically merge multiple observations of the same object while this has to be done in an extra step for point clouds. Since grids also store unobserved cells they can get quite big regarding memory demands, especially in 3D. This problem is mitigated to some extend by the use of quad- or oct-trees.

There are other map representations that do not use points or cells but more complex objects such
2.3 Map Representations

(a) 3D Point Cloud  (b) 3D Plane Map

Figure 2.7: 3D Maps of the city center of Bremen ([Pathak et al., 2010d]).

as lines, polygons or planes. Those usually abstract finer details of the environment and are more general. This data is usually much smaller in size and many algorithms typically perform better on such representations. Figure 2.6(b) shows a 2D line representation while Figure 2.7(b) depicts the city center of Bremen as a set of planes.

Maps that use color or gray-scale images are often represented in a pose graph, where the images are stored in the node. Those images can then be transformed to a common frame of reference and rendered into a big image - the mosaic or photo map. See Chapter 3 for more detailed description.

Topological maps completely abstract from the sensor data and save more general data. They are graphs where the nodes represent landmarks, rooms or junctions and edges reflected connectedness between two landmarks. Common examples for topological maps are plans of public transportation systems which are often very abstract (see Figure 2.9.)

2.3.1 A Fast Octree Implementation

Since 3D maps usually consist of a lot of points, a small representation that supports fast access, namely a special implementation of an octree, was co-developed in the context of this thesis [Poppinga et al., 2007]. The octree data structure [Jackins and Tanimoto, 1980, Meagher, 1982] is a means of storing spatial occupancy information very memory-efficiently, especially more efficiently than in the default solution, a 3D grid.

More precisely, we are interested in a data-structure where for every cell \((x, y, z)\), a value \(occ \in \{\text{occupied}, \text{free}\}\) has to be stored. The word “octree” is formed from “oct” (short for “octant”) and “tree”. As the name suggests, information is stored in a tree.

An octree is able to collapse nodes. This happens when all sibling nodes are either “occupied” or “free”. Then, this node will be represented in the parent by a single value, saving memory.

The FastOctTree (FOT) is an implementation of an octree optimized for memory consumption and speed. The key design feature is the distinction between leafs of the octree and leafs of the FOT: The latter store the values for eight of the former in a bitmap, thus saving one level of nodes.
Figure 2.8: A 3D path (in red) as a 3D Grid Map (from [Ichim, 2011]).

Figure 2.9: Topological map of the subway system in Tokyo (from [Wikimedia and Cid, 2006]).
2.3 Map Representations

FastOcTreeNode is used for the nodes as well as for the leaves. It only has a single one-word member, a union. This union can either be used as a pointer or as a bitmap. As a pointer, it points to a struct containing all children as member variables. As a bitmap, it stores the occupancy of its children. Which role a node plays depends on whether it is a leaf or not, which can be determined by the value of the least significant bit of the union.

As an inner node the variable is a pointer. The class FastOcTreeNode is 4 byte memory aligned, thus ensuring that at least the two least significant bits of pointers to objects of this class are always 0. For FOT leaves, on the other hand, the union stores a bitmap. As an inner octree node always has eight children, only the most significant byte is used. Thus the least significant bit of the least significant byte is always set to 1 to identify this node as a FOT leaf.

The class FastOcTree manages the tree of FastOcTreeNode. No coordinates and edge lengths are stored in the nodes, these have to be maintained by this class. Both iterative and recursive tree traversals are used, depending on the task at hand.

Bresenham’s algorithm

The Bresenham’s line algorithm as well as a 2D Bresenham-like conic drawer are implemented for the FastOcTree. The Bresenham line and also the cone are very fast by using almost no multiplications, no divisions nor square-roots or trigonometric functions. Combinations of the line and the cone algorithm are used to generate more complex primitives like boxes, cylinders or spheres.

Nearest Neighbor

Finding the nearest neighbor is an important algorithm, that can be used to calculate the similarity between two FOTs or for implementations of the ICP registration. Finding the nearest neighbor means, that given a particular cell in one data set, the closest cell with the same property must be determined in a second data set.

Due to the hierarchical structure of the octree, we can quickly find all filled cells and compute the nearest neighbor distances for only those points. For this, an algorithmic optimization is introduced, which extents the work of Hoel and Samet on nearest neighbor search on line segments in a quadtree [Hoel and Samet, 1991].

In a dynamically growing octree, the presence of a node implies, that within the boundaries of that node, there is at least one leaf node. Thus, we can immediately deduce an upper bound to the volume we have to search in the tree. Potentially, a lot of subtrees can be pruned from the search that lie completely outside this bound. While searching the nodes which do fall within this bound, we can progressively tighten said bound when we encounter leaf nodes in lower levels. This way we prune even more nodes from the search. Exploiting this property of octrees makes this algorithm very efficient. There are two phases for our recursive implementation of this algorithm as shown in Algorithm 1. First, finding the octree node from which we want to start the search and finding the upper bound on the subsequent search. Second, we search this node’s children and this node’s parents children, and so forth, until we reach the root node. During the search, we progressively tighten the upper bound to correspond to the closest filled cell found so far, ensuring the pruning of subtrees that lie outside of the current bound.

In the first phase, we first need to find the node which contains the point from where we start our
Algorithm 1 NearestTreeNeighbor – NearestTreeNeighbor from a node.

procedure NearestTreeNeighbor(node, point)
if node.hasChildren() and not at maximum depth then
    node = getNextChild(node, point)
    NearestTreeNeighbor(node, point)
else
    upperBound = 4 \cdot edgeLength(node)
    SearchChildren(node, upperBound)
end if

procedure SearchChildren(node, upperBound)
if not node.hasChildren() then
    upperBound = manhattan distance to filled cell closest to point
    nearestPoint = said cell
else
    if any point in this node falls in current upper bound then
        children = list of children, ordered by distance to point
        for all child \in children do
            SearchChildren(child, upperBound)
        end for
    end if
end if
end if

search. In order to do so, we traverse the tree downwards, choosing the child node which contains that point. We do this until there are no more children, or until we reached the maximum depth. We then set the estimate of the upper bound to six times the edge length of the node we are at (six times because we want the Manhattan distance across the parent in 3D). This procedure is implemented recursively and shown in Algorithm 1.

The upward phase actually performs the search. Going up the stack generated by the recursive downward phase, we search the area around the query point outwards. To ensure this search pattern, the children of the current node are sorted by the Manhattan distance to that point. In order not to replicate work, we do not search the child of the current node which contains the original point, if there is such a child, since we would have visited it before. The children of the current node are only searched if any of them lie within the current upper bound.

In addition to these optimizations in the implementation, a special conceptual optimization is introduced, namely the so-called \textit{LineBurstAccess}. This heuristic starts the traversal to access a specific node from the last visited one. This strategy is more efficient than the default to start from the root as geometrically close points are usually consecutively accessed.
2.4 Applications using Maps

In this section some examples of applications or algorithms that use maps to perform their task are given. For the sake of convenience, those simple examples are all from the Jacobs Robotics Group. As already mentioned in the introduction, path planning is one of the basic tasks needed for an autonomous robot. In Figure 2.10 we can see two different path planning algorithms using a 2D grid map representation of a part of the Jacobs Robotics Lab. In Figure 2.10(a) the A* algorithm is used to find the path between two points [Hart et al., 1968]. In order to keep the robot from bumping into obstacles, occupied cells were grown by the radius of the robot (plus a safety margin) before the start of the A* algorithm. This algorithm works by keeping a list of frontier cells and using a heuristic (usually the geometric or Manhattan-distance to the goal) to decide which frontier cell to explore next.

Figure 2.10(b) shows the Funnel planner which does not use obstacle growing. It is based on the work of Brock and Kavraki [Brock and Kavraki, 2000] and also described by LaValle [LaValle, 2006]. This planner works by growing circles until they hit an obstacle. If this circle exceeds a minimum radius it is added to the frontier. The first circle is generated around the start point. New circles, whose center points are positioned on formerly generated circles, are grown, with a bias towards the goal. If a circle touches the goal point, a path through the center points of the subsequent parent circles is then found.

Multi robot coordination also often relies on maps [Birk and Carpin, 2006a, Carpin et al., 2005]. In Figure 2.11 one can see six robots exploring a (simulated) map in a coordinated manner under communication constraints [Rooker and Birk, 2007]. Explored cells are depicted in green, frontier cells in yellow, unexplored cells in white and obstacles in black. The robots are the red dots. They need to stay within communication range (red lines).

3D maps can be used to determine which part of the environment is drivable for the robot. This task is called terrain classification. Figure 2.12 shows an approach of terrain classification developed at the Jacobs Robotics Group [Vaskevicius and Birk, 2011]. Here the plane map representation is
Figure 2.11: Multi-robot exploration under communication constraints (from [Rooker and Birk, 2007]).

Figure 2.12: 3D terrain classification: A visualization of drivable (green) and non-drivable (red) parts in the 3D plane map (from [Vaskevicius and Birk, 2011]).
2.5 Map Quality

Maps generated by mobile robots are abstractions of the real world which always contain inaccuracies or errors. Especially on extended missions or in rough terrain, maps often contain large errors. But the usefulness of a map not only depends on its quality but also on the application [Leonard and Durrant-Whyte, 1992]. In some domains certain errors are negligible or not so important. That is why there is not just one measurement for map quality.

The derivation of meaningful quantitative assessments of the map quality is desirable for many reasons. They allow a ranking, which can be used in competitions like RoboCup Rescue [Bredenfeld et al., 2006]. Also, quantitative results can be tracked over a long time and thus provide information about the development of the mapping system and the field as a whole. But numerical results, especially when combined to a single number for ranking purposes, have the risk of over-simplifying the problem. This is another reason that the use of a number of different map quality attributes is proposed in this thesis.

2.5.1 Accuracy and Precision

Robotics is not the only area of research where maps play a significant role. In Geographical Information Systems (GIS), the question about the quality of maps is also quite important. GIS are often
defined as information systems that manage, manipulate, and analyze spatial data [Theobald, 2007]. GIS link geographic features represented on a map with attribute data that describe the geographic feature.

In GIS, accuracy and precision are two measures of how well spatial features are located to their true position. This assumes that, for a given feature, several measurements of its location are made. Then the accuracy is a measure of how close the average position of those samples is to the true position. The precision, on the other hand, is the measure of how close the samples are relative to each other [Theobald, 2007]. The bias is the systematic error that affects all data. Figure 2.14 shows examples for high and low precision and accuracy.

In the context of maps generated by robots, these definitions of accuracy and precision cannot be used in a straightforward way. That is because of the assumption in GIS, that one point of interest is sampled with many measures. This is definitely also the case during the process of robotic mapping, but the end product, the map, typically consists of just one value for the position of a feature. GIS and robotic mapping also differ in what is measured. While GIS systems typically measure the position of certain interesting points (and maybe lines between those), robotic mapping usually is about finding any occupied (and free) space.

So for specifying the map quality in robotics, the term accuracy can be used for describing the error between the measured and true position of a feature, meaning an occupied cell or point. Since those measurements (e.g. obstacle cells from a wall) typically cannot be related to a precise location in the real world (we might know that this cell is from this wall, but to which point of the wall is actually corresponds is not readily known), more sophisticated approaches for the measurement of
accuracy are needed and will be described in this thesis.

With just one data point, *precision* cannot be calculated. But of course the relative error between neighboring samples is still interesting. So the GIS concept of precision can be extended to measuring the relative error in robotic maps. When looking at the smallest scale, neighboring cells in a grid-representation or points in a cloud-representation can represent features of the environment (e.g. a straight wall) just to a certain scale - the resolution of the system. This resolution is influenced by many factors, but most importantly by the values returned from the sensor, the mapping algorithm and the map representation. In order to not confuse this value with the resolution of a grid-map representation (which might be much higher than the abilities of the whole mapping system), the term *resolution quality* is used in this work.

On a bigger scale, the relative error between mapped big-scale features (e.g. corners, doors, etc.) might also be of interest. The term *consistency* is used here instead of precision since it corresponds to this derived meaning in a better way.

### 2.5.2 Map Quality Attributes

The quality of maps generated by robots should be assessed by using different attributes. Those attributes should be measured separately and weighed according to the needs of the application [Lee, 1996].

Those attributes can include:

- **Coverage**: How much area was traversed/visited.
- **Resolution quality**: To what level/detail are features visible
- **Global accuracy**: Correctness of positions of features in the global reference frame
- **Relative accuracy**: Correctness of feature positions after correcting (the initial error of) the map reference frame
- **Local consistencies**: Correctness of positions of different local groups of features relative to each other
- **Topological accuracy**: To what level can directions (e.g. "go left at the 2nd crossing") be extracted? Is the topological structure correct or are there broken areas with orientation errors?

Note that the resolution quality is not only depending on the actual size of the grid cells of the map but is also influenced by the quality of the localization. If there are pose errors between scans of the same object its features blur or completely vanish. Depending on the groups chosen there can be multiple local accuracies.

### 2.5.3 Map Metric

The map evaluation algorithms described in this thesis are sometimes also referred to as map metrics. This is because they are supposed to give an unbiased, objective score of the quality of a map. The term metric is also used in other fields like Performance Metric in business [Brown, 2006] or Software Metric for the quality of code in computer science [Kaner et al., 2004].
It can be easily seen that map metrics that compare a robot map with a ground truth map are not metrics in a strict sense if looking at the mathematical definition of a metric [Arkhangelskii et al., 1990]:

There is a distance function $d$ with $d : X \times X \rightarrow \mathbb{R}$. For the variables $x, y, z \in X$ this distance $d$ has to satisfy the following conditions in order to be a metric.

1. $d(x, y) \geq 0$ non-negativity axiom
2. $d(x, y) = 0$ if and only if $x = y$ coincidence axiom
3. $d(x, y) = d(y, x)$ symmetry axiom
4. $d(x, z) \leq d(x, y) + d(y, z)$ triangle inequality

Axiom number three is easiest to break. Given a big ground truth map ($x$) and a very small robot map ($y$), the robot map would get a very bad score (big difference) when compared to the ground truth map ($d(x, y)$). But when we take the robot map as the ground truth and compare it to the ground truth map ($d(y, x)$), we might get a very good score (very small difference in the parts that actually overlap). The two values are different and thus axiom three is broken.

The map evaluation algorithms presented here will return excellent scores up to the value 0 for very good maps, although the maps differ in small details, so also axiom two is broken. The fourth axiom also does not really make sense in the context of map evaluation. If $y$ is the ground truth map and $x$ and $z$ are robot maps, $d(x, z)$ might be very big (e.g. no similarity at all between non-overlapping maps), such that the sum of $d(x, y)$ and $d(y, z)$ can be smaller (both $x$ and $z$ represent parts of the ground truth map). So also the fourth mathematical axiom of the metric definition is broken by our map evaluation algorithms.

Nevertheless, in the spirit of measuring a reliable value of the quality of a map, the map scoring approaches presented in this paper are sometimes called map metrics.
2.6 Map Evaluation Algorithms

The state of the art in map evaluation will be presented in this section. But first a small overview on how other robotic capabilities are measured is given.

2.6.1 Standard Test Elements

Many robotic capabilities can be assessed in a standardized, reproducible and repeatable way in compact test environments. Those standard test elements are developed by the National Institute of Standards and Technology, NIST [Jacoff et al., 2001, Jacoff et al., 2002, Jacoff et al., 2003b, Jacoff et al., 2003a] and published as ASTM International standards (until 2001 ASTM was standing for American Society for Testing and Materials) [ASTM, 2011a].

The development of those test elements is still ongoing. In [Jacoff, 2011] a good overview of the elements that are currently evaluated is given. For example, Figure 2.15 shows three standard test methods for stairs with different inclinations and a test method that is used for testing directed perception and manipulation. Typically when testing a robot, it has to perform a task (like going up and down the stairs) a number of times (often 10 or 30, depending on what significance and confidence is desired). An expert operator is allowed to remotely teleoperate, but not touch the robot. In Figure 2.16 three mobility tests can be seen: crossing ramps, sand and gravel. Here the robot has to perform a number of figure eights around the poles. The same task has to be performed in the symmetric stepfield shown in Figure 2.17, which also features a robot (called “Pointman”) in action.

Other standard test elements evaluate other mobility capabilities (e.g. on an inclined plane, gap-test), the endurance performance or the visual acuity ([ASTM, 2011b]) and sensor performance. In [Jacoff, 2011] also a random maze is described. It is used for evaluating the human-system interaction capabilities with search and navigation tasks. The work on place-based map evaluation using Fiducials presented in this thesis (see Section 5) is often applied to such a maze.

Other robotic capabilities that are intended to be standardized but that are not performed in standard test elements include the radio communication range tests, tow tests and standards for measuring

Figure 2.15: Standard test elements developed by the National Institute of Standards and Technology (NIST). On the left three stairs with different angles (35, 40 and 45 degree) can be seen while the right picture shows a test element used for directed perception and manipulation tests (from [Jacoff, 2011]).
Figure 2.16: Three locomotion test elements during the Response Robot Evaluation Exercise Disaster City 2012: The crossing ramps, sand and gravel.

the cache packaged weight and volume.

Figure 2.17: A symmetric step field test element with a Pointman robot during the Response Robot Evaluation Exercise Disaster City 2012.
2.6 Map Evaluation Algorithms

Figure 2.18: Slalom test with the robot at the start (left) and the robot traversing the corridor (right). Small sticks are scattered on the way to irritate the odometry of the robot.

2.6.2 Test Method Attempt for Map Evaluation: The Slalom Course

An approach to test the mapping performance of robots in a standard test method is described here.

It is an example of a reproducible test metric that robots have to complete. In this test element, the robot is asked to map a special Slalom course. The metrics for the slalom course are straightforward and a corresponding test arena could easily be rebuild. Without using SLAM, the demands on the robots odometry are very high due to the many rotations during the course. The metric for the map quality of the slalom course is the mean square error of the two far-end corners of the course. Figure 2.18 shows the slalom test arena at Jacobs and Figure 2.19 presents two different maps generated by the Jacobs Rescue Robot.

This test is simple and easy to reproduce, but it has obvious limits. It is quite restrictive regarding the robot size (different sized robots have different difficulties) and too small to test the performance of good SLAM algorithms. It also does not include loops that influence SLAM algorithms. Furthermore

Figure 2.19: The results of two different mapping algorithms, concurrently running on the same robot, namely once an evidence grid (left) and once a state of the art SLAM algorithm (right). Being able to use either of the two approaches allows to trade processing speed for precision.
it is not general enough - map metrics that can work in large-scale (application) environments are much more interesting.

2.6.3 State of the Art

Compared to the work on mapping and SLAM, relatively few publications regarding map evaluation exist. One of the earliest systematical approaches was presented by Lee [Lee, 1996]. In his thesis, he defines five properties that a map metric should have:

1. The metric must be clearly defined.
2. The metric must be multi-valued.
3. The metric must reflect the purpose of the map.
4. The metric must balance coverage and detail.
5. The metric must be applicable during the construction of the map.

The “clearly defined” property simply means, that it cannot rely on subjective human judgment but instead should be mathematically defined in such a way, that it generates reproducible and repeatable results. With “multi-valued” Lee stipulates that the values returned by the metric should be of a numerical number type and not just a boolean value.

Furthermore Lee asks that the metric should reflect the purpose of the map. This is, because different applications have different requirements on the maps. So a map metric has to measure what is important for the specific application. Already Leonard and Durrant-Whyte [Leonard and Durrant-Whyte, 1992] realized this and formulated it as follows: “We feel the ultimate test of a map is not ‘does it look good?’ but ‘how accurately, adequately or quickly can X be performed with it?’ ”

In his thesis, Lee proposes the use of the two map attributes coverage and detail, and he is demanding that those are balanced according to the needs of the application. In Section 2.5.2 more map quality attributes have been introduced. Lee’s rule should thus be re-formulated to “The metric must balance the map quality attributes (according to the application)”. In this thesis, it is also proposed that these attributes describe a sufficient amount of properties of maps such that they can be, together with a proper weighting, applied to any application. Thus this combination of quality attributes can always “reflect the purpose of the map”.

Lee used the map metric to tune and evaluate his mapping and exploration algorithms. It was thus important for him to run the map evaluation during the construction of the map. Although this feature is nice to have, it should not be a strict requirement for a map metric.

Lee’s proposed map metric works by using a ground truth map. He selects different locations in the maps and calculates paths between those points. The paths from the ground truth map are then compared to the paths of the generated map.

Recently a number of approaches to the map evaluation problem have proposed. Chandran-Ramesh and Newman [Chandran-Ramesh and Newman, 2008] work in 3D on planar patches. They detect suspicious and plausible arrangements of those planes and classify the map accordingly. So assumptions about the environment, e.g., the presence of planar walls, are made, and the local consistency of this assumption is quantified. A 2D version of their approach also works on lines instead of
planes. The algorithm does not make use of any ground truth information. Thus a map which looks nice might get a good score even if it seriously broken at some point. Also, this approach will not work in unstructured environments, for example in the search and rescue domain.

Kümmerle et al. [Kümmerle et al., 2009] proposed a method that does not compare the maps themselves but compares the robot path estimated by the SLAM algorithms with the ground truth path of the robot. This method is an excellent map metric if the preconditions that this algorithm demands are met. Instead of having to have a ground truth map, now a ground truth robot path is needed. As discussed in the paper, that usually implies human involvement or a good tracking infrastructure. Especially in an event like RoboCupRescue, where the same environment is explored by different robots (using completely different paths), having a ground truth map representation is more easy to get. The second problem is that, next to the actual map, one needs to obtain the actual pose estimations from the SLAM algorithm.

Kleiner et al. [Kleiner et al., 2009] used a similar method. During the RoboCupRescue Interleague Challenge 2009 (see Section 7.4 for a description of the 2010 challenge) scan data was logged from two LRF sensors and an inertial measurement unit (IMU) in the RoboCupRescue maze, using a real robot. The ground truth path data was generated by manually accepting or rejecting individual scan poses and matches that were proposed by a SLAM algorithm. The mapping algorithms were fed this data using the same interfaces as the RoboCupRescue Virtual Robot Simulation server. The poses estimated by the mapping software were then compared to the ground truth poses, using the relative displacements between the poses as the evaluation criteria.

Wulf et al. [Wulf et al., 2007] suggest a similar method. Here the ground truth path is generated using manually supervised Monte Carlo Localization of the 3D scans working on surveyed reference maps. A similarity of those algorithms with the map evaluation algorithms that will be developed in Chapters 5 and 6 is the assumption, that correct localization is a sufficient enough indicator for a good map quality.

SLAM algorithms usually generate uncertainty information about their map. Frese [Frese, 2006] uses this value from the final map as a map metric. This evaluation approach relies on the correctness of the calculation of this value. But the map metric should not rely on the test subject (the SLAM algorithm) itself for its calculation. Also, different mapping algorithms and their maps cannot be compared this way, especially since this uncertainty information is part of the SLAM algorithm but usually not saved in the map.

Using a reference map, also called ground truth map, is a popular way to evaluate maps. Those are very good maps that were generated from a ground truth layout or survey or by another mapping algorithm with known high quality performance.

A very early work using a ground truth map was done by Moravec and Elfes [Moravec and Elfes, 1985]. They use cross entropy to relate the content of each cell of a grid map to content of the corresponding cell in the ground truth map. The main disadvantage of this method is, that only the co-located cells are compared. Thus the maps have to be very similar in order to get meaningful results and small errors such as a slight rotation of the map will result in bad values.
Yairi [Yairi, 2004] proposed to use Least Mean Squares of Euclidean Distances (LMS-ED) to measure the distance between similar points from both maps. This method is computationally expensive and not convenient to use on grid maps.

Lakaemper and Adluru [Lakaemper and Adluru, 2009] also use a comparison with a ground truth map. They create virtual scans out of the target map and measure an alignment energy as map metric. The approach makes use of line- and rectangle-detection which might not be available in unstructured environments. It measures topological correctness and can also quantify the global correctness, so it is using the map attributes the Global Accuracy and the Local Consistencies.

The metric of Wagan, Godil and Li [Wagan et al., 2008] is a feature based approach comparing a map to a ground truth map. A Harris corner detector, the Hough transform and Scale-Invariant Feature Transform (SIFT) are used to extract features from both maps. The features are pairwise matched based on their distance. The quality measure is the number of matched features in the map versus the number of ground truth features. The feature detectors are quite vulnerable to the actual method of rendering the LRF scans. As mentioned in the paper, already changes like noise, jagged lines and distortions pose problems to the feature detectors. This approach can thus only be applied to nearly perfect maps.

Pellenz and Paulus [Pellenz and Paulus, 2008] also propose to use feature extraction. Next to using Speeded Up Robust Features (SURF) they also extract rooms as features from the grid map. The extracted rooms are matched with those from the ground truth map, taking into account the topology and the size and shape of the rooms. The average match error per feature is then the quality metric. This work and the Fiducial approach that will be presented as one map evaluation method in this thesis have in common, that they use few, large and easy to detect features (barrel as fiducial/room). The rooms have to be fully mapped in order to be used. Distortions could happen within one (bigger) room, potentially hindering the correct mapping to the ground truth room. The use of easy to distribute fiducials seems to be advantageous, since their density can be easily controlled and they can be concentrated in areas of interest. Barrels are small enough that they can be registered with a single scan but still big enough to show in low resolution maps. The room detection will not work in unstructured environments.

Balaguer et al. presents a solution for the problem of scoring maps in the RoboCup Rescue Virtual Robots competition. There, different aspects of the map as well as additional information like features are scored. One of them is the Skeleton Quality, which uses a skeleton map that is supposed to be provided by the team and not extracted from the grid map by the judges. No precise definition of this skeleton map is given. Two conflicting interpretations of this Skeleton Quality are presented. In [Balaguer et al., 2009a] the skeleton map seems to be a 2D line segment representation of the obstacles, while in [Balaguer et al., 2009b] a skeleton of the free space, similar to a Voronoi Diagram, is used. This topological information is used to identify false positives and false negatives, which are again used for scoring: “A false positive occurs when a node cannot be accessed whereas a false negative takes place when a clear topological location is available but has not been included in the skeleton map.” More details on how to actually compute the score are not given.
2.6 Map Evaluation Algorithms

Varsadan, Birk and Pfingsthorn [Varsadan et al., 2009] use an image based approach. The $\Psi$-similarity calculates the average Manhattan distance from the pixels of the ground truth map to the nearest pixel of the same color in the map and vice-versa. It allows a very fast computation of level of noise in the map by providing decreasing values for increasing amounts of common error sources like salt and pepper noise and the global effects of translation, rotation, and scaling. Good maps with a very good geo-reference can be evaluated quite nicely, but slightly broken maps or errors in the initial frame of reference lead to bad values, although the map might look very nice.

Birk [Birk, 2010] computes one interesting attribute of maps as map metric - the level of brokenness. This structural error is measured by using the $\Psi$-similarity to find parts of the map which do not fit. This area is then cut-out and registered with the ground truth, thus finding the frame of reference. This is repeated until all of the map is registered. The count of generated sub-maps is the brokenness. This brokenness information can, to some degree, also be found in topological map attributes, for example by comparing different Local Consistencies with each other or the Relative Accuracy.

A more indirect way to assess the quality of maps is to test how well they support applications relying on them. So benchmarks with certain tasks are run and the performance of the whole robotic application is measured. For example, the map coverage is directly linked to the exploration capacity, which in turn relies on a good map accuracy [Makarenko et al., 2002, Zelinsky, 1992]. Since the quality of the algorithms performing the applications task heavily influences the result, this method is not a good choice as a map metric.

Various different map metrics have been proposed in literature. But no generally accepted solution has been developed yet. This is because of the different requirements that various applications ask for and also because the metrics presented are often limited to certain conditions. Through the use of the different map quality attributes and a general applicability to grid maps, the map evaluation approaches that are developed in this thesis aim to provide universal map evaluation metrics.

Three main approaches to the evaluation of mapping systems can be identified: path-based, place-based and structure-based. Path-based methods use the localization result from the SLAM algorithm and compare this to the ground truth path of the robot and are well established [Kümmerle et al., 2009, Kleiner et al., 2009, Wulf et al., 2007]. Place based approaches compare the location of an identified place or feature with its known ground truth position [Lakaemper and Adluru, 2009, Wagan et al., 2008, Pellenz and Paulus, 2008]. The topological structure of the represented environment is used for scoring in the structure-based algorithms [Balaguer et al., 2009b, Birk, 2010].

For path-based approaches robot ground truth data has to be available, while the place- and structure-based algorithms need ground truth data about the environment (e.g. a map). Collecting the a qualitatively good robot ground truth path is typically quite difficult. Also the mapping solution has to provide the location estimates - synchronized to the robot ground truth path.

The robot ground truth data is subjective to the path the robot actually took, which, for example, could have been determined by an exploration algorithm. For every run this data has to be collected, and since the paths the robots took usually differ, it is then difficult to compare the results of the different robots. If a path-based approach wants to also calculate a coverage value, it additionally needs to have some environment ground truth data available.
Environment-based ground truth data, on the other hand, only has to be collected once and can then be applied to maps created by many different robots. Only with this data it is also possible to give an estimate about the area covered in the map. The place-based and the structure-based map evaluation approaches compare the environment-based ground truth data to the maps, while the path-based methods do not make use of a map (a SLAM algorithm needs a map though).

All three approaches for the evaluation of mapping systems are explored in this thesis. In Chapter 3 an image registration algorithm that was developed at the Jacobs Robotics Group is evaluated using the path-based approach and simulated input videos for perfect ground truth path data.

The place-based and structure-based approaches developed in this thesis provide a more general solution: They work on 2D grid maps and return not a single value but the map attributes from Section 2.5.2. The Fiducial Approach from Chapter 5 is a place-based method which finds artificial features (barrels) in the map and compares their location to the known ground truth locations. The structure-based solution from Chapter 6 uses graphs of the topology of the maps and matches and compares those graphs (one from a ground truth map and the other from the map to be evaluated).
Chapter 3

Path-Based Evaluation of iFMI Mosaicing

In this chapter an example for a typical mapping algorithm evaluation is given. Here not the map is evaluated but a path-based approach is used that compares the robots pose estimates with ground truth path data. The use of videos or sequences of images for mapping is the application of interest here. We concentrate on videos captured with aerial or underwater vehicles, because the technique presented here performs best with a top-down view on the environment. The algorithm developed in the Jacobs Robotics Group, called iFMI, is used to register the rotation, scaling and translation along the image plane between two images. Using this information, different applications such as photo mapping, image stabilization or motion detection are possible. The use of GPS or other localization methods is usually not precise enough to allow the aforementioned applications.

