ManifoldSLAM: a Multi-Agent Simultaneous Localization and Mapping System for the RoboCup Rescue Virtual Robots Competition

MSc Thesis (*Afstudeerscriptie*)

written by

Bayu Slamet
(born 17. Feb. 1982 in Gouda, The Netherlands)

Max Pfingsthorn
(born 30. Oct. 1981 in Hannover, Germany)

under the supervision of **Nikos Vlassis** and **Arnoud Visser**, and submitted to the Board of Examiners in partial fulfillment of the requirements for the degree of

Master in Artificial Intelligence

at the *Universiteit van Amsterdam.*

Date of the public defense:
December 11, 2006

Members of the Thesis Committee:
Nikos Vlassis
Arnoud Visser
Leo Dorst
Wouter Jansweijer
Zoran Zivkovic

FACULTEIT DER NATUURWETENSCHAPPEN, WISKUNDE EN INFORMATICA

Abstract

This thesis presents *ManifoldSLAM*, an award-winning multi-robot system that provides simultaneous localization, mapping and exploration functionality. It enables a team of robots to be placed in an unknown environment, to explore it autonomously, and afterwards to produce a detailed map of the explored areas.

The design of our system focuses on the sophisticated Manifold data structure that was published by Howard et al. [27]. The Manifold is a layered data structure that employs a graph organization which decomposes the global map into small-scale local metric maps. This classifies *ManifoldSLAM* as a hybrid SLAM approach that attempts to merge the individual strengths of metrical and topological representations similar to approaches proposed by Bosse et al. [7] and Lisien et al. [31].

While Howard et al. [27] refer to the IDC scan matcher by Lu and Milios [34], in *ManifoldSLAM* we base the SLAM-related functionality on the Weighted Scan Matcher (WSM) published by Pfister et al. [44]. The superior performance of WSM in our domain is demonstrated in an extensive set of experiments specific to our setting that also included MbICP by Minguez et al. [36] and the Normal Distribution Transform by Biber and Straßer [5]. The high speed and accuracy of WSM in our domain enables a light-weight implementation of the parts of the loop-closing and island-merging processes that are executed online. This significantly improves our system's online performance, which finally allowed *ManifoldSLAM* to demonstrate a scalability up to at least 8 robots at the RoboCup Rescue of 2006.

Using *ManifoldSLAM* we have successfully competed in the Virtual Robots league of RoboCup Rescue [9, 10, 1]. During the 2006 RoboCup World Championships we have acquired third place. The accuracy of our maps, the good exploration exposed by our robot team, and the fully autonomous and robust behavior control were key to our achievements. Moreover, the maps that we produced preserved an amount of detail that was unmatched by other competitors in the league. Therefore, we also won the *Best Mapping Award*.

In additional experiments we also illustrate that *ManifoldSLAM* can be applied on real-world data as our system has been demonstrated to deliver equally accurate and detailed maps from raw laser range data that suffers from real-world odometric error and sensor noise.

This thesis contributes a hybrid SLAM approach that advances the current state of the art. We show that by combining the Manifold concept with WSM an efficient multi-robot SLAM system can be implemented, which has been proven successful at RoboCup Rescue.

Acknowledgements

First of all we thank our supervisors Arnoud Visser and Nikos Vlassis, they have been indispensable for our work. We want to thank Nikos for supporting us on this great project and for steering us in the right directions at critical moments. Arnoud has been of tremendous value especially during the RoboCup competitions. His experience, opinions and detailed advice have been vital to our achievements there.

Also we would like to thank Sam Pfister and Stergios Roumeliotis from the California Institute of Technology, Javier Minguez from the University of Zaragoza and Peter Biber from the University of Tübingen for providing us with sample implementations of their scan matching algorithms and their additional support.

The two persons that have most helped us to put the best of ourselves in this project are undoubtedly our lovely wife and girlfriend, Joanna and Marloes. Their continuous patience, endurance and support were implicit and came without asking. We are extremely grateful for that.

On a final note, we would like to express that this graduation project has been a highly enjoyable experience. It has truly been a team effort. Our seamless cooperation has made this project a great pleasure to both of us and was a constant factor throughout this adventure. Especially under pressure our teamwork excelled which has been the key to our success.

This research was partly funded by the AUV Alumni Fonds of the University of Amsterdam.

Table of Contents

Chapter 1

Introduction

Urban Search and Rescue (USAR) involves the location, extrication, and initial medical stabilization of victims trapped in confined spaces. Structural collapse is most often the cause of victims being trapped, but victims may also be trapped in transportation accidents, mines or collapsed trenches. Human USAR teams do their life-saving work in emergency situations throughout the world. Man-made disasters like the London terrorist attacks or natural disasters as the hurricane Katrina that hit New Orleans and the Asian Tsunami are just some recent examples.

Traditionally, the main actors in an emergency response scenario have been human first responders. However, the current realization is that in a number of situations intelligent robots are a better fit for the job at hand or can provide additional support in a cooperative fashion. Robots can have a physical advantage because their size or specific design enables them to get to otherwise unreachable places, or because they can better sustain conditions like fire, lack of oxygen, radioactivity or toxic chemicals. Also, robots can have a sensory advantage as they can be equipped with sensors and appropriate software that give them capabilities that surpass the human senses. Perhaps the most important aspect of robots in the USAR context is that although they may be expensive, in essence they are replaceable, whereas the lives of a human rescuers and victims are not. Thus, a robot could already start exploring an emergency site ahead of the first responders while the site is still considered unsafe for entrance by human rescuers, e.g. due to a release of toxic gases or radioactive radiation.

The RoboCup Rescue project was initiated after the Great Hanshi-Awaji earthquake hit Kobe City (Japan) on the 17th of January 1995 and caused more than 6500 casualties. As of 2002, the RoboCup Rescue Competitions are held as part of the annual RoboCup World Championships. It is the purpose of RoboCup Rescue to promote research and development in this socially significant domain in order to ultimately acquire solutions that can be used by USAR teams under real emergency circumstances.

In June 2006, the RoboCup World Championships were held in Bremen, Germany. This year was the first time that RoboCup Rescue held competitions in the new Virtual Robots League. In this new league, participants compete using virtual robots that are reproduced in a high-fidelity simulator. This relieves the teams from the realities of physical robots like the high entry costs and hardware details and enables them to focus on capable software algorithms such as multi-robot collaboration, localization, mapping, coordinated exploration and autonomous behavior control.

In this thesis we present *ManifoldSLAM*, a software system that was built from scratch and with which we have successfully participated in the RoboCup Rescue Virtual Robots League. *ManifoldSLAM* is a simultaneous localization and mapping (SLAM) system that enables a team of robots to be placed in an unknown environment, to explore and to find victims in this environment fully autonomously, and later to deliver a detailed map of the environment which reports and visualizes the findings of the robot team.

1.1 Motivation

Our research interest lies in several topics that have been at the center of probabilistic robotics research for the last years. Specifically, we are very much interested in simultaneous localization and mapping (SLAM), multi-agent exploration and autonomous behavior control.

These specific topics are at the core of modern robotics systems. Any robot or team of robots which is supposed to achieve any reasonably complex task in the real world needs to incorporate all of them in its control software. Maps are needed to plan on a global scale. Localization is needed to monitor progress and to follow plans. Exploration aims to extend the map and thus to generate even better plans. Planning in general, be it purely reactive or with more consideration, is addressed in the autonomous behavior. Multiple robots working in a team will generally be more robust to unexpected challenges posed by the environment or diverse errors inherent in either hardware or the previously mentioned software mechanisms.

The main efforts in current probabilistic robotics research are directed at least at one of these aspects. We aim to combine these in a comprehensive robotics system and show how they can efficiently act together.

1.2 Research Goals

We set out to design and implement a complete robotics system that supports multi-agent localization, mapping and exploration using fully autonomous behavior control.

Specific to the setting in the Virtual Robots League, a team of virtual robots should be able to be deployed in the simulator. The team should be able to autonomously explore and to find victims in the simulated emergency site. Finally, the system should be able to produce an accurate and detailed map of the environment as well as the other required deliverables listed in the competition rules. Naturally, it was our primary intent to excel in this league using our software.

In a broader sense we intended that our system would be able to produce accurate and highly detailed maps, both under simulated conditions as well as from sensor data that was acquired with real sensors that would be mounted on real robots and therefore suffer from real-world odometric error and sensor noise.

To meet these goals, the system of our design had to be based on the current state of the art in relevant areas of research and should incorporate the best available solutions published to date. We identified the following areas that fall under the umbrella of probabilistic robotics research and pertain to our goals: simultaneous localization and mapping (SLAM), and scan matching.

1.3 Thesis Organization

In Chapter 2, we first outline the challenge we faced in the RoboCup Rescue Virtual Robots League in further detail. The two subsequent chapters review the current state of the art in the relevant areas of research: Chapter 3 discusses SLAM and Chapter 4 deals with scan matching. Both chapters give an in-depth review of the general approaches that have been applied to date to address the specific challenges and also highlight details of several current implementations.

Subsequently, we use this knowledge to design *ManifoldSLAM* in Chapter 5, where we try to combine the strengths of several individual SLAM approaches in a hybrid data structure and use the best available scan matcher we found for our domain. The implementation details of our software are discussed in Chapter 6. Then in Chapter 7, we present the results of our extensive scan matcher experiments, the details of our achievements at the RoboCup and we show results obtained from real-world data-sets.

In Chapter 8, we relate our approach to the current state of the art and to the systems that were employed by our competitors at the RoboCup. We end in Chapter 9 with several concluding remarks and directions for future research.

1.4 Authorship Designations

As part of the fulfillment of the requirements for the Master of Artificial Intelligence degree, the authors of the specific chapters and sections have to be stated. The following list describes the individual writing done by the two authors.

Max Pfingsthorn wrote the following chapters and sections: Ch. 2 (except Section 2.3), Section 3.4, Ch. 4, Ch. 6 (except Section 6.6), Section 7.1.

Bayu Slamet wrote the following chapters and sections: Ch. 1, Section 2.3, Ch. 3 (except 3.4), Ch. 5, Section 6.6, Section 7.2, Ch. 8, Ch. 9.

Chapter 2

The Challenge: RoboCup Rescue

2.1 RoboCup

There are many projects in the world that promote research in robotics and AI, one of them is RoboCup. Annual world championships are held to let researchers from around the world compare their implementations in many different leagues and disciplines. RoboCup consists of multiple soccer leagues with real robots and with simulated players. RoboCup Rescue, as further detailed in the following section, deals with the more significant Urban Search and Rescue setting. Also, RoboCup incorporates leagues for kids with dancing robots as well as soccer playing robots.

Many different areas of robotics research have to be incorporated to successfully participate in RoboCup competitions and eventually achieve this mission. Such areas include mechanical design, sensor fusion, online decision making, strategic planning, and multi-agent coordination and collaboration.

The annual international competitions facilitate research and development tremendously as it is not only an institution for international comparison, but also for the exchange of ideas, approaches, and implementations. Additionally, regional competitions are held during the year in various parts of the world. Each RoboCup event is usually accompanied by a larger technology fair for robotics and related research institutes and companies. This shows how important RoboCup has become as a research facility and proving ground for new robotics technology.

2.1.1 Rescue

Our specific field of interest is in a recent RoboCup discipline called Rescue, which deals with the topic of Urban Search and Rescue. Inspired by the devastation caused in the 1995 earthquake in Kobe City, Japan, RoboCup includes this new discipline in both international and national competitions. It was first incorporated in the RoboCup World Championships in 2002. The discipline consists of a league which employs real robots to explore, map and search buildings and other places after disasters, such as earthquakes or gas accidents. Also, two leagues using simulations exist, one which concerns coordinating emergency workers (police, firefighters, etc.) during a city-wide disaster. The other, which we participated in, simulates the real robot league in software.

The American National Institute of Standards (NIST) Autonomous Systems Devision collaborates closely with the RoboCup Rescue effort. In order to representatively recreate disaster areas for the real robot league, NIST has designed and developed the rules and guidelines for RoboCup Rescue. Their main effort is directed at producing usable solutions for real human rescue workers.

The NIST has designed three different stages of destruction representative for gas leaks or chemical accidents, gas explosions, and earthquakes or collapsed buildings respectively. They are called *Yellow*, *Orange*, and *Red Arenas*.

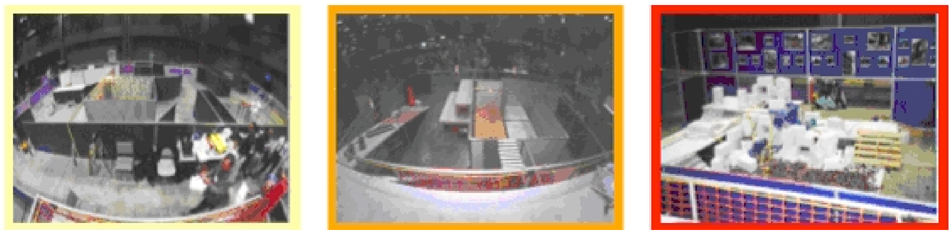


Figure 2.1: Real Rescue Arenas, from [9]

The *Yellow Arena* contains a relatively simple single level office layout which had been minimally damaged. Since it is supposed to represent a gas leak or chemical accident, no structural damage is included. The only damage that is represented would be caused when people collapse due to the nature of the accident. This includes overturned chairs, cupboards, and small tables. Paper is present on the floors, possibly after falling off the tables.

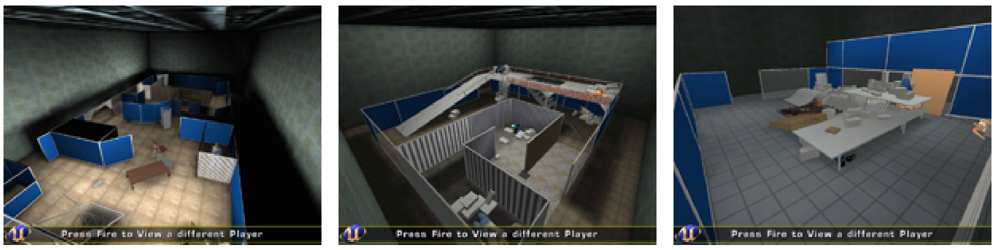
The *Orange Arena* consists of a simple two-story layout and represents gas explosions or similar accidents. Usual damage from such accidents includes relatively small structural damage, but a lot of mess in terms of litter on the floor, broken glass, overturned beds, etc. This kind of rubble makes it difficult to navigate the environment. Another challenge in this arena is getting to the second level. A ramp as well as stairs exist for this purpose.

The final and most difficult *Red Arena* represents the effects of an earthquake. It displays heavy structural damage and a very unstructured environment as a result. This arena is the most difficult as navigating the rubble is very hard and victims, which might be trapped underneath fallen floors, are very hard to detect.

2.1.2 Rescue Virtual Robots

The Virtual Robots league, as briefly introduced before, involves a detailed high-fidelity simulation of the real robot rescue league [9]. In a virtual environment, a single or multiple robots move, sense and act as they would in a real environment. This particular league eliminates the need for physical robot hardware, and therefore takes any related issues out of the development loop. Implementing working solutions needs significantly less time in this hardware-less setting. Therefore, the main focus of this league lies in the autonomous aspect of the robot software.

Another difference is that the simulated areas can be much larger and more diverse than the rather small test arenas in the real robot rescue league. While separate arenas are rarely larger than 10 by 10 meters in the real league, the areas used in this years Virtual Robots competition were close to 100 by 100 meters large. The environment covered wide outdoor areas with streets, sidewalks, and a park as well as extensive, multi-level indoor areas, which featured a very detailed office building. Additionally, due to the larger environment and negligible hardware cost, larger teams of robots can be deployed in the virtual world.


Press Fire to View a different Player
Press Fire to View a different Player
Press Fire to View a different Player

(a) Yellow arena. (b) Orange arena. (c) Red arena.

Figure 2.2: Three test arenas.

The underlying simulator is called USARSim [9, 10, 1, 62, 63, 64, 65] and is based on the game Unreal Tournament 2004^{TM}. This software is discussed in detail in Section 2.2. Programs communicate directly with the simulator server, and each robot has corresponding simulated hardware in the simulator.

Since the simulator provides many of the features that a real system needs in the first place to operate at all, all participants focus on robot control. Also, only a small range of robot designs are implemented in the simulator. This means that there are no hardware differences between competitors. The control of the robots is the only distinguishing factor.

According to the slightly different nature of the Virtual Robots league to the real robot league, the original RoboCup Rescue rules have to be adjusted slightly as well. This scoring function uses several performance measurements, which are acquired in one trial run in the competition. Below, the scoring function that was used during the competitions in 2006 is shown:

$$Score = \frac{VictimScore + ExplorationScore + MapScore}{(1 + Operators)^2}$$

With:

$$VictimScore = \sum ScorePerVictim - \sum BumpPenalty$$

$$MapScore = MetricMapQuality * TopologicalMapQuality$$

The separate factors are computed as follows:

- $\uparrow$ *ScorePerVictim* is computed by rewarding 20 points for every victim that was reported with a status (e.g. “screaming” or “cut on forehead”). For every detected victim with no status report, 10 points are rewarded. Each piece of additional information, such as pictures of the victims, earns a 10 points bonus. An addition was decided on during the competition: Additional 10 points are awarded for locating the victim with an accuracy of up to one meter.
- $\uparrow$ *ExplorationScore* is a value between 0 and 50 based on the total area explored by all deployed robots.
- $\uparrow$ *MetricMapQuality* is a multiplier between 0 and 1 based on the positioning accuracy of special features added to the virtual environment for this purpose.
- $\uparrow$ *TopologicalMapQuality* is assessed manually by a jury based on how accurate topological features such as walls, corridors, etc. are represented in the produced map. The final score is set between 0 and 50 points.
- $\downarrow$ *BumpPenalty*, a penalty of 5 points is deducted for every robot that bumped into a victim. A maximum of one penalty for every robot-victim pair can be applied.
- $\downarrow$ *Operators* is the number of people that actively participate in the creation of material used for scoring, such as map merging or guidance of the robots.

Already a brief examination of the scoring function clearly shows that autonomous approaches are strongly encouraged. Even a total of only one operator introduces a denominator of four.

In addition to this function, the league organization provided several test arenas prior to the competition, depicted in Figure 2.2. One of these, the *Orange Arena*, was a multi-floor building with two different levels connected by stairs and a ramp. The test arenas were supposed to be representative of what the robots would encounter during the competition as far as the obstacles, victim models and physics were concerned. However, the teams were advised to be prepared for much more challenging arenas during the competition caused by drastic changes in size and layout.

Two simulated sensors deserve a special introduction at this point. The *RFID Sensor* can detect the special landmarks used for computing the *MetricMapQuality*. These are called *single-shot RFID Tags* as they can only be observed once per robot. Also, they are detected without noise so their localization accuracy only depends on the accuracy of the map built by the robot. The *Victim Sensor* allows to identify victims in the simulated world without having to do image processing of simulated camera feeds. It works similarly to the *RFID Sensor*, however, every victim can be noisily detected multiple times. At the end of each trial run, a list of victims with additional information as well as a list of detected single-shot RFID Tags had to be reported for use in the scoring function above.

2.2 The USARSim Simulator

2.2.1 Overview

USARSim [9, 10, 1, 62, 63, 64, 65] is the simulator used in the RoboCup Rescue Virtual Robots league. It is implemented as a modification to the popular game Unreal Tournament 2004^{TM} (UT). The complete simulator uses UT's scripting framework and no extra software.

Unreal Tournament was chosen as a base platform because it already provides some key features needed for a high-fidelity simulator: A complete physics engine used to simulate vehicle dynamics, a high-performance graphics engine to visualize large worlds and to simulate cameras, a simple access to the logical representation of the world in order to implement sensors easily, and an efficient synchronization protocol to facilitate parallel rendering of the simulated world. All additional needed functionality is implemented within the scripting framework.

The simulator specific additions include a kinetic model for a differential drive robot. All implemented platforms are derived from this model. The forward kinematics of the generic model have been tested thoroughly, and also allow for dynamic physics such as falling down stairs and platforms.

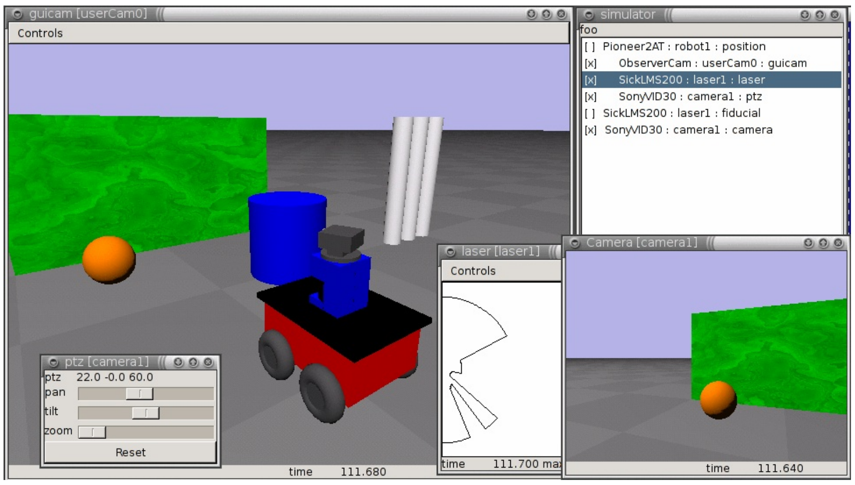
All interactions with the simulator, with one exception, are achieved via a text-based protocol. This makes it very easy to connect to and communicate with the simulator server. The protocol is very structured, so it is easy to write a general parser which can read any message sent by the server. Commands sent to the server correspond to the same structure. The protocol is explained in Section 6.2.

The simulator, while very extensive, does not include every aspect. Specifically, there are no provisions for robot-to-robot communication. This still has to happen outside the simulator via conventional network connections. However, plans exist to include simulated wireless communication with proper signal degradation and similar error sources in the simulator.

Further technical details of the USARSim implementation are discussed in Section 6.2.

2.2.2 Related Projects

There are very few comparable projects currently available. One major open source robotics simulator which is widely used is *Player/Stage*. Actually, the *Player* component of the software is a device server which either runs on a real robot or in a simulator (the *Stage* component). Programs using the architecture connect to the *Player* server to receive sensor data and send commands to actuators via a standard binary protocol. The protocol is implemented with XDR, or eXternal Data Representation, which is a very common representation for serializing typed data for network transmission under Unix. *Stage* is a high-performance, 2D simulator, which is frequently used for large experiments with many robots. It has a high-fidelity, 3D counterpart called *Gazebo*, which also simulates physics and more complex environments. In general, *Gazebo* would be used to create a high-quality simulation of very few robots. A depiction of a sample interface and a running *Gazebo* instance is shown in Figure 2.3.

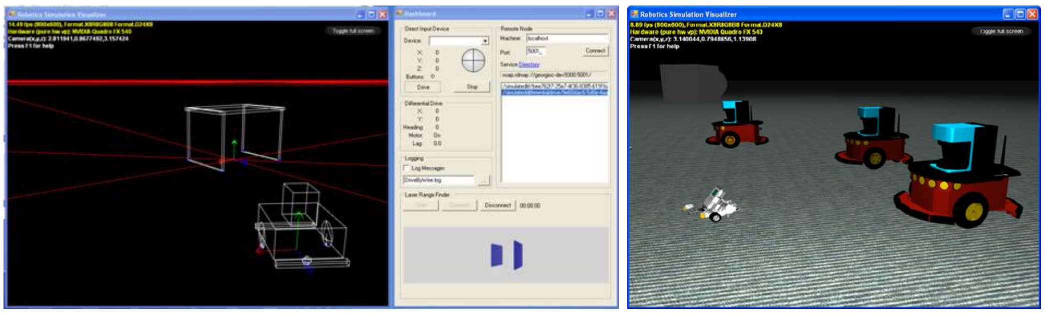


```
[ ] Pioneer2AT : robot1 : position
[x]     ObserverCam : userCam0 : guicam
[x]     SickLMS200 : laser1 : laser
[x]     SonyVID30 : camera1 : ptz
[ ] SickLMS200 : laser1 : fiducial
[x] SonyVID30 : camera1 : camera
```

```
ptz  22.0 -0.0 60.0
```

Figure 2.3: A screenshot of Player/Gazebo with a sample interface (from http://playerstage.sf.net/).

Microsoft© has recently released a similar toolkit entitled "Microsoft Robotics Studio". It is written in Microsoft's own .NET framework, and thus only runs on Windows. This is in heavy contrast to *Player/Stage*, which only runs on Unix, like Linux or Mac OS X^{TM}. MS Robotics Studio is based on SOAP and the REST interaction model. This means all communication in the system is text based and robots and their individual sensors are referenced using URIs (Uniform Resource Identifiers). This allows for a wide range of implementations and is easily extensible. The complete product is very similar to *Player/Stage* in the respect that it allows control over both real and simulated hardware via a unified interface. Its simulator uses a state-of-the-art commercial physics simulation engine and models the virtual world in 3D. Two views of the visualizer are shown in Figure 2.4.



(a) MS Robotics Studio Interface (b) MS Robotics Studio 3D visualization

Figure 2.4: Microsoft Robotics Studio's interface showing the physics models, and the 3D visualization of three Pioneer P2DX robots with laser scanners and a LEGO Mindstorms NXT robot.

Microsoft's product is currently free, however it is a *Community Technology Review* version, meaning a public beta test. It is also very immature piece of software when compared to the seasoned *Player/Stage* package.

2.3 Designing A Successful Approach

The underlying goal of all leagues in the RoboCup Rescue competitions is to work towards intelligent systems that can be applied by rescue workers during real disasters. In the Virtual Robotics League described above, this goal is reflected in the scoring function that is used exclusively to assess the performance of participants. Therefore, our strategy was to maximize each one of the rewards and minimize all penalties of this function.

Note that the scoring function presented above has several rewards that are bounded above to 50 points. However, the victim score grows linearly with the number of victims found and that each victim yields a maximum of 40 points. Thus, this scoring function leaves the possibility to develop a strategy which focuses exclusively on detecting as many victims as possible and provide as many localization-independent details per victim as possible. However, we felt that this would bypass the fundamental goals of the Rescue competitions. An extensive report on victims is of minor use to a human rescue worker when the victims are not depicted on a map or when this map is of poor quality. Therefore we only considered a *complete* solution that aims to optimally address *all* aspects of the challenge.

The scoring function suggests several design guidelines for such a complete system. The most important factor was the number of operators. As noted above, a single operator already increased the denominator in the scoring function from one to four. This implies that an operator should only be considered in the case where his or her presence would mean that more than four times the number of victims can be detected than without an operator. We considered this unlikely, so our primary aim was to develop a fully autonomous solution. The generated map should contain very highly detailed information about the geometry of the environment to optimize *MetricMapQuality*, and to potentially facilitate autonomous behavior. Furthermore, a high degree of map detail would increase the *TopologicalMapScore*. In addition, a multi-agent solution would lead to more exploration, which would benefit the *ExplorationScore* and also increase the chances for a higher *VictimScore*.