For the image registration different methods have been proposed. Often feature-based approaches for the registration of the 6-DoF pose changes are used [Kim and Eustice, 2009, Caccia et al., 2009, Pizarro and Singh, 2003, Gracias et al., 2003]. In many cases, feature descriptors like SIFT [Lowe, 2004, Lowe, 1999] or SURF [Bay et al., 2008], the Lucas-Kanade point tracker [Gracias et al., 2003, Shi and Tomasi, 1994] or intensity patterns [Mikolajczyk and Schmid, 2003, Gool et al., 1996] are used. These methods are best suited for feature-rich environments. But there are often quite homogeneous environment to be mapped like grass, asphalt or sandy ground. Those areas often do not have enough salient features such that the feature based approaches fail.

Image correlation approaches do not require the computation of image features and are therefore often more reliable than feature-based methods. Those approaches were used especially in the underwater settings [Richmond and Rock, 2006, Pizarro et al., 2001, Marks et al., 1995]. Unfortunately with those algorithms, it is no longer possible to compute as many registration parameters. They are usually limited to two, or four motion parameters: the 2D translation [Richmond and Rock, 2006] or the 2D translation, rotation, and scale [Marks et al., 1995].

Figure 3.1 shows two robots from the Jacobs Robotics Lab that are using the algorithms presented in this chapter.

In the first section of this chapter, a short excursus on the improved Fourier Mellin Invariant (iFMI) that has been developed by our group is presented. In Section 3.2 it is then shown how this algorithm is used to generate maps. Finally, a path-based evaluation of the performance of the iFMI algorithm is provided in Section 3.3.
3.1 Improved Fourier Mellin Image Registration

The Improved Fourier Mellin (iFMI) image registration is using a Fourier Mellin Invariant (FMI) [Chen et al., 1994, Reddy and Chatterji, 1996] with two modifications that have been developed at the Jacobs Robotics Group [Buelow et al., 2009]. The two improvements are a) a logarithmic representation of the spectral magnitude of the FMI descriptor and b) a filter on the frequency where the shift is supposed to appear. Those changes are analyzed and validated in [Buelow and Birk, 2009].

The iFMI provides 2D translations, rotations, and scaling, i.e., registrations when the robot is moving horizontally, rotating its yaw, and changing its altitude. It requires in theory a strictly down looking camera and a flat world hypothesis is assumed. But as shown in the experiments, these constraints do not have to be strictly fulfilled to achieve meaningful results.

The basis for the Improved Fourier Mellin (iFMI) is a Phase-Only Matched Filter (POMF). The POMF calculates the signal shift based on the phase difference in the spectra of two video frames. The POMF results in theory in a Dirac pulse that indicates the shift of the signal. If this pulse is clearly detectable as a maximum, it, or more precisely its location, correlates with the transformation parameters between the images.

Both scaling and rotation can be detected between two images, because they are decoupled. This is because the spectrum magnitude is invariant in the translation. That way, the 2D spectrum rotation and scale, which are re-sampled in polar coordinates using a logarithmic radial axis, can be used to detect the rotation and scaling parameters independent of the translation. The translation is afterwards determined by re-rotating and re-scaling one of the input frames and then detecting the signal shift using a FFT.
3.1 Improved Fourier Mellin Image Registration

3.1.1 Principles of the Fourier Mellin Transform

The Fourier Mellin Transform applies a Phase-Only Matched Filter (POMF) on two 2D signals. The 2D signals are the intensity of the 2D images to be matched. The POMF correlation approach makes use of the fact that two shifted signals that are having the same spectrum magnitude are carrying the shift information within its phase. This way a translation between two images can be determined.

But there might also be rotation and/or scaling between two images. Those have to be detected and compensated for first, in order to be able to apply the POMF to detect the translation. Incidentally, the POMF can also be used to detect the rotation and scaling by re-sampling the 2D signals into a polar-logarithmic coordinate system. The re-sampling does not necessarily have to be done in the same resolution as the 2D signals. Thus the algorithm might work with two different resolutions: The image-resolution is the resolution of the 2D matrices with the image intensity values. This resolution might differ from the input resolution of the video frames, so either a cut-out or a resize approach has to be used to get from the input resolution to the image resolution. The second resolution that is being worked with in the iFMI algorithm is the polar-log-resolution of the matrix that is the result of the polar-logarithmic re-sampling.

The POMF works as follows:

1. Calculate the spectra of two corresponding image frames (Figure 3.2).
2. Calculate the phase difference of both spectra.
3. Apply an inverse Fourier transform of this phase difference.

A problem is, that the resulting shifted Dirac pulse deteriorates when the signal (image) content of both input signals differs. As long as the inverse transformation yields a clearly detectable maximum, this method can be used for matching two signals. So it has to be taken care of, that there is enough overlap between the images to avoid too big differences in the input signals. This relation of the two signals phases is used for calculating the Fourier Mellin Invariant Descriptor (FMI).

To detect the scaling and rotation between the images, the scaling of the function/signal using a logarithmically deformed axis is transferred into a shift of its spectrum. The spectrum’s magnitude is
logarithmically re-sampled on its radial axis and concurrently the spectrum is arranged in polar coordinates exploiting the rotational properties of a 2D Fourier transform. Scaling and rotation of an image frame are then transformed into a 2D signal shift where the 2D signal is actually the corresponding spectrum magnitude of the image frame. This intermediate step is called the FMI descriptor [Buelow and Birk, 2009].

Figure 3.2 shows a pair of spectra from two image frames which are rotated, scaled and translated. Furthermore Figure 3.3 shows the corresponding polar/logarithmic re-sampled spectra from 0 to 180 degree from Figure 3.2. The periodical shift of the image content on the x axis is clearly visible, indicating the underlying rotation. To a smaller extent, on the y axis a shift of the image content is also visible, indicating the underlying scaling.

The iFMI algorithm to determine the rotation, scaling and translation parameters between two input images then works as follows:

1. Sample the intensity values of a square part of the input image into a square 2D matrix (in image-resolution).
2. Calculate the spectra of the two signal matrices.
3. Calculate the magnitude of the complex spectral data.
4. Re-sample the spectra to polar coordinates (to polar-log-resolution).
5. Re-sample the radial axes of the polar spectra logarithmically.
6. Calculate a POMF on the re-sampled magnitude spectra.
7. Determine the corresponding rotation/scaling parameters from the Dirac pulse.
8. Re-size and re-rotate the corresponding image frame (the 2D signal) to its reference counterpart (working in image-resolution again).
9. Calculate a POMF between the reference and re-rotated and re-scaled image.
10. Determine the corresponding x,y translation parameters from the Dirac pulse.
3.2 iFMI Mapping

Photomaps or mosaics are image based, metric representations for providing an overview of the environment. The basic approach is to take overlapping pictures of the area and display them nicely matched next to each other. This is related to visual odometry, because the vehicle motion is estimated in this process. Finding a template in an image is also known as image registration [Fitch et al., 2005, Stricker, 2001, Dorai et al., 1998, Brown, 1992, Alliney and Morandi, 1986, Lucas and Kanade, 1981, Pratt, 1973, Rzhanov et al., 2000, Rzhanov et al., 2006]. But since the robot usually does some non-trivial motion, finding out the overlapping parts of images is not sufficient to solve this task. Image stitching [Lowe, 2004] uses features to stitch images together in a similar manner. The term mosaicking is also widely used [Richmond and Rock, 2006, Pizarro and Singh, 2003, Gracias and Santos-Victor, 2001, Marks et al., 1995].

In order to be able to apply the iFMI approach to live real world missions, the author of this thesis reimplemented the MatLab iFMI to C++. The implementation uses either gsl [gsl, 2011] or fftw [Frigo and Johnson, 2005] for the computations of the FFTs. It is multi platform (Linux, Mac, Windows possible but not tested), multi-threaded, supports live mapping and recording as well as mapping after the fact using recorded data. The approach of maintaining the map as a tiled image supports the generation of very large maps without any performance losses. Using the functions from the Jacobs Intelligent Robotics Library (jirlib), we later on added support for graph-based SLAM (see

![Image of a rubble pile at Disaster City generated at RREE-2010 from 630 video frames.](image)

Figure 3.4: A map of a rubble pile at Disaster City generated at RREE-2010 from 630 video frames.
The mapping software uses the iFMI algorithm as follows. First a reference image $I_0$ is acquired. For that and all following frames the distortions of the capture device have to be removed from the image. After that is done, $I_0$ provides the reference frame $F$, which is also the initial pose of the robot in the map $p_0$. For every following image $I_k$ the transformation $T_{k-1}^M$ between the the predeceasing frame $I_{k-1}$ and $I_k$ is determined using the iFMI approach. The robot pose is now updated to $p_k$ using $T_{k-1}^M$ and $p_{k-1}$. Image $I_k$ is transformed (scaled and rotated) to $F$ (the initial frame of reference) by $T_0^F$ and added at the correct position in the photo map. This is repeated for every image.

Figure 3.4 shows a map from the Response Robot Evaluation Exercise 2010 at Disaster City in College Station, Texas [TEEX, 2008]. The data collected by the author of this thesis consist of 630 video frames and depicts one of the rubble piles. Our contributions to the area of mosaicking and image stabilization have been published for underwater [Buelow et al., 2009, Pfingsthorn et al., 2010a] and aerial [Buelow and Birk, 2009, Schwertfeger et al., 2011a] robots.

### 3.2.1 iFMI SLAM

As explained in the introduction to Simultaneous Localization and Mapping (SLAM) (Section 2.2.2), the area of Visual SLAM (VSLAM) does not only take the sequential image registrations into account but also uses informations from earlier images to enhance the map quality. This is done, for example, by doing some loop detection and then registering the images against the map or against older images.

Visual SLAM is an active area of research [Davison et al., 2007, Eustice et al., 2006, Angeli et al., 2006, Davison, 2003, Castellanos et al., 2001, Castellanos et al., 1998, Broida et al., 1990, Grandjean and Robert De Saint Vincent, 1989], with also some work for the underwater domain [Steder et al., 2008, Angeli et al., 2006].

The Jacobs Robotics Group has developed an uncertainty metric for the iFMI spectral registration [Pfingsthorn et al., 2010b]. This allows to embed the iFMI registration in SLAM, which further improves the mosaicking results [Birk et al., 2011]. This was used on the same data as from Figure 3.4. There pose-graph-based mapping was combined with a loop detection algorithm. Using the uncertainties the graph was then optimized and the map improved - see Figure 3.5 for the graph [Birk et al., 2011].
3.2 iFMI Mapping

Figure 3.5: The map with loop closures edges for SLAM (from [Birk et al., 2011]).
3.3 Performance Evaluation of iFMI

The iFMI algorithm is the core of different video registration applications. In [Schwertfeger et al., 2010a] its performance regarding different parameters and circumstances were investigated. This is a good example for a typical path-based evaluation approach, which means that not the result of the mapping system, the map, is scored but the process of procuding this map, the mapping algorithm, is evaluated. As we have seen, the algorithm and its implementation (Figure 3.6) analyzed here are applied not only to the underwater domain [Buelow et al., 2009] but also to unmanned aerial vehicles [Buelow and Birk, 2009]. The purpose of this work is to study the effects of the different resolutions throughout the iFMI algorithm, the methods on how to change these resolutions and and other properties. Changing the resolutions allows to trade off computation speed versus the quality of the map. In the following experiments, ground truth (path- and video-) data is generated in a simulation. The map is represented in a pose graph. The map evaluation metric used in the experiments is the comparison of the end-pose of the mapping algorithm with the known ground truth end-pose.

3.3.1 Generating Ground Truth Data

The ground truth data, a simulated dive of an autonomous underwater vehicle (AUV) pointing a camera to the bottom, is generated by taking a high resolution image of the ocean floor and cutting out lower resolution images representing the frames of a video stream. The frames are cut out using the following parameters:

- Position: the x and y position of the center of the image
- Orientation: the orientation or angle of the frame
3.3 Performance Evaluation of iFMI

Figure 3.7: The figure eight shape. A path with 175 nodes (frames). It is actually path number 1 in Table 3.3.2.

- Height: the height corresponds to a scaling of the cut-out image. The minimum height is 1, which corresponds to no scaling, thus avoiding pixel artifacts in the simulated video data.

A program is used to generate paths of different shape and with different parameters like size of the shape, step width (between the frames), orientation, random variance of pose and orientation and possibly changes in height. Figure 3.7 shows a typical path for the “figure eight” used in all experiments. Figure 3.8 shows one of the four images used for cutting out the frames while Figure 3.9 shows a map generated out of those frames (using a 512x512 pixel resolution).

We are well aware that the data generated lacks any errors occurring in reality like noise, dirt, moving objects, perspective errors or pitch and roll. Nevertheless the experiments performed deliver valid data because they always judge the effects investigated by comparing results gained with equally perfect input data.

3.3.2 Experiments

In the experiments, four different images are used with four different paths, leading to 16 different simulated video streams. Two images have few features and relatively uniform terrain while the other two are rich of features. Two of the paths feature no change in height while the other two oscillate twice respectively three times between height 1 and 1.5. The properties of the paths used can be found in Table 3.3.2.

The smoothness describes how the perfect circles of the figure eight are deteriorated. Smooth paths are not deteriorated - only path number 4 includes additional changes in height. In paths 1 and 3, random changes are added to the poses (including orientation). As a result, the maximum step width of those paths are higher than the average step width.
Path-Based Evaluation of iFMI Mosaicing

Figure 3.8: A uniform (left) and a feature rich (right) image used for extracting the simulated video stream (both from MARUM).

We would like to thank the Marum - Center for Marine Environmental Sciences, University of Bremen, for providing the underwater images.

Figure 3.9: A map generated out of the simulated video stream with 175 frames using a resolution of 512x512 and the figure eight path shown in Figure 3.7 (path number 1 in Table 3.3.2).
3.3 Performance Evaluation of iFMI

<table>
<thead>
<tr>
<th># frames</th>
<th>Average step width</th>
<th>Maximum step width</th>
<th>Height</th>
<th>Height Frequency</th>
<th>Smoothness</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>175</td>
<td>41</td>
<td>1</td>
<td>-</td>
<td>random</td>
</tr>
<tr>
<td>2</td>
<td>211</td>
<td>30</td>
<td>1</td>
<td>-</td>
<td>smooth</td>
</tr>
<tr>
<td>3</td>
<td>129</td>
<td>52</td>
<td>1 - 1.5</td>
<td>2</td>
<td>random</td>
</tr>
<tr>
<td>4</td>
<td>211</td>
<td>30</td>
<td>1 - 1.5</td>
<td>3</td>
<td>smooth</td>
</tr>
</tbody>
</table>

Table 3.1: Properties of the paths used.

The images are cut out of the big input image with a resolution of 512x512 pixel. The other resolutions tested are 448, 384, 320, 256, 192, 128, 92 and 64. In these experiments the polar-log- and image-resolution are always equal. In order to use a smaller resolution two possible approaches are investigated:

- **Resize**: The original 512 pixel image is resized. Thus the overlap stays the same but the details vanish.
- **Cut-out**: The image registration only takes the center of the original frame. The overlap decreases, possibly too much, but a high accuracy is possible if enough overlap is present.

All 16 video streams are run with both approaches and all nine resolutions, leading to a total number of 272 mapping runs - for the 512x512 resolution the resize approach and the cut-out approach deliver the same result (since the resolution is not changed). The resulting camera path generated during the mapping is then compared to the ground truth path. The errors in the position and the orientation are determined together with the standard deviations of those errors.

**Runtimes**

The runtime of the FMI algorithm does not depend on the content of the image and is thus constant for a given pair of resolutions (image and polar-log). The runtime for the pure image registration depends to a great extend on the specific FFT library used. In Table 3.3.2, first the speed of the pure registration is given, which scales with the resolution. The last values show the speed of the GUI.

<table>
<thead>
<tr>
<th>Image &amp; polar-log resolution</th>
<th>Registration</th>
<th>GUI</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>gsl fps</td>
<td>fftw fps</td>
</tr>
<tr>
<td>64</td>
<td>640.8</td>
<td>1168.4</td>
</tr>
<tr>
<td>96</td>
<td>281.0</td>
<td>412.8</td>
</tr>
<tr>
<td>128</td>
<td>159.1</td>
<td>272.5</td>
</tr>
<tr>
<td>192</td>
<td>67.7</td>
<td>117.6</td>
</tr>
<tr>
<td>256</td>
<td>34.8</td>
<td>66.3</td>
</tr>
<tr>
<td>320</td>
<td>21.8</td>
<td>40.6</td>
</tr>
<tr>
<td>384</td>
<td>14.9</td>
<td>27.4</td>
</tr>
<tr>
<td>448</td>
<td>10.2</td>
<td>19.3</td>
</tr>
<tr>
<td>512</td>
<td>7.00</td>
<td>14.3</td>
</tr>
</tbody>
</table>

Table 3.2: The speed on an Intel Core 2 2.8 GHz CPU. The registration itself did not make use of multi threading.
which incorporates various tasks like loading, converting, and undistorting of the images, the actual registration, as well as merging the frames according to the transformation parameters together in one map and displaying it. A screenshot of the GUI is given in Figure 3.6.

Figure 3.10: Average over the errors of all video streams of a given resolution. The standard deviation of the errors is also shown. It is symmetrical but one side is omitted here to ease readability. Please also note the scale and range of both y-axis in this and further diagrams.

### 3.3.3 Results of the iFMI Performance Analysis

In the discussion about the results of the experiments, two different error measurements are of relevance. The first is the error of the orientation as absolute value. The orientation (and also the scale, or height) is determined in iFMI using polar-logarithmic re-sampling in the polar-log-resolution. The absolute difference (in degrees) between the calculated orientation of two successive frames and the actual change in orientation as given in the ground truth path is averaged for every video stream of a given resolution and represents the error of this step.

The second error is the error in translation. After finding the rotation and scaling one of the input frames is transformed according to those values such that afterwards there is only a translational difference between the transformed and the other frame. Thus this step is vulnerable to errors occurring already during the determination of the scale and orientation. For these experiments, the translation error was not assessed independently of the polar-log-resolution. Again, the geometric distance (an absolute value with the unit 'pixel') between the calculated translation and the actual translation defined in the path is averaged for every video stream of a given resolution. This is the translation error.

Looking at Figure 3.10, we see the two results for the average error at a given resolution averaged
over all 16 video streams. The values are the average inaccuracy between two successive frames. Those errors accumulate over time. One could say that in general, down to a value of 192 or even 128, for both resolutions (polar-log- and image-resolution) the algorithm performs quite well.

For the following figures, the standard deviations are omitted either because those are hard to interpret on a logarithmic scale (Figures 3.11(a) and 3.11(b)) or because the graphs would be hardly readable anymore (Figures 3.12, 3.13 and 3.14). Those deviations are anyways very similar to those in Figure 3.10 and scale with the value of the error.

**Resize vs cut-out**

In Section 3.3.2 two approaches to lower the resolution of the input video stream to gain processing speed are discussed: resize and cut-out. Figures 3.11(a) and 3.11(b) compare their performance for different resolutions. The graphs can be explained as follows: For the orientation both approaches have nearly the same errors down to 192x192 pixel. The resized frames perform that well since even small rotations lead to big changes in the outer parts of the image such that the higher details of cut-out lead to no advantage. In lower resolutions cut-out is performing much worse than resize. This is because here the overlap is decreasing too much for cut-out while it does not change at all for resize.

If we consider 192x192 as still acceptably performing for cut-out (which can also be seen in the graph for the translation in Figure 3.11(b)), we can calculate the needed overlap, knowing that the average step width is 30 (see Table 3.3.2): \(1 - \frac{30}{192} \approx 85\%\). With less than this overlap it is likely that other peaks in the Dirac-domain have higher energy than the “correct” one at least once for all the registrations in the video stream. Then the determined result is not just inaccurate but completely wrong. If, in real world applications, further error sources appear more overlap is hence strongly beneficial for iFMI.

The interesting graph is in Figure 3.11(b) for the translation. As long as there is enough overlap it is preferable to cut-out. This is due to the effect that nearly no information is lost in cut-out in resolutions down to 192 since the overlap is always sufficient. Resize, on the other hand, constantly loses accuracy due to the scaling and the interpolation. Even at 128 both approaches perform nearly equally well.

**Further results**

The experiments were deliberately planned such that the other properties of the iFMI algorithm can be investigated. Figure 3.12 is comparing the performance of the image registration between the video streams made from the feature rich and more uniform pictures. It can clearly be seen that both error developments are very similar, leading to the conclusion that the content or the feature richness of the scene does not affect the registration accuracy (as long as there is any non-ambiguous structure).

Figure 3.13 compares the algorithm’s performance between the flat paths and those which oscillate in height. It can be seen that the determination of the orientation (and thus also the scale as it is done in the same step) is not effected by the height changes. The accuracy of the translation suffers from the scaling and interpolation needed in the oscillating paths.

The most surprising result was the one shown in Figure 3.14. Here the effects of the smoothness property of the paths as described in Section 3.3.2 and Table 3.3.2 are investigated. Since the algorithm can handle any combination of rotations and translations (including height changes) it was expected...
to see no differences. Alas, the orientation of the smooth path is showing a strange pattern. It was also checked in the raw data and found that this is not due to one outlier run but consistent with all smooth runs. This is an effect of the circle radius and step width (leading to a certain, constant orientation change in each step) interfering with the chosen resolution. This could lead to rotations being directly on one pixel (matrix element) in the Dirac-domain in some cases (leading to better
results than normal) and in other combinations laying more or less between two (or four) pixel, thus performing worse than average.

### 3.3.4 Interpretation of the Experiment Results

The properties of the iFMI algorithm, especially with respect to the resolutions used, were analyzed here. This is a typical example of using a path-based evaluation, i.e. not the map itself but the process of producing this map are evaluated. It was found that resolutions of 192x192 and up generally perform sufficiently well. It was also concluded that the minimum overlap at this resolution should be at least 85%. With these parameters in mind, the investigated mosaicking program can run at more than 25Hz on a modern laptop, including all tasks like data acquisition, image undistortion, live display, and storage. It should be noted that in this setup only the image registration uses 192x192 for both resolutions - everything else can be done with this speed in the original size (512x512). It can also be seen that changing the resolutions allows to trade off computation speed versus the quality of the map. Furthermore, it was shown that the iFMI algorithm can work with sparse and uniform input data and can also handle changes in height decently well.

![Graph](image.png)

Figure 3.12: Comparison of the results of feature rich and uniform images.
Figure 3.13: Comparison of the results of the flat and height oscillating paths.

Figure 3.14: Comparison of the results of the smooth and the randomized paths.
Chapter 4

Tools for Map Preprocessing and Human Quality Assessment

Path-based methods, such as presented in the previous chapter, are a well established approach to map evaluation, but have hard to meet pre-conditions: very good ground truth path data has to be available (which is especially difficult to gather if the maps of live runs of different robots in the same environment have to be compared), and the pose estimates of the SLAM solutions have to be available and synchronized with the ground truth data. Path-based approaches need to collect robot ground truth data while place-and structure-based algorithms need ground truth data about the environment (e.g. a map).

Robot ground truth data is subjective to the path the robot actually took (e.g. as determined by an exploration algorithm) and does not hold any information about how complete the environment was covered. It also has to be newly gathered for every run. Environment-based ground truth data, on the other hand, only has to be collected once and can then be applied to maps created by many different robots. Only with this data it is also possible to give an estimate about the area covered in the map. A path-based approach would have to additionally also have some environment ground truth data available to calculate a coverage score.

So a more general solution to map evaluation is to take the results of the mapping process, the map, for scoring and compare it to environment-based ground truth data. Those maps come in different formats and with different properties. Even two very good maps of the same area will have certain differences. Those may be due to the resolution that is used (e.g. very thin objects like cables or grass might be represented or not), the properties and height over the ground of the sensors used, dynamic objects (humans) walking through the environment etc.

Map evaluation is predominately interested in the correctness of the significant features of the map. If those are compared to a ground-truth-map, ideally only the objects present in this ideal map are represented in the maps. Thus, it is often useful to preprocess the maps before feeding them to a map evaluation algorithm. Also, it is of interest to provide tools for direct comparison of maps for humans.

The following sections assume two dimensional maps represented in an orthogonal, Cartesian grid. As explained in Section 2.3, the cells in this grid represent the probability that the region is “occupied”.
The map processing algorithms presented here are all implemented in the NIST Jacobs Map Toolkit. This is the software package that was developed during and for the research on this thesis. It is also used to allow Subject Matter Experts (SME) to compare the maps by hand. The tool is described in more detail in Appendix C.

4.1 Image Thresholder

The aforementioned grid $G$ is typically shown as an image. The probabilistic entries $G_{x,y}$ are then represented as a gray scale value with $0 \leq G_{x,y} \leq 255$, thus showing white for free to black for occupied cells. Other colors can be defined to represent additional information. This could be the start pose of the robot(s), robot paths, or locations of objects of interest. The unobserved area typically is defined as the gray value half way between the occupied and free value. Different applications have defined standards for this map representation [Pellenz and Paulus, 2008, Schwertfeger, 2010, DSTO, 2010].

The Image Thresholder generates a grid consisting of just two or three specific values or colors:

- $C_{\text{free}}$ Free (typically white): No obstacle
- $C_{\text{unknown}}$ Unknown (typically gray): Unknown, unobserved area (e.g. voids, also “content” of barrels)
- $C_{\text{occupied}}$ Obstacle (typically black): Obstacle (e.g. walls, barrel)

Figure 4.1: Detail of a map. The original is on the left while the thresholded map is continued on the right. One can see that the ghost-walls in the upper left, which appear due to a localization error, vanish in the thresholded map.
4.1 Image Thresholder

This is done by defining two thresholds $T_{\text{white}}$ and $T_{\text{black}}$ with $T_{\text{black}} \leq T_{\text{white}}$ and applying them in the following way:

$$\text{cell} = \begin{cases} 
C_{\text{occupied}} & \text{if cell} < T_{\text{black}}; \\
C_{\text{free}} & \text{if cell} > T_{\text{white}}; \\
C_{\text{unknown}} & \text{otherwise}
\end{cases}$$

Figure 4.1 shows the details of a map before and after thresholding.

In the NIST Jacobs Map Toolkit, the image thresholder is implemented as an algorithm plugin. The GUI for it shown in Figure 4.2 first offers the “1. Color to” option to manually set certain image colors to free, unknown or occupied. The second step is then to convert the image to gray scale. Two sliders for $T_{\text{white}}$ and $T_{\text{black}}$ are available in the third option. A live preview can be used to refine result.

After thresholding the maps can alternatively be represented in a different form: If one is just interested in occupied cells a planar set of occupied points (also known as point cloud) can be extracted.

Figure 4.2: The GUI for the image thresholder in the NIST Jacobs Map Toolkit.
4.2 Nearest Neighbor Removal

After thresholding, maps can contain a number of occupied cells which do not seem to belong there. Those erroneous entries are introduced by the different error sources, for example by the sensor noise, the SLAM algorithm, or odometry errors. But they could also represent real objects which might be not present in a ground truth map, which can be coarser than reality. Anyways, it is often needed to remove those outliers.

One way to do this is to make use of the k-nearest neighbor algorithm to sort the input points in increasing order of their average (squared) distances to their k nearest neighbors. After that, a certain percentage of points with the largest such distances can be deleted. For that, the NIST Jacobs Map Toolkit uses CGAL [cga, 2011], concretely the remove outliers method of the Point Set Processing package is used [Alliez et al., 2011].

4.3 Alpha Shape Remover

Another approach to remove outliers in the map is to use the alpha shape algorithm [Edelsbrunner et al., 1983]. It can be seen as a way to determine the shape that a point set $S$ is forming. The interesting fact with regard to map processing is, that this algorithm partitions the point set into nested sub sets with a frontier around each set. The length of this frontier is then used to determine whether the sub set is an outlier (and thus to be removed) or not.

4.3.1 Alpha Shape

Alpha shapes are defined over the point set $S$ and a parameter $\alpha$ (which is a squared radius), with $0 \leq \alpha \leq \infty$. The result is a set of edges between points which form the shape or frontier. $\alpha$ is the squared radius of the circles used to construct the alpha shape. An edge between two points is part of the frontier if one of the circles defined over those two points does not contain any other point.

In the implementation in the NIST Jacobs Map Toolkit the “2D Alpha Shapes” package of CGAL is used [Da, 2011]. This library returns one set of frontier edges. Points in the map which belong to small objects or are outliers will get an own frontier. So any objects (or set of points) farther away than $2 \sqrt{\alpha}$ to any other point will get their own frontier. For every point of the map the four points at $x, y \pm 0.25$ pixel are used to ensure that CGAL will create the frontier around every point.

A good value for $\alpha$ has to be found by the user. Generally, very small alpha values will remove small groups of points in the map even if those are close to big objects. But walls observed with just a few beams of an LRF could be represented with evenly spaced points in a map which would get removed with small $\alpha$ values, too. On the other extreme, big values for $\alpha$ will connect outliers to valid objects, thus forming a long frontier which will prevent the removal of those outliers.

The set of frontier edges has to be partitioned to subsets, grouping edges that belong to the same frontier. Towards the end, points within a frontier are to be removed. For that a polygon representation is advisable. So a set of polygons is computed out of the initial set of frontier edges.
4.3 Alpha Shape Remover

4.3.2 Polygonization

Every edge $e_k$ in the unordered set of frontier edges $F$ has a source point $e_{k,s}$ and a target point $e_{k,t}$. In order to efficiently find neighboring edges they are put into a special data structure with two nested maps. The key values are the integer floor values of the x- and y-coordinates of the source point of the edges. The mapped value is a list of all the edges whose source is in the integer coordinates. Those also have a boolean to indicate if they have already been used in a polygon.

Algorithm 2 Polygonization Algorithm – Iterating through all edges and connecting them.

```plaintext
for all edge $e_k \in F$ do
    if $e_k$ marked as unused then
        start new polygon
        set START and CURRENT to $e_k$
        while START source != CURRENT target do
            in edgeMap find edge whose source equals CURRENT target and set to CURRENT
            add point CURRENT target to polygon
            mark CURRENT as used
        end while
    end if
end for
```

Algorithm 2 shows the approach to create the polygons. For all frontier edges which were not yet used a loop is started in which the next connecting edge is added to the polygon till the start was reached again.

4.3.3 Point Removal

After the polygonization, the list of polygons is filtered for only those which are shorter than a maximum length parameter $l_{max}$. All points within the resulting set of polygons are to be removed. This is done by iterating through all points in the map and checking them for inclusion for all of the polygons. In order to speed this process up, first the bounding boxes for all polygons are computed. Thus the computationally more intense checking for inclusion in the polygon is only done for those points which are within the bounding box of said polygon.

4.3.4 Results

An example is given on a map produced during the Multi Autonomous Ground-robotic International Challenge (MAGIC 2010 - see Section 7.1 for a detailed description). The maps produced there have a resolution of 5cm - see Figure 4.3. The parameters for $\alpha$ and $l_{max}$ have to be determined by the user for every given scene, and a value of 1m for $\sqrt{\alpha}$ and 5m for $l_{max}$ have been found to be useful for this example map. They remove all outliers, including the barrels and the posts for a railing which are all not present in the available ground truth map. Another example is given in Figure 4.4.
Figure 4.3: Detail of a map from an autonomous robot competition (MAGIC, see Section 7.1). Polygons (α = 1m) longer than $l_{\text{max}}$ (= 5m) are depicted in blue and shorter ones in red. Points that were removed appear in gray. Those include barrels (top), railing posts (right-bottom), other robots (top left) and other small structures.

Figure 4.4: Detail of a map from RoboCupRescue Singapore. Polygons (α = 10cm) longer than $l_{\text{max}}$ (= 1m) are depicted in blue and shorter ones in red. Points that were removed appear in gray. Those include the center posts of the symmetric step field (see Section 2.6.1 for a description of this Standard Test Element).
4.4 Comparison between Nearest Neighbor Removal and Alpha Shape Remover

Here the Nearest Neighbor Removal from Section 4.2 and the Alpha Shape Remover (Section 4.3) are shortly compared. For that, the algorithms have been applied to a map from phase I of MAGIC. The Nearest Neighbor Removal has been run with a combination of three different percentage removal values (5%, 10% and 20%) and five different values for the number of neighbors used (10, 24, 50, 100 and 300). Details of the resulting maps can be seen in Figure 4.6.

One goal could be to remove the three lonely dots (stilts) from the bottom right. This is only successful when using 300 neighbors - regardless of the percentage of points removed. But with 300 neighbors also most of the structure inside the maze is lost. So it seems that there is no way to achieve the desired result with the Nearest Neighbor Removal approach.

In Figure 4.5 we see the detail of the map before it was processed and after it has been processed with the Alpha Shape Remover. The configuration value is 300 for the alpha value (the map has a resolution of 5cm per pixel) and 5m for the minimum polygon length. It can be seen that the the stilts have been removed successfully (red polygon) while leaving the structure of the maze intact. So at least in certain use cases the Alpha Shape Remover fits the desired tasks better than the Nearest Neighbor Removal.
(b) Alpha shape removal with 300 for the alpha value and 5m for the distance threshold.

Figure 4.5: Original and alpha shape removal approach.
4.4 Comparison between Nearest Neighbor Removal and Alpha Shape Remover

Figure 4.6: Nearest neighbor removal experiment. Shown is one detail from a map from phase I from MAGIC.
4.5 Make Hollow

For most algorithms handling map data, the computation time depends on the number of input points. It is thus desirable to remove unnecessary points, if that removal does not significantly alter the result of the algorithm. The Make Hollow algorithm removes occupied points that are surrounded by other occupied points, thus leaving the outer shape of walls and obstacles intact. There are map algorithms that just rely on this outer shape of obstacles and can thus ignore the inner points. Examples for those algorithms are the Alpha Shape Remover (Section 4.3) and the Voronoi Algorithm (Section 6.2).

The Algorithm 3 slightly modifies the convention that occupied pixels are depicted as black (value 0). Pixels to be removed get the value 1 but are still treated as occupied pixels during the first for loop. This is why in lines two and three of Algorithm 3 it is checked if the color is below 2. In the second loop those marked pixels are set to free (value 255 for white). Figure 4.7 shows one example of a map before and after the application of Make Hollow. The effectiveness, meaning the percentage of removed pixels, of this algorithm varies with the maps and scenarios it is applied to.

Algorithm 3 Make Hollow Algorithm – Mark all occupied pixels with four occupied neighbors and then remove the marked pixels.

```plaintext
for all pixels p do
  if |p| < 2 then
    if for all four neighbors n_p of p : |n_p| < 2 then
      |p| = 1
    end if
  end if
end for
for all pixels p do
  if |p| == 1 then
    |p| = 255
  end if
end for
```

Figure 4.7: A section of a typical RoboCupRescue map on the left (1393 points) and after the removal of the inner pixels on the right (1039 points - about 25% reduction).
Chapter 5

Place-Based Fiducial Approach

As already discussed, the previously presented path-based evaluation is a well established solution, but it is really difficult to employ in real robotics, due to the fact that the ground truth robot pose has to be logged exactly and compared to the pose estimate from the mapping algorithm. Using environment-based ground truth data instead of robot ground truth data has the advantage that this data has to be collected only once, that it is easier to compare different runs and that the coverage can be calculated.

So a more general solution is to evaluate the map. In a place-based map evaluation approach, the position of features of the environment are used [Wagan et al., 2008, Pellenz and Paulus, 2008]. There, naturally occurring features in the 2D map representation like corners or rooms are utilized. They compare the global positions of those features between a ground truth map and the robot generated map.

In a collaboration between the National Institute of Standards and Technology (NIST) and the Jacobs Robotics Group, the author of this thesis significantly contributed to the design and development of the Fiducial map metric that is using artificial markers (dubbed “fiducials”) that are placed in the environment. The roots of this idea had been developed during ICRA 2009 with Adam Jacoff from NIST. While the idea to use barrels as features that can be detected in maps originated from Mr. Jacoff, most other steps such as formalizing the approach and generating maps and results during experiments have been performed by the author. This work has been published in [Schwertfeger et al., 2010b] and [Schwertfeger et al., 2011b].

5.1 Fiducials

Using fiducials is one option to solve the problem of evaluating the quality of maps, without having to resort back to path information. Fiducials are artificial features placed in the environment which can be identified in the resulting map. Using the Fiducial approach, the map quality attributes Global Accuracy, Relative Accuracy, Local Consistencies, Coverage and Resolution Quality introduced in Section 2.5.2 can be measured. The only information needed to score maps are the (ground-truth) positions of all fiducials in the environment. Upon scoring, each fiducial has to be identified in the map together with its position.

This approach completely abstracts from all other information contained in the map like walls,
unexplored and explored areas and other features. However, given a dense enough distribution of fiducials, this method reflects the quality of those features well enough. This is because measuring the localization performance of the SLAM algorithm suffices, since applying sensor data to the map given perfect localization is typically easy.

The fiducials used in the experiments are cylinders placed in the environment. Usually when using the term fiducial scientists refer to reference points that are clearly uniquely identified. This is not the case in this algorithm, as the identification step is not simple and might, in worst case, lead to false results. Nevertheless this slight difference is ignored here and the term “fiducial” is used for the barrels. Those barrels can either be cut in half and (typically) placed on either side of a wall or two cylinders are separated by a short artificial wall. In order to ease the understanding, in the following representations of said cylinders in the actual maps will be refereed to as barrels since barrels since barrels were used as cylinder approximations in the experiments (see Figure 5.1). Fiducials are then the objects in the actual environment and its model - the ground truth map. As mentioned above, there are (usually) two fiducial-parts (A and B) respectively barrel-parts (A and B).

All attributes scored by the Fiducial approach have values between 0 and 100% where 0 means poorest quality while perfect results get a value of 100%. This allows to easily apply application dependent weights to the attributes to come to a simple overall score for maps consisting of just one number. The coverage, resolution quality as well as global and relative accuracies and the local consistencies can be determined using the Fiducial approach. However, first the fiducials have to be identified in the map. Although the Fiducial metric works in principle for 3D maps as well, 2D maps are considered for the rest of this work.
5.1 Fiducials

5.1.1 Identification of Fiducials in the map

The following steps are performed to find the fiducials in the map and to register their position:

1. **Rasterize**: Render the map to a two-dimensional grid with a sufficiently high resolution (if the map is already present in a raster format this step can be skipped).

2. **Colorization**: Use the Image Thresholder from Section 4.1 to remove all probabilistic entries in the grid such that there are exactly two color values left:
   - Free (typically white): No obstacle
   - Obstacle (typically black): Obstacle (e.g. walls, barrel)

3. **Identify barrel parts**: Find all obstacles which form parts of circles with the right radius. The minimum visible angular opening of the part-circle has to be $\frac{2}{3}$rd of the barrel.

4. **Assignment**: For each fiducial part assign one or none of the barrel parts identified in the previous step. Each of those barrel parts can be assigned to maximum one fiducial part.

5. **Determine Position**: For each barrel part assigned to a fiducial part compute the position of the center point of the circle forming the barrel. This is then the position of the barrel part. Thus the positions of two parts of a cut-in-half-barrel are just separated by the thickness of the wall.

5.1.2 Automatic Assignment between Barrel Parts and Fiducials

An automatic assignment of barrel parts and Fiducials was implemented. It makes use of the assumption, that those barrels and fiducials will have roughly the same global position. A first version of this mapping was done as follows:

An exhaustive search is performed over all possible mappings. The squared error from Horns algorithm [Horn, 1987] is used as the decision criterion which mapping is the best one. While this works well for maps with few fiducials, some more will make this approach unworkable, because of the combinatorial explosion of mappings and the resulting computational demands. So the improved algorithm uses a search that is guided and pruned. This is done by taking intermediate results of Horns algorithm and the use of backtracking after a threshold for the squared error has been reached. This way the automatic mapping works fast and quite reliable, even for big maps with many Fiducials.

The following three attributes use distances between two positions to measure the error. For those, there are maximum distances $d_{\text{max}}^{\text{attribute}}$ defined which are considered to be the worst case for the attribute. The values can, but do not have to be the same for those three attributes. Furthermore, the actual distance error $d$ can be discretized to certain values, for example the barrel radius. This can be done in order to avoid differences in scoring which are caused by the inherent error of the ground truth data and to put the resulting scores in bins of similar qualities.

5.1.3 Global Accuracy

For every barrel-part assigned to a ground-truth-fiducial part, calculate the distance $d$ to the (global) position of the corresponding fiducial. Distances $d$ greater than $d_{\text{max}}^{\text{accuracy}}$ are set to $d_{\text{max}}^{\text{accuracy}}$, such that later on no attribute can get a value above 1. The error $e$ is then calculated as $e = \frac{d}{d_{\text{max}}^{\text{accuracy}}}$. Average
over the errors for all those barrel parts. The value for the global accuracy is then $1 - e$ such that perfect maps get a 100% number.

### 5.1.4 Relative Accuracy

The error of the global accuracy is minimized (or the accuracy value maximized) by rotating, translating or even scaling the map. This can easily be done by just changing the positions of the barrel parts, thus eliminating the identification step for each iteration. For this Horns algorithm is used [Horn, 1987]. Often the value for the transformation is just the error in the start pose. If there was no agreement on a global frame of reference, only the relative accuracy can be computed and there can be no score for the global accuracy.

### 5.1.5 Local Consistencies

Local Consistencies are used to determine the correctness of positions of different local groups of features relative to each other. Those groups can be put in different classes. Usually those classes are defined over how difficult it is for the robot to travel from one group member to the other. One group typically consists of the two half barrels on either side of a wall. The difficulty can then be measured by the length of the shortest path that a robot has to drive to see both barrel parts. Those lengths can then be classified, for example in short, medium and long distance classes (resembling easy, medium and hard difficulties for creating consistent results).

So for all groups the distance errors between entries of a group are calculated. For each pair/groups where at least one barrel part has been found:

1. Calculate the geometric distance between the positions of the two barrel parts A and B: $d_{\text{barrel}}$.
2. If one of the barrel parts was not identified in the map set $d_{\text{barrel}}$ to a very high value.
3. Get the distance between the two corresponding (ground truth) fiducial parts: $d_{\text{fiducial}}$.
4. The absolute value of the difference of the distances from step 1) and 2) is the error for this group: $e = \min(d_{\text{max}}^{\text{consistency}}, |d_{\text{barrel}} - d_{\text{fiducial}}|)/d_{\text{max}}^{\text{consistency}}$.

The “short range consistency” is thus one minus the average of the error of all short range groups while the “long range consistency” is one minus the average error of the long range groups.

Using barrels or half-barrels on opposite sides of walls has two advantages. Firstly, it is very easy to judge the quality of the those pairs by just looking at the map and checking if those barrels are properly aligned and form a good circle without big gaps. This already allows a user to quickly assess the rough map quality without any algorithmic computations.

Secondly, one can very easily measure the ground truth distance between those fiducial parts. Thus, even when the ground truth positions of the fiducials are unknown or their measurement contains a great error, one can still compute very accurate local consistency scores. For barrels which are simply cut in half and placed on either side of a wall their distance is equal to the thickness of the wall.

Other local consistencies are also possible, for example based on the arrangement of all fiducials in one room or area.
5.1 Fiducials

5.1.6 Coverage

The coverage is expressed as the ratio of the number of fiducial parts assigned to a barrel part to the total number of fiducial parts. So a value of 100% means that all fiducials have been mapped while for an error value of 0 no barrels have been found.

5.1.7 Resolution Quality

The resolution quality is only measured very coarsely. If most of the barrels can be detected, one can only state, that the resolution is good enough that the identification of the barrels is possible. In that way one can enforce a minimum resolution quality by selecting barrels with a certain, maximum radius. A way to actually measure the resolution quality would be to use a number of easily accessible and mappable fiducials in the first, easy part of an environment. Assuming that this easy part of the arena was completely visited, a coverage value for just that part will predominantly be affected by the resolution quality. This value can then be taken as an upper bound for the resolution quality.

5.1.8 Gathering the Ground Truth Data

The amount and type of ground truth data needed for the fiducial approach differs for the different attributes. Just the number of fiducials identified is sufficient for the Coverage and Resolution Quality attributes. For the Local Consistencies just the distance between two parts of a barrel is needed - that is typically the thickness of the wall. More difficult it the gathering of the ground truth information for the Global and Relative Accuracy, which need the position of the fiducials in a (global) frame of reference.

Several methods have been used to solve this problem. In RoboCup Rescue, the arena is well structured and one can thus derive the positions of the fiducials from the floor plan (see Figure 5.2). But modern SLAM implementations can be more precise than the floor plan, especially since the actual arena often differs from the plan due to shifting of the walls by humans or robots driving against them. Also the construction is typically not done exactly to this floor plan.

The ground truth positions of the fiducials can also be acquired by using very good maps generated from very nice sensor data. This can be done by carrying the sensors smoothly around the environment, omitting all the error sources introduced by the unstructured terrain. When doing so, many loops should be closed while walking around, such that the SLAM algorithms can optimize the map quality using loop detection. The so called “fly through” maps are very precise and can thus be used to reliably determine the position of the fiducials (see Figure 7.12).

At the Multi Autonomous Ground-robotic International Challenge (MAGIC 2010), that was held in September 2010 in Adelaide, Australia [DSTO, 2010] (see Section 7.1 for more details), a georeferenced, high resolution aerial image was used to determine the positions of the outdoor fiducials, even though the fiducials were not yet placed when the picture were taken. This was done by locating the position of the fiducial using local features around its location.

At RoboCup 2011 in Istanbul, Johannes Pellenz introduced the use of tachymeters to determine the positions of the fiducials [Schwertfeger et al., 2011b]. Bright markers, as shown in Figure 5.3, are put at the center point of the barrel circles. A tachymeters is then setup at a place where it can see all the markers. Then the positions of the markers relative to the tachymeter can be easily measured.
Figure 5.2: A ground truth map of the RoboCupRescue arena with iconic representation of the different terrain types. The arena is split into two parts to accommodate two parallel runs during the preliminary rounds. The blue circles depict the locations of the fiducials.
5.1 Fiducials

Modern devices even provide the possibility to reference those positions automatically to another frame of reference.

This approach yields very precise results for the reference coordinates for several reasons:

- The accuracy of tachymeters is much higher than the accuracy of laser range finders used on mobile robots.
- The location of the fiducials is inspected from a single or only a few different locations.
- Since it takes only a short time to capture the data, the reference coordinates can be measured right before each individual run. This way, also small modifications in the environment can be captured.

Yet another approach was tested during the RoboCup German Open 2012. Here the author used a 3D Laser Scanner (Faro) to collect point cloud data. The points from certain heights were sampled and put into a 2D grid map as explained in more detail below. The 3D data was collected from three different spots to avoid almost all occlusions. The three 2D grid maps were then registered to form a 2D ground truth map. From this map the positions of the fiducials can then be easily extracted. Figure 5.4 shows one of the 3D scans and the resulting ground truth map. Very few occluded walls were painted in later on by hand and the walls and fiducials from the 2nd level (1st level wall height: 1.20m, 2nd level wall height: 2.4m) were marked with green color. 2nd level elements are high (2.4m) walls randomly placed in the arena and also next to elevated floors (those are accessible by ramps or stairs). Fiducials placed on the 2nd level walls are used to capture 3D mapping capabilities, by asking the teams to provide a second 2D map in that height.

To extract a 2D slice out of the 3D point cloud the following approach was used. First, the z-value of the ground has to be found in the 3D data. This can be done quite coarsely, since this value is anyways just used for the threshold in the next step. During RoboCup 2012 the author determined this number by looking at the scan with a 3D viewer and selecting a vertex representing the ground. Then, all points with a z value in a certain threshold (e.g. from 0.4 to 1.0 m above the ground) are selected. Using the x and y coordinates of those points, the corresponding cell in the 2D grid is found.
Figure 5.4: On the top a 3D scan of a RoboCup arena. On the bottom a ground truth map generated using three of those 3D scans (the bottom of the scan is on the left in the map). The 2nd level elements are depicted in green color.
5.1 Fiducials

Every 3D point mapped to a cell increases a counter in said cell. At the end all cells that exceed a certain count value represent occupied cells in the 2D grid map while all other are free. This way small objects in the 3D scan (cables, people moving around during the scan, ramps) are not treated as obstacles. Only vertical walls (and fiducials) which lead to many counts in the same 2D cell will be represented in the 2D grid map, which is the desired behavior.

The results of this approach are similar to the tachymeter method. Both use laser range information to find the position of the fiducials from one or very few positions. While the 3D scan method is faster when gathering the data it requires more post processing. Both methods give very precise fiducial locations are thus the use of either of those techniques is the preferred way of measuring the positions of fiducials.

5.1.9 Experiments

Experiments that use the Fiducial Map Metric on 2D grid maps can be found in Chapter 7. For those experiments the fiducial approach and also the metric using topology graphs from the next chapter have been implemented in the NIST Jacobs Map Toolkit.
Chapter 6

Structure-Based Map Evaluation using Matched Topology Graphs

In this chapter another concept to evaluate the quality of maps is presented. The idea is to grasp the structure of the map and to compare it with the structure extracted from a ground truth map. This topological structure [Choset and Choset, 1996, Kuipers and Byun, 1991] is represented in what we denote as Topology Graphs. The topological information we are interested in are hallways, their junctions and dead ends and connectivity.

Using a structure-based approach has several advantages over placed-based methods. First and foremost, we do not have to rely on the detection of features. In case of the Fiducial approach, artificial features have to be place in the environment with a relatively high density. Algorithms relying on natural features are application specific, because those features are typically only found in a sufficient density in certain environments and on certain scales (e.g. indoor vs. outdoor, structured vs. unstructured, etc.). Also we argue that the topological information is often exactly what the maps are used for: finding paths between places of interest. Place-based approaches rely on a good enough resolution of the maps to identify the features, while the structure-based approach presented in this chapter abstracts from the occupied cell level and just uses the topological information in the graph for map evaluation.

To compute the topological graph, a Voronoi Diagram [Voronoi, 1908] that treats the obstacles as Voronoi Sites is used. After some simplification and pruning, the desired Topology Graph can be extracted from the Voronoi Diagram. This is described in Sections 6.1, 6.2 and 6.3.

Once the Topology Graphs have been generated for a ground truth map and the robot map, they are matched to each other. The correspondences between the vertices are not only computed using classical place recognition, but they can also be determined using just the structure of the graph. Section 6.4 describes the different possibilities to compute similarities between the vertices from the two graphs. Those are then used in Section 6.5 to match the Topology Graphs, or, more precisely, their vertices, to each other. Section 6.6 finally shows how to evaluate the maps using the matched graphs.
6.1 Overview for Generating the Topology Graph

Here, a short overview of the Topology Graph generation, which is a form of skeletonization [Ogniewicz and Ilg, 1992], is given. The input to the algorithm is a two dimensional grid-map. After applying some of the map enhancement methods from Chapter 4, a colorization to occupied and free space (see Section 4.1) is needed. The implementation uses the CGAL library [Karavelas, 2011b] to generate the Voronoi Diagram. Following the terminology of CGAL, the occupied cells (more precisely the coordinates of their center) that are the basis for the Voronoi Diagram are called Sites.

After the computation of the Voronoi Diagram, there are far too many edges and vertices, especially those edges passing between directly adjacent obstacle points of the grid map. Section 6.2.1 describes how those are filtered out. After a new graph is being created (see Section 6.2.2), an Alpha Shape Boundary around the map is used to cut off the outside parts of the graph (see Section 6.2.3). Then some more filtering is done on the graph, for example removing edges leading to dead-ends that are shorter than a certain value (see Section 6.2.4).

At this step there are often a number of unconnected sub-graphs present. Section 6.2.5 describes how only the biggest graph (the one with the longest sum of edge-lengths) is kept.
be removed from the graph and, depending on the configuration values, some more simplification steps might be applied. At last the Topology Graph is created (see Section 6.3). Here vertices in close proximity to each other are joined together and vertices are also classified as being Spurious- or Major vertex. Spurious vertices are vertices that exist only because of small scale and local structures in the environment. Those structures are so small and topologically unimportant that they might not be present in another map representation. So they are identified and marked such that they may be ignored during the matching of the Topology Graphs.

As an example of the desired output, Figure 6.1 shows the ground truth map and the Topology Graphs for the two arenas (Arena A and Arena B) of the preliminary runs from the RoboCup Rescue Competition in Singapore 2010. Figure 5.2 from the last chapter shows the according ground truth map with the additional test element information.

6.1.1 Graph Terms

To ease the understanding of this chapter, a small recap on some important elements of graph theory is given here.

- **Graph**: A graph $G$ is a set of vertices $V$ connected by edges $E$: $G = (V, E)$. When comparing two graphs, they are here typically called $G$ and $G'$. In this work planar graphs are used. This property follows directly from the use of Voronoi diagrams to extract the map topology. In a planar graph, the vertices are located on a 2D plane and the edges do not intersect.

  - **Doubly Connected Edge List (DCEL)**: A planar graph in which a connection between two vertices is represented by two oppositely directed half-edges [Berg, 2008].

- **Topology Graph**: A Topology Graph is the graph where the vertices represent locations in (2D) space and the edges represent (drivable) paths between those locations.

- **Vertex**: A node in the graph.
  - **Degree of a Vertex**: The number of edges connected to this vertex: $\text{deg}(\nu) = |e(\nu, \nu')|$. 

- **Edge**: A connection between two vertices (nodes) in the graph - an unordered pair of two vertices $e(\nu, \nu')$. In the Topology Graph it represents a (drivable) path between two vertex locations.

  - **Half Edge**: In a Doubly Connected Edge List, a half edge is a directed edge between two vertices.
  - **Twin**: All half edges have twins. The originating vertex of a half edge is the target of this half edge’s twin, and vice versa.
  - **Ray**: In CGAL, a ray is, in contrast to a half edge, connected to only one vertex and has a direction. Later-on, when the Topology Graph is cut around the boundaries of the map of interest, edges where one vertex is cut and the other is not are turned into rays.

- **Face**: In geometry, the face is the area defined by some polygon. In the Doubly Connected Edge List, the face on an half edge is the left-hand side area that is defined by the polygon that this half edge is part of. The twin of a half edge has the area on the other side as its face. In a Voronoi graph, each face (Voronoi cell) contains exactly one point (also called Site). In this application this point corresponds to an obstacle cell in the grid map.
A few concepts that are introduced in this chapter are also shortly presented here:

- **Dead End Vertex**: A vertex with a degree of one. Also called cut vertex, but in the context of Topology Graphs the name “Dead End” describes the property better, since those vertices appear at dead ends of hallways.

- **Dead End Edge**: An edge connected to a dead end vertex.

- **Spurious Edge**: A dead end edge shorter than a certain threshold.

- **Major Edge**: All non-Spurious edges.

- **Major Vertex**: A major vertex is a vertex with either exactly one edge (the vertex is then additionally also a dead end vertex) or more than two major edges.

- **Spurious Vertex**: A Spurious vertex is a vertex that is connected to exactly two major vertices. It will then also have at least one Spurious edge.

- **Junction**: A vertex that is connected to at least three major vertices.

- **Exit**: An addition to the Doubly Connected Edge List concept. A half edge is the exit of a vertex if it has the vertex as its source. The main reason for introducing the Exit is, that it saves information and distance about both the next Major Vertex and the next Spurious Vertex (if there is one).

All implementations of the algorithms presented in this chapter use Doubly Connected Edge Lists for the graphs. But since the Topology Graph is undirected in nature, the following descriptions are given for an undirected graph. This also simplifies the algorithms to some extend.

### 6.2 Voronoi Graph

The Generalized Voronoi Diagram (GVD) (also called Voronoi decomposition, the Voronoi tessellation, or the Dirichlet tessellation) is a geometric structure which is widely used in many application areas [Klein, 1988, Aurenhammer, 1991]. It is a partition of the space into cells. The cells enclose a site and in this application the site is an obstacle point from the map. The Voronoi cell associated with a site is the set of all points in whose distance to the site is not greater than their distance to all other sites. In this work the interest is not so much in the Voronoi cells but in the graph that is defined by the boundary of said cells. Figure 6.2 shows an example distribution of sites (black dots) and the Voronoi cells (colored areas) for them.

In [Reem, 2011], the stability of Voronoi Diagrams is explored. The question is, given small changes in the position of the sites, how does the Voronoi Diagram change. This is also a fundamental question for this application, since maps generated by robots always differ slightly. Fortunately Reem comes to the conclusion, that small changes in the sites also lead to just small changes in the Voronoi cells. There are some restrictions for his general solution, but for the case of 2D planar geometry the main constraint is, that the sites should not be too close to each other. Actually, for this application, the sites are typically quite close to each other (neighboring obstacle cells of the grid map). But we are not interested in those small cells and they are filtered out anyways.
This section shows how the Voronoi diagram is created and filtered, such that it can be transformed into a Topology Graph. The Topology Graph represents the topological structure of the environment. The topological information we are interested in are hallways, their junctions and dead ends, and connectivity.

CGAL [Karavelas, 2011b] is used for the computation of the Voronoi diagram. For that, all cells from the grid map that are colorized to the “Occupied” value (typically 0), are put into a vector. This 2D set of points (also known as 2D point cloud) is the input for CGAL’s Voronoi_diagram_2 class. After the Voronoi calculation, this class provides access to the Voronoi cells as well as to the graph formed by the boundaries of those cells. This is the graph that is at the beginning of all further computations, which mainly include a) filtering (removing) vertices and edges and b) joining edges (by removing vertices that are connected to just two other edges (or four half-edges)), as described in more detail in Section 6.2.2.

Other methods for computing Voronoi graphs exists. The CGAL 2D Segment Delaunay Graphs [Karavelas, 2011a] has been tried by the author. Here the input are not points (Sites) but lines (Segments) in the 2D space. If the map is represented as lines, this algorithm has the big advantage, that no edges are created through walls and obstacles. This would result in a massively lower amount of edges and vertices, thus speeding up the computation. But a simple approach of connecting adjacent cells from the grid map as Segments resulted in a computation time that was much higher than the grid points implementation. One solution to this problem might be pre-computing large line segments through the walls and using those as input for the Delaunay Graph. But extracting long lines along walls might be quite sensitive to the noise of the mapping algorithms. This is why this approach was not further investigated.
Line thinning approaches [Lam et al., 1992], [Ko et al., 2004] are another alternative for skeletonization. Here the free space is marked and thinned till only a one pixel line remains. But there the result is a list of marked cells from the grid map. The graph structure would to still be extracted from this list, which is a non-trivial task. The point based Voronoi approach does not take more than a few seconds for the maps used here and performs well. Hence the point-based Voronoi approach was used.

The following steps are illustrated using two different maps, shown in Figure 6.3. The left map is from the random maze of the Response Robot Evaluation Exercise 2010 [Texas A&M University, 2008]. It has a resolution of 5 cm and features 4398 occupied cells. The right map was generated by a team of the RoboCup Rescue Competition 2010 in Singapore [Singapore, 2010]. It has 5534 obstacle points and also a resolution of 5cm.

The whole algorithm to create a Topology Graph is described in Algorithm 4. The different steps are described in the next sections. In Appendix A all configuration parameters used in this chapter are listed and shortly explained.

6.2.1 Filter for Distance

As mentioned above, the input to CGAL’s Voronoi calculation are the center points of the obstacle cells of the grid map. As a result, the Voronoi graph will have edges going through (in the map representation) directly adjacent obstacles, i.e. those edges go through walls. That is because the input to the Voronoi Diagram class is a set of points in 2D space. The class does not know anything about the quantization of space in a grid map. See Figure 6.4 for an illustration. Obviously those edges have to be filtered out.

Besides that, even more edges should be filtered. The goal is a graph representing the hallways and rooms. One can think of it as a “connectivity” graph. But when are two rooms “connected”? Is a gap of 10cm enough? Obviously this depends on the application area and the environment that was
6.2 Voronoi Graph

**Algorithm 4** CREATE TOPOLOGY GRAPH – Generating the Topology Graph from the map input.

- Generate Voronoi Diagram, using the center coordinates of occupied cells as input
- Filter the Voronoi Diagram for edges closer than $\delta_{\text{min2Obst}}$ to the obstacles
- Prune the Voronoi Graph from unnecessary vertices (vertices with two edges)
- Use an alpha shape that is fitted around the grid map to remove all vertices and edges outside of the area of interest (the map)
- Filter Dead Ends (depending on the configuration value all of them or only those smaller than $\delta_{\text{deadEnd1}}$)
- A second Dead End Filtering (depending on configuration this step might be skipped)
- Keep the biggest connected graph (criterion: summed length of edges) - all other vertices and edges are deleted
- A third and fourth Dead End Filtering (depending on configuration values)
- Join vertices that are in close proximity to each other (using $\delta_{\text{joinVertices}}$).
- Add attributes to the Topology Graph (Spurious and Major vertices and edges). Also save this information in a graph element called Exit that sits between the vertex and an edge.
- For all vertices, calculate the order of the edges of the current vertex and the relative angles between them.

mapped. As a rule of thumb, the topological map should have edges between connected sites that have enough space to let the user of the map (human or robot) pass through safely.

So a configuration parameter $\delta_{\text{min2Obst}}$ is introduced (see Appendix A). All edges for which the distance to the closest obstacle is less than this value are removed from the graph (see Algorithm 5). Not only the size of the robot but also the desired look of the Topology Graph should influence this variable. In an environment with wide hallways the value could be set higher to have less dead-end edges pointing towards walls.