One consequence of choosing a multi-agent solution is that the robots need to be able to share information by jointly constructing a single map. When all robots start from the same position, this reduces to initializing a shared map. The map is then used by every robot for localization. It is also extended and updated by every robot as they explore the environment. However, in the urban search and rescue setting of our league, robots sometimes enter the environment at different locations. This complicates the map-sharing, as then the relative positioning of the robots is not initially known. In that case, each robot needs to start by constructing an individual map until the relative position of other robots can be determined and the individual maps can be *merged* into a joint map. A typical event that triggers map merging is the identification of a landmark by a robot that was previously observed by one of the other robots.

As described above, the league organizers warned us that we should expect more complex environments as the three test arenas shown in Figure 2.2. Therefore, we added the capability to deal with complex multi-floor arenas as another design guideline.

Based on the analysis so far, the following design guidelines emerge:

- the system should be able to coordinate multiple agents
- the system should work fully autonomously
- the system should use a map representation that is capable of:
  - providing highly detailed information about the environment's geometric proper-

ties

- representing complex and potentially multi-floor environments
- merging multiple maps into one

In order to meet the previous guidelines, we aim for a team of robots. There are several factors to take into account when designing a team of robots. Small robots can, on the one hand, reach places larger robots can not fit into. On the other hand, larger robots typically can carry many more sensors. They are also less susceptible to small obstacles and debris. Covering both these strengths could be attempted in a *heterogeneous* team of robots. However, creating such a team implies a significant amount of additional development effort. Different robot models possibly employing different sensor setups have different characteristics and therefore may require specific adaptations to the software.

Chapter 3

Simultaneous Localization and Mapping

3.1 Introduction

It is near impossible to imagine an autonomous mobile robot that was not programmed with some tasks in mind. Researchers have put forward mobile robots in an extremely wide range of applications which were to be executed under an equally daunting range of circumstances. To date, autonomous robots have been reported to explore environments from regular-shaped office buildings [8, 23] to difficult terrains like the surface of Mars [35] and even underwater, where robots have inspected the Great Barrier Reef [66]. Some robots operated alone [56], others in moderate-sized teams [16] and some robots have worked in teams up to a hundred robots [26, 30]. The range of tasks assigned to these robots has proven to be equally challenging. Robots were programmed to assist or entertain elderly people [45], to act as a museum tour-guide [54], to help locating victims in collapsed buildings [6] and also to complete the very challenging task of traversing several miles of desert road fully autonomously [57].

One factor that brings these different implementations together is that all robots have a need for *maps*. The majority of tasks that robots have been designed to fulfill are location-based, in one way or another. Soccer-playing, terrain exploration, underwater inspection and many other tasks require the robots involved to know their whereabouts. The internal representation and the amount of retained information about the environment differs from application to application. Regardless of the specific setting, however, a map facilitates the *localization* of robots. This localization, in turn, is a prerequisite to a meaningful interpretation of current observations, evaluation of executed motions, analysis of action

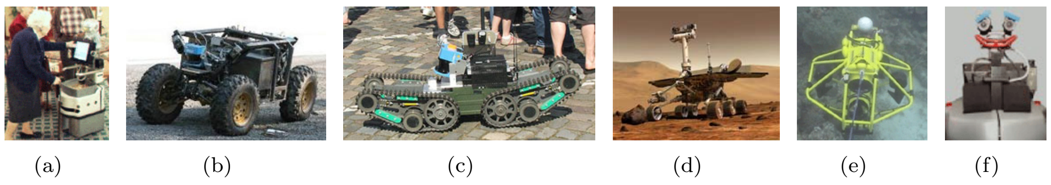


Figure 3.1: Some interesting robots. a) Pearl, robot of Nursebot project [45]. b) Mine exploration robot, courtesy of [19]. c) A rescue robot seen at a demonstration during the RoboCup World Championship 2006 in Bremen, Germany. d) Mars Exploration Rover (MER) [35]. e) Underwater robot, courtesy of [66]. f) Minerva, the museum tour-guide robot [54].

plans, etc. In short, maps provide a spatial context that enables much more elaborate intelligent behavior. From a different perspective, maps enable robots to reach beyond the limitations of purely reactive behavior.

Accurate maps of environments could be provided to robots beforehand. However, in reality this only applies to a small range of applications. Existing blueprints or detailed CAD (Computer Aided Design) drawings might give an accurate description of a building's walls, stairways and doors, for example. Nonetheless, they typically lack information about other objects which *are* definitely obstacles to robots as well, like furniture, debris, low-hanging lamps or steps. In other scenarios it is simply impossible to provide a map beforehand. For instance, rescue robots search collapsed buildings and exploration robots navigate on Mars or deep underground in mines, where they navigate through environments that are simply unknown.

Therefore, in most applications the robot is designed to acquire a map by itself. The robot is then equipped with a number of sensors as appropriate which enable the robot to observe its environment and make a map of it. *Mapping* algorithms address exactly this challenge as they analyze a robot's sensor readings and translate these into a map while the robot explores the environment. Assuming perfect information about the robot's exact current location, mapping is about acquiring the most accurate description of the relevant properties of the physical reality based on sensor readings, despite the associated measurement noise.

In realistic settings such an external source for exact position information is, of course, not available. GPS (Global Positioning System) is sometimes used, but it is not very accurate and only useful outdoors. Therefore, a mapping algorithm is typically paired with an appropriate *localization* algorithm, which brings us to the topic of *simultaneous localization and mapping (SLAM)*. Given a map, localization entails finding the most likely current robot's location and orientation on that map. This is determined by finding the pose that best explains the current sensor readings after comparing these against the information that is stored in the map. In this sense, localization is a global search problem. In most

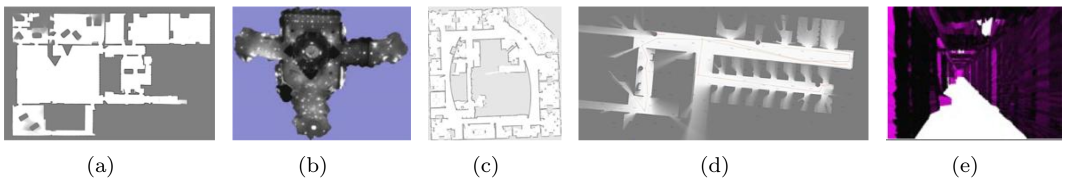


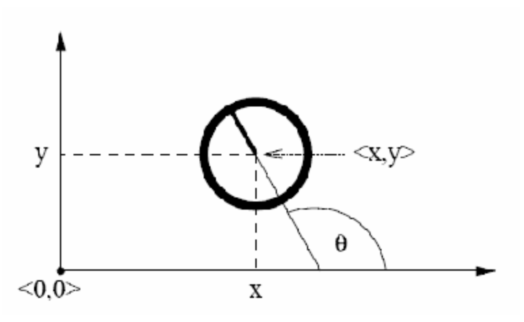
Figure 3.2: Some interesting maps. a) Map of an office environment [26]. b) Minerva uses a ceiling map for localization in the Smithsonian museum, Washington, courtesy of [53]. c) The Intel Research Lab, Seattle, courtesy of [25]. d) Map generated by *ManifoldSLAM*, described in Chapter 5. e) A volumetric map, courtesy of [55].

implementations however, a global search is not feasible without jeopardizing real-time performance. Therefore, in *online* SLAM, localization is usually done incrementally. Each time the localization routine is invoked, it is initialized with the last known robot pose and the motions executed since then. Using this information, a first estimate for the current location can be obtained. Due to noise in the robot's actuators and sensor readings, this is indeed just an initial estimate for the pose. Thus a search for the optimal explanation is still in order, but now only on a *local* scale.

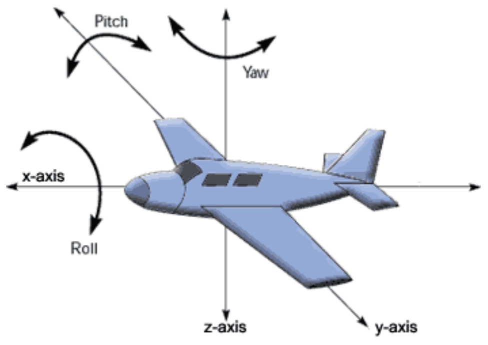
Essentially, SLAM is about having a robot autonomously acquire an adequate answer to the question 'Where am I?'. However, the question about the whereabouts of a robot is hardly ever posed just for its own sake. Rather the contrary, the two pieces of information that are put forward by SLAM, the map and the robot pose, are a key building block towards many intelligent behaviors. For example, they are a prerequisite to many navigation and path-planning algorithms and they can facilitate more elaborate behavior and motion planning. In general they provide a natural context to relate observations, decisions and actions over time. Only by employing SLAM techniques it becomes possible to place an autonomous robot at an unknown location in an unknown environment and then have it autonomously complete location-based tasks successfully. As SLAM alleviates the need for a priori knowledge of the environment, SLAM is sometimes considered the 'Holy Grail' of truly autonomous intelligent systems.

Considering the intelligent behaviors that depend on SLAM, it is arguably *the* primary task of SLAM to capture the robot's whereabouts in representations that *best facilitate subsequent algorithms*. Different representations of the map will have different mapping capabilities and limitations, as well as different practical ramifications in terms of update performance and memory allocation. But perhaps more importantly, some representations are better supported by subsequent algorithms than others. It could even be that particular representations simply enable or disable particular usage scenarios. The choice for a particular representation is a delicate one, and must be considered in a broad context that at least includes the algorithms that will actually make use of the map. Most current approaches represent the acquired information in *planar*, hence inherently two-dimensional, data structures. Some applications though require all spatial dimensions to be incorporated and need *volumetric* maps. Path planning algorithms often work on graph structures and are therefore best served with a topological representation of the map which outlines the connectedness of explored areas. Navigation and obstacle avoidance algorithms on the other hand are not really in need of topologically organized information. They are rather provided with high-resolution information about the local geometry of the environment, which are best encapsulated in an occupancy grid that outlines obstacles at an arbitrary level of detail. These different map types are shown in Figure 3.4.

Regarding the representation of the robot pose, it is sufficient for most scenarios to maintain the pose in 2D. In that case, the robot pose can be fully represented by three variables: An (x, y)-coordinate that denotes the location, and an angle (θ) that denotes the orientation. In 3D, a complete description of the robot pose requires 6 variables: an (x, y, z)-coordinate to denote the location, and a rotation around each of the axes to denote the robot's orientation. These rotations are commonly known as *yaw* (rotation around z-axis), *pitch* (rotation around y-axis) and *roll* (rotation around x-axis). In robotics, the common convention for the coordinate system is to have the x-axis pointing forward, with the y-axis pointing sideways and the z-axis pointing upwards or downwards.



(a) pose in 2D



(b) pose in 3D

Figure 3.3: Pose representations.

The choice for a particular SLAM approach is governed by the choice for a particular map representation and whether poses should be maintained in 2D or 3D. As the capability to serve subsequent algorithms is optimized, SLAM approaches become increasingly biased to particular application contexts. This implies that no single optimal SLAM approach exists. Given a particular use case, different approaches should be compared and contrasted in a search for the optimal solution. To enable the evaluation and discussion of different SLAM approaches on a higher level a generalization that is commonly used in SLAM research will be adopted here as well. A distinction between four families of SLAM approaches is made, which is based primarily on the type of map representation:

- SLAM approaches that employ *Kalman filters* on *feature-based maps*;
- SLAM approaches that provide *occupancy grid maps*;
- SLAM approaches that use *topologically organized maps*;
- SLAM approaches that use *maps with hybrid data structures*.

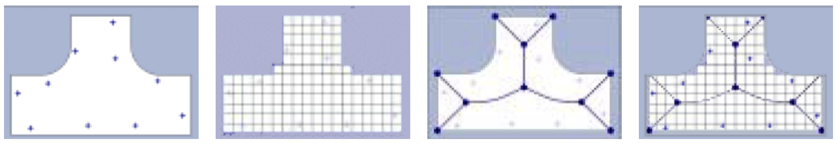

(a) features (b) grid (c) Voronoi diagram (d) Hybrid

Figure 3.4: Different map representations.

The former three families could be seen as the *elementary* SLAM approaches. They stick to a specific map representation and therefore exhibit the typical characteristics that belong to that map representation. During the discussion of these elementary approaches, their strengths, weaknesses and practical ramifications will be outlined. *ManifoldSLAM* though belongs to the fourth family of SLAM approaches. Like most current state-of-the-art SLAM approaches, it mixes elements of the former three representations in a hybrid data structure in an attempt to acquire the optimal solution.

The remainder of this chapter will discuss the the details of these four families of SLAM approaches and several current state-of-the-art implementations. The next section, however, first outlines the details of the challenge that SLAM approaches need to address. This provides the parameters based on which different approaches can then be evaluated in subsequent sections.

3.2 The SLAM Challenge

Before discussing the details of SLAM algorithms, the challenge that SLAM has to address and the complicating aspects are presented in full detail. Some of these aspects that make up the SLAM challenge are typical for any system that deals with real-life realities like sensor noise and odometric error. Several other aspects are typical to SLAM, like the bi-directional dependency between map accuracy and pose uncertainty.

The following are some of the key aspects that make up the SLAM challenge:

- sensor limitations and measurement noise;
- inaccurate motions and actuator noise;

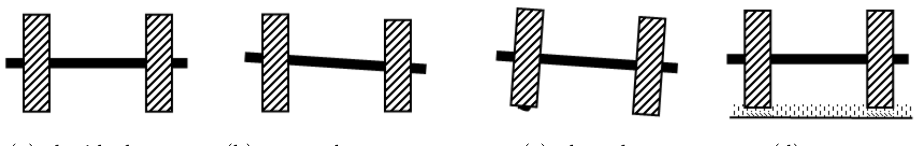
- error accumulation due to the orchestration of online SLAM;
- data dimensionality;
- data association;
- dynamic environments and obstacles;
- exploring the unknown;

3.2.1 Sensor Limitations and Measurement Noise

To acquire a spatial model of the physical environment a robot needs to be able to perceive the world. Sensors commonly brought to bear for this task in robotics include GPS, compasses, range finders using laser, sonar or infrared technology, audio sensors, tactile sensors and cameras. All these sensors are subject to measurement errors and limitations. For example, most of these sensors cannot penetrate walls or have a otherwise limited range, some of them fail to observe obstacles that are transparent or have difficulty sensing objects with particular kinds of surfaces. GPS works excellently in the outdoors but hardly indoors and cameras are often highly sensitive to lighting conditions. In addition to these and other limitations, all of the sensors are bound to be affected by noise in their measurements.

3.2.2 Inaccurate Movement and Odometric Error

To explore an environment, a robot has to navigate through it. This involves some motion commands to be sent to one or more of the robot's motion actuators (e.g. wheels or tracks). This gives rise to its own particular type of problems often referred to as odometric errors: a mismatch between the desired movement as specified by the motion command and the actually achieved movement by the actuator. The difference between the *true* movement and the desired movement can be the effect of any or all of the following: inaccuracy in the actuator, slippery or uneven surfaces, bumps into or over obstacles or other unforeseen challenges yielded by the physics of the environment.



(a) the ideal case (b) unequal movement (c) obstacle (d) carpet

Figure 3.5: Causes for odometric error, images courtesy of Thrun.

3.2.3 Error Accumulation

Clearly the aforementioned noise in measurements and movements pose their challenges on generating accurate interpretations of sensor observations. If these observations were independent then the error would be canceled out as the number of observations increases. In that case acquiring more accurate maps would be just a matter of processing more observations. Unfortunately, with simultaneous localization and mapping we have no such independence, rather the contrary.

The knowledge about robot poses provided by localization serves as the reference frame for mapping to interpret and localize sensor observations and build a map from these. Localization on the other hand, uses the map to compare current observations in order to estimate the current pose of the robot. Thus, localization and mapping each use the other algorithm's output and in turn serve as the other algorithm's input, they are *interdependent*.

Initially both the map and the current pose are unknown. Typically the pose is initialized to zero and from there on, sensor readings are positioned and interpreted to form a map and estimate pose updates. Errors in pose update estimates cause errors in the map resulting in degrading map quality. This in turn has its negative effects on subsequent pose estimates. This is a vicious circle and is referred to as *error accumulation*.

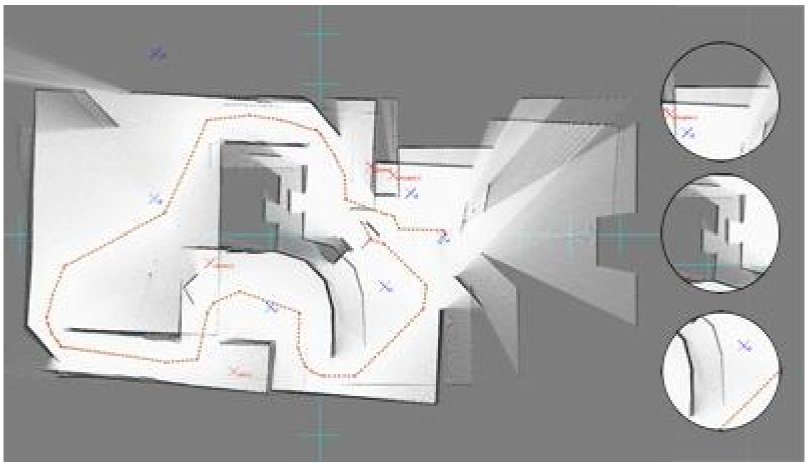


Figure 3.6: Error accumulation. The robot started on the right, moved to the left using the corridor above, then moved downward the the bottom-left corner and returned to its starting position using the corridor underneath. The map clearly shows the effects of error accumulation. The observations near the end of the robot's trail are estimated at a significantly different position from the observations that were acquired near the beginning of the trail. See also the 'ghosting' of the walls in the upper-right and near the center and the curved wall at the bottom.

In figure 3.6, the accumulation of error is clearly visible. Near the end of the robot's path, several parts of the environment are observed again but obviously misplaced with respect to their earlier observations. See also the 'ghosting' in the highlighted regions.

3.2.4 Data Dimensionality

Another challenge that is posed to SLAM algorithms is dealing with data dimensionality. Consider how much data it would take to describe a particular environment up to a certain amount of detail. To preserve memory, one could resort to just a topological description of the major features, for example the logical positioning of rooms and a description of which of those are connected by doors and corridors. However, such a description lacks precision and is of use only in particular scenarios. As more precision is desired, for example when acquiring a two-dimensional floor plan with a resolution of 5 centimeters per pixel or perhaps even a three-dimensional model of an underwater coastline, more and more details about the environment need to be stored. From an algorithmic point of view, every additional piece of data adds to the dimensionality of the mapping and localization algorithms, their memory allocation requirements and their processing and maintenance time complexities.

3.2.5 Data Association

As a robot navigates through its environment it potentially visits the same physical location multiple times. This means that different observations taken at different points in time correspond to the same feature in reality. This is an invaluable piece of information that can be of great benefit towards acquiring accurate maps and pose estimates. The knowledge that a later observation and a prior observation correspond to the same feature in reality enables the SLAM algorithm to detect and correct its accumulated error.

However, to associate two observations also incurs a high risk. Imagine the possibly devastating results when two observations are falsely considered to match to the same thing in the physical environment. Another complicating factor is that as more of the environment has been explored, the number of possible matches grows progressively.

3.2.6 Dynamic Environments and Obstacles

In many real-life circumstances, agents cannot assume past or even current observations to still be representative of current the state of the world. Consider parked cars or pieces of furniture that were observed in the past but have moved in the meantime. Similarly, consider changes that also effect the topological organization of the world like a door that was open before but is closed now. Especially problematic are observations of objects that move already, like people, vehicles or other agents.

All of these circumstances add to the dynamics of the environment and currently there exists no general solution to dealing with these adequately [59]. Mostly agents explore an environment for a constrained period of time, during which the world is simply assumed to be static.

3.2.7 Exploring the Unknown

As it is typically not fully known what challenges will be provided by the environment, the robot should be equipped with robust behavior that enables it to deal with unanticipated situations. There can be unexpected kinds of obstacles, like pits to fall into, or other kinds of environmental aspects not accounted for.

The challenge is then to supply the robot with some adaptive strategies so that despite these unexpected contingencies it still does a proper job of mapping the environment or at least degrade gracefully.

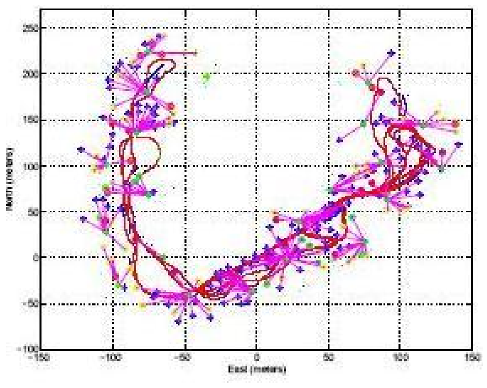
3.3 Approaches

Having outlined the aspects of the challenge to address in the previous section, we will now turn to the general approaches that have been put forward by SLAM researchers to date and discuss how these address the challenges described.

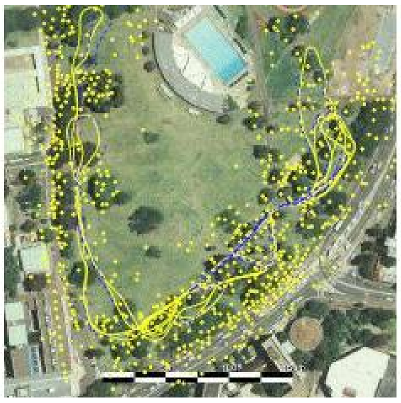
3.3.1 Feature-Based Maps: Kalman Filters

Historically, Kalman filter-based SLAM is the earliest, and arguably the most influential, approach to address the SLAM challenge. The Kalman filter is named after its inventor and was first introduced in [28]. Research to the application of Kalman filters to the SLAM problem traces back to a series of influential papers by Smith, Self and Cheeseman [47, 48]. Based on their initial work, numerous researchers have further developed Kalman SLAM [11, 15, 17, 50] into one of the most popular SLAM approaches currently used.

In essence, Kalman SLAM refers to the application of a Kalman filter to the online SLAM problem. Since Kalman filters are recursive estimators by nature, they lend themselves very well for incremental use as in online SLAM. Maps in Kalman SLAM are typically limited to just the current robot pose and landmark position estimates, hence the name *feature-based* maps. The basic idea is to describe the map, hence the joint space of the current robot pose and all landmark positions, as a Gaussian distribution. The map is therefore transformed into a state-vector μ that holds all the relevant variables and which then constitutes the mean of this Gaussian distribution. This state-vector typically has a length of (3 + 2N), 3 entries to describe the robot pose in 2D (x, y, θ), and 2 entries for every landmark position (x, y), with N denoting the number of landmarks. The covariance of this Gaussian represents the uncertainty over this joint space of robot pose and landmarks. The recursive estimation then entails finding and updating this Gaussian distribution, i.e. finding the most probable map.



(a)



(b)

Figure 3.7: A typical feature-based map of the Victoria Park, Sydney. a) Kalman SLAM typically only retains landmark positions and the robot path. b) A similar map overlaid on a satellite image. Images courtesy of [42] and [40].

Initially, in online SLAM, the algorithm begins with a map with no landmarks. The map then only consists of the current robot pose and a covariance matrix that represents the pose uncertainty. As new landmarks are observed, the state vector (mean) and covariance matrix are extended appropriately to incorporate the new landmark in the map. The initial estimate for the location of a new landmark is simply the projection of the relative distance measurement on a chosen global coordinate frame. The initial covariance values for the newly detected landmark follow from the measurement uncertainty and the pose uncertainty. Note that new landmarks are initially only correlated to the robot pose. These correlations 'spread out' only when the map is re-estimated in subsequent updates to correlate landmarks to each other as well.

A single map update in the Kalman SLAM algorithm involves several steps. First, the robot pose estimate is updated according to a linear motion model that describes how the robot motion actuators (e.g. wheels or tracks) effect the robot pose between the previous time step and the current one. When a landmark is observed, this can involve the detection of a new one, or the re-detection of one that was also detected previously. New landmarks will just be incorporated in the map. And, as long as no landmark *re-detections* occur, the pose estimate will become less and less accurate due to the actuator uncertainty. As depicted in Figure 3.8, the accumulation of this error is reflected in the covariance matrix by increasing values for the robot pose, and through the pose-landmark correlations also on all landmark location estimates. This continues until a landmark is re-detected. Typically, the re-detection of a landmark results in a significant drop in pose uncertainty, which then

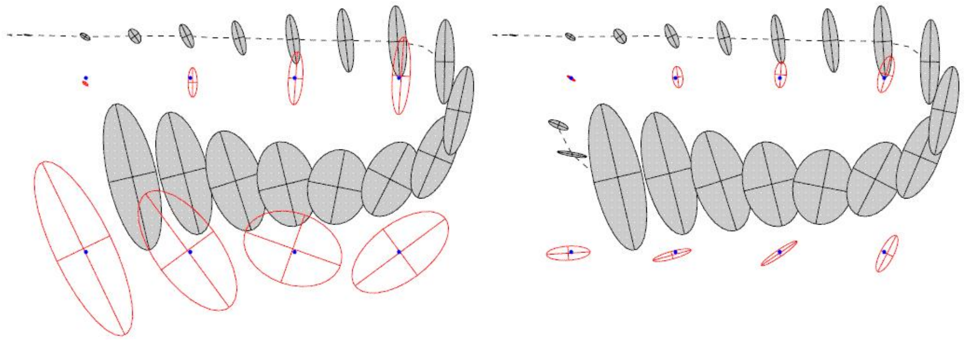


Figure 3.8: Typical rise and drop of map uncertainty with Kalman SLAM. Consecutive robot position estimates are indicated with shaded ellipses and landmark position estimates with unshaded ellipses. On the left: landmark uncertainty increases as robot pose uncertainty increases. On the right: after the re-detection of a landmark the robot pose uncertainty drops significantly, and through the information propagation this affects also all landmark estimates. Images courtesy of [38].

propagates towards increased certainty on all the landmark locations. This jagged pattern of slow rises followed by sudden drops in map uncertainty is typical for Kalman SLAM.

```
   Data: Previous map estimate $\mu_{t-1}$ and $\Sigma_{t-1}$
   Result: Updated map estimate $\mu_t$ and $\Sigma_t$
1  $\overline{\mu}_t = \mu_{t-1} + Bu_t$
2  $\overline{\Sigma}_t = \Sigma_{t-1} + \Sigma_{actuator}$
3  $K_t = \overline{\Sigma}_t C^T (C \overline{\Sigma}_t C^T + \Sigma_{measure})^{-1}$
4  $\mu_t = \overline{\mu}_t + K_t(z_t - C\overline{\mu}_t)$
5  $\Sigma_t = (I - K_t C_t)\overline{\Sigma}_t$
```
Algorithm 3.1: Kalman SLAM

Above, the standard Kalman update routine is summarized in compact form. For a more extensive outline see [59]. On lines 1 and 2, the map estimate is updated according to the linear motion model B, where u_t holds the current motion command and $\Sigma_{actuator}$ denotes the associated uncertainty. Note that this update only affects the robot pose estimate, the equations stated above will leave all landmark position estimates untouched. This is because in most applications the landmarks are either fixed or at least assumed to be static within the robot's operating time. However, it is very well possible to accommodate for dynamic landmarks. In that case, a similar model can be designed which updates the landmark positions in the map at every update as well. Subsequently, on line 3, the *Kalman gain* K_t is computed, which incorporates the measurement uncertainty $\Sigma_{measure}$ that is associated with the current observation z_t. The Kalman gain matrix has a size of 3 by the length of the state-vector, and usually this matrix is not sparse. In effect this is the matrix that propagates changes in pose certainty throughout the map estimate, as the information gain is folded back into the robot's belief at lines 4 and 5. Note that the matrix C is just a convenience matrix that describes a mapping from the state-vector μ to an observation z.