Implementing this edge removal is quite simple. In CGAL every half edge knows about the face it belongs to. Each face has exactly one site, which is the obstacle cell from the grid map. And by definition of the Voronoi diagram, this is also the closest site to this edge. The program simply iterates through all half edges. The ones with a distance bigger then $\delta_{\text{min2Obst}}$ (and all rays) are put into a list, which is the input for the next step.

Figure 6.3 shows two maps on which the steps that lead to the Topology Graph are exercised. Since both maps show a similar environment and use the same resolution, the configuration values for the generation of the Topology Graph are mostly the same. Figure 6.5 depicts the filter step.

**Algorithm 5** FILTER FOR DISTANCE ALGORITHM – Filtering edges close to obstacles out.

```
for all $e \in E$ do
    if distance to obstacle($e$) > $\delta_{\text{min2Obst}}$ || ray($e$) then
        resultList.push($e$)
    end if
end for
```
Figure 6.4: A detail of an unfiltered Voronoi Diagram. In the zoomed in area the centers of the obstacle cells (the CGAL Sites) are shown in black, vertices in blue and edges in red.
6.2 Voronoi Graph

Figure 6.5: The graphs filtered for the distance to the obstacles (using $\delta_{\text{min2obst}}$) are shown for both example maps. A minimum distance of 30cm is configured. This reduces the number of CGAL half edges from 10280 to 2352 for the top and from 29114 to 6482 for the bottom map.
6.2.2 Create Pruned Voronoi Graph

The pruned graph is saved in a new data structure. The graph could be, due to the filtering from the step before, not connected. Also, after removing the edges some more cleaning up has to be done. Vertices that do not have any edges are removed. Additionally, there might now be vertices which are directly connected to only exactly two other vertices, since the edge to another, formerly directly connected vertex has been filtered out. Then the two neighboring vertices should be directly connected, removing the middle vertex and the half edges connected to it. Figure 6.6 illustrates this.

The graph keeps the handles (pointers to the vertices and edges) to the original Voronoi Diagram, especially also the list of skipped middle edges in the new edge. This way it is later-on possible to draw the actual path of the edge in the map, which is useful for illustration purposes.

The algorithm works as shown in Algorithm 6: First, for all vertices, the number of edges connected to this vertex is counted. Now the vertices are iterated. All middle vertices are skipped. Those are vertices that have two edges connected and where none of those edges are rays. All other vertices are added to the new graph. Looking at the example in Figure 6.6, we see that the vertices 4, 6 and 7 are put into the new graph. Vertices 4 and 6 have 3 edges connected and vertex 7 just one. Even if there would be no vertex 7, vertex 4 would still be added to the new graph since one of its then two edges are rays. Vertices 1, 2, 3 and 5 are filtered out because they have exactly two edges.

Now we iterate through all vertices in the new graph. Using the reference to the underlying CGAL graph, the edges are created. Here the skipping algorithm 7 is applied to all half edges leaving the vertex. It is basically following the edges until it reaches a vertex with rays or a vertex with other than two leaving half edges. The vertex where it is stopping is then saved as the goal vertex for the new edge.

In the example, starting at vertex $v_4$, edge $e_{4,2}$ is applied to the edge skipping algorithm. Edge $e_{4,2}$ is leading to $v_2$. Vertex $v_2$ has two edges, so it is skipped. So the next edge is $e_{2,1}$. Since $v_1$ also has two edges, the next edge selected is $e_{1,4}$. $v_4$ has three edges, so the loop is broken and a new edge between the start vertex of the algorithm ($v_4$) to the last vertex (also $v_4$) is created. Now the edge...
Figure 6.7: Voronoi Graphs generated from the Voronoi Diagram. Turquoise edges lead to dead end vertices, rays are green and other edges are blue. The vertices are shown as purple pixel. The edges are, for illustration purposes, drawn along the path of the original Voronoi Diagram. Yellow marks on the path indicate the interval (with $\delta_{\text{angleCalcStart}} = 0$: vertex to mark at distance $\delta_{\text{angleCalcEnd}}$) for which the angle of that edge with respect to its vertex is calculated (see Section 6.3.1). The graphs continue behind the boundaries of this picture and are very big. The top graph has 291 Vertices, 584 half edges and a total length of about 19km. The bottom graph consists of 1060 vertices, 2118 edges with a length of about 6 km.
Algorithm 6 Skip Vertices Algorithm – Skipping vertices.

\[
V_{\text{filtered}} = \{
\]
for all \( v \in V_{\text{VoronoiDiagram}} \) do
  if \( \deg(v) == 2 \) then
    continue
  end if
  if \( \exists e_{\text{ray}} | v \in e_{\text{ray}} \land \text{ray}(e_{\text{ray}}) \) then
    continue
  end if
  \( V_{\text{filtered}} = V_{\text{filtered}} \cup \{v\} \)
end for

skipping algorithm is applied to the other two edges leaving from \( v_4 \), too (those are \( e_{4,3} \) and \( e_{4,5} \)).

Ray edges are just copied from the old graph to the new one.

In the implementation several properties of the edges and vertices are calculated at the same time. The distance from vertex to vertex and the minimum distance to an obstacle are calculated in the Skip Edges Algorithm. Afterwards the distance for all vertices to their closest obstacle is gathered (using the distances from the neighboring edges) and the dead end vertices are marked. Figure 6.7 shows the color coded graphs. Please note that the graphs are actually much bigger than shown here, since they continue behind the boundaries of the images.

6.2.3 Alpha Shape Boundary

In the context of map evaluation, only the parts of the graph inside the map are of interest. The outer parts of the graph thus have to be removed. Since maps typically do not have a simple outer shape, bounding boxes are not good enough here. So an Alpha Shape [Edelsbrunner et al., 1983] is used to define the outer boundary of the map, with \( \alpha_{\text{shape}} \) has the alpha value.

The Alpha Shape and the Polygonization from Section 4.3 are used for this. The biggest polygon generated is the one defining the outer boundary. All vertices outside this polygon are removed. Edges that have one vertex inside and the other outside are turned into rays. Figure 6.8 shows the alpha polygon and the filtered graph, now featuring many rays.

6.2.4 Filter Dead Ends

Voronoi graphs can be very detailed. After the distance filtering, often short dead ends will point towards the walls. The filter dead ends method thus removes all dead end half edges shorter than a certain threshold. It can be beneficial to do this more than once. The corresponding configuration parameters are \( \delta_{\text{deadEnd1}} \) and \( \delta_{\text{deadEnd2}} \). A value smaller than zero will skip the step.

After the removal of dead ends there might be (again) vertices with exactly two neighboring vertices. Those are removed and the half edges joined accordingly. See Figure 6.9 for examples.
Algorithm 7 Skip Edges Algorithm – Skipping edges.

\( v_{\text{start}} \) \hspace{1cm} Save start vertex
\( v_{\text{curr}} = v_{\text{start}} \) \hspace{1cm} Initialize current vertex
\( e_{\text{curr}}: e_{\text{curr}} = \{v_{\text{curr}}, v_x\} \) \hspace{1cm} Initialize the current edge with the original edge.
\( \text{distance}_\text{sum} = 0 \) \hspace{1cm} Initialize the sum of the distances of all edges with zero

\textbf{while} TRUE \textbf{do}

\( \text{distance}\_\text{sum} = \text{distance}\_\text{sum} + \text{distance}(e_{\text{curr}}) \) \hspace{1cm} Add up the distances

\( v_{\text{next}} = v_0|v_0, v_1 \in e_{\text{curr}} \) \hspace{1cm} Set the next vertex to the first vertex of the edge

\textbf{if} \( v_{\text{next}} == v_{\text{curr}} \) \textbf{then}

\( v_{\text{next}} = v_1|v_0, v_1 \in e_{\text{curr}} \) \hspace{1cm} Since the edge is an unordered set, we have to see that we get

the right direction.
\textbf{end if}

\textbf{if} \( \text{deg}(v_{\text{next}}) \neq 2 \) \textbf{then}

\textbf{break}
\textbf{end if}

\( e_{\text{next}} = e_0|e_0(v_{\text{next}}, v_i) \) \hspace{1cm} Set the next edge to be one of the two connected to \( v_{\text{next}} \)

\textbf{if} \( e_{\text{next}} == e_{\text{curr}} \) \textbf{then}

\( e_{\text{next}} = e_1|e_1(v_{\text{next}}, v_j) \) \hspace{1cm} Set the next edge to the other edge connected to \( v_{\text{next}} \)
\textbf{end if}

\textbf{if} \( \text{ray}(e_{\text{next}}) \) \textbf{then}

\textbf{break} \hspace{1cm} If this edge is a ray the loop is also broken
\textbf{end if}

\( e_{\text{curr}} = e_{\text{next}} \)
\( v_{\text{curr}} = v_{\text{next}} \)
\textbf{end while}

\( e_{\text{new}} = \{v_{\text{start}}, v_{\text{next}}\} \) \hspace{1cm} Add the new edge
\( \text{distance}\_e_{\text{new}} = \text{distance}\_\text{sum} \) \hspace{1cm} Save the distance
Figure 6.8: The graphs filtered for all vertices outside the alpha shape enclosing (in blue) of the map are shown here. Rays appear in green. The top map used an alpha value of 2500 (2.5 m) and has an alpha polygon consisting of 133 segments with a total length of 38 meter. The filtered graph features 211 vertices, 424 half edges and a length of 169 meter. The values for the bottom map are a big alpha value of 10000 (5 m), 161 segments with a length of 62 m and a graph of 855 vertices, 1708 half edges and a length of 321 m.
Figure 6.9: Graphs after the first filtering of dead ends. The value of $\delta_{\text{deadEnd}}$ is very high such that all dead ends are removed. The statistics are now as follows: 59 vertices, 120 half edges and 143 m length on the left and 179 vertices, 356 half edges with 184 m on the right side.
6.2.5 Keep Biggest Connected Graph

Only a connected graph can be later-on used for the map evaluation. Thus all edges and vertices not part of the biggest connected graph are removed. The size attribute used to find the biggest graph is the sum of the length of all edges of a connected sub-graph.

The algorithms works by first assigning all vertices and edges the group id zero. Then all half edges are iterated. If a half edge does not have a group id yet, traverseGroupCalcDist is called with a new group id. This method recursively marks all edges and vertices connected with the new group id and additionally sums up the lengths of all edges (rays get a distance of zero).

The list of groups is then searched for the one with the biggest length. All edges and vertices are iterated again and all entities not having the group id of the biggest one are removed. In Figure 6.10 it can be seen that this step not only removes the graphs from the outside of the map but also the small graphs in the void spaces.

6.2.6 Join Adjacent Vertices

Other steps in the simplification of the graphs follow. The rays are removed and after that two more removals of dead ends may follow, depending on the configuration values for $\delta_{\text{deadEnd3}}$ and $\delta_{\text{deadEnd4}}$ - see Figure 6.11. Finally vertices in close proximity to each other are joined together. The parameter $\delta_{\text{joinVertices}}$ is used for that. The new vertex gets the average position of the joined vertices. The edges coming from other vertices to the joined vertices are connected to the new one. The resulting graph is annotated with further information in the next section and then called Topology Graph.
6.2 Voronoi Graph

Figure 6.10: After the removal of all sub graphs not connected to the biggest connected graph. The resulting graphs have 36 vertices, 76 half edges with 54 m on the left and 132 vertices, 268 half edges and a length of 135 m for the right graph.
Figure 6.11: Third and Fourth dead end removal. In this example, the second removal was skipped in the configuration. The configuration for the third step is 0.3 m and for the fourth 0.4 m. The left graph reduces to 18 vertices, 38 half edges and a length of 52 m in the third removal and does not change in the fourth. The right graph reduces to 78 vertices, 160 half edges and 130 m in the third step and to 68 vertices, 140 half edges and 128 m in the fourth.
6.3 Topology Graph

The Topology Graph is the graph structure used for matching two maps to each other. For that it provides additional attributes (such as Spurious and Major Vertices and Edges) and also an additional graph element called “Exit”. This graph represents the topological structure of the environment. Vertices are connected with edges. In the examples those edges are, to aid visualization, drawn using the actual geometric path information from the Voronoi Diagram.

One important attribute that is determined in this step is marking Spurious edges and vertices. For that the dead end property of vertices and edges is used. A vertex is a dead end if it only has one edge connected. An edge is a dead end if it is connected to a dead end vertex. Additionally, an edge gets the property Spurious end if it is a dead end with a length \( \leq \) the value of the parameter \( \delta_{spuriousEdge} \).

A vertex that has exactly two edges that are not Spurious ends is marked as being a Spurious vertex. Since there are no vertices with two edges (those would have been joined away) that means, that a Spurious vertex is connected to one or more dead end vertices in close proximity (via Spurious edges) plus to two more non-Spurious vertices. A Spurious vertex does not add much topology information to the graph. If the short, not so significant Spurious edges and their dead ends would be removed, the Spurious vertex would also be joined away. That is where the name spurious originates from, since the vertex merely holds information about a local feature of the environment and not about the topology as a whole.

Vertices that are not Spurious vertices are called major vertex. Later-on it will be important to access the next Spurious vertex as well as the next Major vertex when going from an originating vertex along an edge. So an additional data-structure between the vertex and the edge, called Exit, is introduced that saves pointers to the next vertex and to the next Major vertex if the direct edge goal is a Spurious vertex. Figure 6.12 shows the generated Topology Graphs for the two example maps.

6.3.1 Calculating the Angle of Exits

The goal of this step is to sort the exits of a vertex according to their angle, such that the left- and right-hand neighbors of exits can be retrieved easily. This geometrical order is later-on of major importance when matching two Topology Graphs using the Isomorphism approach (see Section 6.5). Additionally this will be used during the calculation of the similarity of two vertices (see next Section 6.4). There the biggest and the smallest angle-difference between two neighboring exits is utilized as a descriptor, but also the order is used.

This calculation is not trivial because the position of the goal vertex of the exit cannot be used (it might be somewhere completely different after a long and winding path). So a part of the path associated with the edge close to the originating vertex is used. Since this path might be curved, a number of points are sampled on the path and their angles averaged. The algorithm works as follows:

Edges originating from a vertex get assigned an angle relative to the coordinate system of the map. This is done by calculating the average angle of a number of points in a certain interval on the path of this edge. This interval is given by the configuration parameters \( \delta_{angleCalcStart} \) and \( \delta_{angleCalcEnd} \). Those distances refer to the distance along the path - they are not (necessarily) the geometric distance between the vertex and the points. The points are thus calculated using the original CGAL segments of the Voronoi Graph. The point location is obtained by interpolation in the segment at the desired
Figure 6.12: The resulting Topology Graphs. Spurious edges (the threshold is 80 cm) and vertices are shown in turquoise. The top map has 17 vertices and 36 half edges. The bottom map has 61 vertices and 126 half edges, 28 of which are Spurious. Note that not all Spurious half edges are connected to Spurious vertices!
distance. The configuration file parameter $\delta_{\text{angleCalcStep}}$ is used to iterate over a number of points within said interval along the half edge. The angle of this exit is then the average over the angles of all sampled points.

After the angles for all exits of this vertex have been calculated, they are put into a ring-buffer, sorted by the angles to the map coordinate system. This way, the right- and left-hand neighbors of exits can easily be retrieved. Saved in the ring-buffer is not the global coordinate system angle but the difference in angle to the neighboring exits. This information is, unlike the value of the angle itself, independent of the frame of reference used, as it should be for the use in a Topology Graph. There are two feature values calculated at this step which are used later-on when computing the Local Vertex Similarity. Those are the value of the biggest and the smallest angle-difference between two neighboring exits of this vertex. These values are also rotation-independent.

### 6.4 Calculating the Similarity of two Vertices

Calculating some similarity between two vertices of two different graphs $G$ and $G'$ is an important first step when matching the graphs. Later-on a basic approach will calculate those similarity values between all $i$ vertices from graph 1 ($v_i \in G$) and the $k$ vertices from graph 2 ($v'_k \in G'$), resulting in a similarity table with $i \times k$ entries, hence estimating possible correspondences between vertices in the two graphs.

Similarities, not just those between two vertices, are always calculated such that their value is between 0 and 1, where 0 means a perfect match and 1 means maximum difference. Those values are calculated using an offset $of$ and an upper boundary $ub$ from the configuration:

$$similarity = \begin{cases} 
0, & \text{if } input < of; \\
1, & \text{if } input > ub; \\
input - of, & \text{if } of \leq input \leq ub.
\end{cases}$$

This similarity will guide the search for matching the graphs. It is thus desirable to use an algorithm which gives outstanding results for matching pairs while having a low complexity. Two approaches were investigated: calculating similarities using just the information stored in the graph and using the occupancy information of the local area around the vertices from the original maps (dubbed sensor-based place recognition). For each approach different algorithms and options were developed and tested. As shown later on, it turns out that the sensor-based approach delivers very good results. But using just information from the graph is nearly as good and at least three orders of magnitude faster.

Once the similarities between all vertices are known, one can enhance this information by including the similarity values of the neighboring vertices. This concept, called Propagation Rounds, can be applied to all similarity algorithms and is described in Section 6.4.1.

In Section 6.4.2 the Local Vertex Similarity is introduced. The Enhanced Vertex Similarity from Section 6.4.3 uses the Local Vertex Similarity and a Propagation Rounds approach, which is supplemented by additional information from the graph.

The map based similarity method and the different options are introduced in general in Section 6.4.4, followed by the actual algorithms using ICP (Section 6.4.5) and iFMI (Section 6.4.6) as underlying technique to generate a similarity value.
In Section 6.4.7 the different algorithms are experimentally compared.

6.4.1 Propagation Rounds

Once similarity values (of any kind) between all permutations of pairs of vertices from two maps have been calculated, it is possible to enhance the similarity values by taking the similarity values from the neighboring vertices of the Topology Graphs into account. Thus vertices which have neighbors that also match nicely get better values while vertices with more incompatible neighbors get penalized. This is called calculating one Propagation Round for this similarity value. Consecutive runs of this Propagation Round algorithm, each taking as vertex similarity the values from the previous round, can further improve the result. Those further propagation rounds thus take even more indirect neighbors into account.

Though the concept might sound simple, the actual implementation is more difficult. This is because the matching of vertices \((v_x, v'_x)\) between the two graphs \((G, G')\) is not (yet) determined. The set of direct neighbors of a vertex from \(G\) is \(n(v_x)\) and the \(i\)-th neighbor is \(n(v_x)_i\). Assuming \(v_x\) and \(v'_x\) are the corresponding vertices from \(G\) and \(G'\) and the numbers of their neighbors \(|n(v_x)|\) and \(|n(v'_x)|\) equals, the correspondence between the two sets of neighbors is still unknown. The idea is to try all possibilities, assign a combined similarity for each of them and then take the best combined similarity. Two variants of this Propagation Rounds algorithm are tested:

The Normal Propagation Round algorithm just takes all permutations of assignments between the neighbors \(n(v_x)\) and \(n(v'_x)\), even if the number of neighbors differs. The combined similarity is the mean of the similarities of the matched neighbors plus the similarity of the original vertex pair \(n(v_x)\) and \(n(v'_x)\) divided by two.

\[
s'(v_x, v'_x) \text{ is the similarity between two vertices. } M \text{ is the mapping for vertices } M(n) \text{ and } M'(n) \text{ from } G \text{ to } G' \text{ where } M(n) \in n(v_x) \text{ and } M'(n) \in n(v'_x). \text{ For one specific mapping } M \text{ the combined similarity is }
\]

\[
s_{\text{combined}}(v_x, v'_x, M) = \frac{s(v_x, v'_x) + \sum_{n=0}^{\lfloor |M|/2 \rfloor} (s(M(n), M'(n)))}{|M|/2}.
\]

The Strict Propagation Round algorithm takes the topological constraints of the graph into account. If the number of neighbors differs, the similarity value for two vertices \(v_x\) and \(v'_x\) is heavily penalized without looking at the neighbors. In case of an equal number of neighbors \((|n(v_x)| = |n(v'_x)|)\), only those mappings are checked that are in the correct geometrical order. As mentioned above in Section 6.3.1, the edges leading to neighbors (exits) are put in a ring buffer that is sorted by the angle. The set of neighbors \(n(v_x)\) is sorted in the same way, giving easy access to the neighbors left or right of another neighbor. A correct geometrical order now means that a mapping from \(n(v_x)_i\) to \(n(v'_x)_j\) is only valid if also \(n(v_x)_{i+1}\) is mapped to \(n(v'_x)_{j+1}\). The total number of permutations that comply to this restriction is the number of neighbors present. That is because there are \(|n(v'_x)|\) number of partners to choose for the first vertex in \(n(v_x)\), but afterwards all further mappings are determined by their topological order (their angle).
6.4 Calculating the Similarity of two Vertices

6.4.2 Local Vertex Similarities

The local vertex similarity $s_l$ is the simplest similarity approach. It just works on the Topology Graph. It averages over a number of similarity values to calculate the similarity between two vertices.

First it is checked if the two vertices differ in their type. That is the case if one of the following predicates is fulfilled:

- $\text{differsInBeingSpuriousVertex} \Delta_{spurious}$: true if one vertex is a Spurious vertex and the other is not
- $\text{differsInBeingDeadEnd} \Delta_{deadEnd}$: true if one vertex is a dead end and the other is not
- $\text{differsInBeingRayEnd} \Delta_{rayEnd}$: true if one vertex is a vertex connected to a ray and the other is not

A different type is assigned with a similarity value of 1, equal types get a 0.

The second value $\Delta_{\text{Exits}}$ calculated takes the difference of the number of exits into account. Usually the values for $\sigma_{\text{existsOff}}$ and $\sigma_{\text{existsMax}}$ are 0 and 1, leading to a score of 0 if the number of exits equals and to a score of 1 if they differ.

The third value used is itself obtained from the average of four other sub-scores describing the vertex.

The first and second sub-scores use the values of the biggest ($\Delta_{\text{anglesMax}}$) and smallest ($\Delta_{\text{anglesMin}}$) angle between neighboring exits of a vertex. The difference of those angle-differences for the two to be compared vertices is used together with the parameters $\sigma_{\text{angleOff}}$ and $\sigma_{\text{angleMax}}$ from the configuration file to calculate the scores.

The third sub-score ($\Delta_{\text{obstacle}}$) compares the distance from the vertex to the closest obstacle for both vertices. In case this vertex is a joined vertex, those distances are averaged for all vertices from the Voronoi Graph that form this Topology Graph vertex. The configuration parameters are $\sigma_{\text{obstacleOff}}$ and $\sigma_{\text{obstacleMax}}$.

The fourth sub-score is only applied for joined vertices ($\Delta_{\text{joined}}$). It takes the summed up distances from the original vertices to the joined vertex. Those values are compared using $\sigma_{\text{joinedOff}}$ and $\sigma_{\text{joinedMax}}$.

The first, second and third (which is itself an average of four sub-scores, see above) score values are averaged. This is the Local Vertex Similarity $s_l$:

$$s_l(v, v') = \max(\Delta_{spurious}, \Delta_{deadEnd}, \Delta_{rayEnd}) + \Delta_{\text{Exits}} + \left(\Delta_{\text{anglesMax}} + \Delta_{\text{anglesMin}} + \Delta_{\text{obstacle}} + \Delta_{\text{joined}} \right) / 4$$

This similarity is saved in the vertex table (as are the other similarity values and the values produced by the Propagation Round algorithm). In the implementation this is not actually a table. There, for every vertex from the first graph, a sorted list with the similarity values towards the vertices from the second graph is saved.
6.4.3 Enhanced Vertex Similarities

The Local Vertex Similarity uses only information from the vicinity around the vertices. The Enhanced Vertex Similarity additionally takes the neighbors of this vertex into account. In that, it is very similar to the Propagation Rounds approach. But the Enhanced Vertex Similarity takes additional properties into account.

The enhanced vertex similarity is defined as three fifth of the value of the best exit assignment and two fifth of the Local Vertex Similarities (see 6.4.2) - for the definition of $EA$ see next section:

$$s_e(v, v') = \frac{3 \cdot \max(EA(v, v')) + 2 \cdot s(v, v')}{5}$$

Exit Assignment Similarity

The similarity for an assignment of exits for two vertices is described here.

As mentioned above, one problem is that there are (as long as there is more than one exit) different ways to assign the exits of the two vertices to each other. As a solution, all permutations of exit assignments between two vertices are generated:

The mapping $em$ of one exit $ex_v$ from vertex $v$ from graph $G$ to one exit $ex'_{v'}$ from graph $G'$ is a 2-tuple: $em = (ex_v, ex'_{v'})$.

One exit assignment $ea(v, v')$ is a set of exit mappings $em$. It has $\min(\deg(v), \deg(v'))$ entries. No exit appears twice in this set:

$$\forall em \in ea(\exists em^* \in ea((em_1 = em^*_1 \land em_2 \neq em^*_2) \lor (em_2 = em^*_2 \land em_1 \neq em^*_1)))$$

$EA(v, v')$ is the set of all possible permutations of exit assignments $ea(v, v')$.

This exit assignment similarity is calculated on one exit assignment $sim_{ea}$. First the so called angles similarity $as(ea)$ is calculated. It looks, for every exit, at the angle to the next exit (clock wise). The difference of this angle for the paired exits is calculated into a score using $\sigma_{obstacleOff}$ and $\sigma_{obstacleMax}$ for all the assigned exits. Then the scores for those angle-differences are averaged.

It uses the average of the so called pair-wise Exit Similarity $pe(ex, ex')$ (see next section) to two thirds and the angles similarity $as(ea)$ to one third:

$$sim_{ea}(ea) = \frac{2 \cdot \sum_{n=0}^{\|ea\|} (pe(ea_{n1}, ea_{n2}) + as(ea)) / |ea| + as(ea)}{3}$$

Pair-wise Exit Similarity

The pair-wise Exit Similarity $pe(ex, ex')$ calculates the similarity between the exits $ex$ and $ex'$ and consists of two equally weighted parts. First, there is a comparison of the properties of the half edges and then about the similarity of the target vertices. For the edge comparison the scores of the following attributes are averaged:

- $\Delta_{nextVertex}$ Difference in the distance to the next vertex.
6.4 Calculating the Similarity of two Vertices

- $\Delta_{\text{nextMajorV}}$ Difference in the distance to the next major vertex (that might also be the next vertex) with four times importance for the average.

- $\Delta_{\text{obstacleMin}}$ Difference in the distance to the closest obstacle along the half edge.

- $\Delta_{\text{obstacleAvg}}$ Difference in the value for the average distance to obstacles along the half edge.

- $\Delta_{\text{existsSpurious}}$ Difference in the presence of at least one Spurious vertex along the exit.

The similarity for the target vertices $ts$ is 1 (worst) if exactly of the edges is a ray and 0 if both are rays. Otherwise the local vertex similarity (see 6.4.2) of the target vertex and the next major vertex are used. $\text{target}(ex)$ returns the target vertex given an exit. If this edge is a loop, the target vertex will be the same vertex as the start vertex.

$$ts(ex, ex') = \begin{cases} 0, & \text{if ray}(ex) \land \text{ray}(ex'); \\ 1, & \text{if ray}(ex) \lor \neg \text{ray}(ex'); \\ s_l(\text{target}(ex), \text{target}(ex')) & \end{cases}$$

$$pe(ex, ex') = \left( \frac{\Delta_{\text{nextVertex}} + 4 \cdot \Delta_{\text{nextMajorV}} + \Delta_{\text{obstacleMin}} + \Delta_{\text{obstacleAvg}} + \Delta_{\text{existsSpurious}} + ts(ex, ex')}{8} \right) / 2$$

6.4.4 Range-sensor-based Vertex Similarities

The idea behind the range-sensor-based vertex similarity is to compare the appearance of the map around the vertex to generate a similarity value between the two vertices, i.e. to use in addition a more classical place recognition to estimate vertex correspondences. First the according part of the map has to be extracted from the map, and secondly those parts have to be compared. For the extraction, the two strategies Disk-Extraction and Raytrace-Extraction were tested, while the comparison is done using two variants of the ICP (Iterative Closest Point [Besl and McKay, 1992]) approaches and the iFMI algorithm presented in Section 3.1.

Just the local part of the map around the vertex location is used here. This way errors in other parts of the map do not effect the similarity of this vertex. Both extraction approaches thus have a radius as parameter, specifying the disk from which obstacle points from the maps are extracted into a point cloud. The Disk-Extraction takes all points within this radius and adds it to the point cloud. The Raytrace-Extraction only takes the obstacles that are visible from the location of the vertex, thus omitting obstacle points that are within the radius but that are obstructed by closer obstacles. The idea behind the latter approach is to only use obstacles from this room and not from the area behind a wall. This makes the point cloud even more local and thus localization errors effect this vertex similarity measure even less. Also, the number of points generated is usually much less compared to the Disk-Extraction. On the other hand, since there are less points, it might also be less descriptive and thus turn out to be more difficult to compare in the similarity calculation step.

The similarity of the two vertices has to be independent of the rotation of the point clouds, because errors in the initial orientation of the maps or localization errors might to lead to (significantly) differing orientations. The methods implemented are thus ones that also calculate the relative rotation between the two point clouds - or what is assumed to be the best fit between those.
6.4.5 ICP-based Vertex Similarity

Iterative Closest Point (ICP) is a classical algorithm to calculate the transformation between two point clouds [Besl and McKay, 1992]. Starting with an initial assumption for the transformation, it calculates the transformation with the lowest mean squared error between nearest neighbors from the two point clouds. It then iterates the process with the found transformation as new initial transformation until either a maximum of iterations or a threshold for the mean squared error or its change is reached.

The first ICP Similarity approach (called Simple ICP Similarity) assumes the identity for the initial transformation and uses the square root of the resulting mean squared error as similarity value. The idea is, that this value will be lowest for the vertices from the two maps that are on the same location and higher for all (or at least most) of the other candidates.

Generally ICP is quite sensitive with respect to the initial assumption for the transformation between the two point clouds. If this assumption is off the actual transformation by a larger amount, ICP often is caught in a local minimum. The Simple ICP Similarity is thus depending on the orientation of the maps and thus not rotation invariant, as required. As can be seen in the experiments (see Section 6.4.7) this leads to unsatisfactory results.

The Rotated ICP Similarity mitigates this problem by running the algorithm a couple of times with initial assumptions for the rotation evenly distributed over the 360 degrees of a circle. For example, with a parameter n of 12 for the number of ICP runs, it would increment the initial start angle by 30 degrees for each trial. Thus the actual rotation between the two point clouds will not be off more than 15 degrees from the initial assumption. This gives a very good chance that the ICP run with a starting rotation close to the actual value will not be caught in a local minimum but advance to the global minimum, thus giving the correct and lowest mean squared error. The square root of the lowest
6.4 Calculating the Similarity of two Vertices

of the mean squared errors for all runs is then selected as the similarity value. Obviously the main disadvantage of this approach is the increase in computation time (n-times).