One of the key characteristics that distinguishes Kalman SLAM from other SLAM approaches is that it estimates the *full joint posterior* over the map in an online fashion. This implies that at all times during online construction, the full uncertainty is maintained in the map. Consequently, it is possible, e.g. for navigation algorithms, to adapt to the current uncertainty in robot pose and landmark locations *online*. In addition, through the full posterior, Kalman SLAM has been proven to converge [15]. However, convergence is only guaranteed in the limit, hence when landmarks are observed infinitely often. Therefore, it is not really feasible in practice. Nevertheless, Kalman SLAM currently remains the SLAM approach with the strongest convergence properties.

Another characteristic of Kalman filters is that they assume all noise to be governed by Gaussian distributions. In the context of SLAM, this translates to assuming Gaussian sensor noise, actuator noise and data association error. This imposes a severe limitation of Kalman SLAM. Although sensor noise is often best approximated by a zero-mean Gaussian, odometry is typically governed by trigonometric functions. Furthermore, data association uncertainty is far from a Gaussian function. Consider a particular landmark that is potentially mistaken for another one close by. In such a case, the resulting distribution of potential robot poses is obviously not best approximated by the bell-shaped Gaussian distribution.

The potential data association error as well as the frequently applied means to avoid these, give rise to another common issue with Kalman SLAM. Correct landmark identifications are the sole driving force of the Kalman filter. As outlined in the description of the algorithmic details, landmark re-detections are the only triggers towards higher map accuracy. For the sake of these positive landmark identifications, most Kalman SLAM implementations are tuned to enforce a relatively sparse placement of landmarks. This is simply because a sparser placement of landmarks makes them better distinguishable already just by their location. In addition, it decreases the chance of mistakes due to co-occurrences. However, a sparser landmark placement also creates a sparser set of reference points in the map to ground other map-features or sensor readings to. The less information actually stored in the map, the less geometric properties the map can represent in high detail. Often, a map with high accuracy *and* high geometric detail is desired. Unfortunately, when employing Kalman SLAM, a trade-off between the the two is unavoidable.

Two other issues that pertain to this dilemma are the problems of localizing the robot and the quadratic update costs. As indicated, in Kalman SLAM the map is restricted to representing landmarks. Therefore, localizing becomes harder and less accurate as there are fewer landmarks to be involved in the localization process. As localization degrades, data association in turn becomes harder since larger pose uncertainties have to be overcome. A degrading data association performance negatively influences map accuracy again, and so this vicious circle of degradation continues. The issue of update performance on the other hand directly relates to the size of the covariance matrix. As every update in standard Kalman SLAM typically involves updating the whole covariance matrix through the Kalman gain, maps are often limited in the number of landmarks they will maintain (according to [52] a limit of approximately 1000 landmarks is not unreasonable). When too many landmarks need to be maintained, updating the covariance matrix can become so slow that the SLAM algorithm is no longer able to operate in real-time.

In summary, Kalman SLAM is a complete solution towards SLAM with an incremental algorithm at its heart that makes it applicable to online use. In addition, it has several interesting properties and it is relatively easy to understand and implement. However, some of the Gaussian assumptions that underly Kalman filters are clearly invalid. Also, the approach poses some significant challenges to overcome before it can be applied successfully. The primary challenge involves the significant effort that should be put into the engineering of feature detectors. Another challenge lies in the careful fine-tuning of the landmark placement convention towards a robust balance between the number and sparsity of landmarks and the desired properties of the map.

3.3.2 Metric Maps: Occupancy Grids

Elfes [18] and Moravec [41] were the first to use occupancy grids for mapping purposes. Occupancy grids provide an intuitive representation of the geometry of a robot's physical environment. An occupancy grid is made up of evenly spaced cells, each of which holds a single value that describes if the corresponding region in the environment is occupied. In its simplest form, each cell holds a binary value indicating whether or not an obstacle was detected at the corresponding region in the physical world (see also Figure 3.9, image on the left). It is trivial to implement a simple extension in order to accommodate for the value 'unknown' to also distinguish yet unobserved or unexplored areas in the map. Another popular representation uses floating point values instead of binary values. In that case values between zero and one are used as an indicator of the *belief* about whether an area is occupied or not (see Figure 3.9, image on the right).

Occupancy grids have several benefits over other representations. The most important one is that grid-based representations of a robot's physical environment can be used directly by most navigation, obstacle-avoidance and learning algorithms. Another key benefit of occupancy grids is that the resolution can be tuned to represent the environment's geometric properties at any desired amount of detail. In theory, this is bounded above only by the level of detail originally captured by the sensors. This property of occupancy grids makes them the ideal *metric* representation for maps that should contain a high amount of detail, a feature that most other map representations lack.

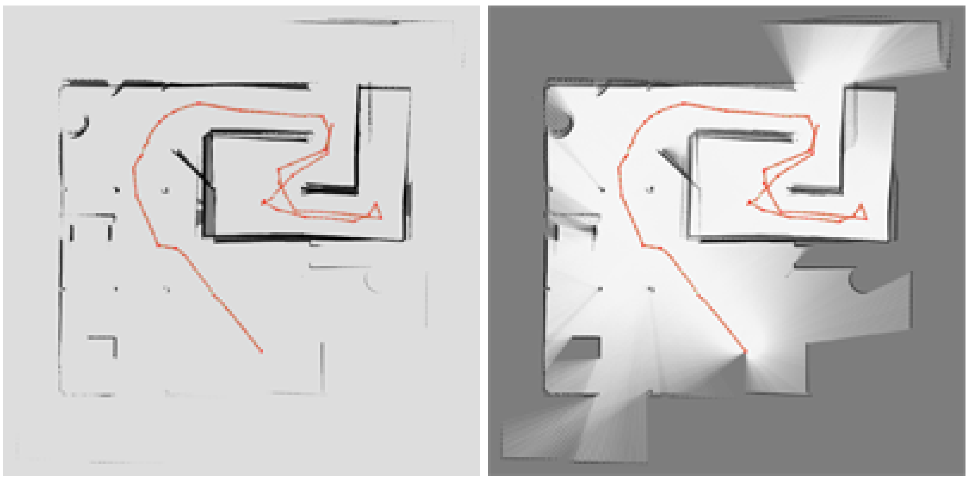


Figure 3.9: Two representations of the same occupancy grid map, the red indicates the robot's trail and the black indicates obstacles. On the left, using binary cells the map cannot distinguish unexplored area's from free-space. On the right, after ray-casting the obstacles the map becomes much more informative.

In its simplest implementation, the construction of an occupancy grid based on sensor readings is straightforward and uses a simple ray-casting technique. The occupancy of cells is then determined by counting *hits* and *misses* for every cell. These counters are initialized to zero and are updated by ray-casting all obstacle measurements as obtained from the sensors involved. Every imaginary ray is drawn from the respective robot pose towards the detected obstacles. For example, if an obstacle is detected at some relative distance, then rays can be cast from the cell at the current position of the robot towards every cell that coincides with the detected obstacle. For all the cells that then intersect with the cast rays before the rays hit on the obstacle the *misses* counter is incremented. Likewise, for the cells that intersect with the detected obstacle the *hits* counter is incremented. Afterwards, the occupancy of every grid cell can be determined by thresholding the ratio of the number of hits over the sum of hits and misses together. In case a notion of 'unexplored' regions is to be maintained, these would be all the cells with zero hits and misses. Similarly, when the cells hold floating point values instead of binary values, the threshold is not necessary and the ratio of hits versus the sum of hits and misses can be stored in the grid directly.

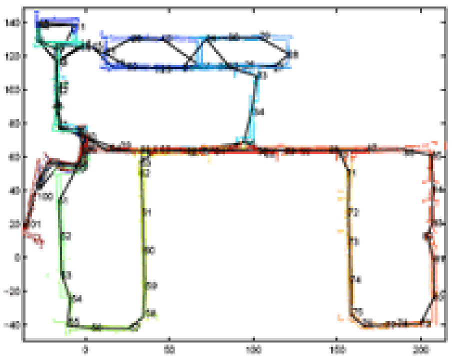
The greatest drawback of using occupancy grids is that their space and time complexities grow exponentially with the grid resolution. Therefore, they are not really well-suited to facilitate online SLAM, especially in large-scale environments. Another issue with occupancy grids is that although they are easily constructed, they are not so easily updated. This is because of the inherent data aggregation in occupancy grid cells. The problem concentrates on the fact that there is no trivial way to undo or alter past grid modifications, i.e. ray-castings, without storing a lot of additional information. As an example, consider the situation where, due to the re-detection of a particular landmark, a large accumulated error in the robot pose estimate is exposed. Naturally, the SLAM algorithm would want to correct for this error and update the map accordingly. This typically involves revised estimates for the poses that contributed to the accumulated error. Updating these implies that also the obstacle detections that are associated with these pose estimates are repositioned. To reflect these changes on the occupancy grid, this would require the repositioned obstacle observations to be redrawn. In other words, the current ray-casting should be undone and then redone based on the updated pose estimate. As an occupancy grid integrates over all plotted information, it does not directly allow for undoing the ray-casting of a single obstacle or observation. This could be made possible by storing additional information for every observation that is plotted in the grid, but then this would lead to excessive memory usage. In addition, the computational complexity of un-casting and re-casting a potentially large number of past observations could preclude the online usage.

In practice the backwards editing of occupancy grids is considered not desirable during online usage. Therefore, pose estimates have to remain fixed once determined. This has led to the situation where approaches employing occupancy grids are often equipped with a means to accommodate for this inflexibility, like particle filters where multiple possible pose sequences are maintained concurrently [25, 22]. See also Chapter 4 for details about scan-matching, which is the de-facto standard online technique for minimizing the influence of odometric errors in the pose estimates.

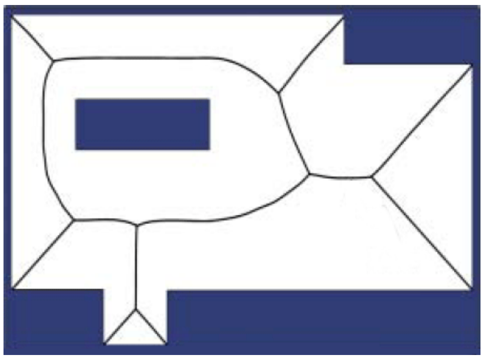
Another way in which occupancy grids are often employed is as a post-processing step after the main SLAM algorithm has finished [59, 58]. In this scenario, occupancy grid mapping is not used for SLAM itself, but merely facilitates subsequent algorithms or the visual reporting of maps. Some SLAM approaches internally make use of a map representation that is not directly suitable for subsequent use by other methods, for example obstacle avoidance or learning algorithms. In other scenarios the map representation that is used by the SLAM algorithm does not easily allow for visual rendering on screen or print. In these situations occupancy grid mapping can be used to transform the information that is stored in the map into a more usable representation.

3.3.3 Topological Maps: Graphs and Voronoi Diagrams

This section discusses a family of SLAM approaches which are based on topologically organized maps. The two types of map organizations mainly used in topological SLAM are graphs, where the nodes and links outline the environment's *connectedness*, and Voronoi diagrams, that *divide* a map into non-overlapping regions based on obstacle detections. Both representations, as all topological representations in general, put a strong emphasis on providing a compact description of the free-space regions in the environment and their interconnectedness.



(a) A graph representation of the traversed pathways, courtesy of [7].



(b) A Voronoi diagram that infers the safest paths from obstacle estimates which are indicated with the shaded polygons, courtesy of [12].

Figure 3.10: Topological maps.

The first key motivation behind employing topological map representations is their superb support for path planning algorithms. Autonomous robots need to to get from place to place in order to complete their designated tasks. Consider the task of exploration that almost all robots will need to follow in order to acquire a map in the first place. It is conceivable that at some point the robot will reach a dead-end of the environment and might therefore wish to continue exploration in another area. Before it can do so, the robot has to get to the other location first. By nature, topological maps facilitate these kinds of tasks excellently. The nodes and links in graphs and Voronoi diagrams tell any path planning algorithm precisely how to get around the explored areas safely.

The second key motivation for using topological map representations is their *compactness*. They are capable of representing huge environments in a very compact way. Whereas the size of occupancy grids and feature-based maps grow exponentially with the size of the environment or the number of detected features, topological maps typically only grow linearly in size as nodes and links are added to denote newly explored areas.

Graph maps are designed to literally capture the navigability of the environment. Along the robot's trajectory, all estimated poses are turned into nodes. Subsequently, links indicate the path the robot has traversed between consecutive poses. By employing a conservative method that refrains from implicit inferences and only adds links whenever a robot actually traversed the path between two locations indicated by nodes, a *consistent* graph is ensured [27]. Topological consistency means that the graph does not indicate connectedness that is not actually in the environment.

Voronoi diagrams are in certain ways the opposite of graphs. While Voronoi diagrams also use nodes and links to outline the estimated navigable free-space, the nodes are positioned using a different methodology. In Voronoi diagrams nodes are positioned at *equidistant* *points*, which are those points that are exactly the same distance away from all near obstacles. For placing these nodes only those obstacles are considered, for which there is no other node at a closer distance then the current one. This procedure allows for the links between any pair of connected nodes to indicate the safest paths between obstacles. The path is a straight line between two nodes that is equally far away from the obstacles on both sides of the link. Hence, like graphs, Voronoi diagrams lend themselves excellently for path planning purposes [31].

The main differences between graphs and Voronoi diagrams follow from the way links are treated. In graphs, links are only constructed for actually traversed paths, whereas in Voronoi diagrams they are estimated and their positioning is determined by the positioning of obstacles. Therefore, graphs can be seen as staying true to the actually traversed paths, which will ensure their consistency. On the other hand, Voronoi diagrams are capable to generalize beyond the actually traversed pathways as they infer the safest pathways based on the obstacle estimates. It is important to note that this feature of Voronoi diagrams imposes a bias towards planar environments, unless of course the obstacles and poses are modeled and maintained in all three spatial dimensions. But this would incur other negative side-effects on the performance and memory consumption of SLAM.

Shortcomings of topological map representations are mostly due to the lack of low-level information in these maps. Localization on a topological map often reduces to localizing to the nearest node, as more detailed information is simply not available. In addition, although the links provide safety paths for navigation, they do not provide the low-level geometric details to actually get the robot safely from one node to the next. Reactive obstacle avoidance is necessary to overcome this. However, as noted in [31], this does not necessarily need to be considered as a drawback, as not relying on map information might make the robot more robust to dynamic environments.

3.3.4 Hybrid Maps: The Optimal Combination

As illustrated in the previous sections, each elementary approach towards representing a map has several positive and negative sides:

- feature-based maps optimize landmark position estimates and pursue a global optimum but also impose a trade-off between geometric detail and map accuracy and the amount of features it can deal with is limited;
- occupancy grids are great for obstacle avoidance, preserving geometric detail and visualization purposes, but memory consumption makes them not well-suited for mapping large environments and they do not lend themselves very well for revisions and updates;
- and topological maps are great for navigation and path-planning purposes and are

capable of mapping large environments, but they typically lack detail.

Based on this it should come as no surprise that many researchers have attempted to acquire better solutions where the strengths of multiple elementary map representations are combined in *hybrid* data structures. Hybrid approaches are categorized into ones that *integrate* multiple representations in a single layer and ones that use a data structure with multiple layers where usually a topological layer at the top is used to *decompose* the lower layer into small-scale feature or grid maps.

In [51], topological and metric (i.e. occupancy grid) map representations are integrated in order to provide a versatile data structure that can serve the map information in multiple forms. The key advantage is that for example a path-planning algorithm can generate an action plan based on the topological data structure, which can subsequently be translated into its grid-based equivalent. This can then be used by a navigation algorithm in order to translate the action plan into a low-level motion plan that will deal with all the obstacle avoidance details.

An example of a hybrid approach that uses a divide-and-conquer strategy, i.e. map decomposition, can be found in the 'Hierarchical Atlas' presented by Lisien et al. [31]. A Voronoi diagram is used to describe the environment on the global level. Locally, on every link in this diagram, so-called 'edge maps' are stored. An edge-map is in fact an occupancy grid that stores the geometric properties of the environment as encountered when traversing the particular edge. As all information on an edge map is stored with respect to a local coordinate frame, the equidistant points and hence the links can be kept dynamic without compromising the validity of the information stored on the edge-maps. This yields two advantages: first, by using grid-maps on a small scale the limitations due to their static nature are avoided and second, grid-maps are only constructed to cover the extent of the actually explored area instead of having them grow on a global scale whenever the boundaries are explored.

ManifoldSLAM, like many current state-of-the-art SLAM approaches (see also Section 3.4), also employs a hybrid data-structure. It uses a graph organization at the global level that decomposes the map in many small-scale grid maps. This yields similar benefits to those observed by Lisien et al. [31]. But since we use a graph instead of a Voronoi diagram at the global level, we have been able to realize an additional significant feature: unlike many other algorithms *ManifoldSLAM* is *not* biased to planar environments and is thus capable of mapping multi-floor environments.

3.4 Current State of the Art

3.4.1 FastSLAM

FastSLAM by Montemerlo et al. [38, 40] is one of the most efficient SLAM algorithms which has been used on real robots. The main advantage of FastSLAM is the insight that all landmark observations in Kalman SLAM are statistically independent if the robot path is known.

Traditional SLAM estimates $p(s^t, \Theta | z^t, u^t)$, meaning the robot path (i.e. all poses s up to time t) and all landmark locations Θ given all observations z and actions u from the beginning to time t. Kalman filters estimate this joint probability completely as a mean and a covariance matrix. Therefore, it demonstrates an update complexity of $O(N^2)$, where N is the number of landmarks in the map.

FastSLAM uses a factored representation of this joint probability:

$$p(s^t, \Theta | z^t, u^t) = p(s^t | z^t, u^t) \prod_k p(\Theta_k | s^t, z^t, u^t)$$

which means that there are $n + 1$ separate estimation problems, where n is the number of landmarks in the map. Intuitively: If we know the path the robot followed through the environment, it is enough to estimate each landmark separately. Interestingly, this factorization is correct since the landmark observations are really conditionally independent given the robot poses. Figure 3.11 shows a graphical model of the SLAM problem showing this property.

The basic implementation of FastSLAM models these two kinds of probabilities in two separate ways. Each landmark is estimated by a 2D Kalman filter. A particle filter represents the posterior over all possible robot paths. This directly corresponds to the factorization given above. This naive implementation would have a complexity of $O(MN)$. Here, N is the number of landmarks in the map. The particle filter which estimates the posterior of the robot pose contains M particles.

The implementation by Montemerlo et al. [38] has an update complexity of $O(M \log N)$. This makes FastSLAM significantly more efficient than any Kalman based SLAM algorithms. Montemerlo et al. [40] prove that FastSLAM with one particle converges in the limit, which makes it the most efficient provably converging algorithm to date. Keeping track of multiple particles should only increase the convergence rate.

FastSLAM has been successfully used in many settings, including real world applications. Nieto et al. [42] and Montemerlo and Thrun [37] extend the basic FastSLAM algorithm to unknown landmark associations, which is important for generating occupancy grid or metric maps. Hähnel et al. [25] show its application to large scale mapping problems and

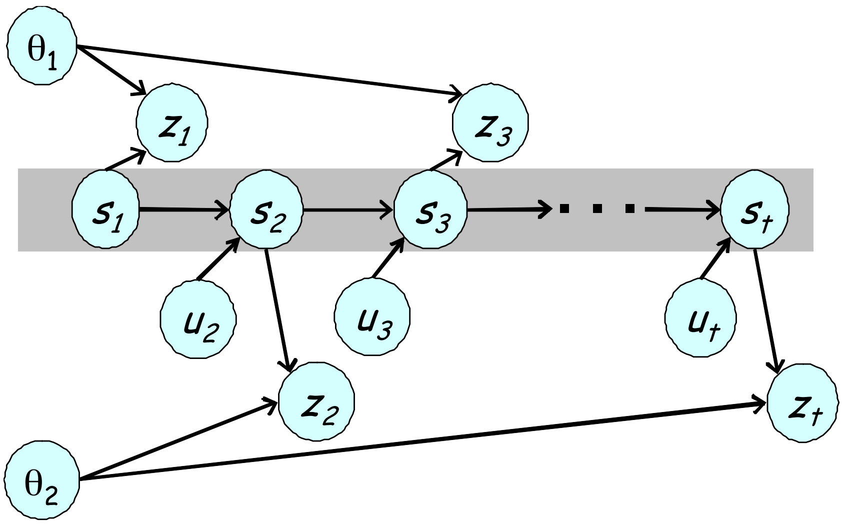


Figure 3.11: A graphical representation of the SLAM problem showing conditional independence of landmark observations θ_i given the robot path s_i. From [38]

the implementation of metric maps using FastSLAM.

3.4.2 Atlas

Another interesting existing approach to SLAM is the Atlas framework proposed by Bosse et al. [7, 8]. Very similar to the SLAM approach developed later in this thesis, it utilizes a graph structure of local maps to represent the complete global map.

In general, each node in the Atlas graph is a map of limited extend, build using a conventional SLAM algorithm. Each such *local* map is limited by some parameter (e.g. number of features in case of a feature-based map). Edges in the graph describe coordinate frame transformations between the two local maps they connect.

An important feature of Atlas is that errors are not bound to only one frame of reference, like in monolithic approaches like FastSLAM. By splitting up the map into sub-maps, estimation errors do not propagate between sub-maps. New sub-maps are seeded with zero uncertainty when they are added to the graph. The only way uncertainty is maintained between sub-maps is via the edges of the graph. Intuitively, correcting these coordinate frame transformations can be done in various ways (global optimization via map fitting, constraint optimization when loops are present, etc). The important point is that estimates within one sub-map do not have to be recomputed.

Another interesting property of this approach is the potential constant time update complexity. Many possible choices for the mapping algorithm on the local level achieve constant time updates when the maximum size of a local map is bounded. Instead of extending the same map by more and more observations, Atlas generates a new local map in its graph structure when the current local map gets too complex (or too uncertain, corresponding to the argument in the previous paragraph). Global localization potentially becomes linear in the number of local maps.

Loop closing, while implicit in algorithms like FastSLAM, has to be dealt with explicitly in Atlas. To this end, the authors develop a generic map matching technique. Two maps are compared by features extracted from them which do not depend on the specific translation and orientation of the coordinate system in the map.

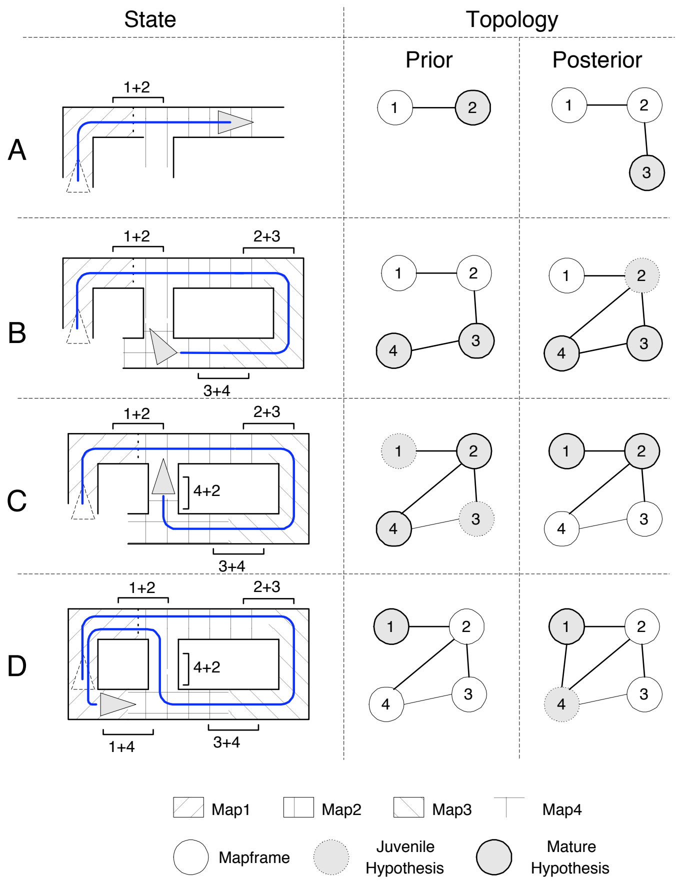


Figure 3.12: Sequential evolution of an Atlas map showing sub-maps and their linking edges. From [8].

Figure 3.12 shows a usual Atlas map being generated. The diagram clearly shows the key aspects of Atlas. New sub-maps are added when the current sub-map becomes too uncertain or too big (A and B). Loops are closed by recognizing similar structures (B and

C). Sub-maps can be extended later on if possible (D).

Lisien et al. [31] introduced the Hierarchical Atlas. In this case, *hierarchical* means exploiting the information in the graph for higher level reasoning tasks. More specifically, the topological graph is a Generalized Voronoi Graph (GVG). In a plane, all points on a GVG are equidistant to the nearest two obstacles. See Figure 3.13 for an example.