6.4.6 iFMI-based Vertex Similarity

The iFMI algorithm has been introduced in Section 3.1. It is used in this context to calculate the similarity between the two point clouds of obstacles from the area around the respective vertices from the two maps. First the point clouds have to be put into an image (or grid, or matrix), since the iFMI works on 2D image data. Then the registration is applied. The signal to noise ratio around the Dirac pulse of the translation step is then used as the similarity value.

6.4.7 Vertex Similarity Experiments

Experiments to compare the performance of the different vertex similarity algorithms have been conducted. The different approaches are applied to a number of maps that show the same environment, namely the random maze from the Response Robot Evaluation Exercise at Disaster City 2010. For those experiments nine different maps have been used. One example map (map H) is shown in Figure 6.14 - all maps are shown in Appendix B. Map A is a ground truth map, while maps B, C, D and E where generated by different mapping algorithms. All those maps are roughly in the same orientation. Map F is the map B rotated by exactly 90 degrees, while map G is map B rotated by 180 degrees. Map H is map B rotated by approximately 140 degree and map I is map C rotated by about 37 degrees.

The maps show the Topology Graphs and numbers for the vertices. The numbering simply starts at the top left and goes down to the bottom right. Since the maps all show the same environment, it is
expected that the Topology Graphs are very similar and this is indeed the case. This is used to assess
the performance of the vertex similarity experiments. The vertices from two maps at corresponding
locations should have very good similarity values while non-corresponding vertex pairs should have
higher values. So in these experiments the simple criterion is, whether for any given vertex from one
map, the best (lowest vertex similarity value) of all vertices from another map is the one corresponding
to that location. For that the correctly corresponding vertices between the maps were determined by
hand. In appendix B.1 one such mapping from map A to map H is given. For example, one can see that
the vertex number 1 in map A corresponds to the vertex number 20 in map H. So in the experiments,
vertex number 1 in map A will be compared with all the vertices from map H, using one of the vertex
similarity methods. If the vertex from map H with the lowest (best) value is vertex number 20, then
this was a success, otherwise a failure.

All maps feature 17 vertices that always appear at the same location, so those are the ones used for
the success calculation. Some maps have some additional Spurious vertices (e.g. for map H vertices
3, 5, 6, 7, 14 and 16) that are included in the similarity calculations (and could thus have the lowest
similarity value for another vertex, making this comparison a failure), but that do not otherwise effect
the success calculation.

Not all combinations of maps were used. This is because the Topology Graphs for maps that
are just rotated by 90 or 180 degrees are exactly the same. Thus the Simple and Enhanced Vertex
Similarity have a 100% success rate. This seems like an unfair advantage to those algorithms, since
in reality the exact same maps with just global rotations will not occur. Note, that maps rotated at
an angle other than 90, 180 or 270 degree always have differences due to the grid interpolation and
discretization and thus the Topology Graph differ to some extend.

Furthermore, a distinction for map pairs with a similar orientation and rotated maps has been made
to show the effects of the rotation for the different algorithms. The success rate is averaged over all
vertices and map pairs for the “No Rotation” case, for the “Just Rotation” case and for the combined
results called “Both”. Appendix B.2 shows the 10 combinations used for “No Rotation” and the 23
map pairs for ”Just Rotation”.

**Tested Vertex Similarity Algorithms**

The 38 approaches to compute a vertex similarity that have been tested during the experiments are
shown in Table 6.1. The Local Vertex Similarity (Section 6.4.2) and the Enhanced Vertex Similarity
(Section 6.4.3) are compared to the Simple ICP Similarity (Section 6.4.5), the Rotated ICP Similarity
using 12 different rotations, and the iFMI Similarity (Section 6.4.6), which are employing the ray-
extraction and the disk-extraction methods. When Propagation Rounds (Section 6.4.1) are applied,
the abbreviations are extended with ”_1” for the first and ”_2” for the second round. The abbreviations
for Strict Propagation Rounds are additionally extended with a ”_s”. The Enhanced Vertex Similarity
is not applied to a second round since it is in itself already a round-like algorithm.

All 38 algorithms have been applied to all 33 map pairs. The configuration parameters in Ap-
pendix A are used for the creation of the Topology Graphs. The ICP algorithms use a maximum num-
ber of iterations of 100 and an error reduction factor tolerance of 0.01. The radius for the Raytrace-
and Disk-Extraction is 50 pixel or 2.5 meter. First some statistics and speeds are shown, using just the
maps A and B. The average point cloud size for the Raytrace-Extraction for both maps combined is
76 points, while there are 619 points on average in the Disk-Extraction.
6.4 Calculating the Similarity of two Vertices

<table>
<thead>
<tr>
<th>Abbr.</th>
<th>Local Vertex Similarity</th>
<th>Enhanced Vertex Similarity</th>
<th>Simple ICP Similarity</th>
<th>Rotated ICP Similarity</th>
<th>iFMI Similarity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extraction</td>
<td>Simple</td>
<td>Enhanced</td>
<td>ICP</td>
<td>ICP-12</td>
<td>iFMI</td>
</tr>
<tr>
<td>without Rounds</td>
<td>-</td>
<td>-</td>
<td>ray</td>
<td>disk</td>
<td>ray</td>
</tr>
<tr>
<td>First Normal Propagation Round</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Second Normal Propagation Round</td>
<td>✓</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>First Strict Propagation Round</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Second Strict Propagation Round</td>
<td>✓</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 6.1: The 38 vertex similarity calculations that were tested in the experiments. A ✓ indicates that this experiment was run. The ray- and disk-extraction have the same abbreviation because their results are shown in two different diagrams (Figure 6.15). In the diagram, the abbreviation of the similarity algorithm is appended with "_.1" when one Propagation Round is applied to it and with "_.2" for the second Propagation Round. Strict Propagation Rounds are additionally appended with "_.s".

Experiment Results

The 17 x 17 vertices from the maps have 289 combinations of pairs, so there are 578 extractions needed (2 per pair). The ICP approaches actually use a caching strategy that saves already generated extractions, reducing the number of extractions needed to 34. Table 6.2 shows the speeds for the different algorithms - for the complete computation of the 17 x 17 similarity values. The computation was done in a single thread on a 2.8 GHz Core 2 processor.

The Local Vertex Similarity does not only calculate the similarity but is also responsible for creating the different data structures in which the results are saved. The Enhanced Vertex Similarity includes the time needed for the necessary Local Vertex Similarity. It also computes the combinations of neighbors possible for a pair of vertices and saves this in a special data structure. This is also why it is much slower than the Propagation Rounds - those actually use this precomputed data.

The extraction times are already included in the ICP and iFMI speeds. The main result of the speed comparison is, that the calculation of a propagation round is about 2 orders of magnitude faster than the Local Vertex Similarity which in turn is one order of magnitude faster than the Enhanced Vertex Similarity. Nevertheless, all this is still two orders of magnitude faster than the Map-based approaches (The Simple Raytraced ICP has very bad results, as will be shown below.)

Figure 6.15 shows the results of the similarity experiments. The higher the bars the better the similarity. Both graphs show the results from the Local and the Enhanced Vertex Similarities, also with Propagation Rounds. Those are the same results - they are included in both graphs such that they are easier to compare. The graphs differ in the results for the map-base approaches (ICP and iFMI) - the upper graph (6.15(a)) shows the results acquired using the Raytrace-Extraction while the lower graph (6.15(b)) depicts the values for the Disk-Extraction.
Structure-Based Map Evaluation using Matched Topology Graphs

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>FPS in Hz for 17x17 similarities</th>
</tr>
</thead>
<tbody>
<tr>
<td>One Propagation Round</td>
<td>190,000</td>
</tr>
<tr>
<td>One Strict Propagation Round</td>
<td>196,000</td>
</tr>
<tr>
<td>Local Vertex Similarity</td>
<td>1,560</td>
</tr>
<tr>
<td>Enhanced Vertex Similarity</td>
<td>89</td>
</tr>
<tr>
<td>Raytrace-Extraction</td>
<td>7.6</td>
</tr>
<tr>
<td>Disk-Extraction</td>
<td>7.8</td>
</tr>
<tr>
<td>Simple ICP (Raytrace-Extraction)</td>
<td>4.3</td>
</tr>
<tr>
<td>Rotated ICP Similarity (12 rotations) (Raytrace-Extraction)</td>
<td>0.41</td>
</tr>
<tr>
<td>Simple ICP (Disk-Extraction)</td>
<td>0.37</td>
</tr>
<tr>
<td>Rotated ICP Similarity (12 rotations) (Disk-Extraction)</td>
<td>0.038</td>
</tr>
<tr>
<td>iFMI</td>
<td>0.31</td>
</tr>
</tbody>
</table>

Table 6.2: Speed of the different algorithms

It should be kept in mind, that the percentages reached here are those acquired with the simplest matching algorithm possible: Just matching a vertex against the one vertex from the other map with the best similarity score. Later on, better methods of matching will be introduced. The values computed using the similarity algorithms presented in this section are then used to guide the search and thus do not need to be perfect.

Interpretation of the Results of the Vertex Similarity Experiments

First the effects of the Propagation Rounds are evaluated. It can be seen that for most approaches adding Propagation Rounds improves the result. Often the Strict Rounds approach is better than using Normal Rounds. The Enhanced Vertex Similarity, which basically is a Local Vertex Similarity with one Propagation Round and additional constraints such as the edge lengths, is superior to the Local Vertex Similarity with one Propagation Round, proving that the additional information is quite helpful. The Enhanced Vertex Similarity is already as strict as a Strict Propagation Round and therefore there is very little difference between the additional Normal and the additional Strict Propagation Round.

A very clear result is, that the range-sensor-based place recognition (ICP and iFMI) perform much better using the Disk-Extraction than with the Ray-Extraction. As mentioned above, the Disk-Extraction has approximately ten times as much points. It seems that raytracing omits too much information and is just too sparse. The additional points from the Disk-Extraction help separating the results much more.

The effects on the similarity algorithms of a rotation of or in the map can be seen when comparing the "No Rotation" bars with the "Just Rotation" ones. One can see that the graph based approaches (Local and Enhanced Vertex Similarity) have very similar values for both cases - this is especially true for the Enhanced Vertex Similarity. The place-based solutions have more problems. The Simple ICP Similarity is for the "No Rotation" cases relatively decent (for the Disk-Extraction even excellent), but for the "Just Rotation" combinations it performs very badly. Only for the information-rich Disk-Extraction the Rotated ICP Similarity is rotationally independent. The same is true for the iFMI approach.

The best results are clearly achieved using the Rotated ICP Approach with Disk-Extraction.
6.4 Calculating the Similarity of two Vertices

(a) Using the Raytrace-Extraction for the range-sensor-based approaches.

(b) Using the Disk-Extraction for the range-sensor-based approaches - the values for the simple and enhanced algorithms are equal to the above values and are just included for better comparison.

Figure 6.15: Combined average results and standard deviation of the Vertex Similarity Experiments.
Adding one strict Propagation Round seems to improve the result - but just by a little bit. With or without any kind of Propagation Rounds - the values are close to or above 90 Percent, regardless of the rotation. If for any application the runtime is also important, the Enhanced Vertex Similarity with one additional Propagation Round is also very interesting. The values are in the high eights while this algorithm is about 2000 times faster (three orders of magnitude) then the Disk-Extracted Rotated ICP method.

6.5 Matching of two Topology Graphs

In order to compare the Topology Graph (TG) of the ground truth map and the one from the robot-generated map, those graphs have to be matched against each other. The bigger graph (the one with more vertices) is called second graph and the smaller first graph. This way the algorithmic treatment is easier, since less special cases have to be checked for compared to the case when the graph with the fewer vertices is matched against one with equal or more vertices.

In the last section, algorithms to calculate the similarity values between vertices from the two graphs have been introduced. Those values are used now - to a lesser extent when searching for Isomorphisms (Section 6.5.2), but heavily when applying the neighbor-growing (Section 6.5.4).

6.5.1 Graph matching

Two strategies to match the two Topology Graphs are described here. The first one is based on finding a strict isomorphism between the two graphs. It is targeted as a quite fast way to find parts of the maps that do not diverge in the connections of the graph. The only missing correspondences allowed here are those of the Spurious Vertices and Edges.

The second, more heuristic approach is the recursive neighbor growing algorithm (see Section 6.5.4), which makes heavy use of the vertex similarity. It mostly ignores the constraints that guide the isomorphism algorithm and uses a similarity value for the whole match to guide the search.

Graph Isomorphism

The word isomorphism comes from Greek and means "equal shape". In mathematics, an isomorphism is a mapping between objects that has a relationship between two properties or operations. So constructs that are isomorphic are structurally identical and only small, well defined differences are ignored.

If there exists a bijective map $f$ such that $f$ and $f^{-1}$ preserve the structure between two algebraic structures, then those structures are isomorph.

Graph Isomorphism is a concept of graph theory [Zemlyachenko et al., 1985, Fortin, 1996]. The problem here is, given two graphs, is there a 1-to-1 mapping between the vertices such that the adjacency is preserved? So, given $G$ and $G'$ with the edges $E$ and $E'$, does there exist a $f$ such that:

$$\exists f \forall v_x, v_y \in G, (v_x, v_y) \in E \iff (f(v_x), f(v_y)) \in E'$$
6.5 Matching of two Topology Graphs

The graph isomorphism problem has a long history in science, especially in mathematics, chemistry (matching the structures of molecules) and computer science. One important aspect is, that the problem is not known to be in P and it is not known to be NP-complete [Schönig, 1987]. However, its generalization, the subgraph isomorphism problem (see below), is known to be NP-complete [Cook, 1971].

Subgraph isomorphism

The problem of finding an isomorphism between a graph and a sub-graph of a second graph is called subgraph isomorphism. The map evaluation using Topology Graphs needs to find matches between Topology Graphs. That is slightly different than the problems stated above: We are only looking for subgraphs from both graphs that form an isomorphism. This is called maximum common subgraph isomorphism problem. The problem can be stated as: Given two graphs $G$ and $G'$, what is the largest induced subgraph of $G$ isomorphic to an induced subgraph of $G'$? Induced means, that edges between two vertices of the subgraph can only exist, if there is also an edge between the corresponding vertices in the other graph.

One possible solution for this problem is to build a modular product graph, in which the largest clique represents a solution for the MCS problem [Bunke et al., 2002]. An important aspect is, that we are not only interested in the decision problem (is there such an isomorphism or how big is it) but in the actual matching of vertices.

Matching of Topology Graphs

Besides all the research mentioned above, the task of matching two subgraphs from the Topology Graphs has certain, specific properties:

- The Topology Graph is planar. That means, that laid out on a 2D space, no edges intersect. This is easily shown by the fact that the origin of the Topology Graph is a Voronoi Diagram in which, by definition, if two edges are touching, a vertex is added. More precisely the Topology Graph is even a plane graph since it is laid out in 2D.
- There is a very good heuristic, namely the Vertex Similarity from Section 6.4, for the matching of the vertices.
- The Topology Graphs may contain Spurious Vertices and Edges which might or might not be used for the isomorphisms. Those Spurious elements may appear in both maps. This is a major divergence from the constraints typically asked for the isomorphisms.
- We are not just interested in the biggest common connected subgraph, but in all common subgraphs - down to certain minimum size.

Due to the differences in the application-specific details, an own solution to the maximum common subgraph isomorphism problem has been developed. As we will see, this algorithm is actually very fast, since it efficiently uses the constraints of the 2D space (the angular order of the edges connected to a vertex) and the vertex similarity as a heuristic to guide the search. For the sake of shortness, the problem is just called "Find TG-Isomorphisms".
6.5.2 Finding TG-Isomorphisms

There is one strict rule for finding the isomorphism: that the number and the geometric order of the exits (half edges leaving the vertex) of the vertices matched must be identical. The length of the edges is the only other criterion for finding the isomorphism. The vertex similarity is used to guide the search.

The Vertex Assignment Struct

The vertex assignment structure saves the information about the assignment of vertices and exits between the two graphs that are being compared. The mapping between vertices is saved as two hash maps for both directions for quick access. Another mapping is saved for the Exits already assigned.

The similarity value for the whole assignment is the sum of the similarities of the already assigned vertex pairs, using one of the vertex similarity algorithms presented above.

There are three methods defined for the struct. The methods usedFirst(TopoVertex *) and usedSecond(TopoVertex *) gather the information whether the vertex in the parameter has already been assigned to a vertex from the other graph - for both graphs respectively. The method isConsistent(TopoVertex *first, TopoVertex *second, bool &found) checks if assigning those two vertices to each other would be consistent. If both vertices have not yet been assigned then the result is true. If one of them has been assigned and the other was not, then the result if false. If both have been assigned, then the result is true if they have been assigned to each other and false if they have been assigned to different vertices. The boolean found is true if both assignments have been found and false otherwise.

Create Sorted Candidate Vertex list

The matching works on the vertices. It is desired to start with the candidate vertex pairs with the best chances for belonging to an isomorphism. So a sorted list of the vertices from the first graph with the best vertex similarities as comparison value is generated. One additional tweak here is that dead end vertices are always pushed to the end of the list - regardless of their vertex similarity value. That is because those similarities might not be very distinctive and because they are at a dead end (which with respect to the structure all look alike) and have just one neighboring vertex, which is bad for the Propagation Rounds algorithm.

Wavefront Propagation Initialization

After creating the sorted list, an empty vertex assignment is initialized. Now the sorted candidate list is iterated, starting with the best vertex. For every vertex which has not yet been assigned, it is tried to find a new isomorphism starting at said vertex $v_x$. This works by trying to match this vertex with every vertex from the other graph that has the same number of exits.

The recursive Wavefront Propagation algorithm explained below is then applied to this pair. Should this algorithm result in a complete isomorphism, it is checked if the number of matched vertices in this assignment is at least as big as the configuration value $\sigma_{isoMinVert}$. If this is the case the result is accepted. The biggest accepted assignment for $v_x$ is then added to the final assignment. Typically there will be just one accepted assignment for $v_x$, since the exploration algorithm will only
6.5 Matching of two Topology Graphs

return very good matching isomorphisms. Only if there are large scale repetitive patterns in the environment that are reflected in the Topology Graph, taking the maximum match can help finding the correct correspondences.

Recursive Wavefront Propagation

In the Wavefront Propagation algorithm, vertices that have edges to other, not yet matched vertices form the wavefront. In the beginning just the proposed pair (the best vertex from the sorted candidate vertex list, see Section 6.5.2) is in this frontier (the terms wavefront and frontier are used synonymously here). Although this is a recursive algorithm by design, most matches of vertices will not meet the strict parameters for being an isomorphism, such that mainly an iterative search is done, thus saving computation time.

In Algorithm 8 the exit assignment table, which is generated during the calculation for the Enhanced Vertex Similarities (see Section 6.4.3), is used in the function getGoodAssignments. One exit assignment is the pairwise matching of exits of one vertex from the first graph with exits of the matched vertex from the second graph. The exit assignment table consists of all permutations of matches between the exits of the first and second graph (of said vertices).

The function checkExitAssignmentForConsistency checks if one exit assignment is consistent with the current vertex assignment. This is only true with the exits are equal in number and are matched in the correct (geometric) order. Otherwise the function immediately returns false. Then it is checking if the vertices that the edges are leading to are already matched and if so, if they are respectively matched with the correct vertices. In this step the possibility that an edge might lead to a Spurious vertex is taken into account. The function thus also returns consistency if a Spurious vertex matches with a major vertex or if the major vertices match. The functions usedFirst, usedSecond and isConsistent from the Vertex Assignment Struct are heavily used here. Additionally, the length of the edges are compared. The following formula is used to be more lenient on shorter edges:

\[
\text{ratio} = \frac{\text{firstDistance} + \sigma_{\text{isoEdgeDistMinAllow}}}{\text{secondDistance} + \sigma_{\text{isoEdgeDistMinAllow}}} \\
\text{if}(\text{ratio} > 1) \text{ratio} = \frac{1}{\text{ratio}}
\]

If the ratio is then smaller than \(\sigma_{\text{isoEdgeDistFactor}}\) this exit assignment is rejected.

6.5.3 The TG-Isomorphism Example

Assume two maps with Topology Graphs like in Figure 6.16. Les us assume that both mapping algorithms delivered pretty good results, so the graphs are quite similar. We now have two graphs: \(G\) and \(G'\), both having seven vertices \(v_x\) and \(v'_x\) with \(0 < x < 8\) as well as the edges from Table 6.3.

We want to match graph \(G\), the first graph, with the second graph \(G'\). The vertex table with matches for the vertices from \(G\) to \(G'\) holds the results of the vertex similarity calculation and the exit assignment table in every cell.

Assume that \(v_1\) has the best vertex similarity value of all vertices from the first graph. It will thus be the first entry in the sorted candidate vertex list and matched first. The search for the isomorphism
Algorithm 8 Recursive Wavefront Propagation - The Recursive Wavefront Propagation

```
function recursiveWavefrontPropagation( frontier, currentVertexAssignment )
  while not frontier.empty() do
    currentVertexPair = frontier.pop()
    goodAssignments = getGoodAssignments( currentVertexPair, currentVertexAssignment )
    if goodAssignments.size() == 1 then
      addToVertexAssignmentAndFrontier( goodAssignments.top(), frontier, currentVertexAssignment )
      continue
    else if goodAssignments.empty() then
      continue
    end if
    // goodAssignments has more than one entry - recursion!
    for all assignment in goodAssignments do
      frontierCopy = frontier
      assignmentCopy = currentVertexAssignment
      addToVertexAssignmentAndFrontier( assignment, frontierCopy, assignmentCopy )
      recursiveWavefrontPropagation( frontierCopy, assignmentCopy )
    end for
  end while

function getGoodAssignments( currentVertexPair, currentVertexAssignment )
  rtnVarGoodAssignments.empty()
  exitAssignmentTable = vertexTable[currentVertexPair.first][currentVertexPair.second].exitTable
  for all exitAssignment in exitAssignmentTable do
    if checkExitAssignmentForConsistency( exitAssignment, currentVertexAssignment ) then
      rtnVarGoodAssignments.insert( exitAssignment )
    end if
  end for
  return rtnVarGoodAssignments
```

120
then starts with the match that has this good vertex similarity value: \( v_1, v'_1 \). This match is added to the frontier and to the vertex assignment list and then Algorithm 8 is applied.

The recursive Wavefront Propagation algorithm now uses the Table 6.4 to try out all exit assignments. Although there is an exit assignment similarity available, it is not used here, since for the isomorphism algorithm all possibilities are tried. For this example those values are made up - they are just included as a reminder that they have been calculated. They are used during the neighborhood growing explained in Section 6.5.4. Let us assume here it starts out with entry number 4 from Table 6.4. First it is checked if this entry is in the right angular order. Since that is not the case the algorithm now continues with entry number 5. Again it is checked if the exit assignment is in the right order and consistent with what we already have - that means with the current vertex assignment. There is currently only \( v_1 \Rightarrow v'_1 \) in the vertex assignment. For all further considerations in this example we assume that the distances are always compatible - otherwise the consistency check would return false. So **checkExitAssignmentForConsistency** returns true since all other vertices are still unmatched.

Table 6.3: Edges of the example graphs. Each edge appears twice - once for every vertex it is connected to. Notation: \( e_{n,k} = \{v_n, v_k\} \) with \( n \leq k \)
Structure-Based Map Evaluation using Matched Topology Graphs

Table 6.4: The exit assignments possible for the pair $v_1, v'_1$ with example exit assignment similarities.

<table>
<thead>
<tr>
<th>Number</th>
<th>Assignment</th>
<th>exit assignment similarity</th>
<th>is in right order</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$(e_{1,2}, e'<em>{1,2})$ $(e</em>{1,4}, e'<em>{1,4})$ $(e</em>{1,5}, e'_{1,5})$</td>
<td>0.1</td>
<td>yes</td>
</tr>
<tr>
<td>2</td>
<td>$(e_{1,2}, e'<em>{1,2})$ $(e</em>{1,4}, e'<em>{1,5})$ $(e</em>{1,5}, e'_{1,4})$</td>
<td>0.4</td>
<td>no</td>
</tr>
<tr>
<td>3</td>
<td>$(e_{1,2}, e'<em>{1,5})$ $(e</em>{1,4}, e'<em>{1,4})$ $(e</em>{1,5}, e'_{1,2})$</td>
<td>0.6</td>
<td>no</td>
</tr>
<tr>
<td>4</td>
<td>$(e_{1,2}, e'<em>{1,4})$ $(e</em>{1,4}, e'<em>{1,2})$ $(e</em>{1,5}, e'_{1,5})$</td>
<td>0.5</td>
<td>no</td>
</tr>
<tr>
<td>5</td>
<td>$(e_{1,2}, e'<em>{1,4})$ $(e</em>{1,4}, e'<em>{1,5})$ $(e</em>{1,5}, e'_{1,2})$</td>
<td>0.8</td>
<td>yes</td>
</tr>
<tr>
<td>6</td>
<td>$(e_{1,2}, e_{1,5})$ $(e_{1,4}, e'<em>{1,2})$ $(e</em>{1,5}, e_{1,4})$</td>
<td>0.7</td>
<td>yes</td>
</tr>
</tbody>
</table>

The exit assignments from entry 5 are now added to the frontier and the vertex assignments while $v_1 \Rightarrow v'_1$ is removed from the frontier (but of course it stays in the vertex assignment list). The situation, depicted in Figure 6.17, is now as follows:

- Vertex Assignment List: $v_1 \Rightarrow v'_1, v_2 \Rightarrow v'_4, v_4 \Rightarrow v'_5, v_5 \Rightarrow v'_2$
- Frontier: $v_2 \Rightarrow v'_4, v_4 \Rightarrow v'_5, v_5 \Rightarrow v'_2$

Now the recursion starts and the recursive Wavefront Propagation algorithm is called again with the new frontier and vertex assignment. One vertex pair is picked to be expanded, for example $v_4 \Rightarrow v'_5$. The method `checkExitAssignmentForConsistency` will return false for all exits assignments for those two vertices since they differ in the number of exits! Since there is no valid exit assignment, the next frontier pair is iterated. But also $v_5 \Rightarrow v'_2$ has different amounts of exits. In the next iteration $v_2 \Rightarrow v'_4$ is tried. Both $v_2$ and $v'_4$ have three exits. Table 6.5 shows the exit table for this pair.

So the exit assignments for $v_2, v'_4$ are checked with `checkExitAssignmentForConsistency`. The first one is in order, i.e. the clockwise angular ordering of the exits is correct. So the first exit match $(e_{1,2}, e'_{1,4})$ is checked. The exit $e_{1,2}$ is pointing to $v_1$. $e'_{1,4}$ is pointing to $v'_1$. Both $v_1$ and $v'_1$ are already in the vertex assignment list (see above). So we have to check if they have been matched to each other. This is actually the case so we continue and check the second exit match from the first exit assignment from Table 6.5, which is $(e_{2,3}, e'_{4,5})$. We see that there is no entry for $v_3$ in the vertex assignments, but one for $v'_5$. This is an inconsistency so `checkExitAssignmentForConsistency` returns false.

The recursive Wavefront Propagation method will now try the other exit assignments for $v_2, v'_4$. Entries 2, 3 and 4 in Table 6.5 are not in order and thus skipped. Entry 5 starts with $(e_{1,2}, e'_{4,5})$. Both $v_1$ and $v'_5$ are in the vertex assignment list. But they are not pointing to each other so this entry is also
Figure 6.17: During the Wavefront Propagation. The matching of the vertices between the two graphs is shown. $v_1, v'_1$ is not in the frontier (solid line) while the other three matches are in the frontier.
not consistent. The last entry number 6 starts with \((e_{1,2}, e'_{1,4})\). \(v_1\) is found in the vertex assignment list while \(v'_6\) is not, so also the last entry is inconsistent.

Since no consistent exit assignment could be found for \(v_2, v'_4\) the recursive Wavefront Propagation continues to work on the frontier \((v_2, v'_4\) has been removed from the frontier). Unfortunately the other assignments in the frontier \((v_4 \Rightarrow v'_5\) and \(v_5 \Rightarrow v'_2\)) both also have no consistent exit assignment. This recursion of the frontier exploration is then stopped. The result (the four assignments from the vertex assignment list) is saved.

We are back to just one vertex assignment and frontier \((v_1 \Rightarrow v'_1)\) and continue to work on the original Wavefront Propagation with the exit assignments from Table 6.4. We had just worked on entry 5. Entries 2, 3 and 4 are not in order and Entry 6 will also not advance beyond 4 matches. Only if the correct entry number one is tried a better result will be achieved.

Figure 6.18 shows the situation after the first entry of Table 6.4 has been tried:

- **Vertex Assignment List:** \(v_1 \Rightarrow v'_1, v_2, v'_2, v_4, v'_4\) and \(v_5, v'_5\)
- **Frontier:** \(v_2, v'_2, v_4, v'_4\) and \(v_5, v'_5\)

Let's assume that the recursive Wavefront Propagation now works on \(v_2, v'_2\). Table 6.6 shows the exit table.