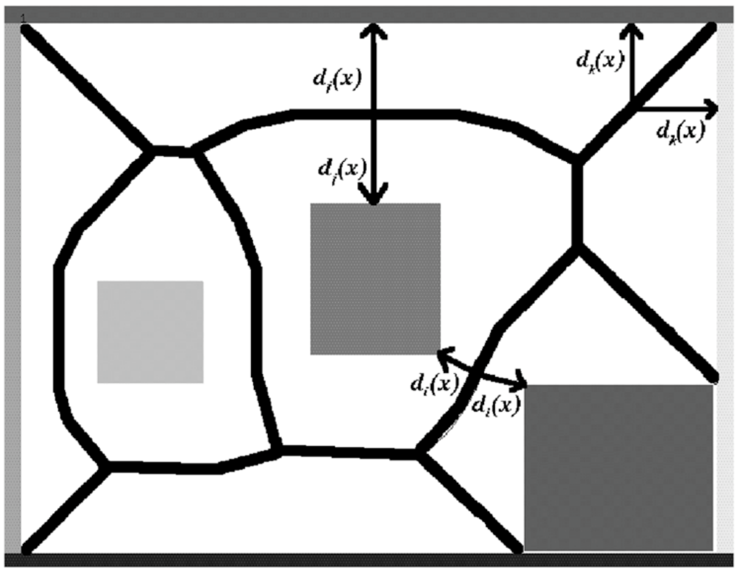


Figure 3.13: An example GVG in a plane. All points on the GVG are equidistant to the nearest two obstacles. From [31].

The GVG encodes important information for higher level tasks such as safe path planning and exploration. Path planning is possible with simple shortest path algorithms. Since the paths on the GVG represent the safest paths in the environment (farthest possible distance to obstacles), the robot can neglect exact localization until the node closest to the goal location is reached. Additionally, the remainder of the path from the last node to the final goal is known to be free because of the GVG property. Similarly, exploration needs to expand parts of the GVG that do not end at an obstacle.

Lisien et al. [31] and Bosse et al. [8] include some results for data sets collected with real robots. Dragone et al. [16] show an application of Atlas in dynamic environments and multi-agent robot soccer.

3.5 Conclusion

This chapter has explained the challenge faced by simultaneous localization and mapping algorithms in detail. The general solutions to address this challenge have been presented and their positive sides, negative sides and practical ramifications have been discussed.

During the discussion of the *elementary* approaches, feature-based, grid-based and topological, we observed that each has its own strengths and drawbacks. This explains the current trend to combine individual strengths and overcome the mentioned drawbacks in so-called *hybrid* approaches. The details of two such current SLAM approaches, FastSLAM and Atlas, have been discussed.

ManifoldSLAM follows the current trend towards hybrid approaches and combines a graph organization with metric representations. The graph is used at the global level to decompose the map in small-scale metric maps. In Chapter 5 we will further present the details of our approach.

Chapter 4

Scan Matching

4.1 Overview

When a robot moves in an unknown environment it will potentially encounter obstacles and other hazards, like slippery or uneven floors. In order to know or learn where the robot is situated, conventional methods are not sufficient in such an environment. Odometry tries to extrapolate motion from wheel rotations or acceleration sensors. Both approaches, which are often combined, have their individual drawbacks. The measurement of wheel rotations is highly affected by slippery or uneven floors as wheels could turn without actually moving the robot accordingly. The inherent error in acceleration sensors is amplified by the double integral necessary to extract motion. While odometry is a valuable tool and can gather important information on smooth and flat floors, it can be quite erroneous in its belief about real motion in more unstructured environments. Additionally, since all sensor measurements are of a completely relative nature, errors will accumulate over time and impact the robustness of odometry significantly.

A robot needs to sense distances to objects around itself to avoid obstacles. Not only can this distance information be used to avoid certain hazards but it can also aid motion estimation. Instead of, or in addition to, conventional methods, the robot would be able to infer its own motion from changing distances to surrounding objects. See Figure 4.1 for a visual description. Distance measurements depend only on the real motion of the robot, so the quality of the floor does not influence it. They also directly relate to motion, much like the acceleration sensors. Unlike the case of acceleration sensors, however, no integration of sensor values or similar processing is needed, which reduces the possible error significantly.

One such sensor is a sonar distance sensor. It will measure the distance to the nearest object in the direction it is pointing. Unfortunately, these kind of distance measurements

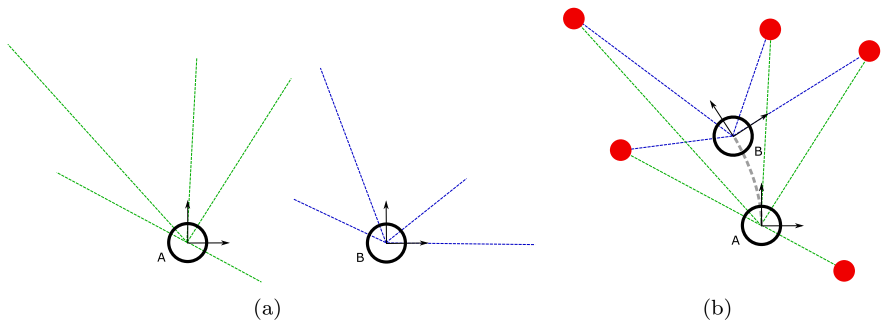


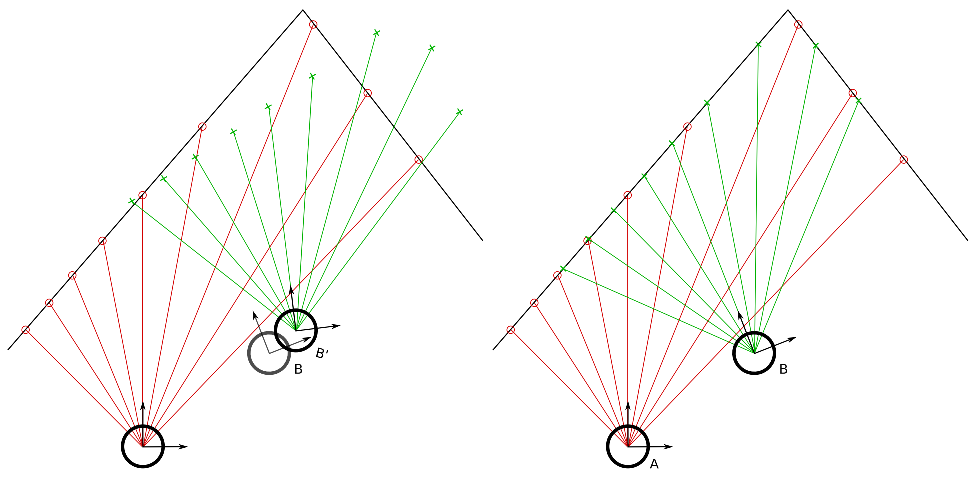
Figure 4.1: Motion from distances to features: a) Measurements taken at two consecutive poses **A** and **B**. b) Motion that caused the measurements to change and is thus computable from the changes it introduced.

are usually not precise or numerous enough to be used in accurate and efficient algorithms. Another important sensor is the laser range sensor. Just like the sonar sensor, it measures the distance in the direction it is pointing, but instead of sound it uses a beam of light. This makes it much more precise, often up to a couple of millimeters, and measurements can be taken very rapidly.

The laser range sensor can be used in a rotating setup which will allow the robot to observe almost its complete surrounding generally as a 2D slice. In the most common case, it will rotate horizontally. In this case, it could not sense the floor or the ceiling, but it can easily detect walls. Other setups exist, such as combined horizontal and vertical rotation, but are less common. In indoor environments, horizontal slices are a very efficient method to describe the surroundings. Plans like that are often used by humans as well, for example in architecture. Outdoors, however, more elaborate methods are needed. The main difference between in- and outdoors is the much fewer number of observable objects outdoors and that the ground is possibly not as flat. Laser range sensors have a maximum range, above which readings are unreliable. Outdoors, many distance readings will be unusable for the motion estimation or mapping methods because not many objects are in range of the sensor. We will focus on the indoor and simple outdoor case (with relatively many objects) and discuss methods applicable to those settings.

Laser Range Scanners (or LADAR), as described above, are often the main means of acquiring spacial maps with mobile robots. This is due to their high resolution and precision. Laser Range Scanners measure the distance from the sensor to the nearest object at many different angles, which is also shown in Figure 4.2. Most commonly, such sensors measure 181 distances, one for each degree over a total of 180 degrees. However, there are also Laser Range Scanners which have a much higher number of beams per degree or that scan a larger total field of view. These types of sensors have been in use for a long time and the literature on their use in mobile robotics is thus quite extensive. [33, 32, 52, 38, 59]

In this chapter, we are concerned with using two or more Laser Range Scanner observations to estimate the difference in pose between those observations. This problem is generally phrased as “Laser Scan Registration” or “Laser Scan Matching”. An example of a successfully completed matching of two scans is shown in Figure 4.2. It is a very important problem in the scope of mapping and localization.



(a) (b)

Figure 4.2: Two scans A (red) and B (green). a) The second scan B' at an (erroneously) estimated location. The failure of alignment is obvious. B shows the true location of the second scan. In b), the scans have been successfully matched. An integral part of scan matching is to manipulate the origins of the scans so both scans describe the same geometry as shown in this figure.

The general problem that Scan Matching tries to solve is to optimize the pose parameters of a second scan relative to a previous scan such that the two scans describe the same geometry. While laser scanners produce polar coordinates (an angle and a distance), the general representation used in these algorithms are so-called “point clouds”, a set of 2D Euclidean coordinates. This is not only useful in Robotics, but also in Computer Graphics and Computer Vision. The most important algorithm (see Section 4.2.1) discussed in this chapter originates from these disciplines.

This chapter will first introduce basic approaches to the Scan Matching problem and then highlight specific algorithms that were evaluated. Finally, some experimental results are presented which will motivate the choice for the final implementation discussed later in Chapter 6.

4.2 Approaches

4.2.1 Point-Correspondence Methods

Some methods try to estimate corresponding points in the two scans to be matched. The difference between these methods is the metric they use to find corresponding points. Like shown in Figure 4.3, it is the rule which finds the "closest", or corresponding, point which governs how well the algorithm can recover rotation, translation and how efficient it is at it. The general point-correspondence method will try to minimize distances between the corresponding point pairs. This minimization might actually change the points such that new correspondences emerge or old ones are discarded, see panel (b) in Figure 4.3. Generally, such methods find local minima only if the initial guess of the displacement is too far away from the global minimum.

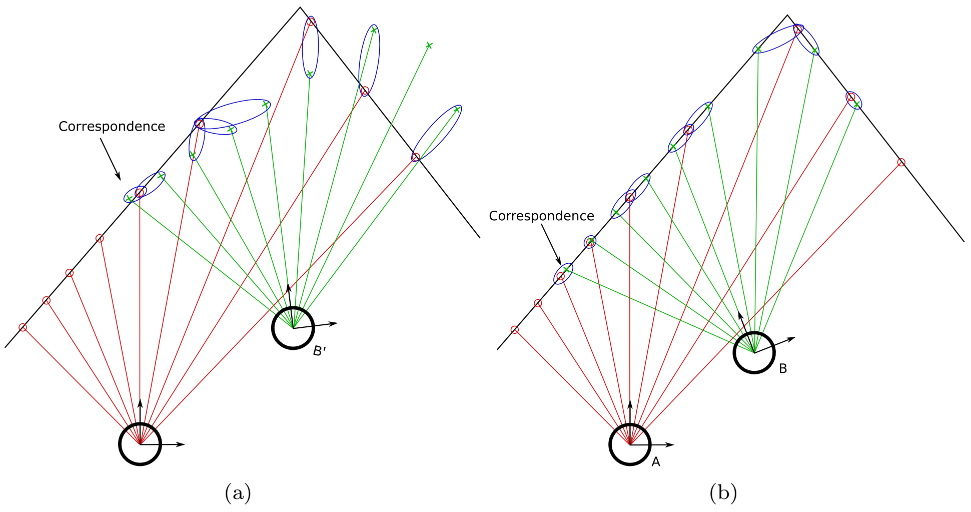


Figure 4.3: The points of one scan must be associated with points of the other (blue ellipses). How this is done is different between methods. a) Unaligned scans. b) After alignment.

The most prominent algorithm in this family is the Iterative Closest Point algorithm, described below. However, it is important to note that there are many algorithms derived from the above idea. Many try to remedy the inherent difficulties, such as how to find the "right" corresponding points early in the process to speed up convergence. Another common problem with this kind of algorithm is that real laser scanners almost never measure the same point twice, which is one of the assumptions made above. It is easy to see this problem in the right picture of Figure 4.2. Even though both red and green scans consist of the same number of points, their laser rays with which they measure the distances do not hit the wall at exactly the same points. At least light offsets are present and as a result, different distances are reported. It might even happen, that one scan has rays that hit obstacles such as trees, doors or table legs while the other does not. This introduces a non-trivial error which makes many algorithms very inaccurate in small indoor environments. For future reference, we will call this error *Correspondence Error*, a term borrowed from Pfister et al. [44].

Iterative Closest Point

This algorithm originated in the Computer Vision and Computer Graphics community [4] and is rather closely related to the Expectation Maximization (EM) Algorithm of general Machine Learning. In these disciplines, it is used to combine scans of 3D objects in order to represent them as 3D computer models.

The Iterative Closest Point (ICP) algorithm is very simple in itself.

We will define a mapping given a pose difference estimate $q = (x, y, \theta)$ that projects a given point p according to q.

$$q(p) = \begin{pmatrix} x + cos(\theta)p_x - sin(\theta)p_y \\ y + sin(\theta)p_x + cos(\theta)p_y \end{pmatrix} \tag{4.1}$$

```
Data: Laser Scans $a$ and $b$
Result: A pose which describes the transformation from $a$ to $b$
1 initialize pose estimate $q$;
2 while fit not good enough do
3     for each point $b_i$ do
4         find the closest point $a_j$ to $q(b_i)$;
5         save relation between points $b_i$ and $a_j$ as pair $(i, j)$ in set $R$;
6     end
7     find pose which minimizes the mean square error
      $E(q') = \frac{1}{|R|} \sum_R distance(a_j, q'(b_i))^2$;
8     let $q = q'$;
9 end
```

Algorithm 4.1: Iterative Closest Point Algorithm

This algorithm will eventually converge to a good estimate of the pose difference between the origins of the two laser scans. The main parameters to take into account in this approach is the distance metric used and any sort of exclusion criteria of relations computed in the for loop. These might include distance thresholds and other ways of excluding outlying points. The most important property of this algorithm is that pose updates can be computed in closed form once the distance metric is known.

Later, we will evaluate three methods that are based on this general algorithm.

4.2.2 Function Optimization Methods

As it was introduced before, scan matching is nothing but an optimization problem. The function to be optimized is the "fit" of one scan to another given a transformation. It is therefore straightforward to phrase scan matching as such a problem. Algorithm 4.2 shows the most general method using this scheme.

```
  Data: Laser Scans $a$ and $b$
  Result: A pose which describes the transformation from $a$ to $b$
1 initialize function $f_{ab}(q)$ such that it has a minimum when $b$ and $a$ are aligned using
  transformation $q$;
2 $q = argmin_{q'} f_{ab}(q')$ ;
```

Algorithm 4.2: Scan Matching using Plain Optimization

Following this scheme, there are several different ways to define $f_{ab}(q)$. This choice will significantly impact the optimization algorithm that is to be used in the last step above. In some cases, an analytical optimization might be possible, however numerical optimization might be necessary in others.

While not many methods rooted in this paradigm exist, the most recent ones will define $f_{ab}(q) = \sum_i f_a(q(b_i))$. In this case, $f_a(c)$ describes how well one single point measurement c would fit the previous scan a. This function is usually a probability distribution, but it can be anything meaningful. Sometimes, it is also called "Energy Function". Scan matching is then done via numerical or analytical optimization over q.

Such energy functions can be derived using Gaussian kernels to estimate the probability distribution over possible repeated observations in later scans. Other methods to estimate such distributions exist and the choice of the method lies at the heart of this approach. The better the distribution, the faster the optimizer will converge.

Not many efficient implementations of this method exist, presumably because of the complex functions that need to be optimized. Such functions are often not smooth enough for numerical optimization or give rise to many local minima. Global optimization algorithms, which are needed to compensate for these kind of problems, are inefficient when compared to point-correspondence methods for scan matching. In the case of analytical optimizations, this might still be a problem if the function does not behave well around the global minimum.

4.3 Specific Algorithms

Four distinct algorithms are discussed in this section. We chose exactly these methods because of their individual qualities:

The Normal Distribution Transform (NDT) is interesting because it would inherently avoid the *Correspondence Error* of point-based algorithms. Iterative Dual Correspondence (IDC) was chosen because of its historical importance and because it still is a very popular method. An interesting feature of Metric-based Iterative Closest Point (MbICP) is that it includes rotation in the distance metric of the general ICP algorithm. Finally, the Weighted Scan Matcher (WSM) is attractive because it explicitly models all possible error sources in the general point-based case, which apparently aids robustness significantly.

4.3.1 Normal Distribution Transform

Biber and Straßer [5] describe an algorithm using the *Function Optimization* approach discussed in the previous section. In order to be able to optimize this function analytically, the authors decided to approximate the probability distribution as a sum of Gaussians. The process of generating this probability distribution from the laser range scan is called the *Normal Distribution Transform*, or NDT, and is the main contribution in this paper.

The *Normal Distribution Transform* divides the area the scan covers into a grid. For each grid cell, it computes the mean and covariance matrix for all points within that cell. Basically, each grid cell knows how likely it is to measure any point inside itself. To counteract descritization effects, the authors choose to four grids instead of one. The extra grids are shifted by half a grid edge length to the right, down and both right and down, respectively. That means that each point falls into four grid cells, one from each grid.

This gives rise to a piecewise continuous and differentiable function. An example is shown in Figure 4.4. It can be seen that the laser range scan is well represented, but the descritization effects can still be noticed. Clear cutoff lines are present where the grid organization introduces discontinuities. However, the grid organization also allows to evaluate less Gaussians, which speeds up the method.

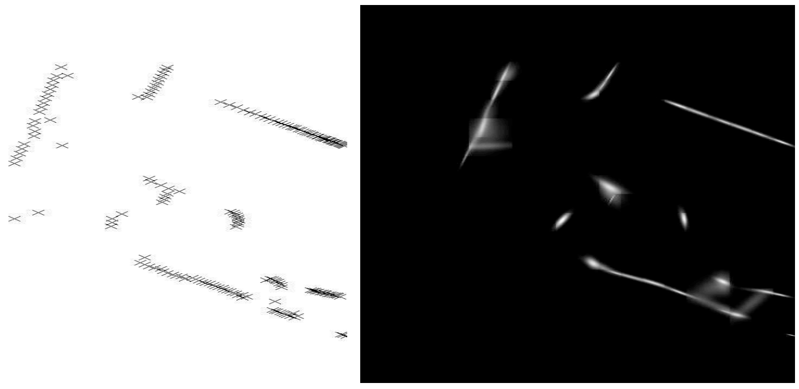


Figure 4.4: The effects of the *Normal Distribution Transform*. Left: The original laser range scan. Right: The resulting probability density. From [5].

The basic algorithm to generate the NDT is shown in Algorithm 4.3.

```
1  Let $l$ be the edge length for one cell;
2  Let $x$ be the minimum $x$-coordinate of points in the scan;
3  Let $y$ be the minimum $y$-coordinate of points in the scan;
4  while there are still points with an $x$-coordinate greater than $x$ do
5      while there are still points with an $y$-coordinate greater than $y$ do
6          Get all points in the rectangle defined by the points $(x, y), (x + l, y + l)$;
7          Compute the mean and covariance matrix given these points;
8          Store the mean and covariance matrix together with the rectangle for later
           lookup;
9          Advance $y$ by $\frac{l}{2}$;
10     end
11     Let $y$ be the minimum $y$-coordinate of points in the scan;
12     Advance $x$ by $\frac{l}{2}$;
13 end
```

Algorithm 4.3: The *Normal Distribution Transform*

Evaluating the match of a scan to another scan's NDT representation is nothing more than evaluating that probability distribution at all points of the new scan and summing up all results. For the complete scan matching process, the resulting function is optimized with Newton's algorithm, which requires the first and second derivatives. Given those derivatives, one step of the optimization will move the parameter estimate towards the minimum. For the calculation of first and second derivatives, as well as the specifics of Newton's algorithm, see [5] and references therein. For sufficiently smooth functions, only a few Newton steps are usually necessary to find the minimum, which makes this algorithm very efficient. Biber and Straßer claim to be able to process almost 100 scans per second on a regular desktop machine.

The main advance of this algorithm is that there is no need for explicit point correspondences as there is for point-correspondence methods. As discussed previously, it is very hard to find the right corresponding point, if it even exists. Therefore, this method should be more robust than point-correspondence based ones. Interestingly, Biber and Straßer show an example result generated by their algorithm which shows that it was very robust to environmental change such as opening or closing doors.

However, one main disadvantage remains: The numerical properties of the NDT are not as favorable as desired. It seems that a lot of extra effort must be spent on rejecting outliers, tuning the cell size, and implementing extensions to the basic Newton algorithm to achieve the reported efficiency. Such fine-tuning is partly situation-dependent and thus not acceptable for a general scan matching method.

4.3.2 Iterative Dual Correspondence (IDC)

Lu and Milios were two pioneers in the field of scan matching. Their paper from 1994 introduced one of the first efficient scan matching algorithms [33]. "Iterative Dual Correspondence" (IDC) tries to remedy some of the previously mentioned pitfalls of the classic ICP algorithm by first defining two different distance metrics for finding point correspondences and then solving for translation and rotation separately.

The two distance metrics used have two significantly different goals: The first describes the translation very well, the second can cope especially well with rotation. The first method is called *closest-point rule*, which essentially selects two points with the smallest Euclidean distance as pairs. The second method, called *matching-range-point rule*, pairs two scan points if their distance to the scan origin is roughly equal and their angles are at most some specified number (called B_ω in the figure) of degrees apart. This method is illustrated in Figure 4.5.

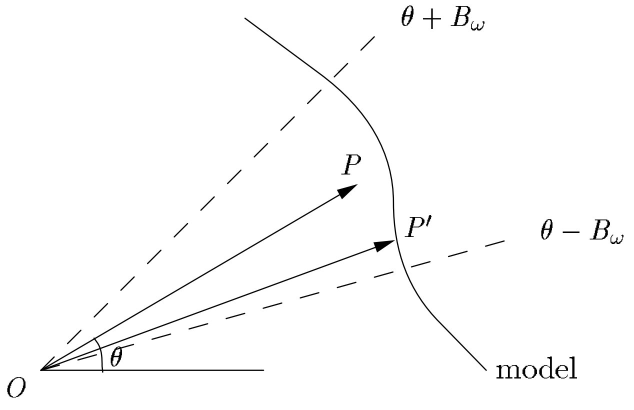


Figure 4.5: Lu and Milios' *matching-range-point rule* illustrated. B_ω is the maximum angle difference. From [33].

The basic assumption of the *matching-range-point rule* is that the translation has already been solved and finding the rotational component of the target transformation is the main concern. Since that is the case, comparing polar distances already gives a good indication of correspondence. The maximum angle difference allowed to make a correspondence will effectively limit the maximum rotational error the method will be able to correct. However, it will also prevent false correspondences between two unrelated pairs. Such an important parameter obviously needs tuning, but Lu and Milios' experimental results show that the method is rather insensitive to small variations in this parameter.

For both methods, a least-squares optimizer is used to find the complete transformation. The transformation is split into two parts, the translational part T_i, and the rotational part ω_i. The final result is assembled as the translation from the method using the first metric and the rotation from the method using the second. This is outlined in Algorithm 4.4. Note the subscripts of T and ω.

```
   Data: Laser Scans $A$ and $B$
   Result: A pose which describes the transformation from $A$ to $B$
1  for each point $a_i$ in $A$ do
2      find a correspondence $b'_i$ in $B$ according to the closest-point rule;
3      find a correspondence $b''_i$ in $B$ according to the matching-range-point rule;
4  end
5  Compute the least-squares solution $(T_1, \omega_1)$ from the corresponding pairs $(a_i, b'_i)$
   (from the closest-point rule);
6  Compute the least-squares solution $(T_2, \omega_2)$ from the corresponding pairs $(a_i, b''_i)$
   (from the matching-range-point rule);
7  Let $p = (T_1, \omega_2)$;
```

Algorithm 4.4: The Iterative Dual Correspondence Algorithm

This "dual" approach combines the two special separate strengths of both correspondence rules. Also, IDC tries to counteract the previously highlighted flaw of point-based methods, the *Correspondence Error*. Contrary to the classic ICP, IDC finds closest points on lines connecting two points in the first scan with both correspondence rules. This might not necessarily be an existing point in the previous scan, but a point on an approximated connection between two existing points. While the assumption of continuous and partially linear geometry in the environment is rather strong, it does solve the said problem at least to some extend. Such assumptions are usually violated in small indoor environments, for example in the case of table legs or open doors.

The IDC algorithm is still a very popular method. It is efficient and achieves reasonably accurate results. However, it has its weaknesses and the above illustration already highlights one of the first downsides. The computation of the rotation and translation is decoupled, which might result in a non-optimal result. Either, the method finds a local minimum or it might even terminate with an estimate that is not a minimum at all. The authors state that the *closest-point rule* is very weak in correcting rotational error and the *matching-range-point rule* is exceptionally weak at correcting translational error. That means the discarded translation and rotation might differ significantly from their counterparts which are used to construct the final transformation estimate. In such a situation, it is possible that both solutions approximate the correct transformation insufficiently. A coupled approach can be computationally more complex but probably generates better results.

Since our main area of interest is the high-resolution indoor environment, this algorithm is not adequate for our needs as shown later in the experimental section. However, it is included for completion and as a reference to today's de-facto standard for the other tested algorithms.

4.3.3 Metric-based Iterative Closest Point (MbICP)

Minguez et al. [36] introduce a novel distance metric to be used in a regular ICP setup, as described above. The main motivation behind MbICP is to couple the rotation and translation estimation process, which was the main flaw in the IDC algorithm. As described above, the Euclidean distance metric does not perform well in the ICP algorithm to solve rotational errors. For that reason, a distance metric was designed to address this issue.

This distance metric is given with respect to the transformation from one scan point to another in the other scan. The Euclidean distance metric would only utilize the translational component. MbICP's metric will also include the rotational component of the transformation.

The *norm* of a transformation $q = (x, y, \omega)$ is

$$||q|| = \sqrt{x^2 + y^2 + L^2\omega^2}$$

and the distance metric is then defined as (using the transformation defined in equation (4.1))

$$d(a_i, b_i) = \min\{||q|| \text{ } \textit{such that } q(a_i) = b_i\} \tag{4.2}$$

L is a parameter which describes how much the rotational component ω should be taken into account. Contrary to intuition, it will be clear later that when $L \to \infty$, this new distance metric tends towards the Euclidean metric. This is because of the minimization of the norm defined above. See Minguez et al. [36] for details of the derivation.