Looking at entry 5 we see that \((e_{1,2}, e'_{2,3})\) is not consistent because \(v_1\) is assigned and \(v'_3\) is not. Entry 6 \((e_{1,2}, e'_{2,5})\) is also not consistent because both \(v_1\) and \(v'_5\) are present in the vertex assignment list but are not pointing to each other. It turns out that only entry 1 is consistent: For \((e_{1,2}, e'_{1,2})\) both entries \(v_1\) and \(v'_1\) exist and point to each other, for \((e_{2,3}, e'_{2,3})\) both \(v_3\) and \(v'_3\) are not yet matched and

<table>
<thead>
<tr>
<th>Number</th>
<th>Assignment</th>
<th>is in order</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>((e_{1,2}, e'<em>{1,4})) ((e</em>{2,3}, e'<em>{4,5})) ((e</em>{2,5}, e'_{4,6}))</td>
<td>yes</td>
</tr>
<tr>
<td>2</td>
<td>((e_{1,2}, e'<em>{1,4})) ((e</em>{2,3}, e'<em>{4,6})) ((e</em>{2,5}, e'_{4,5}))</td>
<td>no</td>
</tr>
<tr>
<td>3</td>
<td>((e_{1,2}, e'<em>{4,6})) ((e</em>{2,3}, e'<em>{4,5})) ((e</em>{2,5}, e'_{1,4}))</td>
<td>no</td>
</tr>
<tr>
<td>4</td>
<td>((e_{1,2}, e'<em>{4,5})) ((e</em>{2,3}, e'<em>{1,4})) ((e</em>{2,5}, e'_{4,6}))</td>
<td>no</td>
</tr>
<tr>
<td>5</td>
<td>((e_{1,2}, e'<em>{4,5})) ((e</em>{2,3}, e'<em>{4,6})) ((e</em>{2,5}, e'_{1,4}))</td>
<td>yes</td>
</tr>
<tr>
<td>6</td>
<td>((e_{1,2}, e'<em>{4,6})) ((e</em>{2,3}, e'<em>{1,4})) ((e</em>{2,5}, e'_{4,5}))</td>
<td>yes</td>
</tr>
</tbody>
</table>

Table 6.5: The exit assignments possible for the pair \(v_2, v'_4\).
Figure 6.18: During the Wavefront Propagation. The vertices 1, 2, 4 and 5 are matched between the two graphs. \( v_1, v'_1 \) is not in the frontier (solid line) while the other three matches are in the frontier.
for \((e_{2,5}, e'_{2,5})\) both \(v_5\) and \(v'_5\) exist and point to each other. Again the distances of the edges are also checked and found to be compatible. So this exit assignment is accepted and taken. Since there is just one valid exit assignment, the recursive Wavefront Propagation will actually continue to iterate through the frontiers. The situation is now as follows (see Figure 6.19):

- **Vertex Assignment List:** \(v_1 \Rightarrow v'_1, v_2, v'_2, v_4, v'_4, v_5, v'_5, (e_{2,3}, e'_{2,3})\)

- **Frontier:** \(v_4, v'_4, v_5, v'_5\) and \((e_{2,3}, e'_{2,3})\)

It turns out that from now on there will always be at most one consistent exit assignment! All vertices that are added to the frontier have a ”source, meaning the vertex that added them to the frontier. This “source” is obviously connected to the vertex. And only one of the exit assignments that are in order can consistently match this origin for the vertices in \(G\) and \(G'\). So after the first recursion, the Wavefront Propagation is not branching anymore but just iterating through the frontier.

In the example, at some point all vertices will be correctly matched and the frontier will be empty. Then the call to the Wavefront Propagation for the pair \(v_1 \Rightarrow v'_1\) will return. The number of vertices matched is hopefully bigger than the configuration value \(\sigma_{isoMinVert}\) - then it will be accepted and added to the final list of matches. Should there be unmatched vertices left (not in this example) it will be tried to find isomorphisms for those, too. In these next tries, the found vertex assignments from the previous runs will be kept. There are no more unmatched vertices in this example, so find isomorphisms will be finished.

<table>
<thead>
<tr>
<th>Number</th>
<th>Assignment</th>
<th>is in order</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>((e_{1,2}, e'<em>{1,2})) &lt;br&gt;((e</em>{2,3}, e'<em>{2,3})) &lt;br&gt;((e</em>{2,5}, e'_{2,5}))</td>
<td>yes</td>
</tr>
<tr>
<td>2</td>
<td>((e_{1,2}, e'<em>{1,2})) &lt;br&gt;((e</em>{2,3}, e'<em>{2,3})) &lt;br&gt;((e</em>{2,5}, e'_{2,5}))</td>
<td>no</td>
</tr>
<tr>
<td>3</td>
<td>((e_{1,2}, e'<em>{2,5})) &lt;br&gt;((e</em>{2,3}, e'<em>{2,3})) &lt;br&gt;((e</em>{2,5}, e'_{1,2}))</td>
<td>no</td>
</tr>
<tr>
<td>4</td>
<td>((e_{1,2}, e'<em>{2,3})) &lt;br&gt;((e</em>{2,3}, e'<em>{2,5})) &lt;br&gt;((e</em>{2,5}, e'_{1,2}))</td>
<td>no</td>
</tr>
<tr>
<td>5</td>
<td>((e_{1,2}, e'<em>{2,3})) &lt;br&gt;((e</em>{2,3}, e'<em>{2,5})) &lt;br&gt;((e</em>{2,5}, e'_{1,2}))</td>
<td>yes</td>
</tr>
<tr>
<td>6</td>
<td>((e_{1,2}, e'<em>{2,5})) &lt;br&gt;((e</em>{2,3}, e'<em>{1,2})) &lt;br&gt;((e</em>{2,5}, e'_{2,3}))</td>
<td>yes</td>
</tr>
</tbody>
</table>

Table 6.6: The exit assignments possible for the pair \(v_2, v'_2\).
6.5 Matching of two Topology Graphs

Figure 6.19: During the Wavefront Propagation. The vertices 1, 2, 3, 4 and 5 are matched between the two graphs. $v_1, v'_1$ and $v_2, v'_2$ are not in the frontier (solid line) while the other three matches are in the frontier.

6.5.4 Grow Neighbors

The grow neighbors algorithm is quite similar to the TG-Isomorphism algorithm from above. It uses a recursive Wavefront Propagation approach to extent the match according to the exit assignments possible for a match in the frontier. Unlike the isomorphism method, the exit assignment does not have to be in order and also vertices with different numbers of exits are expanded. The consistency check (growNeighboursCheckExits) only returns false if an inconsistency in the vertex assignments is detected. The advantage of the grow neighbors algorithm is, that it is more lenient. It might match vertices that could not be matched with the TG-Isomorphism. But of course also the danger of making false matches is increased. That is why the grow neighbors algorithm is only used to grow the already found matched from the TG-Isomorphism, such that the likelihood of false matches is greatly reduced.

The vertex similarity is used heavily. If a new match is found it is checked if the similarity of the whole match falls below a certain threshold. If that is the case the match is not added.
6.5.5 Example Match of two Topology Graphs

Figure 6.20 shows an example match between the ground truth map from RoboCup 2010 in Singapore and a robot generated map just using the Topology Graph isomorphism. 24 vertices were matched to each other and five matches that a human would very likely have made are missing. One can see some differences between the ground truth map and the robot map - for example around the car that was placed in the top of the competition arena. Actually the robot map represents the actual scene better than the ground truth map which was just coarsely modeled after the arena. This results in some differences in the graphs which make the isomorphism stop at these points.

This also happens on vertex 18 in the left map, which is correctly mapped to vertex 20 in the right map. Due to the differences in the map, vertex 18 has five exits (to 17, 20 and 9 and two from the loop) while vertex 20 from the right map has only four exits (to 18, 23, 17 and 11). Thus the isomorphism algorithm stops here.

Figure 6.21 shows the match after applying the grow neighbors algorithm. The four matches that were grown are shown in red.
6.5 Matching of two Topology Graphs

Figure 6.21: Example match using the isomorphism algorithm and then the grow neighbors approach. There are 28 assignments made and one assignment is missing from the left graph (vertex 11).

6.5.6 Discussion of the Configuration Parameters

For the creation of the Topology Graphs, the vertex similarity calculation, and the TG-matching a number of configuration parameters are used. They are listed with a short description of each of them in Appendix A.

Eleven parameters are used for the extraction of the Topology Graph from the grid map. Those have to be determined by the user for the specific environment. The resolution of the maps also has to be taken into account, but this is not a problem since all maps can be scaled to a common resolution before applying them to the algorithm. For future work it is anyways planned to go from pixel coordinates (of the CGAL Sites) to the coordinates of the geo-registration of the map, using a simple transformation. Then the parameters that are currently in pixel will be in meter. The determination of those parameters has to be done only once for one environment, for example by testing them with the ground truth map.

Some are easy to find (e.g. $\alpha_{\text{shape}}$ just depends on the biggest opening of the mapped area to the outside), while others depend on each other. For example, a small value for $\delta_{\text{min2Obst}}$ will leave many short edges, such that more dead end removals will be required. The threshold for those dead end removals (e.g. $\delta_{\text{deadEnds}}$) influences the choice of $\delta_{\text{angleCalcEnd}}$ ($\delta_{\text{angleCalcEnd}}$ has to be smaller than
\( \delta_{\text{deadEnd}} \) to ensure that there is actually enough path left in all edges to calculate an angle). Also if \( \delta_{\text{spuriousEdge}} \) is bigger than \( \delta_{\text{deadEnd}} \) no dead end will be marked as Spurious.

Typically, \( \delta_{\min2\text{Obst}} \) should be chosen quite big, but small enough such that no desired edge is filtered out, and a little extra space (since the maps will not be as good as the ground truth map). The further values can then be adjusted such that the graph “looks good”.

Changes to the four sets of offset and maximum parameters for the vertex similarity calculation only moderately effect the similarity values, such that their default values as given in Appendix A are good choices in most cases.

The three values for the graph matching, on the other hand, have heavy influence on the match result. \( \sigma_{\text{isoMinVert}} \) is the minimum number of vertices to be matched for that an isomorphism is accepted. Small values (like four or five) increase the danger of not corresponding vertices to be matched, while big values (eight and above) can reject good matches. The value is influenced by the size of the environment and of how well connected the graph is, e.g. how many loops are there. The graph is bigger and well connected, bigger values for \( \sigma_{\text{isoMinVert}} \) can be chosen because even if some unmatchable vertices are in the graph (because they have the wrong number of edges connected, for example), the isomorphism can grow around them using the loops. Also the variables that are used to determine if the length of two edges are similar enough such that they might be matched (\( \sigma_{\text{isoEdgeDistMinAllow}} \) and \( \sigma_{\text{isoEdgeDistFactor}} \)) influence the error rate of the matches. Being more lenient helps to still match similar vertices if maps contain larger errors, but also the risk of matching wrong vertices is increased.

### 6.6 Map Evaluation using Matched Topology Graphs

Evaluating the map quality after matching the Topology Graph \( G \) of the map with the Topology Graph \( G' \) of a reference map is similar to the approach of the Fiducial Map Metric (see Section 5.1). Again there are a number of attributes that we are interested in:

- Coverage: How much area was traversed/visited.
- Resolution quality: To what level/detail are features visible
- Global accuracy: Correctness of positions of features in the global reference frame
- Relative accuracy: Correctness of feature positions after correcting (the initial error of) the map reference frame
- Local consistencies: Correctness of positions of different local groups of features relative to each other

All map attributes scored get values between 0 and 100%. The value 0 represents very bad quality and perfect results get a value of 100%. Some attributes use distances between two positions to measure the error. For those, there are maximum distances \( d_{\text{max}} \) defined which are considered to be the worst case for the attribute.

#### 6.6.1 Coverage

The straight forward approach for coverage is to relate the number of vertices matched from \( G \) to \( G' \). Only non-Spurious Vertices are counted here because, as explained earlier, those vertices might or
might not appear in the other graph. This directly translates into a score with:

\[
\text{coverage} = 1 - \frac{|V|}{|V'|}
\]

But the Topology Graph might cover long hallways with one very long edge. It would be “unfair” to count the vertex in this dead end the same way than a very short one. So taking the length of the edges is another possibility. But here one has to take care not to count edges from \( G \) that lead to unmatched vertices, since those might be created by erroneous map parts. So a better approach to calculate the coverage is to use only the length of the edges that have been matched.

Unfortunately the edges were not matched explicitly, so this information is not readily available. The solution is to visit every matched vertex in \( G \). For every such vertex \( v_m \) we look at all its edges. If those lead to vertices that have also been matched they contribute to the total sum. While doing this, the fact that Spurious Vertices might have been skipped has to be taken into account.

For the Topology Graph of the ground truth map \( G' \) the whole length should be counted, since this is the ideal reference map that should be aimed for. So at the end, the coverage score is the ratio of matched edge length by the edge length of the reference graph. This should be capped at 100%, in case very complete maps feature slightly longer paths due to mapping errors.

The coverage calculation is the only attribute that takes unmatched parts of the graphs into account. For all other attributes the score does not decrease even if a lot of vertices are not matched.

### 6.6.2 Resolution Quality

The Topology Graph is a very general representation of the environment. So very little can be said about the resolution quality using this approach. The only information a good matching gives about the resolution quality is, that it is at least good enough to represent the topological structure of the environment. Therefore no score is computed for this attribute.

### 6.6.3 Global Accuracy

The global accuracy is about correctness of the pose (position and orientation) of the map in its global reference frame. It is computed by calculating the mean square error of the position of the matched vertices from \( G \) and \( G' \). It is enough to consider the position of the vertices, since an error in orientation will automatically lead to position errors of the vertices.

The relation of the found squared error to a configured maximum error \( d_{\text{GlobalAccuracy}} \) is the accuracy value. This value is capped at 100% for very bad maps.

### 6.6.4 Relative Accuracy

The relative accuracy corrects the map pose such that there is a maximum overlap with the reference map. This is done by rotating, translating or even scaling the Topology Graph accordingly. The parameters for this transformation are determined using Horn’s Algorithm [Horn, 1987] on positions of the matched vertices.

After this correction step, the same formula as for the Global Accuracy is used - potentially with a different configuration parameter \( d_{\text{Rela}} \) instead of \( d_{\text{GlobalAccuracy}} \).
6.6.5 Local consistencies

The local consistency is a rather fuzzy measure. It is used to describe the topological correctness of the map. Obviously the Topology Graph is predestined for this use, as it has been designed for this purpose. So there are many attributes for the Topology Graph that could be used here.

The vertices are the natural starting point for this calculation. The topology is given by their connections, the edges. The most relevant topological information is the order of the edges originating from a vertex (exits) and their length. This determines the algorithm for computing the local consistency.

We go through all matched vertices. We look at the vertices \( v'_{x} \) of the reference map \( G' \). For every exit \( e'_{x,y} \) from \( v'_{x} \) we determine the exit score \( s_{e'_{x,y}} \). Rays are skipped and the score gets assigned a 1.0 (very bad) if the exit leads to an unmatched vertex. Otherwise the score \( s_{e'_{x,y}} \) is calculated using the edge lengths of \( e'_{x,y} \) and the matched counterpart \( e_{i,j} \):

\[
s_{e'_{x,y}} = \frac{\left| \text{length}(e'_{x,y}) - \text{length}(e_{i,j}) \right|}{d_{\text{max}}^{\text{LocalConsistency}}}
\]

If \( s_{e'_{x,y}} \) should be above 100% it is capped at this value.

There is another alternative which also supports the different consistency classes that were introduced in the Fiducial Map Metric. There the classes were defined as different difficulty classes for consistencies, i.e. how hard is it for a robot to keep the consistency. They were determined by hand by taking the path length from needed to traverse between the features into account. The Topology Graph represents those path length in its edges. The next section (6.6.6) describes how a table that contains the minimum path distance between any two vertices of a graph is computed. This distance information is used to put all vertex pairs from the reference Topology Graph \( G' \) into a number of difficulty classes - similar to those from the Fiducial Map Metric. Those difficulty classes are defined by thresholds for the path distance that the user has to specify according to the needs of his application of the map and the map quality value.

Then the distance of the geometric locations is computed for every such pair. The same is done for the vertices in the map graph \( G \). Now for every vertex pair from the reference graph where both vertices are mapped to the map graph the error in the distance is computed. This error is summed to its according class such that at the end the average of all such errors can be calculated for every class.

6.6.6 Calculating the Distance Table

For every vertex in a Topology Graph its shortest distance along the edges to every other vertex is calculated. This is done with a wavefront based approach [Dijkstra, 1959]. We iterate through all vertices. At the beginning of each iteration, each vertex gets assigned infinity as the distance to the current vertex. Now all edges leading from the vertex are added to the frontier. The frontier is being worked on till it is empty. We pop an edge from the frontier. Then the distance saved in the source of the edge (0 for the start vertex) and the edge length is added. The result is compared with the value saved in the target vertex. If the result is smaller, the target value gets updated and all edges leading from that vertex are added to the frontier.

At the end every vertex has the shortest distance to the start vertex saved. Now the same thing is continued with the next start vertex. See Algorithm 9.
Algorithm 9 Distance Algorithm – Calculate the distances between all vertices in a Topology Graph.

for all \( v_x \) in \( G \) do
\[
v_x^d = 0
\]
Initialize all other distances with infinity: \( \forall v_y \in G, v_y \neq v_x : v_y^d = \text{INF} \)
Add all exits of \( v_x \) to the frontier
while \(!\text{frontier.empty()}!\) do
\[
e_{i,j} = \text{frontier.pop()}
\]
\[
tmp = v_i^d + e_{i,j}
\]
if \( tmp < v_j^d \) then
\[
// A shorter path has been found!
\]
\[
v_j^d = tmp
\]
Add all exits of \( v_j \) to the frontier
end if
end while
end for

6.7 Computing the Brokenness using matched Topology Graphs

In [Birk, 2010] another map attribute, the brokenness, was proposed. It is defined as "the degree with which a map can be partitioned into regions that are locally consistent with ground truth but off relative to each other." Figure 6.22 shows maps of the same environment but with different numbers of broken regions.

Calculating the brokenness using a Topology Graph that is matched against a ground truth Topology Graph is relatively simple. A subset of connected vertices, a Brokenness Group, has to be found whose minimal mean square error regarding the location of the vertices compared to the matched ground truth vertex locations does not exceed a certain threshold. This minimal mean square error is easily computed using Horns algorithm. This subset is saved. The process is iterated over the remaining vertices that are not yet part of a subset. A subset is only accepted if it contains at least a certain number of vertices.

The number of subsets found minus one is then the brokenness degree as defined by Birk.

6.7.1 Finding Brokenness Groups

The goal is to find a set of vertices that geometrically fit to each other. For this, two thresholds are defined. The \( \epsilon_{\text{brokenThres}} \) describes the upper bound on how big the Mean Squared Error (MSE) of the vertex locations with regard to the matched ground truth vertices can be. And the \( \epsilon_{\text{brokenMinV}} \) is the minimum number of vertices needed in this found set such that it is considered a Brokenness Group.

The idea behind the brokenness concept is, that the part of a map that is broken has in itself no big local errors. Maps are typically broken because of either bump noise (the wheels/ tracks slip on the ground, maybe because the robot hit an obstacle, thus the odometry is broken) or because of a failed registration of the sensor data. Even in SLAM algorithms those large local residual errors might occur. Areas of maps where the mapping consistently failed and which thus show no large-scale identifiable structures are not considered broken areas. Since broken parts are internally relatively intact, the Topology Graph should match nicely, too. This idea guides the implementation for the brokenness
Figure 6.22: Maps based on data from simulations. The undisturbed map together with maps with increasing degrees of brokenness (from [Birk, 2010]).
search, which is a Wavefront Propagation approach.

We are starting with one matched vertex pair \((v_x, v'_y)\) from \(G\) and \(G'\). The neighbors of this match have to be added to the frontier of to be tried matches.

**Frontier adding**

The incident vertices (neighbors) of \(v_x\) from graph \(G\) are iterated. If those were matched to a vertex from \(G'\) during the Topology Graph Matching, they are about to be added to the frontier. But before that, it is checked that this vertex match is not already in another Brokenness Group, and not in the current Brokenness Group, nor already in the frontier.

If the Graph Matching also used the Neighbor Growing approach, it could happen (though quite unlikely), that a neighbor of \(v'_y\) is matched to a vertex which is not a neighbor of \(v_x\). That is why the same process of adding matches to the frontier is repeated for the neighbors of \(v'_y\). Most often this will not add anything to the frontier, because the matches were already added when the incident vertices of \(v_x\) were added.

**Adding the Best Match to the Brokenness Group**

Now a new match from the frontier should be added to the current Brokenness Group. So we iterate through the frontier matches. For every frontier match it is calculated what the error value of the current Brokenness Group would be, if it were added to the Brokenness Group. This error is the Mean Squared Error (MSE) of the best fitting between the vertex locations of the vertices in the set from \(G\) and the matched vertices in the set from \(G'\). It is calculated using Horn’s algorithm.

At the end the best match (the one with the lowest error value) is taken. If this error value is above the threshold \(\epsilon_{\text{brokenThres}}\) we return with the current Brokenness Group. Otherwise the best match is added to the Brokenness Group and its neighbors are added to the frontier using the approach described above. This process of calculating the error values of the frontier and adding the best match is repeated until either the frontier is empty or \(\epsilon_{\text{brokenThres}}\) has been exceeded.

**Finding all Brokenness Groups**

So far it was described how to grow one Brokenness Group using a Wavefront Propagation approach. Finding all Brokenness Groups is then fairly simple. All matches of vertices between \(G\) and \(G'\) are iterated. If the match is not already in a Brokenness Group it is taken as the initial match for the Wavefront Propagation. The resulting Brokenness Group that this match created is saved.

After that the biggest Brokenness Group is selected. If it has at least MinNumberVertices it is added to the final set of Brokenness Groups. This process is repeated until no valid Brokenness Group can be extracted anymore.

The reason why the number of vertices is taken as a selection criterion and not the error value is, that the error value will always tend towards the threshold. Matches are added until there is none left that would not exceed the threshold. So the values of the errors are often similar. Taking the biggest group is a good choice, since one can be fairly certain that a big group within the error threshold is actually a good matching map part.
Improvements

Sometimes it could happen, that a part of a map that is broken is not represented by a connected Topology Graph. This can be, because there can be some unmatched vertices due to too big map differences. In such a case there are two Brokenness Groups generated, because the Wavefront Propagation cannot "skip" through unmatched vertices. There are two approaches to mitigate this.

In the first approach, after finding a Brokenness Group of at least the minimum size, all other unused matches between vertices from $G$ and $G'$ are tried. The one with the lowest error is taken and added if the threshold is not exceeded. This is repeated until no more unused matches are left or all of them exceed the threshold.

The second approach is used after all Brokenness Groups have been extracted. Here all groups are pair-wisely tested if the union of the group does not exceed the threshold. If that is the case the two groups are joined.

The first approach adds any match from anywhere in the map to the Group as long as it fits. So if any match should fit just by coincidence, it would be added here and thus not be available for another Brokenness Group. This could prevent the proper finding of the other Brokenness Group. But that scenario is quite unlikely. The second approach only adds full Brokenness Sets (which have at least MinNumberVertices), so small groups of matches will not be added to the group. If one is just interested in the brokenness number and not the actual vertices belonging to a set this is fine. Another property of the second approach is that two Brokenness Groups might belong to the same broken map part. But each Brokenness Group tries to maximize its error threshold by potentially adding not matching vertices until that is not possible anymore. So the union of the Brokenness Groups could have even more vertices that theoretically should not belong in this group. Those could exceed the threshold and thus prevent the merging of those Groups.

The first approach seems to have less disadvantages and performed well during the experiments, and it is thus preferred by the author.

6.7.2 Brokenness Experiments

The maps from Figure 6.22 were taken and applied to the Image Thresholder from Section 4.1 and the Alpha Shape Remover from Section 4.3 (with 16 for the alpha value and also 16 for the length). Maps 1 through 5 were then applied to the Topology Brokenness calculation, using Map 0 as ground truth. The computation was done in pixel coordinates with a SquaredErrorThreshold of 80 and a MinNumberVertices value of 5.

In Figure 6.23 Map 2 is shown together with the matches to the ground truth map and the found Brokenness Groups. The green matches were generated by the Topology Graph Isomorphism and the red matches by the Neighbor Growing. There are 61 Isomorphism matches and 6 neighborhood matches. Eight vertices from map two could not be matched against any of the vertices from the ground truth map (this one has seven unmatched vertices).

The numbers represent the Brokenness Group the vertex belongs to. The biggest group with 44 matches (which also means 44 vertices per map) is number 0 - this part represents the unbroken map. Horns algorithm of course also calculates the transformation between two point sets. The angle from this transformation is 0 degrees for set 0 and 20.6 degrees for set 1, which is correct.

Table 6.7 gives some statistics about the matching between the ground truth map and the broken
6.7 Computing the Brokenness using matched Topology Graphs

Figure 6.23: The Ground Truth Map (Map 0 - at the bottom) matched to Map 2 (at the top left), which has two broken areas. The green matches are made by the Topology Graph Isomorphism and the red matches by the Neighbor Growing. The numbers represent the brokenness group the vertex belongs to. As can be seen, both the degree of brokenness and the related partitions are correctly determined.

<table>
<thead>
<tr>
<th>Map</th>
<th>Number of Vertices</th>
<th>Number of Half Edges</th>
<th>Matching Vertices with Ground Truth Isomorphism</th>
<th>Matching Vertices with Ground Truth Neighbor Growing</th>
<th>Missing</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>75</td>
<td>176</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>74</td>
<td>172</td>
<td>58</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>76</td>
<td>178</td>
<td>61</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>79</td>
<td>188</td>
<td>49</td>
<td>9</td>
<td>17</td>
</tr>
<tr>
<td>4</td>
<td>79</td>
<td>186</td>
<td>50</td>
<td>9</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>79</td>
<td>186</td>
<td>47</td>
<td>9</td>
<td>19</td>
</tr>
</tbody>
</table>

Table 6.7: Data about the Topology Graphs of the different maps and their matches against the ground truth map.
In Table 6.8 the different Brokenness Groups for the matches are presented. Please also refer to Figures 6.23 and 6.24 for a visualization of those results. Maps 1 and 2 deliver the expected result and detect a brokenness of 1 respectively 2.

For maps 3, 4 and 5, the left-most brokenness cannot be detected. It is very small and only represented by three vertices. Since MinNumberVertices is 5 this broken part cannot form its own set. All other broken parts from maps 3, 4 and five are nicely detected.

Map 5 is interesting. Groups 1 and 4 have nearly the same angle, which is also true for the actual brokenness at those places in the map. They are not in the same group because they have different centers of rotation, such that the vertex positions differ. But close to the actual center of rotation this effect is quite small. Due to that fact and some "bad luck“, both groups had one vertex added to their set at the other groups place during the global vertex search described in approach one of the Improvement section above.

The runtime for those experiments are very short. The extraction of the Topology Graphs takes much longer than the finding of the Brokenness Groups. Everything together is done for one of the experiment maps in under one second. The algorithm finds the brokenness almost as reliable as [Birk, 2010]. Only very small scale broken parts which are not represented by enough vertices in the Topology Graph cannot be detected. But the image registration approach relies on similar "looking“ maps, i.e. local features. In contrast to that this approach also works on maps with differing resolution qualities much noise on the walls, as long as the topological structure can still be extracted.

Table 6.8: The different brokenness groups.

<table>
<thead>
<tr>
<th>Map</th>
<th>Detected Brokenness</th>
<th>Brokenness Set Number</th>
<th>Number of Matches</th>
<th>Squared Error</th>
<th>Angle in Degrees</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>44</td>
<td>20.8</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>18</td>
<td>57.3</td>
<td>20.6</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0</td>
<td>44</td>
<td>58.5</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>11</td>
<td>8.9</td>
<td>-69.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>9</td>
<td>79.9</td>
<td>21.8</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>0</td>
<td>35</td>
<td>68.5</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>8</td>
<td>69.9</td>
<td>23.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>6</td>
<td>44.4</td>
<td>-70.8</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>0</td>
<td>28</td>
<td>77.7</td>
<td>0.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>10</td>
<td>79.7</td>
<td>7.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>9</td>
<td>79.9</td>
<td>21.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>7</td>
<td>77.3</td>
<td>-70.5</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>0</td>
<td>19</td>
<td>76.8</td>
<td>0.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>9</td>
<td>71.4</td>
<td>7.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>9</td>
<td>79.9</td>
<td>21.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>7</td>
<td>77.3</td>
<td>-70.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>6</td>
<td>53.8</td>
<td>7.8</td>
</tr>
</tbody>
</table>
6.7 Computing the Brokenness using matched Topology Graphs

(a) Brokenness degree 1  
(b) Brokenness degree 3  
(c) Brokenness degree 4  
(d) Brokenness degree 5

Figure 6.24: The brokenness groups for the different maps.
Chapter 7

Map Evaluation Experiments

In this chapter the place-based Fiducial algorithm and the structure-based Map Evaluation using Matched Topology Graphs are tested using experiments. First two examples of field test that used the Fiducial algorithm are presented. Those are nice demos, but they are not sufficient to validate the Fiducial approach, mainly because not enough different maps were generated at these events. In Sections 7.3 and 7.4 more significant experiments with more maps are conducted.

7.1 Fiducials at MAGIC 2010

The Multi Autonomous Ground-robotic International Challenge (MAGIC 2010) was held in September 2010 in Adelaide, Australia [DSTO, 2010]. This was an event jointly sponsored by the Defense Science & Technology Organization (DSTO) in Australia and the Research Development & Engineering Command (RDECOM) in USA. The goal was to test multi-vehicle robotic teams that can execute an intelligence, surveillance and reconnaissance mission in a dynamic urban environment. More than 1.5 million US$ of price money were distributed to the teams.

The National Institute of Standards and Technology was asked by the organizers to technically support the live adjudication of the event and to provide the map scores. During his research visit at NIST the author of this thesis designed and implemented, together with a colleague from NIST, the adjudication software and was subsequently invited by NIST to do the map evaluation and to assist in adjudicating the event in Adelaide.

This challenge was split into three phases which had to be cleared of Objects of Interest (OOI) subsequently. After the team ends a phase (be declaring so), the robots batteries could be replaced and they were positioned at the new start point. Each phase featured a different environment and different OOIs. In order to clear a phase, a team of robots had to coordinate. Sensor robots would search the area for static and mobile OOIs. Static OOIs (Figure 7.1(a)) are red barrels put in the environment.

Mobile OOIs are persons following fixed paths wearing red clothes. There is an activation zone for OOIs (5m diameter for static OOIs and 10m diameter for mobile OOIs). If robots enter the activation zone of a static OOI, everything within its lethality zone (everything within line of sight and a distance of less than 10m away) gets blown up - including the static OOI itself. For mobile OOIs the lethality zone has the same extension as its activation zone. Robots entering this zone will be disabled by the OOI.
Map Evaluation Experiments

(a) A static Object of Interest (OOI - the red barrel) from MAGIC 2010. Visible at the bottom is the pink circle of the activation zone of that OOI (2.5m radius). If robots enter this zone the OOI and everything within its lethality zone (everything within line of sight and a distance of less than 10m away) gets “blown up”.

(b) A fiducial in the maze of phase 1.

Figure 7.1: Objects of Interest and Fiducials at MAGIC 2010.

Once a sensor robot has found a static OOI it can call a disruptor robot that has to enter the neutralization zone (4.5m) and mark the OOI with a laser for 30 seconds. Then the static OOI is deactivated. Mobile OOIs have to be tracked by two sensor robots for 15 consecutive seconds. After that the mobile OOI is disabled. Everything within a radius of 5m will also be disabled - including potentially present non-combatants (persons following specific paths like mobile OOI, but dressed in blue), which would cause a big penalty.

Mobile OOIs and non-combatants are constantly being tracked by an ultra-wide-band (UWB) tracking system (see section 7.1.2). Robots are tracked if they are within the coverage area of the tracking system.

Static OOIs (red barrels) and the stands for the tracking system are barrels that can act as a kind of fiducial. Since there were not enough fiducial parts available to outfit the whole challenge area (roughly 300m x 200m, albeit with big non-accessible areas) the above mentioned barrels were used for map scoring. Only in the indoor part of phase one fiducials were used (see Figure 7.1(b)). The locations of all such fiducials/barrels are shown in Figure 7.2.