Unfortunately, this particular form from (4.2) is very hard to compute since it implies a search over all possible transformations which transform a_i to b_i. In the paper, the authors already provide an approximation to the above distance function.

$$d^*(a_i, b_i) = \sqrt{\delta_x^2 + \delta_y^2 + \frac{(\delta_x a_{iy} - \delta_y a_{ix})^2}{a_{iy}^2 + a_{iy}^2 + L^2}}$$

with $\delta_x = b_{ix} - a_{ix}$ and similarly for δ_y. Now it is obvious that when $L \to \infty$, d^* tends towards the Euclidean distance. A comparison to Euclidean distance is shown in Figure 4.6.

Figure 4.7 shows how this metric captures the correct correspondences under rotation quite well. In general, this method converges faster and more reliably than the previously discussed IDC algorithm [36].

However, this algorithm is still prone to produce incorrect correspondences. In general, there is no mechanism to interpolate between existing scan points or solve the *Correspondence Error* in any other way. This means that this method completely relies on the distance metric and very dense scans to estimate the transformation between the two scans sufficiently well. Thus, it is very susceptible to local minima when the scans are not dense enough or

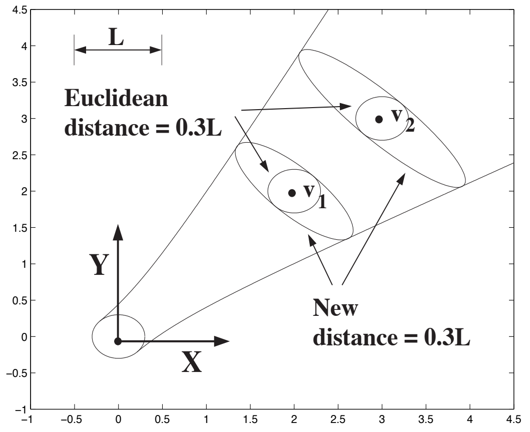


Figure 4.6: The contours of the d^* distance in comparison to Euclidean distance. From [36].

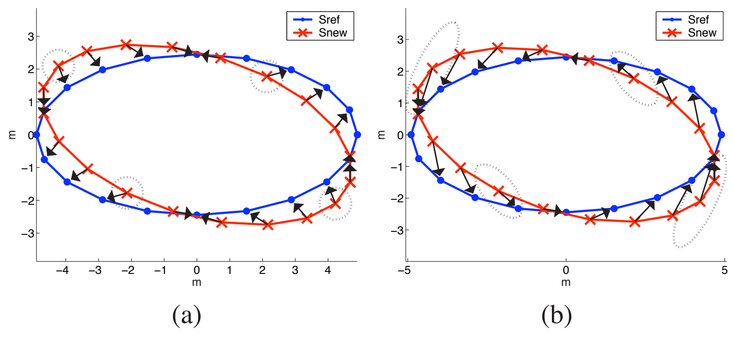


Figure 4.7: a) Correspondences computed using Euclidean distance between to identical but rotated simulated scans. b) Correspondences computed with the metric used in MbICP. The dashed ellipses show contours for each distance metric. From [36].

the environment is too irregular, which would often be the case in unstructured environments. Some optimizations, like detection and rejection of outliers, can improve robustness and convergence. Our trials show that these optimizations are in fact necessary for this algorithm to be of any use at all.

4.3.4 Weighted Scan Matching

The *Weighted Scan Matcher* (WSM) by Pfister et al. [44] can be considered a hybrid method. It combines the basic ICP idea with some properties of the *Function Optimization* approach. Instead of assuming perfect point correspondences, the WSM algorithm weights the influence of each point pair by the uncertainty of their correspondence. The background of the WSM algorithm is firmly grounded in probabilistic models and explicitly includes all possible error sources in the calculations.

The general error associated with point correspondence scan matching is broken down into three parts: Correspondence Error, Measurement Error, and Bias Error. We already introduced the notion of Correspondence Error above. The error between two points a_i and b_i from two different scans with a transformation (R, t) (R being the rotation matrix), the error can be written as

$$\epsilon = a_i - R \cdot b_i - t \tag{4.3}$$

This is easily verified as this equation only states that the general error when matching two points to each other is the difference in coordinates after the transformation of the second point.

Assuming Gaussian additive noise for the measurement process, a_i and b_i are written as

$$a_i = r_{a,i} + \delta r_{a,i} + d_{a,i} \tag{4.4}$$

similarly for b_i. In the previous equation, r_i is the actual distance from the sensor to the object, $\delta r_{a,i}$ is the noise in the measurement process itself, and $d_{a,i}$ is a bias term which differs between measurement methods.

Finally, substituting (4.4) into (4.3), we get

$$\epsilon = \underbrace{(r_{a,i} - R \cdot r_{b,i} - t)}_{CorrespondenceError} + \underbrace{(\delta r_{a,i} - R \cdot \delta r_{b,i})}_{MeasurementError} + \underbrace{(d_{a,i} - R \cdot d_{b,i})}_{Bias} \tag{4.5}$$

From this error model, WSM draws its uncertainty model. This model in turn is used in a *Maximum Likelihood* fashion, which is in fact a *Function Optimization* problem. A log-likelihood function is defined which has a minimum when pose estimate q correctly aligns both scans as uncertainty will be zero in that case. This function is then optimized analytically.

An example of the correspondence uncertainty is shown in Figure 4.8. It shows how well this probabilistic approach allows for errors in correspondences, something previous methods did not consider at all. Such explicit modeling makes the WSM algorithm as robust to *Correspondence Error* as any point-correspondence method can be. As numerous experiments in [44] show, this algorithm achieves great accuracy. Even after matching a sequence of scans one by one, the accumulated error is very small. Also, the results presented there show that this algorithm performs significantly better than the IDC algorithm described above.

In addition to the estimated pose difference between two scans, this algorithm also reports a covariance matrix. This is a key advantage over all other scan matching techniques discussed in this section. Not only can we compute which pose is the most likely one, but we

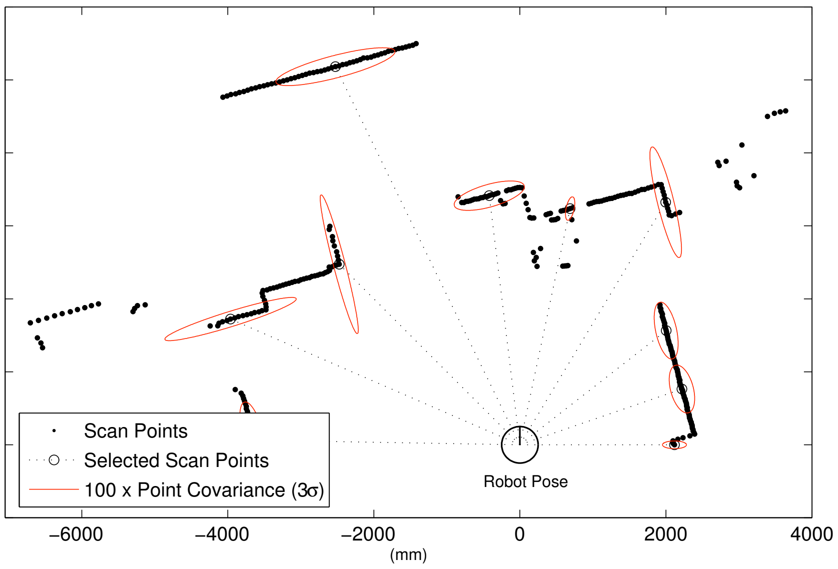


Figure 4.8: Correspondence uncertainty in the WSM algorithm, from [44].

also know how certain we are of this estimate. This is especially useful for situations where the output of scan matching is used for SLAM. More detailed uncertainty information can only help methods like FastSLAM (described in Section 3.4.1). The benefit is very visible in Figure 4.9. Not only does the WSM algorithm on average converge to an estimate much closer to the real pose, but it does so very consistently and well within its own uncertainty estimate.

4.4 Conclusion

From the respective authors' experiments and results, we can already see that the Weighted Scan Matcher (WSM) algorithm by Pfister et al. [44] will be a very suitable candidate for use in the Robocup Rescue Virtual Robot league. The robots encounter smalls spaces in indoor environments in that setting, so the presumably high accuracy of WSM will be of great benefit.

We received MATLAB implementations for all presented algorithms and were thus able to verify the actual quality of each algorithm. Our detailed experiments conducted with these implementations are described in Section 7.1.

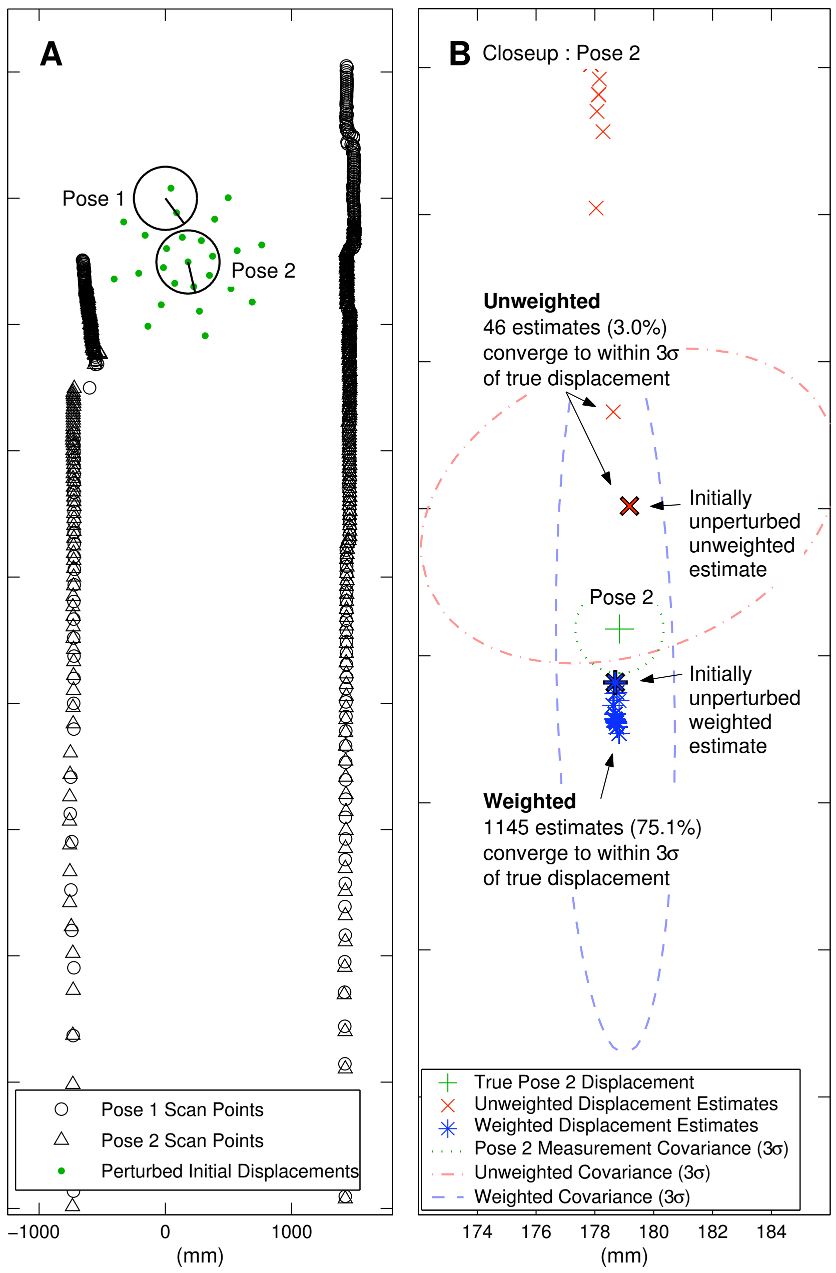


Figure 4.9: Visualization of the reported covariance matrix and comparison to the IDC algorithm (referred to as "Unweighted" in the legend). Notice how the blue ellipse representing WSM's final estimated covariance matrix is mainly uncertain along the direction of the corridor. From [44].

Chapter 5

Our Approach: ManifoldSLAM

5.1 Introduction

Our main aim for this project was to successfully compete and achieve a high ranking in the RoboCup 2006 championships held in Bremen, specifically the Virtual Robots League of RoboCup Rescue. As described in Chapter 2, this league focuses on two aspects of performance for scoring: the ability to explore and map an urban search and rescue scene, and the number of detected victims as well as the amount of detailed information that is reported per victim.

In the two earlier chapters, the current state of the art in SLAM and scan matching research has been summarized. In addition, various dimensions enabling the comparison and evaluation of different approaches have been mentioned. This chapter describes a complete SLAM solution to the Robocup 2006 urban search and rescue challenge based on that evaluation of different approaches. The nature of the challenge gives higher importance to certain aspects of the approaches over others. Therefore, these aspects of interest will be the focus of the approach described in this chapter.

It is important to note that hardly any SLAM implementation is ready for immediate use. Although some implementations can be found online, their efficient usage depends to a large extent on a detailed understanding of their configuration. Thus, in order to arrive at a solution that addresses all the specific aspects of the challenge, a considerable amount of time must be spent on fine-tuning parameters and optimizing modifications or extensions. In addition, since all available implementations come without support or guarantees, it should not come as a surprise that debugging would be very time consuming.

We reviewed all the current state-of-the-art SLAM algorithms discussed in Chapter 3. In the few cases where source code was available, some basic tests were conducted. However, no single existing SLAM algorithm turned out to be adequate for our case. Therefore, we designed a new method, drawing inspiration from the individual advantages and trying to avoid the disadvantages of the reviewed algorithms. Our new hybrid approach is based on a map that employs a graph organization, which is similar to Howard's Manifold data structure [27] and the organization used in Atlas [7]: *ManifoldSLAM*.

5.2 Design Essentials

Following Howard's initial descriptions [27, 26], we have designed a sophisticated data structure that meets all map-related requirements. As in many other hybrid SLAM approaches [51, 60, 7], we aim to cover the strengths of both topological and metric representations. Similarly to [31] and [60], the representation incorporates a *layered* data structure with a topological organization at the global level and detailed metric maps at the local level.

Globally, the Manifold relies on a graph structure that grows with the amount of explored area. Nodes are added to the graph to represent local properties of newly explored areas. Links represent traversable paths from one node to the next. One key design aspect is that the Manifold data structure does *not* assume a planar environment. When a robot navigates in a circle and this fact is represented by different parts of the graph that start to overlap, no correspondences between the overlapping sections are assumed and a spiral structure emerges. This design was inspired by Howard [27] and is key to several features exhibited by the Manifold. The most important feature that follows from this design is that it enables the Manifold to map an environment with multiple overlapping levels.

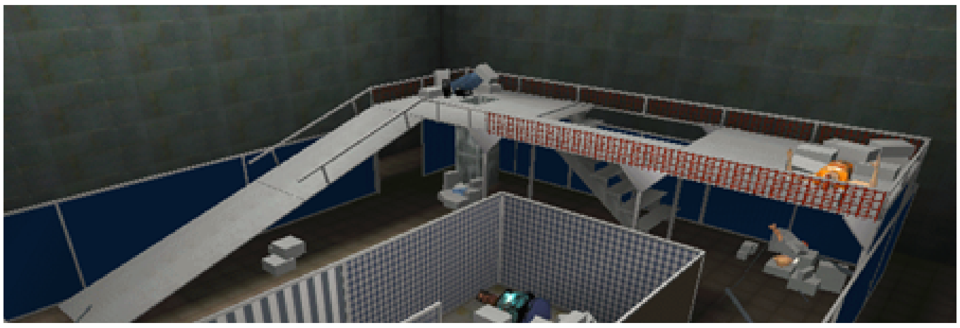


Figure 5.1: Multiple floors in the Orange arena.

While other map structures suffer from the *cross-over problem* when faced with multiple overlapping levels in an environment, the Manifold handles this gracefully. For example, consider a two story office-like building like the Orange Arena depicted in Figure 5.1 and a robot which first explores the lower level and subsequently the upper level. Figure 5.2(a) outlines the typical problem that planar maps suffer from as soon as the robot navigates over a location in the upper level which is directly above a previously mapped other location on the lower level. As planar map representations fail to distinguish between the different floors, it will assume that the observations should correspond to the same place in the physical environment. It is not hard to imagine how this *cross-over problem* leads to distorted, inconsistent or otherwise degraded maps when it is not accounted for. Figure 5.2(b) shows how the Manifold handles cross-over gracefully.

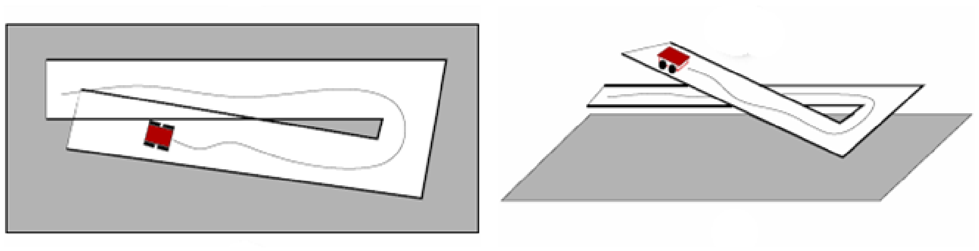

(a) (b)

Figure 5.2: Cross-over. Images courtesy of [27]

Another aspect of the Manifold design that makes it so successful is that the same landmark may be stored multiple times on different nodes. Consider a robot that traverses an environment and observes the same landmark twice. Also assume that the second detection occurs on a different node from the first. Upon the second detection, a graph structure will have been built up that depicts the area explored so far. It makes sense now to conclude that the current node should correspond to the one that holds the previous landmark observation and that the *closing of a loop* is in order. Essentially this would mean to add a link to the graph that connects these two nodes in order to represent the landmark identification. However, Chapter 3 showed that data association is one of the toughest challenges to deal with and is prone to false positives. In case of an incorrect correspondence identification, the erroneous addition of this link will degrade the graph's consistency significantly and thereby might have severe negative impact on planning, navigation or any other algorithms that build on the Manifold. Thus, in order to preserve the Manifold's consistency, great care must be taken when inferring links between nodes which were *not actually traversed by the robot*. As the Manifold does not assume a planar environment, the graph structure is free to grow without requiring loops to be closed prematurely. Therefore, as multiple nodes can hold observations of potentially the same landmark, loop-closing can be delayed until more certainty is acquired through additional observations.

The landmarks that are stored on different nodes also play an important role when the Manifold is shared between multiple agents. Different agents possibly enter the environment at different locations or at different points in time. The Manifold's data structure easily facilitates this by having each agent start on its own new node. The particular robot will develop this node into a disconnected component of the graph, also called an *island*.

Initially this will result in as many islands as there are agents. This exactly represents the fact that the relative positioning of the robots is unknown at that time. Hence, the Manifold's consistency is preserved. Certain events can then trigger the merging of two islands into one. A typical example for such an event is the detection of the same landmark by two different robots. At that moment, the relative positioning of the two robots can be determined and subsequently their islands can be merged appropriately. When two robots have been exploring the same parts of the environment the island-merging process will typically result in the addition of multiple links between nodes of both islands. The set of landmarks observed by both robots can be used to guide this process. Island-merging bears a clear resemblance to loop-closing and has the same major impact on the graph structure. Note that in order to avoid jeopardizing the Manifold's consistency, also island-merging can be delayed until a certain level of certainty is acquired.

All localization and mapping related functionality relies on scan matching. The geometric details stored at the node-level are in fact the range scans acquired from the laser range scanner. The details are discussed in Section 5.3, but the consequence that we wish to mention here is that using the scan matcher we can compare the information that is stored on nodes. Subsequently, the covariance matrix that is returned by the scan matcher is stored on the link between the nodes. This way the Manifold embeds uncertainty information throughout the graph. As discussed later in Section 5.4, this locally available uncertainty information is used extensively during localization and mapping. Note also that because of the scan matcher, localization is not limited to maintaining the current node. By comparing the current observations to the information stored on the nodes using the scan matcher a full pose estimate with associated uncertainty can be maintained.

By nature, the Manifold's graph structure greatly facilitates navigation algorithms. The implementation of path planning and search algorithms on the Manifold are straightforward. Shortest paths can be obtained using e.g. Dijkstra's algorithm [14], but also heuristic searching is supported. The uncertainty information that is stored on the links, or confidence values that can be derived from them provide, a great heuristic in order to find secure paths. To translate path plans into motion plans, the nodes provide the necessary information about the local geometric properties of the environment. Thus, it is possible to construct, analyze and evaluate path plans up to the obstacle avoidance details already beforehand. Therefore, the Manifold data structure accommodates and facilitates the development of advanced navigation, exploration or behavior.

5.3 Data Structure Details

In this section the Manifold data structure is discussed in detail. The three core concepts that make up the Manifold will be introduced: *patches*, *relations* and *local sub-maps* [27].

Patches form the nodes of the overall graph structure while relations form the links. A local sub-map is always acquired from the perspective of a single patch and will contain a local subset of nearby and *well-fitted* patches.

We will denote a patch with π and a link with ϕ. The Manifold can then formally be defined as the set of all patches and relations together: $\Pi = \{\{\pi\}, \{\phi\}\}$. Local sub-maps will be denoted with Π^*.

It should be noted that although the Manifold will be described for specific use with laser range scans, the concept could be generalized to other types of sensor measurements effortlessly. Incidentally, the implementation used at RoboCup 2006 integrates victim observations and other information in a single patch.

5.3.1 Patches

Patches are the fundamental building blocks of Manifolds. They form the nodes of the graph structure, where each patch is of finite extent and defines a local planar coordinate system. In effect, the patches discretize the full map into small, possibly overlapping, local, metric maps. Let π denote a single patch, then:

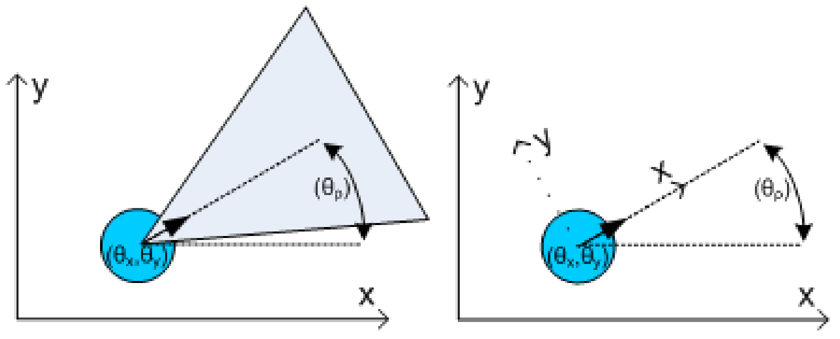
$$\pi = (\theta, s) : \theta = (x, y, \rho)\,, s = \{(\alpha, d)^n\}$$

A single patch stores a single laser range scan observation s together with the estimated global robot pose θ from where this scan was taken. A single scan as returned by the laser range sensor will consist of a set of n angle-distance pairs (α, d), which are easily translated into local (x, y) coordinates relative to the patch origin. Thus, the pose denotes as the origin of the local coordinate frame and it provides the transformation from the global coordinate frame to the local measurement frame and vice versa. Let r_{π_a} be a robot pose estimate relative to patch π_a, then $\oplus$ is defined as the coordinate transformation operator that projects this pose estimate on the global frame and $\ominus$ as the inverse operator that projects it back to a patch-relative pose estimate [27]:

$$r_{global} = r_{\pi_a} \oplus \theta_a$$

$$r_{\pi_b} = r_{\pi_a} \oplus \theta_a \ominus \theta_b$$

Note that these transformations in fact decouple the local measurements from the positioning of the patches on the global frame. So during loop-closing and island-merging patch positions can be re-estimated without having to update the measurement values. In essence, the patches institute the lower layer of the Manifold as they hold the geometric details of the environment.



(a) Patch as pose $\theta_{(x,y,\rho)}$ with scan (b) Patch as local coordinate system

Figure 5.3: Patches.

5.3.2 Relations

Relations form the links in the graph and represent the navigability between patches. They are mostly constructed between consecutive robot poses and sometimes additional relations are inferred during loop-closing and island-merging. Every relation stores a Gaussian probability distribution over the estimated pose-difference $\Delta\theta_{ab}$ between two related patches π_a and π_b. This Gaussian distribution with mean $\Delta\theta_{ab}$ and covariance matrix Σ_{ab} is estimated from the set of *pair-wise point-correspondences* between the two patches. Typically, the parameters of this Gaussian are estimated by a scan matcher, see also Chapter 4. Note that the common convention in robotics it to decompose a pose difference like $\Delta\theta$ into a rotation $\Delta\theta_\rho$ followed by a translation $\Delta\theta_\tau$, each with their own associated (Gaussian) uncertainty. The covariance stored on relations should therefore be interpreted accordingly, as an uncertainty in a consecutive rotation and translation. Therefore the resulting probability distribution typically looks like a bent ellipse, see Figure 5.4.

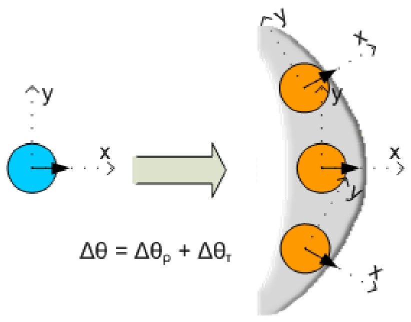


Figure 5.4: Motion uncertainty shaped as a bent ellipse.

Subsequently, let ϕ_{ab} denote the relation between two patches π_a and π_b, then:

$$\phi_{ab} = (\pi_a, \pi_b, \Delta\theta_{ab}, \Sigma_{ab})$$

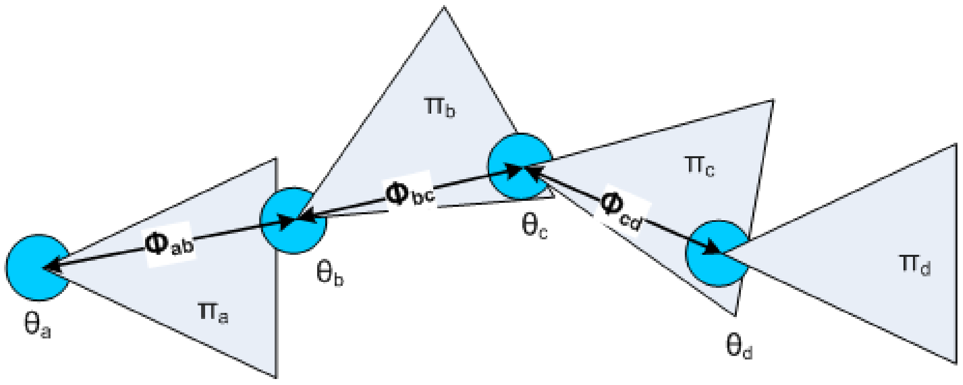


Figure 5.5: Relations.