Figure 7.3 shows another ground truth map for MAGIC 2010. The organizers provided a CAD drawing made by a survey company that represented some of the obstacles. This information was then integrated with the changes made to the environment made afterwards (e.g. the maze from phase one and all the additional fences) by the author. For Figure 7.2 a geo-referenced photograph was provided. Using this and the CAD drawing, the author supplemented all other information seen. Figure 7.3 shows all major obstacles in black. This is thus an approximation of what an ideal 2D map
Figure 7.2: The ground truth map of the Adelaide Fairgrounds - the place of the competition. The blue dots mark the locations of the fiducials. The blueish areas are inaccessible to the robots.
generated by robots should look like. Only small scale objects (also the fiducials/barrels) are missing.

In Figure 7.4 the best map that has been created by a team is shown. This map only shows phase 1. The start location was in the top left. The obstacles are depicted in blue, free space in white and the paths of the different robots in purple. The locations of the static OOI s are marked in red. One can see, that the maze has not been fully explored. Furthermore is the map distorted a bit and parts of the walls towards the end of the passage way (in the center of the map) are missing.

7.1.1 Map evaluation during MAGIC 2010

The barrel locations, the fiducials and 37 different Lexicon Items defined by the organizers of MAGIC 2010 were used for map scoring. Lexicon Items include features such as cars, building entries and gates. The rules of MAGIC 2010 already defined a map scoring system. This was a variant of the Fiducial approach. The main differences are that different features are used, which have to be identified by the teams. The global positions of those features were thus directly provided by the teams.

All other map attributes are calculated in framework of the Fiducial rules. To aid this calculation, which was done by hand, the ground truth map seen in Figure 7.2 was created. Each feature is indicated by a red dot surrounded by a 1m radius yellow circle and a 3m radius blue circle. Then the global positions of features found by the teams were classified in three categories: inside the yellow circle, inside the blue circle or outside, and were scored accordingly.

The indoor environment of phase 1 was sufficiently covered with barrel fiducials. Here it was tried to directly apply the Fiducial scoring from Section 5. Unfortunately, the only good map produced for this area had a resolution quality just too bad to be able to extract the fiducials from the map. In retrospective one can conclude, that for the performance shown in this competition the fiducials should have had a bigger diameter. This unexpected problem was caused by the fact that mapping such a big area as in MAGIC with multiple robots is a task that is much more difficult than the following experiments.

7.1.2 Adjudication software for MAGIC 2010

The MAGIC 2010 adjudication software, developed to a great extend by the author of this thesis during his research visit at NIST, is a networked system containing multiple judging computers and servers. The ultra-wide-band (UWB) tracking system provides a stream of positions for tags that are in the coverage area of the receivers. This stream is usually put through a filter to smooth the paths of the tags and to remove outliers. Then the data is fed into the adjudication server. Several adjudication clients, featuring a GUI, are connected to the server. They can perform actions on the different entities (object-of-interest (OOI), non-combatant or robot) like disabling, freezing or scoring simultaneously. The other task of the users of the clients is to update the positions of the robots if they are not within the covered area of the UWB tracking system. For this they used the cameras distributed over the area, the video feeds provided by the teams and informations provided by the judges in the field.

The adjudication software was written using Qt and the Neutral Messaging Language NML [rcs, 2006].

Figure 7.5 shows a test that was performed in the so called “red square”. A number of tags from the UWB tracking system used for localization were put on the ground. The adjudication server would
7.1 Fiducials at MAGIC 2010

Figure 7.3: Obstacle map extracted from the ground truth map. This is approximately how an ideal map should look like - only a small scale objects such as fiducials are not captured in this map.
Figure 7.4: A map of phase 1, generated by multiple robots. The red lines are the paths of the robots.
get the position, id and time of those tags from the UWB System. The adjudication clients connected to the server then correctly draw the positions of the different entities.

Figures 7.6 and 7.7 show screen-shots from the action during the missions. One can see that OOs and/or robot were disabled or “blown up”. The string “Path-Data” behind a robot indicates that the tag placed on that robot was inside the coverage of the UWB tracking system. “Manual” means that the robot position was traced by one of the adjudication software users by hand. “Manual” control of the robot position overrides the UWB system, even if the robot is (maybe just barely) inside UWB coverage.
Figure 7.5: A screenshot of the GUI of the MAGIC adjudication software during tests performed with the UWB tracking system. A great number of tags are laid out in the “red square”. Roles like object-of-interest (OOI), non-combatant or robot were already assigned to the tags. Refer to the legend of the software for the meanings of the different icons.
Figure 7.6: A screenshot of the GUI of the MAGIC adjudication software at the end of phase 1. Two static OOls are scored and three robots are disabled. The lethality-, neutralization-, and activation-zones of the static OOls can be seen. No robot is present in the map. The colored triangles are roughly the field of view and names of the cameras that the judges can use to observe the robots.
Figure 7.7: A screenshot of the GUI of the MAGIC adjudication software from phase 3. Some robots are all still near the start point at the top. Others are in the “South Boulevard”, on their way to the “Grassy Area”. Five sensor robots have already been disabled by the mobile OOI's (red disk). The paths for the mobile OOI and the non-combatants are shown as red respectively blue dotted lines.
7.2 Fiducials at Disaster City 2010

The data presented here has been gathered during the Response Robot Evaluation Exercise Disaster City 2010 [Jacoff and Messina, 2006, Birk et al., 2009a]. There, a maze in a building on an inclined plane has been mapped as well as an adjoining hall and the area in front of this building. Figure 7.8 shows the ground truth map for the maze. Two different types of fiducials have been applied there. Only for the maze the exact poses of the fiducials are known and thus only this part of the maps is used to evaluate their quality here.

As fiducials, barrels with a radius of thirty centimeters and a height of one meter are used. They come in two different configurations.

**Percent Fiducials** consists of two barrels and one piece of square plywood (about 1.2m x 1.2m or 4ft x 4ft). Those are mainly used outdoors where walls are less present.

**Wall Fiducials** are built by cutting one barrel in half and putting both halves on opposite sides of a wall, forming a nearly exact circle when viewed from the top. They come also in a variation where the barrel is cut into a 1/4th and a 3/4th piece which are placed on corners.

The sensors used are a Hokuyo UTM-30LX laser range finder (LRF) with a field of view of 270°, an angular resolution of 0.25° and a range of above 30 m as well as a Xsens MTi gyro and accelerometer. Those were mounted on a stick and connected to a Laptop. The sensor data was collected by a person holding the stick with the sensors slowly walking through the maze and the environment.

Two different mapping algorithms were tested. Two maps were created for each mapping algorithm. Those maps are shown in Figure 7.9.

Figure 7.8: The maze and the fiducials in the ground truth map. The ranges needed to traverse from one fiducial part to the other correspond to the color of the void: gray=four pallets; white=eight pallets; black=twelve pallets; striped=single barrel (no group = no distance).
Figure 7.9: The maps from Disaster City 2010. Red crosses indicate Fiducials that could not be identified.
7.2 Fiducials at Disaster City 2010

Figure 7.10: The Map colored with the Thresher to free (white), unknown (gray) and occupied (black).

The results of the Fiducial map scoring for the four maps gathered are compared here. In Figure 7.8, the ground truth map including the location of the fiducials is shown. There are a total of 16 fiducial parts present. 14 of those form seven groups (pairs) since they are on opposite sides of walls while two do not belong to a complete fiducial.

Each group was assigned a distance, measured in “pallets” (the 1.2 m square area for each element). This distance reflects the minimum number of pallets that has to be traversed to get from one part to the other in the group. The values are 12 for one fiducial, 8 pallets for two more and 4 for the other four fiducials. The 12 and 8 pallet groups are used for the long range consistency while the four four pallet groups are used for the short range consistency.
Table 7.1: Count of identified barrels in the second column. Complete and found short- and long-range groups in the third and fourth column (e.g. 3 of 6 means that for 6 groups at least one barrel was found while for three groups both barrels are identified). One of the long-range groups of Map 4 has a distance $d$ bigger than $d_{\text{max}}$.

Table 7.2: Measured distances/ errors for the identified barrel parts for the relative accuracy attribute.

### 7.2.1 Coverage and Local Consistency Results

Figure 7.10 shows the maps after the colorization step from Section 4.1. In Table 7.1, the counts for identified barrels are provided. The crosses in the Figures 7.9 mark the missing fiducial parts. Those are used to calculate the coverage as shown in Table 7.3. The first table furthermore contains data for the local consistencies. For the sake of simplicity, the value for $d_{\text{max}}$ and for the discretization are chosen to be one barrel radius (30 cm). This way it is easy to just count all those groups/ pairs which are within said distance.

### 7.2.2 Relative Accuracy Results

Table 7.2 contains the data for the relative accuracy. The distances between the barrel- and fiducial parts were summed up. The chosen value of the barrel diameter (60 cm) for $d_{\text{max}}$ was in no case exceeded.

Table 7.3 also contains the values for the relative accuracy calculated after the formula from 5.1.3. Global accuracy cannot be calculated because there was no global frame of reference.

### 7.2.3 Resolution Quality Results

The resolution quality cannot be directly measured with the Fiducial approach. Just the indirect fact, that most fiducials are detectable provides evidence, that the resolution quality is at least good enough to detect the barrel halves.
7.2 Fiducials at Disaster City 2010

<table>
<thead>
<tr>
<th>Map</th>
<th>Coverage</th>
<th>Consistency</th>
<th>Relative</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Short</td>
<td>Long</td>
<td>Accuracy</td>
</tr>
<tr>
<td>1</td>
<td>63 %</td>
<td>50 %</td>
<td>33 %</td>
<td>83 %</td>
</tr>
<tr>
<td>2</td>
<td>63 %</td>
<td>50 %</td>
<td>50 %</td>
<td>92 %</td>
</tr>
<tr>
<td>3</td>
<td>94 %</td>
<td>100 %</td>
<td>66 %</td>
<td>93 %</td>
</tr>
<tr>
<td>4</td>
<td>81 %</td>
<td>75 %</td>
<td>33 %</td>
<td>81 %</td>
</tr>
</tbody>
</table>

Table 7.3: Results of some attributes. The coverage, short-range and long-range consistencies as well as the relative accuracy are shown.

7.2.4 Result Discussion

In this section the results of the Fiducial approach as shown in Table 7.3 are discussed. The $1/4^{th}$ barrels are more difficult to map and identify than the other sizes. This could be used to measure the Resolution Quality. But those might be too difficult to automatically detect, which is why they were avoided in later experiments.

The coverage values for Maps 1 and 2 are significantly lower than the other two. But the maze, the area of interest, has been explored in all mapping runs. The reason for this is, that the fiducials do not appear in a good enough quality in the first two maps. So it has to be noted, that the coverage attribute of the Fiducial approach only measures the area covered with good enough map quality.

The consistency values reflect the map quality quite well. The best map in the set, Map 3, achieves the highest score. It can also be seen that the short-range consistency is always at least better than the long-range consistency, which is an expected result. The short-range consistency values for the first two maps again suffer from the unrecognizable fiducials. But Map 2 did not see any of the percent-fiducials. Those were also not counted for the long-range consistency such that the average value for the remaining two groups is better. The broken barrel in Map 4 is a long range one and has, in accordance with the algorithm, lowered the long-range consistency score.

The attributes from Table 7.3 were averaged in the last column, giving each attribute the same weight. The average results reflect the subjective map quality quite well, such that the Fiducial map scoring algorithm seems to be a viable algorithmic metric for mapping algorithms.
7.3 Fiducials at RoboCup Rescue League 2010

In order to verify the fiducial map metric, the evaluation of different maps of different quality is presented here. The results are compared to a ranking provided by human judgment made by the author. The maps were created during the RoboCupResuce World-cup 2010 competition in Singapore and during the RoboCup Rescue Interleague Mapping Challenge 2010 that used sensor data collected in Singapore. Please refer to Section 7.4 for a detailed description of the Mapping Challenge. The maze is depicted in Figure 5.1 and a schematic ground-truth map is shown in Figure 5.2. This ground truth map does not reflect the exact arrangement of the arena. Furthermore, the arena has been reconfigured throughout the competition and the positions of some of the fiducials were also changed. The experiments are from the “Preliminary A” arena. The software used is a second generation of the Jacobs Map Analysis Toolkit [Varsadan et al., 2009].

As fiducials, barrels with a radius of thirty centimeters and a height of one meter are used. They are build by cutting one barrel in half and putting the halves on both sides of a wall, forming a nearly exact circle when viewed from the top. One pair is put on both sides of a void, which could be interpreted as a very thick wall (1.2 meter).

Figure 7.12 shows a map generated using nearly perfect sensor data (the lightweight robot was carried through the maze). The maps created during the rescue competition were created on different robotic platforms using different sensors, navigation techniques (autonomous as well as teleoperated) and mapping algorithms. In contrast, for the interleague challenge maps, sensor data was collected in the maze and later-on fed to the algorithms.

Due to different arena configurations, three groups of maps are distinguished. For the preliminaries the arena was divided into two parts (A and B) to allow simultaneous runs, while later-on one big maze was made for the finals. For part A ten maps were scored while part B has 7 maps. For the whole arena 24 maps were tested, of which 16 were generated during the interleague challenge.

Two local consistencies are defined. Short-range consistency and long-range consistency. For those, the two barrel parts on opposite sides of a wall form one group. Each group was assigned a distance, measured in “pallets” (the 1.2m square area for each test element, see Section 2.6.1). This distance reflects the minimum number of pallets that has to be traversed to get from on part to the other.
Figure 7.12: A very good map created by a human carrying the sensors around. The arena is now configured as one big maze.
other in the group. The long-range consistency is thus the average value for all groups which are more than six pallets away from each other, while the short-range consistency is calculated for fiducials with six or less pallets distance.

The value for $d_{\text{max}}$ is set to 4 times the radius of the barrels for the experiments, while the actual distance value $d$ is discretized to the radius of the barrels (30 cm).

Figure 7.13: Maps 3, 4, 5 and 6 of the “Preliminary A” arena of RoboCup 2010.
7.3 Fiducials at RoboCup Rescue League 2010

Figure 7.14: Maps 7, 8, 9 and 10 of the “Preliminary A” arena of RoboCup 2010.

7.3.1 Results

Only the results for one group of maps (part A), consisting of ten maps, are presented in more detail here, while the experiments performed on the other two groups in general support the results of this one. The maps 1 and 2 (Figure 7.11) are from the RoboCupRescue Interleague Mapping Challenge. Since the sensor data collected there contained all of this part of the maze, a 100% score was possible for this arena. Unfortunately map 1 suffered from a localization error, overwriting parts of the map.

The other maps (Figures 7.13 and 7.14) are from the RoboCupRescue competition. Since no global coordinate system was defined, the global accuracy cannot be calculated. The robots took different paths exploring different amounts of the arena, thus often corresponding barrels on the other side of walls have not been mapped (properly). This leads to low scores for the consistency attributes. Note the big differences in the representation of the map data, although a precise standard was defined. This is the reason why the Image Tree thresholder from Section 4.1 cannot be automated.

Table 7.4 shows the results of the Fiducial Map Metric. To calculate the average, first the two
Table 7.4: Attributes for the maps measured using the Fiducial map metric.

<table>
<thead>
<tr>
<th>Map</th>
<th>Coverage</th>
<th>Relative Accuracy</th>
<th>Consistency Short-range</th>
<th>Consistency Long-range</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>56%</td>
<td>75%</td>
<td>100%</td>
<td>0%</td>
<td>60%</td>
</tr>
<tr>
<td>2</td>
<td>100%</td>
<td>97%</td>
<td>100%</td>
<td>100%</td>
<td>99%</td>
</tr>
<tr>
<td>3</td>
<td>44%</td>
<td>94%</td>
<td>100%</td>
<td>0%</td>
<td>63%</td>
</tr>
<tr>
<td>4</td>
<td>67%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>89%</td>
</tr>
<tr>
<td>5</td>
<td>33%</td>
<td>100%</td>
<td>0%</td>
<td>100%</td>
<td>61%</td>
</tr>
<tr>
<td>6</td>
<td>67%</td>
<td>71%</td>
<td>38%</td>
<td>25%</td>
<td>57%</td>
</tr>
<tr>
<td>7</td>
<td>56%</td>
<td>95%</td>
<td>100%</td>
<td>0%</td>
<td>67%</td>
</tr>
<tr>
<td>8</td>
<td>33%</td>
<td>67%</td>
<td>75%</td>
<td>0%</td>
<td>46%</td>
</tr>
<tr>
<td>9</td>
<td>44%</td>
<td>81%</td>
<td>100%</td>
<td>50%</td>
<td>67%</td>
</tr>
<tr>
<td>10</td>
<td>44%</td>
<td>100%</td>
<td>100%</td>
<td>0%</td>
<td>65%</td>
</tr>
</tbody>
</table>

consistency values are averaged and then this value is used to calculate an average score also using the coverage and the relative accuracy. The coverage value is sometimes surprisingly low (e.g. maps 1 and 9), compared to the area actually visible. This is due to the fact, that the fiducials are not properly visible in the map due to mapping errors. A low coverage value also means that the consistency calculation will often miss the corresponding barrel of a fiducial pair, thus generating a low score. If there is, on the other hand, just one pair with a good consistency, the score will be very good. So for maps with a good coverage it is more challenging to get a good consistency value. This is less of a problem with the bigger maps with more fiducials (or a higher fiducial density).

Figure 7.5 shows the results of the human judgment of the maps, made by the author. This judgment was not done using the fiducials but using the mapped area (for coverage) and locations and consistencies of walls. This judgment is of course quite subjective, but the ranking generated out of it should reflect the actual map attributes quite well.

The results of the fiducial algorithms and the human scoring are compared in Table 7.6. The first rank in each cell is the one gathered with the Fiducial Map Metric (e.g. tie between places 4 and 5 for coverage of map 1), while the number after the slash is the judges rank (e.g. place 2 for coverage of

Table 7.5: Attributes for the maps approximated by a human judge.

<table>
<thead>
<tr>
<th>Map</th>
<th>Coverage</th>
<th>Relative Accuracy</th>
<th>Consistency (Topology)</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>80%</td>
<td>50%</td>
<td>70%</td>
<td>67%</td>
</tr>
<tr>
<td>2</td>
<td>100%</td>
<td>90%</td>
<td>90%</td>
<td>93%</td>
</tr>
<tr>
<td>3</td>
<td>50%</td>
<td>50%</td>
<td>70%</td>
<td>57%</td>
</tr>
<tr>
<td>4</td>
<td>60%</td>
<td>90%</td>
<td>90%</td>
<td>80%</td>
</tr>
<tr>
<td>5</td>
<td>40%</td>
<td>50%</td>
<td>90%</td>
<td>60%</td>
</tr>
<tr>
<td>6</td>
<td>50%</td>
<td>40%</td>
<td>50%</td>
<td>47%</td>
</tr>
<tr>
<td>7</td>
<td>50%</td>
<td>90%</td>
<td>90%</td>
<td>77%</td>
</tr>
<tr>
<td>8</td>
<td>30%</td>
<td>60%</td>
<td>80%</td>
<td>57%</td>
</tr>
<tr>
<td>9</td>
<td>60%</td>
<td>70%</td>
<td>80%</td>
<td>70%</td>
</tr>
<tr>
<td>10</td>
<td>40%</td>
<td>90%</td>
<td>90%</td>
<td>73%</td>
</tr>
</tbody>
</table>
The big differences (two ranks) in the coverage values for the maps 1, 6 and 9 are again due to the fact, that for those maps fiducials are often missing due to mapping inaccuracies. In so far, the fiducial metric is even advantageous to the human judgment, since only the area that has been properly mapped without errors should be counted for the coverage calculation.

The relative accuracy values for maps 5 and 8 differ significantly. This is due to the fact that those maps are really small. The few (3) fiducials for map 5 are actually at the right places while the subjective appearance of the map is not so good. The walls for map 8 overlap quite good with the ground truth, but some of the actual fiducial positions are off.

The consistency values differ to up to three ranks. This is due to the relatively low amount of fiducial pairs (two for each short and long range), which yield to extreme results depending on whether the pair is complete or not. For the maps with greater coverage this is effect is less prominent and the results are thus better.

Nevertheless, the average of the results delivers a fairly descent result. As can be seen in Table 7.6, the actual rankings for the averages correspond quite nicely, having a rank difference of one in four cases, of two in two cases and the same rank for four maps.

<table>
<thead>
<tr>
<th>Average</th>
<th>Map</th>
<th>Coverage</th>
<th>Relative Accuracy</th>
<th>Consistency (Topology)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Human</td>
<td>Fid.</td>
<td>Human</td>
<td>Fid.</td>
<td>Human</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1-4</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td>1-4</td>
<td>1-3</td>
</tr>
<tr>
<td>3</td>
<td>3-4</td>
<td>7</td>
<td>1-4</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>10</td>
<td>1-4</td>
<td>1-3</td>
</tr>
<tr>
<td>5</td>
<td>3-4</td>
<td>9</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>1</td>
<td>7-9</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>5</td>
<td>7-9</td>
<td>1-3</td>
</tr>
<tr>
<td>8-9</td>
<td>6</td>
<td>3</td>
<td>7-9</td>
<td>6</td>
</tr>
<tr>
<td>8-9</td>
<td>10</td>
<td>8</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>9</td>
<td>6</td>
<td>10</td>
<td>9</td>
</tr>
</tbody>
</table>

Table 7.6: Rank Comparison ordered by the average human judgment.
7.4 Matched Topology Graph Evaluation of RoboCup Interleague Mapping Challenge 2010

The first experiments utilizing the Topology Map Evaluation were performed on maps generated during the Interleague Mapping Challenge 2010 [Kleiner et al., 2009, Schwertfeger, 2010]. The challenge was designed to compare the performance of mapping algorithms. All other factors that influence the quality of maps, as mentioned in the Introduction, such as sensors, the environment, the path taken or the processing power, are equal for all contestants, due to the special setup of the competition. This is achieved by the organizers by providing the sensor data and the computation environment.

The sensor data was gathered during the RoboCup Rescue Competition 2010 in Singapore, while the training data was collected at the test facilities at the National Institute of Standards and Technology (NIST) in Gaithersburg, Maryland, earlier that year. The data consists of the readings of a Laser Range Finder (LRF) and an IMU. Some datasets were used twice, with the second set altered in two specific ways. The first change was, in order to simulate a LRF with shorter range, that the range data was capped at 5 meters. Second, the order in which the data was presented to the algorithm could be reversed.

The teams provided their mapping solution to the judges. Quite often those programs rely on a multitude of external libraries and are compiled for different operating systems. Therefore, teams were asked to provide their software together with their OS as a virtual machine image. The judges started the provided systems on their computer. The sensor dataset was uploaded to the virtual machine and then the mapping software would be started. The resulting maps are used in the following experiments to evaluate the matched TG metric.

The sensor data was provided to the mapping algorithm in a defined, binary format provided by the University of Koblenz. The sensors used for the data collection of the Interleague Mapping Challenge are a Hokuyo UTM-30LX laser range finder (LRF) with a field of view of 270°, an angular resolution of 0.25° and a range of slightly above 30m as well as an Xsens MTi gyro and accelerometer. Those were mounted on a stick and connected to a Laptop. The sensor data was collected by a person, holding the stick with the sensors, slowly walking through the maze and the environment.

While the Fiducial Experiments from Section 7.3 only presented two maps of arena A from the Mapping Challenge, in this experiment, all eight maps representing the “mapping competition final”

<table>
<thead>
<tr>
<th>Number</th>
<th>Experiment Maps</th>
<th>Map Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>0</td>
<td>Ground truth</td>
<td>No dataset, but the ground truth map from Figure 7.15.</td>
</tr>
<tr>
<td>1</td>
<td>1 &amp; 5</td>
<td>mappingFinal</td>
<td>The original dataset including LRF (up to 30m range) and IMU data.</td>
</tr>
<tr>
<td>2</td>
<td>2 &amp; 6</td>
<td>mappingFinal-5m</td>
<td>The above dataset with the LRF range capped at 5 meter.</td>
</tr>
<tr>
<td>3</td>
<td>3 &amp; 7</td>
<td>mappingFinal-backwards</td>
<td>The original dataset in reverse replaying order.</td>
</tr>
<tr>
<td>4</td>
<td>4 &amp; 8</td>
<td>mappingFinal-backwards-5m</td>
<td>The above, reversed dataset capped at 5 meter.</td>
</tr>
</tbody>
</table>

Table 7.7: The “mapping final” datasets from the Interleague Mapping Challenge 2010.
7.4 Matched Topology Graph Evaluation of RoboCup Interleague Mapping Challenge 2010

environment are evaluated against the ground truth map, shown in Figure 7.15. The data for the “mapping competition final” environment was collected in the maze configuration of the last day of RoboCup 2010 in Singapore, in which the runs for the autonomy and mapping awards were performed. Table 7.7 lists the datasets used.

Two mapping algorithms, one provided by the University of New South Wales, Australia, and one from the University of Koblenz, Germany were tested. Figures 7.16 and 7.17 show the resulting maps and how they were matched against the ground truth map. No global reference was used, so in the map toolkit their relative position varies. Thus the general direction of the lines representing the matches with the ground truth topology graph, which is not shown, is random (depending on where the team map is positioned relative to the ground truth map). But non parallel lines indicate wrong matches. For Map 1 the brokenness algorithm from Section 6.7 detected a brokenness degree of one with an associated angle of 7.6 degrees. The brokenness groups for this map are shown, but omitted for all other maps, since for those no brokenness was detected.

7.4.1 Results

Table 7.8 shows the results of the experiments. The graph of Map 8 differs too much from the ground truth graph, such that it could not be matched. This is because of blocked or too narrow passages in the map and other mapping errors that can be seen when carefully comparing this map to the ground truth map. The ability to sense such circumstances although the map doesn’t look too bad on first sight is an important aspect of this map evaluation algorithm. Three Local Consistency attributes have been calculated: One for distances between two vertices of a graph of less than 6 meters (short), one for distances between 6 and 18 meters (medium) and one for higher distances (long).

The map attributes have also been combined into a weighted result. The coverage was given the most importance, because it is the only attribute that penalizes missing matches. The other attributes work on the matched vertices and could thus return good values even if half of the map has really bad quality. So for the weighted result, the Relative Accuracy and the Consistencies were averaged together in one value which was in turn averaged with the Coverage attribute.

In Table 7.9, the rankings among the maps that were derived from the different attributes are shown. A ranking made by the author is included as a basis of comparison for the map metric. This human ranking is motivated as follows.

<table>
<thead>
<tr>
<th>Map</th>
<th>Number Vertices</th>
<th>Number Matches</th>
<th>Number Major Matches</th>
<th>Coverage</th>
<th>Relative Accuracy</th>
<th>Consistency Short</th>
<th>Consistency Med.</th>
<th>Consistency Long</th>
<th>Weighted Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>57</td>
<td>36</td>
<td>30</td>
<td>0.68</td>
<td>0.89</td>
<td>0.83</td>
<td>0.81</td>
<td>0.45</td>
<td>0.71</td>
</tr>
<tr>
<td>2</td>
<td>53</td>
<td>29</td>
<td>23</td>
<td>0.52</td>
<td>0.83</td>
<td>0.77</td>
<td>0.58</td>
<td>0.47</td>
<td>0.59</td>
</tr>
<tr>
<td>3</td>
<td>62</td>
<td>32</td>
<td>26</td>
<td>0.59</td>
<td>0.88</td>
<td>0.78</td>
<td>0.59</td>
<td>0.30</td>
<td>0.61</td>
</tr>
<tr>
<td>4</td>
<td>55</td>
<td>34</td>
<td>28</td>
<td>0.64</td>
<td>0.95</td>
<td>0.85</td>
<td>0.80</td>
<td>0.76</td>
<td>0.74</td>
</tr>
<tr>
<td>5</td>
<td>55</td>
<td>39</td>
<td>34</td>
<td>0.77</td>
<td>0.96</td>
<td>0.85</td>
<td>0.81</td>
<td>0.82</td>
<td>0.81</td>
</tr>
<tr>
<td>6</td>
<td>59</td>
<td>43</td>
<td>35</td>
<td>0.80</td>
<td>0.95</td>
<td>0.89</td>
<td>0.77</td>
<td>0.78</td>
<td>0.82</td>
</tr>
<tr>
<td>7</td>
<td>38</td>
<td>20</td>
<td>16</td>
<td>0.36</td>
<td>0.87</td>
<td>0.79</td>
<td>0.73</td>
<td>0.44</td>
<td>0.53</td>
</tr>
<tr>
<td>8</td>
<td>31</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 7.8: The results of the Topology Map Evaluation for the Interleague Mapping Challenge maps.
Figure 7.15: The ground truth map. Slight changes made in the environment between the original map (Figure 7.12) and the actual environment at the time the sensor data was collected have been edited in by hand.
Figure 7.16: Maps 1 (top left), 2 (top right), 3 (bottom left), and 4 (bottom right). The matching with the ground truth graph is shown by lines (green for isomorphism, red for neighbor growing).
Figure 7.17: Maps 5 (top left), 6 (top right), 7 (bottom left), and 8 (bottom right). The matching with the ground truth graph is shown by lines (green for isomorphism, red for neighbor growing).
Maps 6 and 5 are very good and reflect the geometry of the environment better than Map 1. Maps 3 and 4 as well as 7 and 8 have problems in the lower half of the map. The dataset seems to have been quite challenging there. But Maps 3 and 4 are still better than 7 and 8 - the paths to the free standing barrel in the lower right corner is correctly shown, which is not the case for maps 7 and 8. Map 4 is better than Map 3, because of the better represented dead end on the left and also because of the top parts, that are shown more accurately. Map 2 contains severe errors in the right side (e.g. location of the free standing barrel) and is put into the worst class together with Maps 7 and 8.

The rankings from Table 7.9 show that the Topology Map Metric corresponds quite well with the human judgment of the maps. Only the coverage attribute takes the unmatched vertices into account. But also the other attributes reflect the overall quality of the map quite well, since the missing of matched vertices is usually caused by problems of the mapping algorithm which also leads to localization errors of the decently mapped areas.

Brokenness could be detected in the upper part of Map 1. This is a very nice result, because this fact could have been easily missed by a human judge. A video comparing the ground truth map and Map 1 using the NIST Jacobs Map Toolkits available on http://robotics.jacobs-university.de/multimedia/movies/mappingChallengeAlpha.avi. Although maps 2, 7 and 8 contain large errors in the bottom parts, those areas don’t qualify as broken, because they also contain in itself lots of errors.