The uncertainty described by the covariance matrix could also be interpreted as a *confidence measure* that describes how well the two patches fit together. A better fit means that correspondences between the two range scans were found with higher associated certainty and thus that the relative positioning of the patches could be determined with higher confidence. The covariance matrix gives a full description of the uncertainty, but for some purposes a single confidence measure is more convenient. Good examples of single confidence values that can be derived from a covariance matrix are the determinant $det(\Sigma)$ or the trace $trace(\Sigma)$. The relations lend themselves very well for use in path-planning and search algorithms. In addition, heuristic algorithms can take advantage of the uncertainty information stored on the links.

5.3.3 Local Sub-Maps

Local sub-maps are defined from the perspective of a single patch [27]. They aggregate all nearby and *well-fitted* patches. The primary purpose of local sub-maps is to aid SLAM. Both mapping and localization on the Manifold heavily rely on scan matching. Traditionally scan matching is about matching pairs of scans. However, we have extended the scan matcher of our choice (see also section 5.4) so that it is also capable of matching a scan to a sub-map.

The concept is illustrated in Figure 5.6 where a robot zig-zags through a corridor. In the example the robot is configured with a laser range scanner that has a field of view of 180 degrees, which is indicated using different shades of gray. Only the time steps at which the range scan is used to create a patch are depicted, so the robot poses and range scans in the illustrations can also be seen as the corresponding patches in the Manifold. At time

t_1, a patch is created as the robot enters the corridor. Then at time t_2, the range scan is compared to the one stored at time t_1 and another patch is created. At time t_3, the robot could proceed in a similar way by comparing the new scan against the scan previously stored at the patch of time t_2. However, in the rightmost image, the new scan is not just matched against the previous patch, but against the *sub-map* of the previous patch. Assume that this sub-map will at least include the patch of time t_1. Then it is not hard to see how a much more complete reference frame is provided to match the new scan to, this difference is also marked in red in the rightmost image.

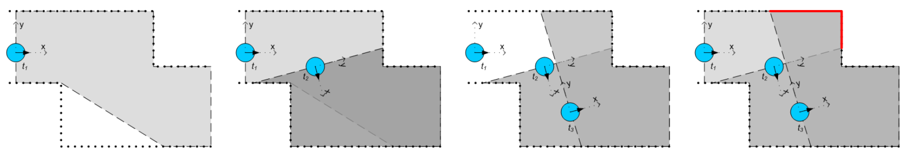

(a) t_1 (b) t_2 (c) t_3 regular (d) t_3 using submap

Figure 5.6: The effect of using local sub-maps.

The local sub-map of a particular patch is obtained by a breadth-first search through the Manifold's graph structure that starts from the particular patch for which the local sub-map is acquired. The frontier of this breadth-first search expands on patches that are well-fitted and halts at ill-fitted patches. The quality of the fit between two neighboring patches is determined by analyzing the covariance matrices that are stored on the relations. For convenience, we first compute the *Fisher information matrix* from a particular covariance matrix. Following standard statistics literature, we define the Fisher information matrix for a multivariate Gaussian distribution as the expected outcome of the second derivative of the log-likelihood:

$$I_{ab}\left(\Delta\theta_{ab},\Sigma_{ab}\right) = E\left[\nabla^2 L\left(\Delta\theta_{ab},\Sigma_{ab}\right)\right] : L = \ln\left(P\left(\Delta\theta_{ab},\Sigma_{ab}\right)\right)$$

For a more elaborate treatment of Fisher information matrixes refer to [13], and see [20] for a discussion of Fisher information in the context of SLAM. Here we limit ourselves to postulating that for Gaussians, the Fisher information matrix reduces to just the inverse of the covariance matrix:

$$I_{ab}\left(\Delta\theta_{ab},\Sigma_{ab}\right) = E\left[\nabla^2 L\left(\Delta\theta_{ab},\Sigma_{ab}\right)\right] = \Sigma_{ab}^{-1}$$

Having an equation to compute the Fisher information on the links now enables us to exploit the *additive nature* of information. For patches that are direct neighbors of the starting patch, the fitting quality can be determined by analyzing the Fisher information of the single link involved. For patches that are not direct neighbors, the analysis can now be based on the *sum* of all Fisher information matrices that lie on the search-expansion path between the patches. An individual patch is deemed well-fitted by applying a threshold on a confidence interval on every pose-variable. The confidence interval of a particular pose-variable is given by:

$$\sigma_i = c\sqrt{\left(I_{ab}\left(\Delta\theta_{ab}, \Sigma_{ab}\right)^{-1}\right)_{ii}}$$

We set c to 1.96 for a 95% confidence interval. The i index below the σ refers to the pose-variable that the threshold is applied to. Correspondingly, the ii index below the inverted Fisher information matrix refers to the on-diagonal element corresponding to the particular variable. We had two thresholds in our configuration that could be tuned: $\tau_{threshold}$ that was applied on the translational variables $\Delta\theta_x$ and $\Delta\theta_y$ and $\rho_{threshold}$ that was applied on the rotational variable $\Delta\theta_\rho$. We considered two patches well-fitted when the confidence value was below the corresponding threshold on all three variables. This is similar to the reasoning in [27]. See Algorithm 5.1 for the $GetSubmap\left(\pi_s, \tau_{threshold}, \rho_{threshold}\right)$ procedure in pseudo-code.

```
Data: the start patch $\pi_s$ for which to acquire the sub-map, the translational
      threshold $\tau_{threshold}$ and the rotational threshold $\rho_{threshold}$
Result: the sub-map $\Pi^*$ composed of all well-fitted nearby patches transformed to
        the coordinate system of $\pi_s$
1  $\Pi^* = \{\}$;
2  $\Pi_{frontier}$ = set of patches neighboring $\pi_s$;
3  while $\Pi_{frontier}$ not empty do
4      $\Pi_{nextfrontier} = \{\}$;
5      for each patch $\pi_t \in \Pi_{frontier}$ do
6          $Links_{st}$ = all the links on the search-expansion path from $\pi_s$ to $\pi_t$;
7          $I_{st} = I\left(Links_{st}\right) = \sum\left(I\left(link_{st}\right)\right)$ for all $link_{st} \in Links_{st}$;
8          $x_{ok} = \left(1.96\sqrt{\left(I_{st}^{-1}\right)_{xx}} < \tau_{threshold}\right)$;
9          $y_{ok} = \left(1.96\sqrt{\left(I_{st}^{-1}\right)_{yy}} < \tau_{threshold}\right)$;
10         $\rho_{ok} = \left(1.96\sqrt{\left(I_{st}^{-1}\right)_{\rho\rho}} < \rho_{threshold}\right)$;
11         $wellfitted = x_{ok}$ and $y_{ok}$ and $\rho_{ok}$;
12         if $wellfitted$ then
13             $\pi'_t = \pi_t \oplus \theta_t \ominus \theta_s$;
14             $\Pi^* = \Pi^* \cup \pi'_t$;
15             $\Pi_{next}$ = patches $\pi$ neighboring $\pi_t$ for which $\pi \notin \Pi^*$;
16             $\Pi_{nextfrontier} = \Pi_{nextfrontier} \cup \Pi_{next}$;
17         end
18     end
19     $\Pi_{frontier} = \Pi_{nextfrontier}$;
20 end
```

Algorithm 5.1: $GetSubmap\left(\pi_s, \tau_{threshold}, \rho_{threshold}\right)$

5.4 Localization and Mapping

5.4.1 Weighted Scan Matching

Scan matching is the fundamental algorithm to most of the SLAM-related functionality that is implemented on the Manifold. As indicated in Chapter 4 on scan matching and as demonstrated in the extensive experiment discussed in Section 7.1, the Weighted Scan Matcher by Pfister et al. [44] is the logical choice. Amongst the scan matchers evaluated, it has superior accuracy without sacrificing performance.

The core algorithm that matches scan-pairs is outlined in Section 4.3.4 and the original paper by Pfister et al. [44]. In addition, we implemented an extension that also enabled us to match a scan against a local sub-map using the same algorithm. Local sub-maps are explained in detail in Section 5.3.3. In essence, we have modified the scan matcher so that it matches arbitrary *point-clouds*, where each point-cloud can have any number of points. Recall that Weighted Scan Matching belongs to the family of Iterative Closest Point algorithms, so after correlating points between the two point-clouds the maximum likelihood estimation can proceed as usual. The maximum number of point correlations is naturally constrained to the number of points in the smallest point-cloud. Note that along the same lines a similar extension could be implemented where also local sub-maps can be compared to each other. For our approach though, this was not necessary.

One interesting consequence of having a scan matcher that matches point-clouds is that it no longer assumes range scans to be of a certain resolution or to have a certain field of view. In the context of a multi-agent system this yields an interesting benefit as this allows different robots to have different configurations for the range scanners. The benefit lies in the fact that the configuration of the range scanner can be optimized with respect to the robot's intended strategy or processing capabilities. Also in the simulated environment of the Rescue Virtual Robots League this is of interest. Some robots were spawned from a three year old laptop while other robots were spawned from a high-end workstation. When desired, we could configure the robots that were spawned from less capable hardware to work with lower scan resolutions. This would have no repercussions on the shared Manifold as all functionalities like localization, mapping, loop-closing and island-merging rely on the scan matcher.

5.4.2 Incremental Localization and Mapping

Incremental localization and mapping refers to the main process that is executed online by every robot as it explores the environment. Formally, localization is about maintaining the tuple (θ_t, π_t) where the subscript t refers to the current time index and θ_t is the current robot pose with respect to the current patch π_t. Mapping refers to the process that maintains the

Manifold Π over time, where Π_t denotes the Manifold as constructed up to time step t.

Localization and mapping is an incremental process that is triggered whenever a new scan s_t is obtained from the laser range scanner. Initially, the robot is not localized and the Manifold contains no patches. Therefore, localization is initialized with an arbitrary starting pose θ_0, for example at the global origin $(0, 0)$ with rotation 0. The Manifold is then initialized by immediately transforming the first incoming range scan s_0 into a patch π_0 which is positioned at this initial pose estimate.

From then on, incoming laser range scans are compared with the current patch using the scan matcher. Note that we actually compare against the *local sub-map* of the current patch, but as there is only one patch to begin with, the sub-map is initially reduced to just that patch. The scan matcher will return a Gaussian probability distribution $N(\Delta\theta_t, \Sigma_t)$ over the disposition estimate $\Delta\theta_t$. Localization then proceeds by first updating the current robot's pose estimate $\theta_{t+1} = \theta_t + \Delta\theta_t$. The patch π_{t+1} where the robot will be localized to is set to be the patch in the sub-map that has its origin nearest to the estimated projected pose. Note that if π_{t+1} differs from π_t the pose estimate needs to be transformed accordingly $\theta_{t+1} = \theta_{t+1} \oplus \pi_t \ominus \pi_{t+1}$. Some buffer could be put in place to avoid oscillations on equal-distance points, but note that due to the concept of local sub-maps the decision of which of the nearest patches to localize to is usually not important.

Mapping is more involved and is based on the *quality of fit*. To avoid a lot of redundant data being stored on the patches, the Manifold is only extended with new patches when this quality of fit drops below certain thresholds. The idea behind basing this decision on the quality of fit is straightforward. In the case of a bad fit, the scan matcher did not have sufficient reference data in the local sub-map to fit the new range scan with more certainty. The new range scan apparently contains details not yet covered by the Manifold and hence it is desired to add a new patch with this range scan. To asses the quality of fit of a new range scan, the same notion of being *well-fitted* is used as during the construction of local sub-maps (refer to Section 5.3.3). Recall that we already obtained the covariance matrix Σ_t of the Gaussian distribution that was estimated by the scan matcher. The Fisher information matrix I_t is computed by taking the inverse of this covariance matrix $I_t = \Sigma_t^{-1}$ and we compute the 95% confidence interval for every pose variable as follows:

$$\sigma_i = 1.96\sqrt{\left(I_t^{-1}\right)_{ii}}$$

The details behind this equation were already discussed in Section 5.3.3 and are assumed familiar here. The Manifold is extended when the confidence value of *any* pose variable drops below the corresponding threshold. For the translational variables we use a threshold τ_{ext} and for the rotational variable we use ρ_{ext}. Note that the thresholds that determine when to extend the Manifold are usually much stricter than those that determine which patches to include in the local sub-map. If the algorithm decides to extend the Manifold, a new patch π_* is added at the estimated current robot pose and at the same time a new relation ϕ_* is inserted that connects the newly added patch to the patch the robot is currently localized to. We store the parameters of the Gaussian distribution on this relation and we reset the localization to the origin of the newly added patch.

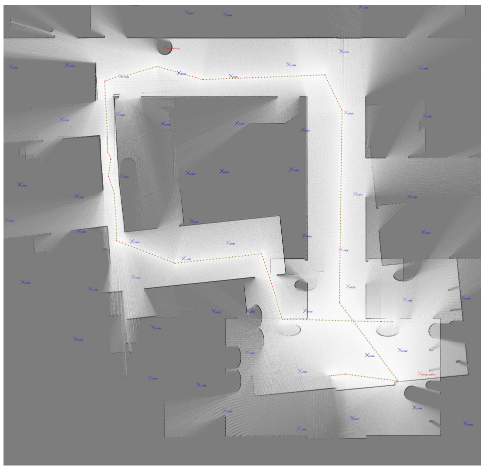
```
**Data**: the current Manifold $\Pi_t$ and localization estimate $(\theta_t, \pi_t)$, the newly obtained
       range scan $s_t$ and the extension thresholds: $\tau_{ext}$ for translations and $\rho_{ext}$ for
       rotations
**Result**: the updated Manifold $\Pi_{t+1}$ and localization estimate $(\theta_{t+1}, \pi_{t+1})$
1  $\Pi^* = GetSubMap\,(\pi_t)$;
2  $P\,(\Delta\theta_t, \Sigma_t) = WeightedScanMatcher\,(\Pi^*, s_t)$;
3  $I_t = \Sigma_t^{-1}$;
4  $x_{exceeded} = \left(1.96\sqrt{\left(I_t^{-1}\right)_{xx}} > \tau_{ext}\right)$;
5  $y_{exceeded} = \left(1.96\sqrt{\left(I_t^{-1}\right)_{yy}} > \tau_{ext}\right)$;
6  $\rho_{exceeded} = \left(1.96\sqrt{\left(I_t^{-1}\right)_{\rho\rho}} > \rho_{ext}\right)$;
7  $extend = x_{exceeded}$ or $y_{exceeded}$ or $\rho_{exceeded}$;
8  **if** $extend$ **then**
9      $\theta_* = \theta_t + \Delta\theta_t$;
10     $\pi_* = (\theta_*, s_t)$;
11     $\phi_* = (\pi_t, \pi_*, \Delta\theta_t, \Sigma_t)$;
12     $\Pi_{t+1} = \Pi_t \cup \{\pi_*, \phi_*\}$;
13     $\theta_{t+1} = (0, 0, 0)$;
14     $\pi_{t+1} = \pi_*$;
15 **end**
16 **else**
17     $\Pi_{t+1} = \Pi_t$;
18     $\theta_{t+1} = \theta_t + \Delta\theta_t$;
19     $\pi_{t+1} =$ nearest $\pi$ of $\pi \in \Pi^*$;
20 **end**
```

Algorithm 5.2: Incremental localization and mapping

When using just incremental localization and mapping the graph structure will actually grow into a tree. Whenever new areas are explored trails of patches will be formed on the Manifold. Branches are formed when a robot backtracks, is localized to non-leaf patches, and then chooses a new direction. As discussed in Section 3.2, due to sensor and actuator noise every patch will have some uncertainty in the estimate of its position and orientation. And as the Manifold is developed incrementally the error due to this uncertainty will propagate to future pose estimates. This implies that the error will have increasingly degrading effects on the Manifold's accuracy. In the following section the traditional and explicit means to address this issue in graph-based SLAM will be discussed: *Loop-closing*.

5.4.3 Loop Closing

Loop-closing refers to a family of algorithms related to topological SLAM that aim to identify and resolve accumulated errors in the map. Metric SLAM methods solve this problem implicitly. As the name suggests, the piece of information that triggers loop-closing is the detection that a loop has been traversed in the environment. This detection is usually the result of observing a particular landmark again. Due to error accumulation, the two estimates are not likely to project to the same global coordinate. Even worse, as the error can build up indefinitely, their projections could be very far apart. By identifying a previously observed landmark, this error is now uncovered and loop-closing can start to resolve it.


(a) before loop closure.

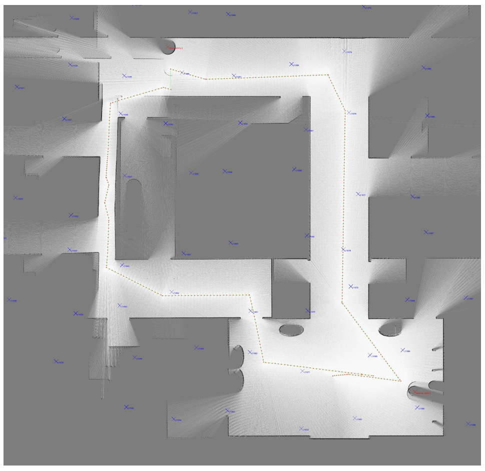

(b) after loop closure.

Figure 5.7: Loop-closing. The robot starts at the bottom right and moves up. Then the robot turns left several times until it returns in the bottom right and observes a particular landmark again and detects the loop.

In terms of the Manifold, the detection of an already known landmark will inform the loop-closing algorithm that two distinct patches have been found to correspond to the same part of the environment in reality. The accumulated error ϵ is then set to be the difference between the globally projected position estimates of these two observations: $\epsilon = (\lambda_1 \oplus \pi_{\lambda_1}) - (\lambda_2 \oplus \pi_{\lambda_2})$. The different position estimates of landmark λ are denoted with λ_i. π_{λ_i} refers to the patch from where the i-th landmark observation was made.

In our approach, loop-closing starts by relating the two patches π_{λ_1} and π_{λ_2}. For this purpose, they are overlaid on each other based on the landmark positions, i.e. π'_{λ_2} is π_{λ_2} translated by ϵ, and then π'_{λ_2} is compared against the local sub-map $\Pi^*_{\lambda_1}$ of π_{λ_1} using the scan matcher. Subsequently a relation $\phi_* = \left(\pi_{\lambda_1}, \pi'_{\lambda_2}, \Delta\theta, \Sigma\right)$ is inserted into the Manifold that connects the two patches and stores the Gaussian distribution $N(\Delta\theta, \Sigma)$ that is returned by the scan matcher. Note that because we overlaid the patches based on the landmark estimates, we *know* that the newly inserted relation has *zero accumulated error*. What remains now is to use this knowledge in order to revise the Manifold and improve its accuracy.

```
   Data: the current Manifold $\Pi_t$, the two landmark observations $\lambda_1$ and $\lambda_2$ together
         with their respective patches $\pi_{\lambda_1}$ and $\pi_{\lambda_2}$
   Result: the updated Manifold $\Pi_{t+1}$
1  $\epsilon = (\lambda_1 \oplus \pi_{\lambda_1}) - (\lambda_2 \oplus \pi_{\lambda_2})$;
2  $\pi'_{\lambda_2} = \pi_{\lambda_2}$ translated by $\epsilon$;
3  $\Pi^*_{\lambda_1} = GetSubMap\,(\pi_{\lambda_1})$;
4  $N\,(\Delta\theta, \Sigma) = WeightedScanMatcher\left(\Pi^*_{\lambda_1}, \pi'_{\lambda_2}\right)$;
5  $\phi_* = \left(\pi_{\lambda_1}, \pi'_{\lambda_2}, \Delta\theta, \Sigma\right)$;
6  $\Pi_{t+1} = \Pi_t \cup \{\phi_*\}$;
7  $\Pi_{frontier} = \{\pi_{\lambda_1}, \pi'_{\lambda_2}\}$;
8  $\Pi_{t+1} = RefitManifold\,(\Pi_{t+1}, \Pi_{frontier})$;
```

Algorithm 5.3: Loop closing

The Manifold's accuracy is improved by *back-propagating* the new knowledge, starting from the relation ϕ_* that closed the loop. In essence this back-propagation involves a *breadth-first refitting* of the Manifold. The frontier of this breadth-first process is initialized with the two patches π_{λ_1} and π_{λ_2} on which the loop was closed: $\Pi_{frontier} = \{\pi_{\lambda_1}, \pi_{\lambda_2}\}$. As the frontier iterates through the Manifold, every patch encountered is refitted using the scan matcher until all patches have been processed. In effect, this process ripples the error correction throughout the Manifold.

```
   Data: the current Manifold $\Pi_t$ and the initial frontier $\Pi_{frontier}$ from which to start
         refitting the Manifold
   Result: the refitted Manifold $\Pi_{t+1}$
1  $\Pi_{t+1} = \{\}$;
2  while $\Pi_{frontier}$ not empty do
3      $\Pi_{t+1} = \Pi_{t+1} \cup \Pi_{frontier}$;
4      $\Pi_{nextfrontier} = \{\}$;
5      for each patch $\pi_f \in \Pi_{frontier}$ do
6          $\Pi_{next}$ = patches $\pi$ neighboring $\pi_f$ for which $\pi \notin \Pi_{t+1}$;
7          $\Pi_{nextfrontier} = \Pi_{nextfrontier} \cup \Pi_{next}$;
8          for each patch $\pi_n \in \Pi_{next}$ do
9              $P\,(\Delta\theta, \Sigma) = WeightedScanMatcher\,(\pi_f, \pi_n)$;
10             $\phi_* = (\pi_f, \pi_n, \Delta\theta, \Sigma)$;
11             $\Pi_{t+1} = \Pi_{t+1} \cup \{\pi_n, \phi_*\}$;
12         end
13     end
14     $\Pi_{frontier} = \Pi_{nextfrontier}$;
15 end
```

Algorithm 5.4: Refitting the Manifold

The key motivation behind this choice of loop-closure algorithm was the ease of implementation. Most parts of the algorithm are already available from the incremental localization and mapping. One drawback is that this only *pushes back* the error to the other side(s) of the Manifold and that along the way the error is bound to accumulate again. Note though that the maximum path-length over which the error can accumulate is basically cut in half, so accuracy will still improve significantly (see also Figure 5.7). Refer to Section 7.2 for more detailed results.

When implemented as described, loop closure is a costly process, where most of the processing time is spent in the scan matcher. Several instruments can be put in place to speed up loop-closing. One could compare the quality of fit before and after the refitting to determine when the refitting no longer appears to yield significant improvements. Note that although refitting is then no longer considered necessary, in order to avoid gross discontinuities in the Manifold the frontier should still proceed to transform all remaining patches appropriately. Another optimization would be to not refit patches that were already matched with high confidence, the patch involved would then only need to be transformed to maintain the relative positioning with respect to the previous patch.

The process of closing a loop has so far been described as a single-agent technique. In the next section we discuss *island-merging*, which could be considered the *multi-agent* version of loop-closing. The similarity lies in the fact that island-merging also aims to exploit the information that can be deduced from observing a particular known landmark again. The key difference is that island-merging is triggered when the observations are made by *multiple* robots.

5.4.4 Island Merging

Island-merging is a multi-agent technique that merges two disconnected components in the graph, the *islands*. The information that steers the merging process is the set of landmarks that have been observed on both islands. Since island-merging has a sense of *direction*, we will distinguish between the source island $\hat{\Pi}^S$ and the target island $\hat{\Pi}^T$ in the sense that we will merge the source island *into* the target island in order to acquire the merged island $\hat{\Pi}^M$. Note also that we use the "$\hat{\ }$" to distinguish the notations for local sub-maps and complete Manifolds. Given $\hat{\Pi}^S$ and $\hat{\Pi}^T$, let Λ^S and Λ^T denote the landmarks observed on the each island accordingly. Then the set of landmarks that will be used by island-merging are those landmarks occurring in both sets: $\Lambda^M = \Lambda^S \cap \Lambda^T$. Since landmark position estimates typically come without a rotation, at least two landmarks are needed in this set to be able to also determine the relative rotation of the islands.

Given the relative positioning $\Delta\Theta$ of the two islands, the algorithm starts by putting the islands roughly in alignment. This is done by treating the source island $\hat{\Pi}^S$ as a rigid body

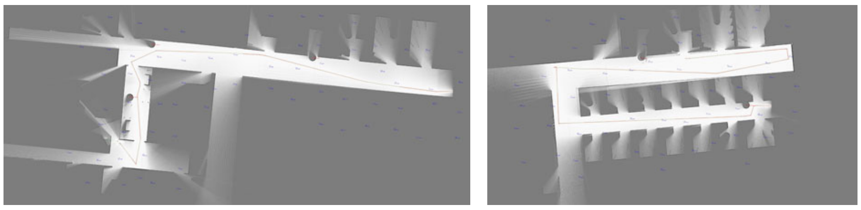

(a) Northern island. (b) Eastern island.

Figure 5.8: Two islands acquired in the Hotel arena.

which is transformed as a whole by $\Delta\Theta$. Then the actual merging of the islands is performed. This part of the process can be seen as if we are *stitching* the two islands together. For every landmark $\lambda_i \in \Lambda^M$ the corresponding patches $\pi_i^S \in \hat{\Pi}^S$ and $\pi_i^T \in \hat{\Pi}^T$ are obtained. Using the scan matcher each source patch π_i^S is matched with the local sub-map Π_i^* of the respective target patch π_i^T. Then a new relation $\phi_* = \left(\pi_i^S, \pi_i^T, \Delta\theta_i, \Sigma_i\right)$ is constructed that stores the Gaussian distribution $N(\Delta\theta_i, \Sigma_i)$ as obtained from the scan matcher. At this point the two islands have been connected and the robots can continue their localization and mapping on the merged island $\hat{\Pi}^M$.

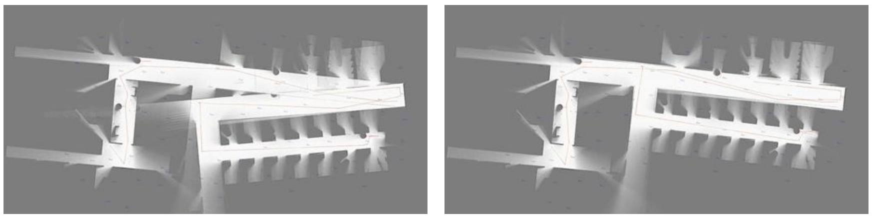

(a) Overlaid based on known starting poses in the upper-right end of the corridor. (b) After rigid-body alignment the islands are connected by newly constructed edges.

Figure 5.9: Island merging.

However, we can further exploit the inherent information in duplicate landmark detections, especially since we already have an algorithm in place to refit the Manifold (refer to Algorithm 5.4). Instead of using just a single relation as the starting point for refitting the Manifold, we can now use *all* the newly inserted relations as the initial seed. The frontier of the breadth-first refitting is then initialized with all the patches that were used to connect the two islands. From there loop-closing proceeds as described in the previous section. The greater the number of landmarks that are used during island-merging and the more widespread their positions, the more accurate the merged map will be after refitting. This is because then the path-lengths over which the error can accumulate during the refitting are likely to be much shorter.

Like loop-closing, island merging is a costly process. After the initial alignment and connection, a refitting procedure is executed which iterates through the entire Manifold.

```
**Data**: the source island $\hat{\Pi}^S$ and target island $\hat{\Pi}^T$ that will be merged
**Result**: the merged islands $\hat{\Pi}^M$
1  $\Lambda^S$ = all landmarks on $\hat{\Pi}^S$;
2  $\Lambda^T$ = all landmarks on $\hat{\Pi}^T$;
3  $\Lambda^M = \Lambda^S \cap \Lambda^T$;
4  $\check{o}^S$ = average position of $\lambda_i$ for all $\lambda_i \in \Lambda^S \cap \Lambda^M$;
5  $\check{o}^T$ = average position of $\lambda_i$ for all $\lambda_i \in \Lambda^T \cap \Lambda^M$;
6  $V^S$ = set of all vectors $v_i^S$ from $\check{o}^S$ to all $\lambda_i \in \Lambda^S \cap \Lambda^M$;
7  $V^T$ = set of all vectors $v_i^T$ from $\check{o}^T$ to all $\lambda_i \in \Lambda^T \cap \Lambda^M$;
8  $\check{\rho}$ = average $\rho_i$ for all $\rho_i = atan2\left(v_i^S, v_i^T\right)$ for $v_i^S \in V^S$ and $v_i^T \in V^T$;
9  $\hat{\Pi}^{S'} = \hat{\Pi}^S$ rotated over $\check{\rho}$;
10 recompute $\check{o}^S$ based on the rotated island $\hat{\Pi}^{S'}$;
11 $\check{\tau} = \check{o}^T - \check{o}^S$;
12 $\hat{\Pi}^{S''} = \hat{\Pi}^{S'}$ translated by $\check{\tau}$;
13 $\hat{\Pi}^M = \hat{\Pi}^{S''} \cup \hat{\Pi}^T$;
14 $\Pi_{frontier} = \{\}$;
15 **for** *each landmark* $\lambda_i \in \Lambda^M$ **do**
16     $\pi_i^S$ = the corresponding patch $\in \hat{\Pi}^S$;
17     $\pi_i^T$ = the corresponding patch $\in \hat{\Pi}^T$;
18     $\Pi_i^* = GetSubMap\left(\pi_i^T\right)$;
19     $N\left(\Delta\theta_i, \Sigma_i\right) = WeightedScanMatcher\left(\Pi_i^*, \pi_i^S\right)$;
20     $\phi_* = \left(\pi_i^S, \pi_i^T, \Delta\theta_i, \Sigma_i\right)$;
21     $\hat{\Pi}^M = \hat{\Pi}^M \cup \{\phi_*\}$;
22     $\Pi_{frontier} = \Pi_{frontier} \cup \{\pi_i^S, \pi_i^T\}$;
23 **end**
24 $\hat{\Pi}^M = RefitManifold\left(\hat{\Pi}^M, \Pi_{frontier}\right)$;
```

Algorithm 5.5: Island merging

However, usually there will only be as many islands as there are robots so merges occur at most ('the-number-of-robots' - 1) times. This mirrors Howard's findings from [27].

5.5 Multi Agent Support

As indicated earlier, the Manifold was explicitly designed to be able to support a *team* of robots. Especially since our Behavior Control is mostly reactive and coordinates a single agent as opposed to being greedy for exploration and coordinating multiple agents, we had a lot to gain by deploying larger teams of robots. For this purpose we have adopted a strategy that maximizes the Manifold's support for multiple agents.

As described in Chapter 2, we did runs of 30 minutes during the competition, of which 20 minutes were allotted for the robots to explore the virtual rescue arena, and 10 minutes

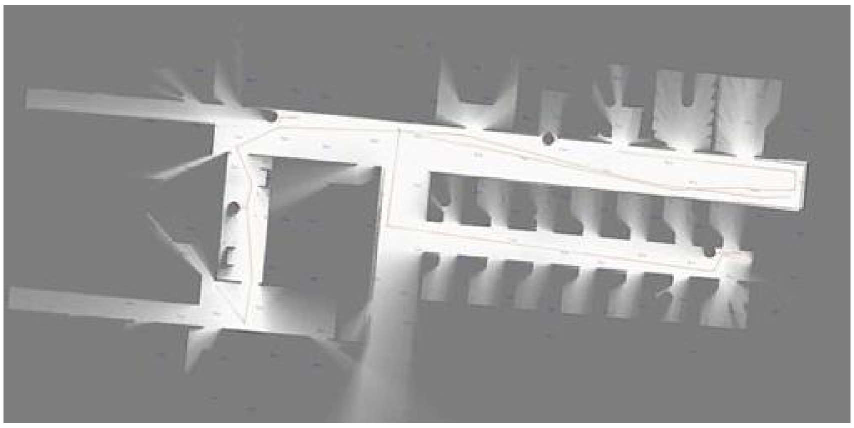


Figure 5.10: Merged islands after refitting.

were allowed to be used to generate any deliverables. Deliverables included information like the map, the victims report, and single-shot RFID report. The rule that we made use of is that during the time allotted for deliverables, we were still allowed to have processes running as long as they run autonomously and only involve post-processing in order to generate the required deliverables. It should be noted that we were allowed "a single interaction" with our system in order to inform the software that the running time was over and that it should switch to getting the deliverables.

There are several ways in which a team of agents can share information. Some approaches adopt a leadership strategy where one leading agent operates as the information aggregator and supplier. The approach that we followed is the one where a separate non-agent process is run where information is accumulated and made available. We refer to this process as the *map-server* where each agent connects to as a client, see also Chapter 6. The advantage is that the map-server relieves the agents of work as all heavy processing like loop-closing and island-merging only need to be done once on the central process, after which the changes can be communicated back to the agents. The disadvantage is that the map-server could become a limiting bottleneck of the system and determine the maximum capacity of the system as a whole.

Based on our client-server process model, our strategy to maximize multi agent support involved the following design decisions:

1. Every agent maintains its own Manifold and performs localization and mapping locally.
2. Any extension to the local Manifold is forwarded to the central map-server.
3. The loop-closing and island-merging processes are only executed on the map-server, where they are decomposed into a part which operates during the run and a part which generates deliverables.

The first two items are straightforward. The main process run by every agent is that of incremental localization and mapping where the performance is almost exclusively determined by scan matching, refer also to Section 5.4. Whenever the local Manifold gets extended, the update is forwarded to the central map-server as well. Since the agent already did the required processing for its own map, the map-server only needs to append the incoming patches and relations to the central Manifold, which takes almost no time.

The third item is of main interest as it is the one that concerns the potential bottleneck. By centralizing all loop-closing and island-merging processes we have reduced the computational load on the agents and are therefore able to run more agents on the same hardware. In addition, to improve the map-servers capacity, we have *decomposed* these processes which means that we can postpone most of the work until delivery time. The decomposition follows from the realization that refitting the Manifold mostly serves to improve the *global accuracy*. In case of loop-closing, after the relation that closes the loop has been added, *local accuracy* is already near optimal. The same holds true for two islands that have been aligned and connected. The notion of *local accuracy* refers to the fact that the probability distributions that are stored on the relations store the *fit* between the two related patches and its quality. It is true that during the refitting of the Manifold, better fits are usually found, but mostly these improvements are only marginal when compared to original fit. More concretely, this observation has led us to the design where the map-server runs Algorithms 5.3 (loop-closing) and 5.5 (island-merging) during running time but then *without* the refitting part. The execution of Algorithm 5.4 (refitting) is postponed until after the active run, refer to Section 5.4 for details about these algorithms.

There is one consequence to be aware of when refitting is postponed as described. Since global accuracy is only pursued *after* the active run, the coordinate transformation operators $\oplus$ and $\ominus$ that use the global frame as mediator should be handled with care. We introduce a new operator $\odot$ which we define as the coordinate transformation *through a relation*, i.e. without the global frame as intermediate. Instead of

$$r_b = r_a \oplus \theta_a \ominus \theta_b$$

the pose r_b relative to patch π_b from the pose r_a relative to patch π_a is

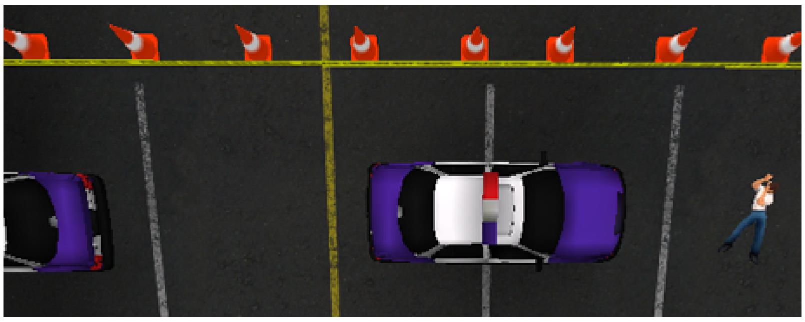
$$r_b = r_a \odot \Delta\theta_{ab}$$

where $\Delta\theta_{ab}$ is obtained from the relation $\phi_{ab} = (\pi_a, \pi_b, \Delta\theta_{ab}, \Sigma_{ab})$ that relates the patches π_a and π_b. When algorithms make use of the Manifold online, this $\odot$ operator is used for their coordinate transformation as they then rely on the local accuracy which *is* maintained online.

The map-server's capacity, i.e. the number of simultaneous agents it can handle, is greatly improved when all refitting operations are postponed. In fact, loop-closing is reduced to a single scan matching operation followed by the insertion of one relation. Island-merging is reduced to a single scan matching operation and link insertion for every pair of landmark observations from both islands. A logical but relatively minor further improvement might be to move the responsibility for loop-closing to the agent to further decentralize the processing load.

5.6 Visualization

Highly detailed visualizations of the Manifold map can be acquired due to the amount of information that is stored per patch. The range scans are easily rendered as point clouds. By ray-casting the individual scan beams from the patch origins, the more informative occupancy grids can be derived. Rendering visualizations of the map can be done online as well as offline, although we preferred offline rendering in order to improve the online performance of our hardware.


(a) Part of a simulated arena.

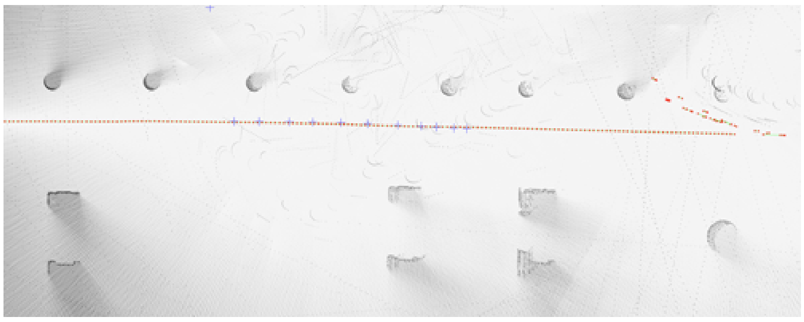

(b) The corresponding visualization of the acquired map.

Figure 5.11: The highly detailed occupancy grid visualization preserves a lot of information. Note that the robot's laser range scanner is mounted at a height such that it scans below the car's chassis and only observes the wheels.

The patches lend themselves reasonably well for online rendering as they are updated only rarely. While feature-based SLAM re-estimate the landmark positions at every time step, *ManifoldSLAM* only moves patches during the refitting of the Manifold after loop-closing and island-merging operations. Refitting operations occur much less frequently than the Kalman filter updates and in Section 5.5 we even presented a strategy were refitting only occurs offline.

Offline rendering of the full map can be as easy as a single pass over all the patches. It is possible to render any desired subset of the graph, but an interesting case is to consider the situation where the Manifold still holds several disconnected islands. They could be rendered separately, but their disconnectedness can also be used for other purposes. Consider for example a robot that gets lost, falls down stairs and lands upside down or in any other situation where the robot is useless at some point. In some of those scenarios, the island that was developed by the robot becomes of low informative quality and is best kept from the other islands and from the map in general. While rendering the Manifold, this can be taken into account and certain islands can be flagged by some means such that they are ignored. In other map representations it is not always possible to adequately deal with such errors that were introduced by a robot whose failure was detected only later. With the Manifold representation, however, there are possibilities to back-track and degrade gracefully.

Chapter 6

Implementation

6.1 Overview

This chapter describes the implementation of the complete system that was used during the RoboCup Rescue Virtual Robots competition at the RoboCup World Championships 2006 in Bremen, Germany. It was designed and implemented in C++ without directly reusing existing SLAM or other robotics software.

As Chapter 5 states, the only parts of the system that we did not invent ourselves are the scan matcher and the general underlying idea for our conception of *ManifoldSLAM*. The scan matcher was originally implemented in MATLAB^{TM*} scripts, which we ported to C++. The only other supporting library used was the Qt Toolkit, a cross-platform Graphical User Interface (GUI) library. This library is described in more detail in Section 6.4.1.

The system is designed in an Object Oriented way [21], such that swapping implementations at different levels is very straightforward. We apply this pattern throughout our system, for example in the extensible simulator interface (see Section 6.3.1). In general, our system is designed to be extensible at all levels, starting the the very low level of simulator input and output, up to high levels such as behavioral control and SLAM. This flexibility is very important and valuable during development as we are able to test and contrast different implementations easily. It additionally allows for a much simplified design since no special cases have to be made during processing.

This chapter first discusses the specific details of the USARSim simulator. This is followed by a discussion of the architecture of the complete system by highlighting several key factors in its design. Then some interesting implementation details are described. The SLAM and Control modules are discussed in the end.

*see http://www.mathworks.com

6.2 USARSim Technical Details

In order to implement a suitable client program to interact with the USARSim server, we have to understand its communication protocol and organization. Our robot control software has to connect to the simulator, sent commands and receive sensor data.

In addition to the text-based protocol used in USARSim, we present all simulated robot models as well as the general process of connecting to, communicating with, and disconnecting from the USARSim system.

6.2.1 Protocol

The text-based protocol employed in the simulator consists of different message types and data segments (see also [65] for the complete manual). The general format of one message, which is terminated by a *return* and a *line-feed* character, is:

`TYPE {segment1} {segment1}...`

Possible messages types are described below. Each segment consists of one or more space-separated strings, some are names, some are values. Some examples are:

- `{Type RangeSensor}` - *Type* is the name, *RangeSensor* the value.
- `{Location 3.212,2.5331,1.221}` - *Location* is the name, the three floating point numbers are the values.
- `{Name F1 Range 4.2134}` - The name of the specific sonar is *F1*, and its measured range is *4.2134*.
- `{Name Camera Location 0.2,0.0,0.3 Orientation 0.0,0.0,0.0 Mount CameraTilt}` - The *Camera* is mounted on the *CameraTilt* object, 20cm in front and 30cm above.

It depends much on the message type what kind of data segments will be included. Possible message types are:

- **INIT**: A command which initializes the connection and the kind of robot to be simulated, along with the starting location.
- **DRIVE**: A command to set rotational speeds for the two motors of the differential drive, and to toggle the lights on and off.
- **SET**: A multi-purpose command to manipulate actuators, or interact with sensors (like resetting odometry).
- **CAMERA**: A command to control the camera's rotation and zoom settings.

- **MISPKG**: A command to control one *Mission Packages*, which is a series of connected joints (like an arm). Instead of using **SET** on each joint, this command allows to specific the pose of the last joint, and the poses of all other connected joints is computed automatically.
- **GETGEO**: Requests a **GEO** message for a certain part. See **GEO** below.
- **GETCONF**: Requests a **CONF** message for a certain sensor.
- **STA**: A status message with information about the simulator and the robot itself, like the power level.
- **MIS**: A general status message of the *Mission Packages*, which is a series of connected joints (like an arm). See **MISPKG** above.
- **SEN**: A message which contains sensor data.
- **GEO**: A message which describes the geometric properties of sensors and other fixtures on the robot, with translational and rotational offsets, as well as information which objects are attached where.
- **CONF**: A message which contains configuration values for different sensors, like the field of view of a laser range scanner.
- **RES**: A general response to **SET** commands.

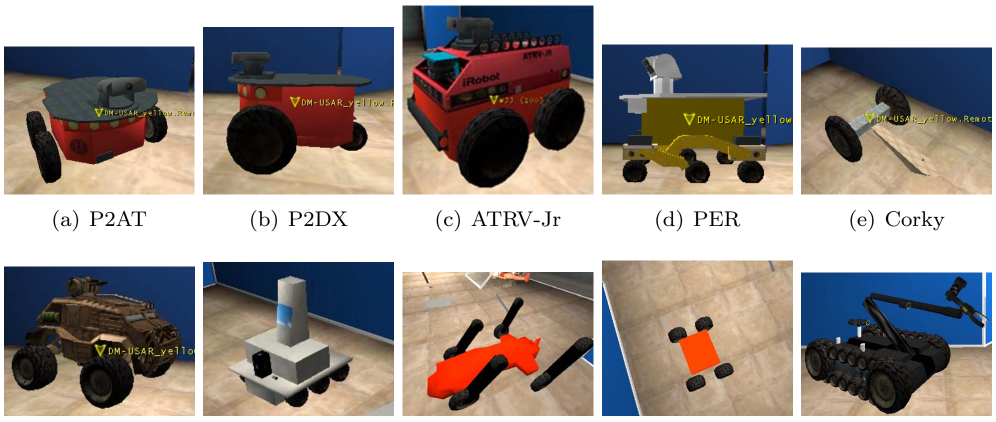
See Wang [65] for more information.

6.2.2 Robots, Sensors and Actuators

USARSim currently supports ten different robot designs.

Figure 6.1 shows all available robot designs, which are described in more detail in the following list. Depending on its size, each platform has the capability to carry a specified amount of sensors.

1. **Activ Media Robotics' Pioneer P2AT** is a compact platform with a 50 cm by 50 cm footprint. It has a high sensor load capability.
2. **Activ Media Robotics' Pioneer P2DX** has a small footprint of 44 cm by 38 cm. It's two wheel setup makes it unstable when navigating over uneven terrain. It has a medium sensor load capability.
3. **iRobot's ATRV-Jr** is one of the biggest platforms with a footprint of 55 cm by 77 cm. It is very mobile and through its size has a very high sensor load capability.


(a) P2AT (b) P2DX (c) ATRV-Jr (d) PER (e) Corky

(f) Four-Wheeled Car (g) Papagoose (h) Tarantula (i) Zerg (j) Talon

Figure 6.1: All simulated robot designs in USARSim.

4. **Carnegie Mellon University's Personal Exploration Rover (PER)** is a small robot without many distinguishing details. It also seems to have an unstable wheel configuration, at least in the simulated world. Also, it has a small sensor load capability.

5. **Carnegie Mellon University's Corky** has two wheels with a counter weight to keep the platform level. This platform is obsolete.

6. **The generic four-wheeled car** is a model of a remote controlled car. It has a very small sensor load capability. This platform is obsolete as well.

7. **International University Bremen's Papagoose** is a rather large platform with six wheels. However, the above Pioneer robots show superior stability on uneven terrain. It also has a high sensor load capability.

8. **University of Freiburg's Tarantula (toy-based)** is a very small robot, and its most distinguishing features are its four flippers. They also serve as the platforms main locomotion. Its maximum sensor load is small.

9. **University of Freiburg's Zerg** is the smallest robot platform in this list. It is a very small four-wheeled robot with space for very few sensors. Its steering is hard because the physics simulation makes it flip very easily with any reasonable acceleration.

10. **Foster-Miller Talon** is the only real tracked platform in this list. It has a very complex gripper arm attached, and includes multiple cameras. It is the largest robot with a footprint of 86 cm by 57 cm. It has a very high sensor load capability.

An almost complete list of sensors is implemented with reasonably realistic characteristics.

1. The **Sonar Sensor** is a generic unidirectional range finder. It is modeled after the general concept of a sonar distance sensor, but it neglects real-world constraints such as incidence angle or surface texture which usually impacts the real sonar's accuracy significantly. The simulator implementation simply casts a ray in the world model and reports the distance to the first object hit by that ray if it is below some distance threshold. The additive noise is uniform: $r_{measured} = r_{real} + r_{real} * random(-noise, noise)$.

2. The **IR Distance Sensor** is very similar to the sonar sensor, but it is based on the infrared distance sensor. Its default maximum measurement threshold is therefore much smaller, and its ray may pass through transparent objects. The noise model is the same as above.

3. The **Laser Range Scanner** models real-world range scanners such as the SICK family of sensors. It is basically a rotating *sonar* sensors (not IR sensors, counter-intuitively) such that its rays sweep over one horizontal line. All other previous comments apply.

4. The **Tilting Laser Range Scanner** is simply a *Laser Range Scanner* which also tilts and produces a depth image. Since each pixel means one ray-casting operation, only small resolutions are feasible with the current implementation. This sensor is motivated by recent sensor designs such as the Swiss Ranger 3D camera.

5. The **Camera** is a special sensor as it will provide a rendering of the world from its specific point of view. That is why it cannot be accessed through the same interface as all other sensors, but needs a separate Unreal Tournament Client^{TM} instance and a special "camera server" to capture the client's display, encode it as a JPEG picture, for example, and transmit it to the robot control program. Otherwise, it works just like a real world camera would.

6. The **RFID Sensor** is a rather unusual sensor with two functions. First, it is used to assess the quality of produced maps by allowing to detect so-called "single-shot" RFID tags with absolute accuracy, as described in Section 2.1.2. That way, the smaller the error of localization of the robot, the smaller the error of the estimated location of these tags. Also, it can detect RFID tags released by the robot itself, but with limited accuracy and limited maximum range.

7. The **Victim Sensor** is meant to allow victim identification and localization without a camera. It was included due to above mentioned shortcomings of the camera implementation. It is very similar to the RFID sensor, but has a field of view which is similar to that of a camera. Also, with decreasing distance, more information is

available for a victim. There are also distinct false positives, which are identified as such with decreasing distance as well.

8. The **Inertial Navigational Unit (Gyro)** measures rotations around each of the three axes, just like its real counterpart.

9. The **Odometer** keeps track of planar motion by simulating motor encoders and counting wheel rotations. According to the wheel configuration, measurements are combined into motion in all three directions.

The simulator also implements a small range of actuators.

1. The **Motor** has the obvious task to control the wheels, and therefore the movement of the robot.

2. **Gripper with Arm**: A specific device is implemented to control the arm on the Talon robot. Each joint and the opening angle of the gripper claw can be controlled separately.

3. A **Pan-Tilt-Zoom Camera Mount** can control the direction of a mounted camera, as well as the zoom level of the same camera. Each of these variables can be controlled separately.

4. The **RFID Releaser** is able to place new RFID tags in the environment at the robot's position.

6.2.3 Organization

Figure 6.2 shows how processes are distributed when running USARSim. There are two main components: The simulator server and the clients which control the simulated robots.

Each client process controls exactly one robot in the simulated world. Once a new client connects to the server, a new robot controlled by that client is inserted into the world. Via this network connection, all important data is communicated to and from the simulator. Sensor values and manipulator commands are exchanged with the server via the previously introduced protocol.

As mentioned before, the **Camera** sensor needs special attention. A separate Unreal Tournament^{TM} (UT) client process needs to be started. This game client will render the picture as seen from the camera's point of view. It has direct access to the geometry of the world, robot models and their position through UTs special online game protocols. Since the game is focused on online use, it is a very efficient way to distribute the simulation processing and rendering to different computers. However, the game client only renders the

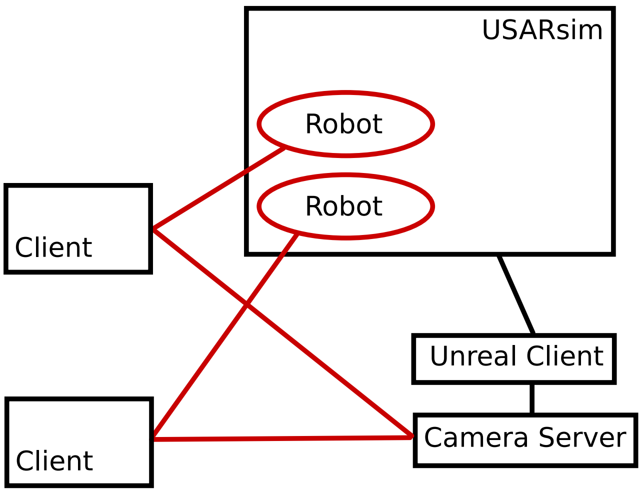


Figure 6.2: Organization of the USARSim processes.

scene to the main screen. This means another application has to read the image directly from the graphics card memory (achieved via a Direct3D^{TM} hack), compress it for example into a JPEG image, and send it on to the correct client.

Once the client disconnects from the simulator, the robot is removed from the world.

6.3 Architecture

There are two major parts to the system architecture: The *Agent* and the *Server*. The Agent represents one robot with its sensors and actuators in the USARSim simulator, which is described in Section 6.3.1. The Server gathers mapping information from all connected agents and automatically merges it into one map. It is described in Section 6.3.2.

This decentralized setup allows us to spread the computations across many machines and still run a relatively large team of agents. Only one server instance is required, which will record the pre-processed mapping information from the agents. It will only merge everything into one map once it is instructed to do so, hence it does not require much processing before that point in time.

6.3.1 Agent Organization

The *Agent* process encapsulates everything one robot in the simulator represents. It hosts the network connection to the simulator, reads the sensor data sent by the simulator, and controls the simulated robot.

One agent consists of multiple connected sensors and some actuators. This corresponds exactly to the organization of the simulated robot in USARSim. The agent also holds the SLAM module and a Behavior module. These will be discussed in greater detail in the next section.

The main function of the agent is an organizational one. It keeps track of all connected sensors and actuators. In addition, it knows the general data format in which the USARSim server sends sensor information. The information packets are read from the network connection as fast as possible, parsed in a generic format and passed on to the right sensor implementation, identified by type and name. That sensor can interpret the data, which depends on the type of sensor.

The agent also provides a method to send commands to the simulator server via its network connection. This function is used by the actuators which send specially formatted requests to the server, for example to drive forward.

Implemented sensors are (also illustrated in Figure 6.3):

- Sonar
- Laser Range Scanner
- Odometry / INU (position and orientation)
- RFID
- VictimRFID

One agent is configured in a subclass. Each uniquely configured robot class on the simulation server will be represented by exactly one *Agent* subclass. This subclass will instantiate the right kinds of sensors and actuators, as well as the SLAM and Behavior modules. Our current implementation only consists of one such configuration as we used a completely homogeneous team during the RoboCup competitions.