### 7.4.2 Comparison with the Fiducial Approach

The Interleague Mapping Challenge maps are also applied to the Fiducial Approach from Chapter 5. This Fiducial experiment used the same parameters as the experiment from Section 7.3. The results for the different attributes are presented in Table 7.11. The ranking and its comparison with the human ranking are shown in Table 7.10. Except that there is no medium Consistency in this experiment, this table can be directly compared to Table 7.9 with the results from the Topology Graph approach.

Despite the bad quality of Map 8, nine barrels could be identified there. The short range consistency is then actually not that bad, such that this map outperforms Map 2 in the weighted ranking. Comparing the Fiducial approach with the human ranking and with the Topology Graph approach one can see, that the results are quite similar.

<table>
<thead>
<tr>
<th>Map</th>
<th>Human Ranking</th>
<th>Weighted Ranking</th>
<th>Coverage Ranking</th>
<th>Rel. Acc. Ranking</th>
<th>Consistency Ranking</th>
<th>Ranking</th>
<th>Short</th>
<th>Med.</th>
<th>Long</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1-2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2-3</td>
<td>1-2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1-2</td>
<td>1</td>
<td>1</td>
<td>2-3</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>1-2</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>2-3</td>
<td>2-3</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>6-8</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>6-8</td>
<td>7</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>6-8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 7.9: The rankings of the Topology Map Evaluation for the Interleague Mapping Challenge maps, sorted by the human ranking.
## Map Evaluation Experiments

<table>
<thead>
<tr>
<th>Map</th>
<th>Human Ranking</th>
<th>Weighted Ranking</th>
<th>Coverage Ranking</th>
<th>Rel. Acc. Ranking</th>
<th>Consistency Ranking Short</th>
<th>Consistency Ranking Long</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1-2</td>
<td>1-2</td>
<td>1-2</td>
<td>1-3</td>
<td>1-4</td>
<td>1-2</td>
</tr>
<tr>
<td>6</td>
<td>1-2</td>
<td>1-2</td>
<td>1-2</td>
<td>4</td>
<td>1-4</td>
<td>1-2</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>4-6</td>
<td>3-4</td>
<td>5-7</td>
<td>1-4</td>
<td>5-8</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>3</td>
<td>3-4</td>
<td>1-3</td>
<td>1-4</td>
<td>5-8</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>4-6</td>
<td>5-6</td>
<td>5-7</td>
<td>5-6</td>
<td>5-8</td>
</tr>
<tr>
<td>2</td>
<td>6-8</td>
<td>8</td>
<td>8</td>
<td>5-7</td>
<td>8</td>
<td>3-4</td>
</tr>
<tr>
<td>7</td>
<td>6-8</td>
<td>4-6</td>
<td>5-6</td>
<td>1-3</td>
<td>7</td>
<td>3-4</td>
</tr>
<tr>
<td>8</td>
<td>6-8</td>
<td>7</td>
<td>7</td>
<td>8</td>
<td>5-6</td>
<td>5-8</td>
</tr>
</tbody>
</table>

Table 7.10: The rankings of the Fiducial Approach for the Interleague Mapping Challenge maps, sorted by the human ranking.

<table>
<thead>
<tr>
<th>Map</th>
<th>Coverage</th>
<th>Relative Accuracy</th>
<th>Consistency Short-range</th>
<th>Consistency Long-range</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>81%</td>
<td>79%</td>
<td>94%</td>
<td>0%</td>
<td>69%</td>
</tr>
<tr>
<td>2</td>
<td>38%</td>
<td>81%</td>
<td>33%</td>
<td>25%</td>
<td>49%</td>
</tr>
<tr>
<td>3</td>
<td>75%</td>
<td>84%</td>
<td>75%</td>
<td>0%</td>
<td>66%</td>
</tr>
<tr>
<td>4</td>
<td>88%</td>
<td>96%</td>
<td>100%</td>
<td>0%</td>
<td>78%</td>
</tr>
<tr>
<td>5</td>
<td>100%</td>
<td>99%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>6</td>
<td>100%</td>
<td>89%</td>
<td>97%</td>
<td>91%</td>
<td>94%</td>
</tr>
<tr>
<td>7</td>
<td>69%</td>
<td>94%</td>
<td>50%</td>
<td>25%</td>
<td>67%</td>
</tr>
<tr>
<td>8</td>
<td>56%</td>
<td>67%</td>
<td>75%</td>
<td>0%</td>
<td>53%</td>
</tr>
</tbody>
</table>

Table 7.11: Attributes for the Mapping Challenge maps, measured using the Fiducial approach.
7.5 Topology Evaluation of Broken Maps

In this experiment, the five broken maps from the Brokenness Section (Section 6.7) were evaluated against the unbroken map (see Figure 6.22) using the Topology Map Metric. The results of this experiment can be seen in Table 7.12.

The inherent ranking of the maps (maps with higher numbers being more often broken and thus worse than lower number maps) is nicely reflected in the map attributes that were calculated. That is why no extra ranking table is provided. The coverage of all maps, including the ground truth, is the same. Due to the brokenness, some vertices cannot be matched and thus the coverage is not 100% for the maps. But still the coverage is and remains quite high for the different maps.

One can see that the Relative Accuracy is also not a very strong indicator for the level of brokenness. This is because sometimes the next brokenness is bend towards the “correct” direction, thus potentially even improving the Relative Accuracy.

As expected, the Consistency Attributes are the strongest indicator for the brokenness. The medium and long range consistency values significantly decrease in value with every brokenness that is added to the map. Since the broken parts do not have any other errors, the short range consistency is quite high and only slowly decreasing (due to the errors around the start areas of the broken parts).

The weighted result uses the same formula as in the above section. It also reflects the increasing errors in the maps.

<table>
<thead>
<tr>
<th>Map</th>
<th>Number Vertices</th>
<th>Number Matches</th>
<th>Coverage Matches</th>
<th>Relative Accuracy</th>
<th>Consistency Short</th>
<th>Consistency Med.</th>
<th>Consistency Long</th>
<th>Weighted Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>74</td>
<td>66</td>
<td>0.88</td>
<td>0.71</td>
<td>0.95</td>
<td>0.88</td>
<td>0.41</td>
<td>0.81</td>
</tr>
<tr>
<td>2</td>
<td>76</td>
<td>67</td>
<td>0.89</td>
<td>0.62</td>
<td>0.94</td>
<td>0.84</td>
<td>0.29</td>
<td>0.78</td>
</tr>
<tr>
<td>3</td>
<td>79</td>
<td>58</td>
<td>0.77</td>
<td>0.63</td>
<td>0.87</td>
<td>0.68</td>
<td>0.25</td>
<td>0.69</td>
</tr>
<tr>
<td>4</td>
<td>78</td>
<td>59</td>
<td>0.79</td>
<td>0.63</td>
<td>0.86</td>
<td>0.70</td>
<td>0.17</td>
<td>0.69</td>
</tr>
<tr>
<td>5</td>
<td>79</td>
<td>56</td>
<td>0.75</td>
<td>0.59</td>
<td>0.82</td>
<td>0.53</td>
<td>0.13</td>
<td>0.63</td>
</tr>
</tbody>
</table>

Table 7.12: The results of the Topology Map Evaluation for the broken maps.
Chapter 8

Conclusion

In this thesis, the evaluation of the quality of maps generated by mobile robots was the main subject. Leading towards this topic were Chapters 1 and 2, which gave an introduction to mapping, maps, and map evaluation. This was done by giving an overview of different types of sensors and algorithms used for mapping, of map representations and also the different applications that maps are used in. Factors influencing the quality of maps were identified and different attributes that can be assigned to maps were defined. At the end of Chapter 2, a survey on previous work on map evaluation was assessing the current state of the art in this field.

Chapter 3 presented work done in the area of mosaicking as an example for path-based map evaluation. The rest of this thesis was then dealing with 2D grid maps. Chapter 4 introduced some pre-processing steps that can be applied to the maps. Then two map evaluation approaches were developed: the Fiducial Map Evaluation in Chapter 5 and the Topology Graph approach in Chapter 6. At the end, a number of experiments and field tests using these map evaluation algorithms were presented (Chapter 7).

In the introduction it was explained how the registration of sensor data (finding the spatial displacement of the sensor between two data sets) is a first step in mapping and SLAM (Simultaneous Localization and Mapping). In Chapter 3 it is first shown how this registration can be done for video data by using a spectral algorithm called Fourier Mellin Invariant (FMI). Then mapping and SLAM using FMI are shortly introduced. In Section 3.3 the performance of the photo registration is analyzed by applying the photo mapping to a virtual video with known ground truth movement.

Some processing algorithms for 2D grid maps are then described in Chapter 4. Maps often are not in a state that is suitable for automated processing. They might, for example, contain additional data like robot paths or start positions or probabilistic entries for the occurrence of obstacles. The Image Thresholder is used to colorize the maps to just free and occupied (and maybe unknown) cells. Outlying obstacle points should also be removed from maps to aid the automatic processing. The Nearest Neighbor Removal and an algorithm using Alpha Shapes were developed for that and their performance is compared. The maps can also be simplified by removing unnecessary, occluded cells using the Make Hollow approach. At the end of this chapter, the NIST Jacobs Map Toolkit, which is used for most of the algorithms in this thesis, is introduced.

The first novel map metric, the Fiducial Approach, is described in Chapter 5. After an introduction to the Fiducial Map Metric, the algorithm on how to match the fiducials from the known ground truth positions to those found in the map is explained. Then the calculations for the different map attributes
Conclusion

are done. It is also described how the ground truth data can be acquired.

Map evaluation using Topology Graphs is then developed in Chapter 6. After an introduction to graphs, topology graphs and Voronoi Graphs the algorithm is explained in detail. First, it is shown how the Voronoi Graph is extracted from the 2D grid map and then how the Topology Graph is created and attributed. The calculation of the similarity of two vertices from two graphs is explored in Section 6.4. There, also experiments comparing the different options are performed. This then leads to the description of the matching of two Topology Graphs in Section 6.5, using graph isomorphism and neighbor growing. It is then shown how the matched graphs can be used to calculate the different map attributes, including the map brokenness.

In Chapter 7 a number of field tests and experiments are discussed with the different algorithms.

Results Discussion

The registration step of the photo mapping algorithm (iFMI Registration) was the area of research in Chapter 3. A path-based evaluation was used for a quantitative analysis of the iFMI algorithm. The effects of the resolution of the iFMI matrices and the different options to down-scale the input image (resize vs. cutout) were evaluated. The resize approach scales in input image to the desired resolution while the cutout method uses cropping around the center of the video frame to reduce the resolution. As expected, it was found that the error of the algorithm increases with lower resolutions. It was also shown that, given enough overlap, cutout works better. The minimum overlap between two consecutive frames that results in good registrations for all the frames was found to be 85 %. If this value cannot be ensured, it is suggested to use resize instead of cutout for more reliable results. The presented program can run at more than 25Hz on a modern laptop, including all tasks like data acquisition, image undistortion, live display, and storage. A resolution of 192x192 pixel was used for the image registration - everything else was done with this speed in the original video size (512x512 pixel). The experiments also showed that the claim of the iFMI algorithm to work with very featureless environments is true - differences of performance could not be found between the feature-rich and the sparse feature video streams.

The experiments in Sections 7.1 (MAGIC 2010), 7.2 (Disaster City 2010) and 7.3 (RoboCup 2010) illustrate the performance and limits of the place-based Fiducial Map Evaluation method. The results of this algorithm were compared against a human judgment of the maps. This judgment was not done using the fiducials but using the mapped area (for coverage) and locations and consistencies of walls. This judgment is of course quite subjective, but the ranking generated out of it should reflect the actual map attributes quite well.

Differences in the rankings occurred when the maps were quite small (with very few fiducials) or when the fiducials were not identifiable in the map. Although it might be counterintuitive, this is actually intended because it is argued, that the fiducials are of a size that corresponds to the size of the details that is needed for the specific application the map is used in. If the fiducials cannot be identified in a part of a map, that also means that the details the users of the maps need cannot be recognized, thus this part of the map should not count as covered.

One big advantage of the Fiducial approach is the low amount of ground truth information needed in order to compute a score. Other algorithms need a ground truth path or a ground truth map. In the proposed approach, just the fiducial positions relative to each other (for Global Accuracy in a global reference frame) have to be provided. If only the Local Consistencies are to be scored, in
the proposed wall-barrel-system, the only information needed is an estimate of the thickness of the walls. The metric can be fully automated while still allowing quick quantitative assessments of the map quality by just looking at the image of the map.

The only part of the maps actually evaluated are the fiducials. Thus, as long as the fiducials are detectable in the map, all other mapping errors like noise or broken parts do not effect this algorithm. This metric does not rely on naturally occurring features, although those could be used if they are dense and large enough. This is also the biggest disadvantage of the Fiducial approach, meaning that only environments with such fiducials can be scored.

An approach that does not rely on the detection of features in the map and that uses a comparison with a ground truth map was presented next. The second map evaluation algorithm developed in this thesis is using the structure-based approach of matched Topology Graphs. Besides not needing any modifications of the environment, the fact that no feature detection is needed, but that the topology of the free space is used, enables us to score maps with a low resolution quality (that could be otherwise very good).

The generation of those graphs is not trivial and thus first discussed here. The graphs generated in Section 7.4 (Interleague Mapping Challenge 2010) all nicely represent the topological structure of those maps. We believe that all examples in this thesis show that the generation of the Topology Graphs works very well.

Next, experiments were performed to characterize all the options that were developed for the calculation of a vertex similarity measure. Section 6.4.7 shows the maps and the results for the algorithms. The main result was, that the best algorithms perform very good (over 85 % correct matches purely based on the single similarity value). The Rotated ICP Approach with Disk-Extraction works best, closely followed by the Enhanced Vertex Similarity with one Propagation Round. Since the latter is more than 2000 times faster, it depends on the application and the speed demands which method is preferred.

The next crucial step is the matching of two Topology Graphs (TG). Finding the TG-Isomorphism is explained in detail and also shown in a step by step example. Looking at all the examples and experiments, it can be seen that the matching performs very well. The matches in the brokenness experiments (Section 6.7.2) were all perfect, while there were just a few wrongly matched vertices in the experiments of Section 7.4, which did not effect the results of the map evaluation. Also, the Fiducial approach was applied to the maps form Section 7.4. The comparison between the TG evaluation, the Fiducial metric and a human judgment showed, that all three methods come to very similar results, which is a further reassurance that the developed methods work well.

Using the matched graphs, finally the attributes for the maps can be calculated. One of those attributes, the map brokenness, was described in more detail, since it was not as easy to compute as the others. The experiments in Section 6.7.2 discuss the performance of this metric. It was shown, that the brokenness as defined by [Birk, 2010] could be detected. Only one broken part of the experiment maps that was very small was not recognized. Also, in the RoboCup experiments from Section 6.4.7, a broken part in Map 1 was detected, which could have been easily missed by a human judge. The main advantage of the topology approach over the original image registration approach from [Birk, 2010] is, that because of the abstraction to just the structure of the environment, the topology approach is robust against smaller error sources such as sensor noise and different methods of representing the sensor data in the map. The image registration approach, on the other hand, is vulnerable to such noise, because it uses all the (possibly diverging) information represented in the maps.
In this experiment also the other map attributes that were calculated using the matched Topology Graphs were evaluated. Again, those results were compared to human judgment and it turned out that they corresponded quite well. Even though only the coverage attribute takes the unmatched vertices into account, the other attributes still reflect the overall quality of the map quite well, since the missing of matched vertices is usually caused by problems of the mapping algorithm which also leads to localization errors of the decently mapped areas. It was also argued that the attributes calculated for the maps from the brokenness experiments correlated very well with the expected results.

Scientific Contributions

The contributions to the scientific community by presenting novel work in this thesis can be summarized as follows:

- Propose the extensive use of different weighted map quality attributes for general map scoring.
- A path-based evaluation of the iFMI mapping algorithm.
- Presentation and (partially) evaluation of map pre-processing algorithms.
- Co-development and definition of the Fiducial Map Metric.
- Development of the Topology Graph Map Evaluation Approach:
  - Generation algorithm for an attributed Topology Graph from a Voronoi Diagram.
  - Development and comparison of several Vertex Similarity approaches.
  - Design of two Topology Graph matching algorithms.
  - Algorithms for the computation of different map quality attributes.
  - Algorithm for the computation of the Map Brokenness attribute.
- Experiments and evaluation using the Fiducial and the Topology Graph approach, including the Map Brokenness attribute. Comparison of the results with a map evaluation performed by a human.

Final Conclusions and Future Work

Mapping is an important function for most mobile robots. Thus the development of good metrics that allow the evaluation of the results of the mapping process, the maps, is needed. Using those metrics, the mapping systems can be compared and improved. It was shown that both metrics, the Fiducial Map Metric and the Topology Graph Metric, give reproducible results that correspond to a human judgment. In Chapter 2 it was argued that different robotic application have different demands on the quality of the maps. The attributes defined on the maps allow, that individual scores can be calculated per application by weighing those attributes differently. The Fiducial Metric and the Topology Graph approach both calculate those attributes and give good results, such that it is up to the user to decide which one to use (or even both). This decision is influenced by the factors of how difficult it is to place fiducials in the environment versus how difficult it is to produce an exact ground truth map for the Topology approach.
In the "State of the Art" (Section 2.6.3) the few approaches to map evaluation that have been published are presented. Those either do not work on the maps themselves or just calculate one attribute for the map quality. No map metric has been generally accepted yet. Through the use of the map attributes and the good results that the two algorithms in this thesis show in calculating those attributes, the author believes that the two metrics are very good candidates for general map evaluation.

The two methods have been presented using 2D grid maps. But 3D mapping is a very hot topic in the robotics community. Both algorithms can be adopted to 3D. For the Fiducial Map Metric, a first step already done at RoboCup 2011 and 2012 is to ask the teams for additional horizontal 2D maps (a slice through the 3D map) at the height of the second level fiducials (2m). A more sophisticated approach is to directly identify the Fiducials in the 3D map. Using matched Topology Graphs is also possible in 3D. The map evaluation can then be done in a similar way.

Other work to be done in the future is to extend the Topology Graph to better represent open areas. Those could be detected using an alpha shape approach and added as additional elements (special type of vertex) to the graph. This would help with the stability of the topological representation and also improve the matching process. Matched Topology Graphs can also be used in other applications. Using those graphs one can, for example, merge maps quite easily, even if they overlap just a little bit and/or contain broken areas.
Appendix A

Configuration Parameters for the Topology Graphs

A number of configuration parameters are used during the creation and matching of the Topology Graphs (see Chapter 6). Those are shortly discussed here.

First the parameters that influence the creation of the Topology Graphs are presented. Those variables are used in Section 6.2.

The parameters are presented as follows: First the variable (e.g. $\delta_{\text{min2Obst}}$) used in this thesis is given, followed by the corresponding name of this variable in the configuration file (e.g. voronoiMinimumDistanceToObstacle). Then follows a short discussion of the variable.

- $\delta_{\text{min2Obst}}$ voronoiMinimumDistanceToObstacle In pixel. The value determines the distance the Topology Graph keeps from obstacles. Its value depends on the resolution of the map and on the environment represented in the map.

- $\alpha_{\text{shape}}$ alphaShapeRemovalSquaredSize In pixel. The value is used to cut the parts of the Voronoi Diagram that are outside of the map. Before squaring it, the value should be a little bigger than the largest opening of the maps.

- $\delta_{\text{deadEnd1}}$ firstDeadEndRemovalDistance In pixel. A value of 0 removes no dead ends, very high numbers remove all dead ends. Typically the first dead end removal is set to remove all dead ends.

- $\delta_{\text{deadEnd2}}$ secondDeadEndRemovalDistance In pixel. Depending on how open the area is and the ratio between the average space and the value chosen for $\delta_{\text{min2Obst}}$, subsequent dead end removals might be used to get a cleaner graph with less Spurious edges.

- $\delta_{\text{deadEnd3}}$ thirdDeadEndRemovalDistance As above.

- $\delta_{\text{deadEnd4}}$ fourthDeadEndRemovalDistance As above.

- $\delta_{\text{joinVertices}}$ topoGraphDistanceToJoinVertices In pixel. The distance that determines if two neighboring vertices are merged into one.

- $\delta_{\text{spuriousEdge}}$ topoGraphMarkAsSpuriousEdgeLength In pixel. The threshold that determines if a dead end edge is marked as being Spurious.
Configuration Parameters for the Topology Graphs

- $\delta_{angleCalcStart}$ `topoGraphAngleCalcStartDistance` In pixel. Used together with $\delta_{angleCalcEnd}$ to determine the part of the geometrical path of an edge (always close to a vertex) that is used to determine the angle of said edge towards the vertex.

- $\delta_{angleCalcEnd}$ `topoGraphAngleCalcEndDistance` See above.

- $\delta_{angleCalcStep}$ `topoGraphAngleCalcStepSize` The step size that is used to sample points from the interval described above in $\delta_{angleCalcStart}$.

The following variables are used in the calculation of the Local Vertex Similarity (see Section 6.4.2). The offset and the maximum values are used together in a formula to determine a similarity value between 0 and 1 (0 means a perfect similarity, 1 is very bad).

- $\sigma_{existsOff}$ `simiVerNumberOfExitsOffset` (default 0) When comparing to vertices, these values are used for the similarity value that reflects the similarity of the number of edges connected (the degree of that vertex).

- $\sigma_{existsMax}$ `simiVerNumberOfExitsMax` (default 1)

- $\sigma_{angleOff}$ `simiVerExitAngleRadOffset` (default 1 degree) The biggest and the smallest angle between edges of a vertex are used as descriptors. When comparing these descriptors between two vertices, this configuration value is used to compute the similarity value. This is also used for the Enhanced Vertex Similarity. In radians.

- $\sigma_{angleMax}$ `simiVerExitAngleRadMax` (default 12 degrees)

- $\sigma_{obstacleOff}$ `simiVerDistanceToObstaclesOffset` (default 0.5 pixel) The average distance to the obstacle is a descriptor for edges and vertices. When comparing those descriptors, this configuration value is used to compute the according similarity value.

- $\sigma_{obstacleMax}$ `simiVerDistanceToObstaclesMax` (default 2 pixel)

- $\sigma_{joinedOff}$ `simiVerDistanceBetweenJoinedVerticesOffset` (default 0.5 pixel) For vertices that were created by joining other vertices together, the average distance to the joined vertices is a descriptor. This configuration value is used to compute the according similarity value.

- $\sigma_{joinedMax}$ `simiVerDistanceBetweenJoinedVerticesMax` (default 2 pixel)

During the matching (Section 6.5) and the evaluation (Section 6.6) these configuration values are used:

- $\sigma_{isoMinVert}$ `simiIsomorphismMinNumberOfVertices` During the isomorphism algorithm, this value is the minimum number of vertices that have to be matched for that a matched subgraph is accepted and kept as a result. Typical values are between five and eight.

- $\sigma_{isoEdgeDistMinAllow}$ `simiConsistencyEdgeDistanceMinAllowance` During the isomorphism algorithm edges are only matched if their distance is compatible. This and the following value are used to determine if two edge length are still compatible.

- $\sigma_{isoEdgeDistFactor}$ `simiConsistencyEdgeDistanceFactor`
• $\epsilon_{\text{brokenThres}}$ **SquaredErrorThreshold** During the calculation of the brokenness groups this is the threshold for the mean squared error.

• $\epsilon_{\text{brokenMinV}}$ **MinNumberVertices** During the calculation of the brokenness groups the minimum number of vertices found in a group such that this group is accepted.
Configuration Parameters for the Topology Graphs
Appendix B

Maps for Vertex Similarity Experiments

Those are the maps used in the Vertex Similarity experiments from Section 6.4.7.

Figure B.1: Vertex Similarity experiment Maps A and B

Figure B.2: Vertex Similarity experiment Maps C and D
Maps for Vertex Similarity Experiments

Figure B.3: Vertex Similarity experiment Maps E and F

Figure B.4: Vertex Similarity experiment Maps G and H
B.1 Example Vertex Mapping from Map A to H

This is one example mapping for the vertices from Map A of the Vertex Similarity experiments to Map H (for maps see Appendix B, for the experiments see Section 6.4.7).

1, 20
2, 23
3, 19
4, 22
5, 21
6, 15
7, 18
8, 12
9, 18
10, 13
11, 11
12, 4
13, 10
14, 2
15, 9
16, 8
17, 1

B.2 Map Combinations used

Those are the combinations of maps used in the Vertex Similarity experiments from Section 6.4.7. The no rotation combinations refer to maps which have the same global orientation while the just rotation pairs have differing global orientations.

- No rotation (10): AB, AC, AD, AE, BC, BD, BE, CD, CE, DE
- Just Rotation (23): AF, AG, AH, AI, BH, BI, CF, CG, CH, CI, DF, DG, DH, DI, EF, EG, EH, EI, FH, FI, GH, GI, HI
Appendix C

NIST/ Jacobs Map Analysis Toolkit

The NIST Jacobs Map Toolkit is a plugin-based program to show, compare, manipulate and score maps. The first version of this program was developed at the Jacobs Robotics Group [Varsadan et al., 2009]. During the work on this thesis and an eight month research visit at NIST, the author re-implemented and extended the software presented here (because of the new 3D and plugin architecture), re-using some of the old code. This program is used for the different map evaluation experiments - from Disaster City, over RoboCup to MAGIC (see Section 7.1). It supports the Fiducial Map Metric described in Section 5.1 as well as the Topology Map Evaluation from Section 6.6 and the algorithms described in this Chapter.

Since it is difficult and time-consuming for one group to implement and maintain code for the different input and output formats and to keep track of the improvements for the different algorithms applied to the map, the toolkit is using a plug-in system. This allows external authors of such code to independently maintain their contributions. It also allows to provide closed-source algorithms if that is wished.

The software is:

- open source: allow contributions from other groups
- allow closed-source contributions: advanced algorithms with IP restrictions should nevertheless be available for the community
- extendability: The sources should be well documented and understandable such that writing extensions or plug-ins should be easy
- multi platform: Linux, Mac and Windows (untested)

C.1 Implementation

The implementation uses a 3D library throughout the program, i.e. also the 2D data is displayed using the 3D environment. The 3D library of choice is OGRE [OGRE, 2010]. The user interface is written using Qt 4.5. All data (viewed or modified maps/ vector data/ 3D data/ etc.) is maintained in a modifiable (order, visibility, alpha-value) tree structure using the model/ view approach. The proprietary data (fiducials, sessions, etc.) is saved in XML using the Qt DOM classes.
A simple plug-in system loads plug-ins during run-time. Different plugins may depend on each other (no circular dependencies are allowed). These plug-ins hold most of the features of the project. The core software just provides the following:

- management of plug-ins
- initialization and management of the 3D OGRE structure
- management of different displayed maps/items in a tree with alpha-blending-capabilities
- most of the user interface (only some renderers/iodrivers/geoObjects have additional GUI elements)
- interfaces for common data like:
  - GeoreferencedObject
  - Georeferenced-VectorData
  - IODrivers
  - Renderers
  - ScoringAlgorithms

Figure C.1: A screenshot of the GUI of the NIST Jacobs Map Toolkit.
C.2 Usage

- Alpha Animation
- Create Videos
- Saving/ Opening of sessions (groups of MapLayers)

The following lists contain plugins which are currently implemented.

- **IODrivers**
  - loading/ saving GeoTIFF
  - loading huge GeoTIFF files using fast pyramid access
  - loading/ saving images + export as GeoTIFF
  - loading/ saving fiducials ground truth
  - loading/ saving fiducials marked in a map

- **Map processing plugins**
  - Image Thresholder (Section 4.1)
  - Make Hollow (Section 4.5)
  - Nearest Neighbor Removal (Section 4.2)
  - Alpha Shape Removal (Section 4.3)

- **scoring algorithms for 2D maps**
  - the Fiducial Map Metric (Section 5.1)
  - Topology Map Metric (Section 6.6)
    - Generation of Topology Graphs (Section 6.3)
    - Vertex Similarities (including ICP and iFMI) (Section 6.4)
  - a plug-in that uses other scoring algorithms to compute a over-all score using weights for the different map quality attributes and map evaluation algorithms

**C.2 Usage**

The program starts by re-opening the last used session. This session is saved on exit as `default.session`. All data not saved in this XML file is always saved in a folder with the base name (default in this case) plus “.data”. On exit, before creating the new `default.session`, a backup called `default.bck.session` will be made of the old `default.session` (the old backup will be deleted).

The program window, as seen in Figure C.1, consists of the map area, a tree view of the different map layers and other windows elements like tool bars, menus and status lines. There are different interaction modes with the map area, selectable in the “Map View” tool bar. With the first the user can switch between 2D and 3D mode. The 3D mode is still experimental and will not further be described here. Interaction with the map windows can be either in view mode where one can pan, rotate or scale the view on the entire scene. In the “Geo Reference Edit” mode one can edit the geo reference for a map layer if “Edit Georeference” is chosen from its context menu before. A widget
allows manual input of global values or values relative to the parent map layer and has some other nice options (locking of certain values, undo and identity with the parent or the global frame of reference). It is also possible to change the position, orientation and scaling using the mouse and the map widget.

The third mode for interaction with the map widget is the “Data Edit” mode which is used by the renderer of the specific data type. For example, with the fiducial renderer one can add, edit and remove fiducials and change fiducial options here.

The tree view displays the different map layers. The order in which they appear there determines the order they are alpha blended in the 2D view. Using alpha blending one can make one map layer partially transparent, such that the map is easy to compare with the one from an underlying map layer (if they are in the same global reference frame). Figure C.1 shows two maps overlayed over each other. Also videos from the map display and the alpha animations can be produced. One such video is available on http://robotics.jacobs-university.de/multimedia/movies/mappingChallengeAlpha.avi. It shows the ground truth map and Map 1 the experiment from Section 7.4.

Additionally, the tree represents the geo reference relations. If the geo reference of one parent is changed, all its children change their (global) geo reference accordingly. The map layers can be dragged and dropped to other map layers (their global geo reference will stay the same). To add a dropped mapLayer as a child hold Ctrl while dropping. A child’s global alpha value is relative to the alpha values of its parents unless “Absolute Alpha Value” is checked in the context menu.

Each map layer has a geo object containing the data that is displayed (except the group layers). Geo Objects are loaded and saved using io drivers. Some can be newly created. Different io drivers may load the same GeoObjects (like image and gdal (geotiff) both load imageGeoObjects) and one GeoObject might be exported using different io drivers (like export imageGeoObjects as images or gdal geotiffs). There might be different renderer for the same GeoObject - select with render with.

Groups (part of a session) are saved/ loaded using the XML standard. External data is saved in the name_data directory created. It is currently not possible to overwrite old sessions. Renderer or Data (e.g. FiducialGeoObject) might provide the possibility to edit options using the context menu. It is also possible to disable (the display of) certain map layers.
References


190


192


193


197


198


199


200


201


202