Originally, we evaluated the usage of Player[†] for this exact purpose, but it proved too cumbersome to add new devices in the Player framework. As there are some special devices used in USARSim for scoring (like the RFID sensor), we needed a framework which easily support new types of sensor data. We needed a specially designed and flexible framework to implement all sensors and actuators for use in the agent.

Figure 6.3 shows the final class hierarchy and associations for our agent and related packages. *Sensors* are divided into two classes: Those from which we need to process every single observation and those from which we only need the latest reading. The former is represented as a *MultiStateSensor*, such as the laser range scanner, the latter is called a

[†]see http://playerstage.sf.net

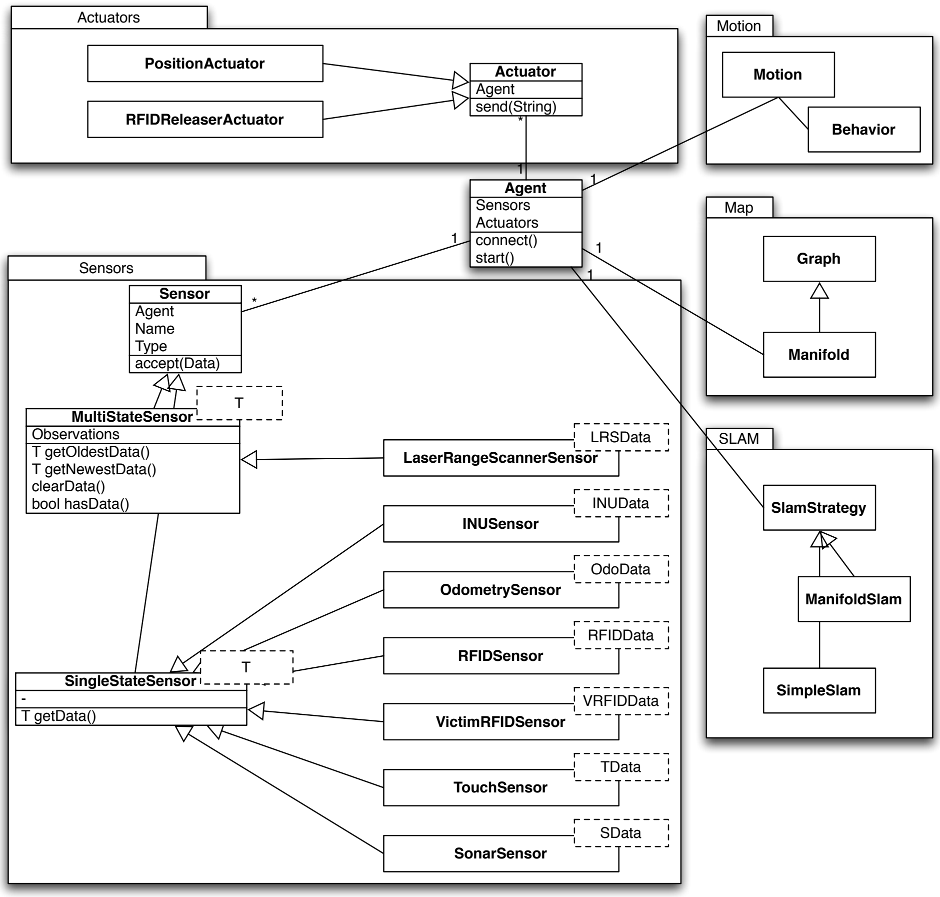


Figure 6.3: The UML diagram of the Agent, Sensor and Actuator hierarchies with implementing subclasses. Also shows the SLAM and Motion modules.

SingleStateSensor. All sensors are implemented as templates (parameter types are depicted as dashed boxes in the UML diagram), which means that we can later decide which exact type of data they will provide. This gives us great flexibility in the organization of sensor data structures, as well as type safety. Polymorphism would not fit this specific need well as it assumes similarity between derived classes, while sensor data structures may radically differ between sensors.

The *Motion*, *SLAM*, and *Map* packages, which are only briefly summarized in Figure 6.3, are described in more detail later in this chapter in Section 6.5 and 6.6.

Actuators are another package represented in Figure 6.3. Currently, only two actuators are implemented: One to control the robot's movement, and one to release RFID tags at the robot's location. Since actuators in general only share the use of the agent object to send commands encoded as strings to the server, all actuators are highly specialized and expose methods that only relate to their specific function. For example, the *PositionActuator* only exposes one method to manipulate the differential drive of the robot: `void drive(float translationalSpeed, float rotationalSpeed)`. Other possible actuators, like one to operate a pan-tilt-zoom camera unit, would expose different functions.

The main advantage in this encapsulating design is that the complete set of knowledge needed to operate and use one sensor or actuator is contained in one specific class. Other approaches might share such responsibility between different entities in the system, which would slow down development and impair maintainability.

6.3.2 Mapping Server Organization

The *Server* represents shared or aggregated knowledge between all agents. Initially, it was intended that the agents organize decentrally. However, due to networking and portability issues with the implementation of multicast messaging, the Server was implemented to provide the same or similar facilities. Broadcast messaging could not be used since multiple agent instances were to be used on one machine with one IP address.

In its current implementation, the server stores and later post-processes mapping information gathered by the individual agents. It can be extended to provide communication between agents, or a global planning module. All these concerns were taken into account while designing the server.

Figure 6.4 shows an UML diagram of the classes involved in the server. One Server object consists of multiple connections to one agent each. Here, our specialized cross-platform high-performance networking abstraction was used, which is described in Section 6.4.2. An agent connection also hosts a *SLAM* module, through which all mapping information is forwarded from the agent. This information has already been pre-processed by the sending agent and can be directly inserted into the combined map in the Server object. This process is very inexpensive, and a server was successfully run with eight connected agents, while one physical machine was only able to host at most two agents.

After agents finish sending mapping information, the server uses its *Map* module to merge the still separated individual maps from each agent. While this is implemented as an offline procedure, it can be an online one. Such an online map merging is detailed in the description of *ManifoldSLAM* in Chapter 5.

6.4 Details

There are some noteworthy implementation details that are described in this section. Firstly, the cross-platform API *Qt* is described, followed by our custom networking implementation.

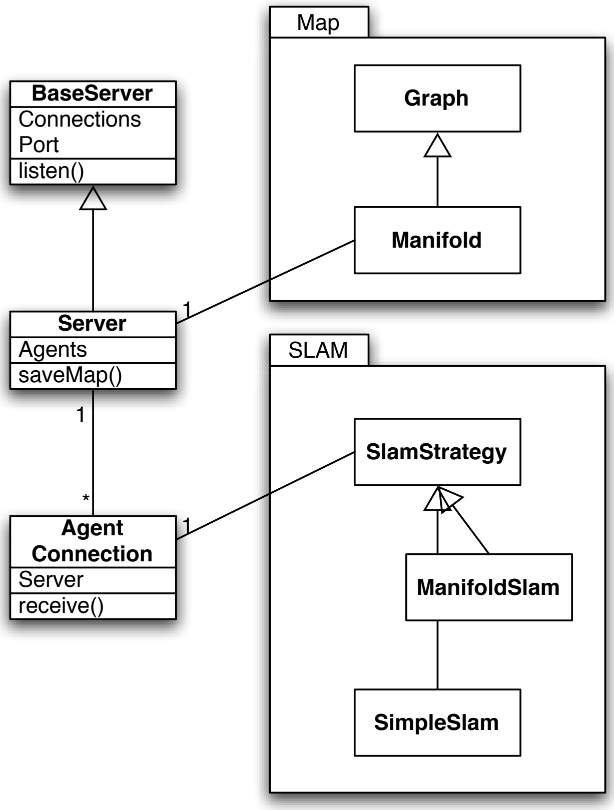


Figure 6.4: The UML diagram of the Server hierarchy with implementing subclasses.

6.4.1 Qt Cross-Platform API

The Qt Toolkit is very popular in the Linux community. The widely known KDE (K Desktop Environment) window manager is based on this library. Since version four, it is also freely available for the Windows and Mac OS X platforms for non-commercial use. While it is mainly a windowing and graphical toolkit, it provides many additional functions that are commonly needed in modern desktop applications. It supports cross-platform multi-threading, many collection classes similar to the Standard Template Library (STL), and an extensible object serialization mechanism to send objects over network connections or save them to a file.

We extensively used this API to develop the required functionality quickly. As our computers had various different operating systems installed, it made development very speedy since we did not have to worry about using the right platform-dependent APIs or even writing abstractions ourselves.

As our application is very resource intensive, we required multi-threading support and thus used Qt's thread and thread synchronization facilities in many places. Qt's threading model is inspired by the Java programming language and works by subclassing the abstract *QThread* class. Our applications require up to around ten threads in some cases, which makes this a very important feature of Qt.

Object serialization is also an important feature Qt provides. Such facilities allow us to use the C++ stream operators ("<<" and ">>") to send data contained in objects over the network connections or store it in binary format. This feature is used extensively to assemble data containing mapping information that is to be sent to the server.

6.4.2 Networking

While Qt does provide some cross-platform network socket classes, they were too heavy-weight and too dependent on Qt's own event system. Since Qt is usually used within desktop applications, their networking support is adequate for that area of application, but not for high speed data processing as we require.

Therefore, we implemented our very own cross-platform high-performance network sockets. Using a common class interface, two versions supporting MS Windows (using the WinSock API) and Unix-like (using the Berkeley Socket API) operating systems respectively were developed. These classes wrapped the low level platform-specific code. Other features, like synchronous communication over asynchronous sockets, were built on top of these abstractions.

The system was divided into two parts: First, two base classes for outgoing (connecting socket) and incoming (listening socket) connections were implemented wrapping only the platform specific system calls. Secondly, classes to represent a connection, a server and a client were designed. These used the low level functions of the socket implementations and extended it with higher-level constructs such as threads, buffers, and message formatting. Qt's byte manipulation facilities are used to store and retrieve typed binary data in network transferrable form.

One extra feature which facilitated development substantially was being able to start a synchronous conversation over an asynchronous connection. This works by using the thread synchronization mechanisms such that the calling thread is suspended while the worker thread of the connection waits for a reply for this conversation. Safeguards such as timeouts are not implemented yet, so a program may hang if the connection is lost or no answer is received after the calling thread is suspended. However, this was never a problem in our trials as we used high-speed cable ethernet connections in our tests as well as in the competition.

Synchronous calls help development because even complex queries to the mapping server could be expressed as simple function calls instead of having to supply a callback function.

A query for changes in the global map might be formulated as a function `MapUpdates getMapUpdates(long timestamp)`. Here, an object which contains all recent updates to the maps would be returned from the function call. This object is generated from the mapping server's response. Since we use multi-threading extensively, only one thread waits for this response while we can keep processing data with the other threads. This network communications model helped us a lot in developing the mapping server.

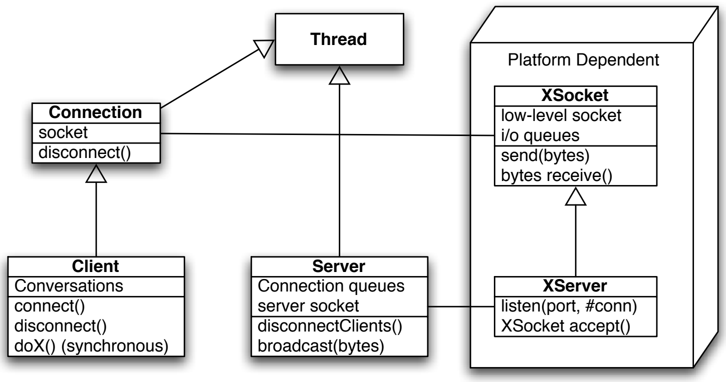


Figure 6.5: A UML diagram of the networking classes. The box on the right depicts the platform-dependent code based on either the Berkeley Socket or WinSock APIs.

6.5 SLAM Module

As shown in Figures 6.3 and 6.4, the *SLAM* module is part of both the *Map Server* and *Agent*. The module consists of two parts, the map itself and the algorithm implementation to manipulate it. A more detailed overview over the two involved packages is given in Figure 6.6.

The map is represented as a specific implementation of an abstracted graph data structure. All graph-specific implementation detail is encapsulated in the *Graph* superclass, which keeps the actual map free of any distracting code and therefore more maintainable. The graph is stored in an edge-list form. This is beneficial for later communication with the server as single vertices and edges can be easily accessed for synchronization and partial updates.

Superseding and extending a *Graph*, the *Manifold* class allows manipulation of the map data by the SLAM implementation described in the next paragraph. It also implements important functionality like island merging (described in Section 5.4.4) directly. Such methods which operate directly on the map data would have been too cumbersome to implement as part of the SLAM methods. Additionally, the *Manifold* can generate so-called sub-maps.

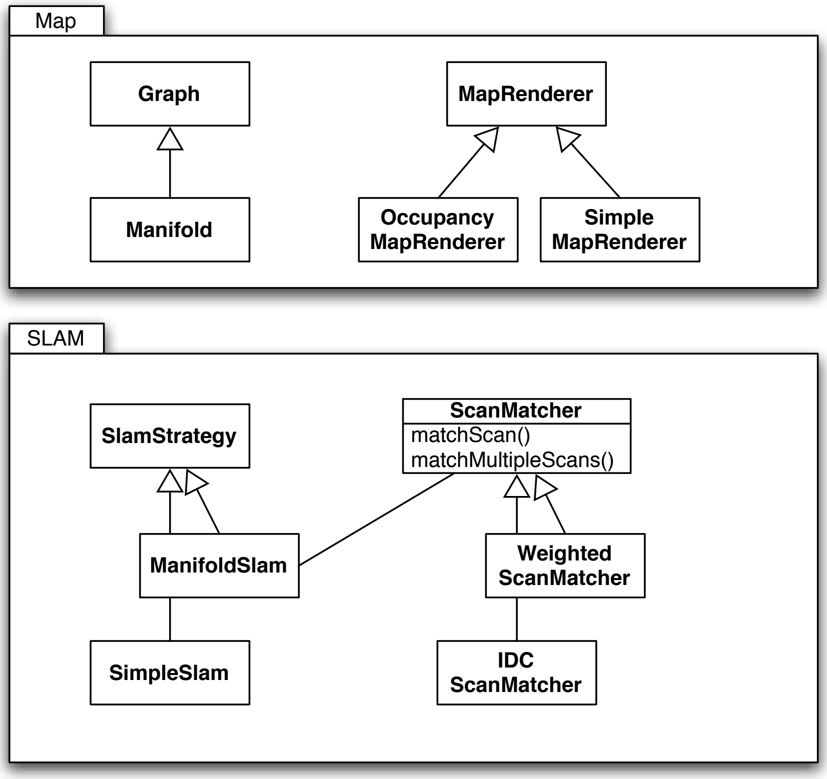


Figure 6.6: A more detailed UML diagram of the *SLAM* and *Map* packages.

These can be seen as local neighborhoods of given nodes in the graph and are central to the inner workings of the *ManifoldSLAM* method shown in Chapter 5. The class uses standard graph traversal algorithms to generate these sub-maps. The graph traversal is also important for map rendering, which is discussed next.

Rendering the *Map* as an occupancy grid (or any other form) is implemented in the *MapRenderer* hierarchy. It is not central to the general architecture and thus not shown on previous UML diagrams. Two forms of renderers are implemented: The first one is a simple renderer which is optimized for speed and was used for debugging purposes. It only shows occupied space like walls. See the left side of Figure 3.9 for an example. The second one is a real occupancy grid renderer. Using Qt's 2D drawing features, it draws white lines with a black dot at the end for each laser scan ray. The right side of Figure 3.9 shows this style of rendering. All sample maps shown in this thesis are rendered using this implementation.

The SLAM algorithms are implemented in the *Strategy* pattern. These classes usually do not have any state variables but only encapsulate a specific way of doing something. In this case, they implement different ways to generate maps. SLAM strategies have one common

SlamStrategy superclass which defines a function to process one observation. The simplest strategy, *SimpleSLAM*, is to just add the new observations without any processing. This is only correct if there is no noise on odometry and laser range scanner data. *ManifoldSLAM* is implemented as a strategy as well.

One last component which is indirectly included in the *SLAM* strategies above is scan matching. Scan matchers are implemented in a similar way as the SLAM strategies. The abstract *ScanMatcher* class defines two functions, one to match two consecutive scans and one to match a new scan to a history of scans which are already aligned. The second version is of course more robust to occlusions and discontinuous environments as the first, but with some algorithms it is much harder to implement. Two different scan matchers are available: The Weighted Scan Matcher (Section 4.3.4) and the Iterative Dual Correspondence (Section 4.3.2) algorithms. The common interface allows them to be used interchangeably.

6.6 Autonomous Behavior and Motion Module

In our complete approach, we focused on Multi-Agent SLAM. However, another important topic is autonomous exploration and its connection with SLAM. There are two dimensions on which exploration algorithms are typically compared and contrasted: exploration rate and multi-agent support. The former refers to a ratio that measures the amount of explored area over time where high exploration rates are usually preferred. The latter refers to whether an algorithm can take advantage of multiple agents by coordinating their joint actions. We set out to be on the high side of both these dimensions as initially we included *frontier-based multi-agent exploration* in our design [68, 70].

Frontiers are defined by Yamauchi [67] as the ‘regions on the boundary between open space and unexplored space’. Intuitively it is easy to understand why these regions are of interest: Essentially *all* unexplored areas are of interest but robots can only start exploring these from where they are accessible, i.e. where the unexplored area borders on known free area. The notions of free space and unexplored space are well-defined concepts on occupancy grids. As it was illustrated in Section 5.6, the Manifold can produce these easily. Unfortunately, time did not permit us to further develop the idea of frontier-based exploration on the Manifold and we have had to move this to Future Work, see also Section 9.2.

The remainder of this section discusses our current implementation that was inspired by the Dutch Aibo Team [61], which has successfully participated in the 4-Legged League of the RoboCup Soccer competition since 2004. By using a similar approach to control a robot’s behavior, we were able to develop a very robust exploration strategy that can easily be extended and elaborated on in the future.

Following the popular sense-reason-act paradigm to analyze a robot's behavior, we could decompose exploration as follows:

- *sense*: is about acquiring all necessary sensory information.
- *reason*: the *Behavior Control layer* determines which behavior should currently be executed by the agent based on the sensory inputs. The term *current behavior* refers to a higher level description like '*explore-environment*'.
- *act*: the *Motion Control layer* bridges the semantic gap between the higher level behavior descriptions and the low-level motion commands that trigger the actual movements. To hide the low-level actuator details from the behavior control layer, the motion control layer exposes several *elementary motion patterns* like '*go-to-victim*' that can be activated by the behavior control.

In bottom-up fashion, the following sections first discuss the sensor input that was used by both Motion Control and Behavior Control. Then the details of the Motion Control layer and the elementary motion patterns it exposes follow. How and when these are activated is described in the final section on the Behavior Control layer. For an illustration of how the three components collaborate refer to Figure 6.9.

6.6.1 Sensor Input

The used P2AT robot was equipped with the following sensors:

- A laser range scanner
- Several sonar range sensors
- An odometry sensor
- An INU sensor
- A RFID sensor
- A Victim sensor

The laser range scanner was pointing straight ahead and was configured with a field of view of 180 degrees. The resolution was sometimes 1 degree, resulting in scans of 181 laser beams and sometimes 0.5 degree, which resulted in scans of 361 beams. Of the 16 sonar sensors, 8 were mounted on the front and 8 on the back. They were evenly spaced and some of them were slightly tilted up or down so that the robot would also be able detect obstacles hanging above as well as holes in the floor.

The INU sensor and odometry sensor were able to be placed arbitrarily and are not visible in Figure 6.7. Both RFID sensors were attached to the bottom of the robot's chassis. They were positioned in the center and were both pointing straight ahead.

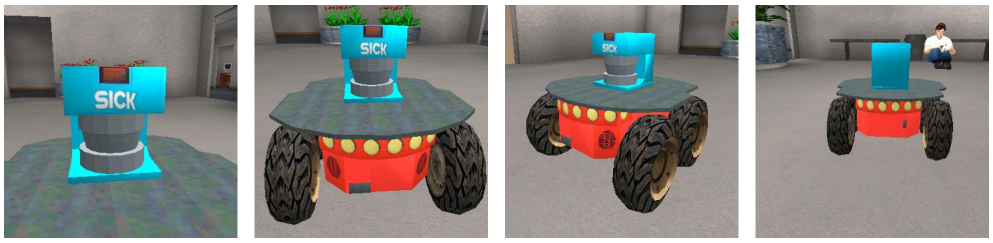


Figure 6.7: Sensors. The SICK laser range scanner on top and the front and back sonar rings are clearly visible. The other sensors like the VictimRFID sensor and INU sensor are hidden or attached to the bottom of the robot's chassis.

6.6.2 Motion Control

For the behaviors of our design three elementary motions were sufficient. An overview is shown in Figure 6.8.

Figure 6.8: Elementary motions.

JustGo can be seen as the default motion. Robots executing this motion move straight ahead for as long as possible. When the sonars or laser range scanners detect an obstacle in front of the robot, it will stop, move backwards a random amount of time, turn left or right randomly and move ahead again. Simplistic as it may seem, it is excellent for covering long distances in corridors or outdoors. A fall-back scenario was included to make the robot more robust against unnoticed obstacles or other unforeseen challenges. In these situations where the robot may have missed an obstacle in its sonar or range scanner input, the INU will eventually inform the robot of significant changes in the pitch or roll values. The robot will then initiate an immediate retreat to avoid falling over and choose a new direction.

GoToVictim will make the robot move towards the specified victim. This motion relies on the high-resolution laser range data which the robots uses to cautiously approach the victim while avoiding bumping into it. It also reads the victim sensor to potentially adjust its heading.

Stop will stop the robot immediately.

6.6.3 Behavior Control

The currently executed behavior is managed in a finite state machine as outlined in Figure 6.9. Only one behavior-state can be active at a time. Each behavior controls the Motion Control layer when it is active. The transitions from one behavior-state to the next are determined by the behavior-state that is currently active. At startup, the *Explore* behavior is activated by default and from there everything is based on sensor input.

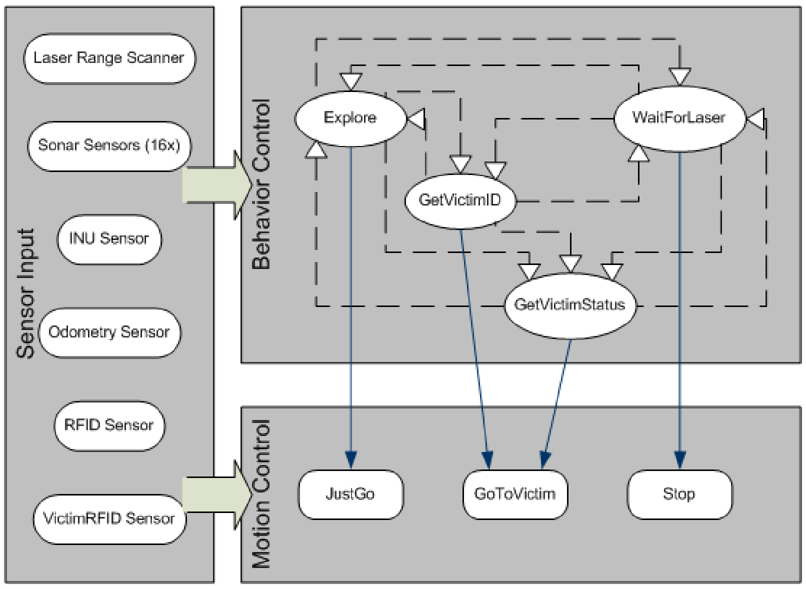


Figure 6.9: Behavior Control. The dashed arrows indicate the possible behavior-state transitions and the solid lines indicate which elementary motion is executed during the activation time of a particular behavior-state.

The states *Explore*, *GetVictimID* and *GetVictimStatus* aim to get a maximum *ScorePerVictim*. Recall from Section 2.3 that 10 points are awarded for every victim that is reported. At least the victim's ID is required to get a valid victim report, so the transition (*Explore* → *GetVictimID*) is made when the Victim sensor reports that a victim is observed but that the robot is still too far away to be able to read the corresponding ID. Another 10 points are earned when the victim's status is also reported. When the ID is sensed, the transition (*GetVictimID* → *GetVictimStatus*) is triggered. If a robot observes a victim's ID while in the *Explore* state, which could happen for example when it encounters a victim just after turning around a corner, then *GetVictimStatus* is activated directly through (*Explore* → *GetVictimStatus*). Both *GetVictimID* and *GetVictimStatus* are programmed such that in case they fail to achieve their objective in a certain time, they transfer control back to *Explore* through either (*GetVictimID* → *Explore*) or (*GetVictimStatus* → *Explore*).

The *WaitForLaser* state is implemented to increase the accuracy of RFID position estimates. Especially for the single-shot RFID tags that were used to measure the *MetricMapQuality*, this behavior-state is of great value. Whenever a single-shot RFID tag is observed, the robot is stopped immediately and the tag is stored on a new patch that is formed with the range scan acquired during the stand-still. Such range scans are not affected by the usual time-of-flight issues, rotational distortions or additional motion noise, so the patch-relative estimate for the RFID position is improved. The same is done for first-time victim observations that contain the victim's ID. The *WaitForLaser* state can be triggered from any other state and runs for at most a second because the laser range scanner is configured to generate at least one scan per second. After the scan is obtained and stored as a newly inserted patch together with the RFID tag, control is transferred back to the state which activated *WaitForLaser*.

Chapter 7

Experiments and Results

7.1 Scanmatcher Evaluation

7.1.1 Introduction

The scan matcher algorithms described in Chapter 4 already brought along some results in the respective papers that introduced them. However, none of the mentioned results was representative in the setting of the RoboCup Rescue Virtual Robots league. Since we received implementations for four different algorithms, we were able to produce more relevant results.

An important note is that we received three implementations written or endorsed by the authors of the respective algorithms. The fourth one, implementing the IDC algorithm, was used as a comparison to the WSM algorithm by its author.

The results of the NDT based scan matcher are included for completeness as we encountered some problems with the implementation. Instead of converging towards the real solution, it diverges or terminates inexplicably. After some study of the code, it was possible to at least make the implementation not diverge, but it still does not converge very reliably. This is evident in the section describing the results.

In the next section, we describe the setup of the experiment. Then, the results of the trials are described for each method. Afterwards, we discuss some implications of the results in respect to the best choice for our SLAM method.

7.1.2 Experimental Setup

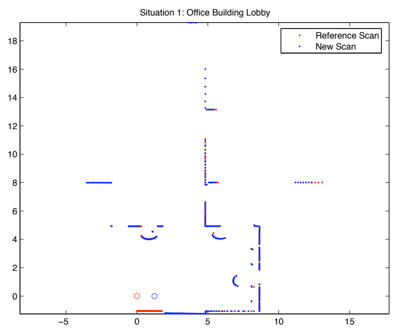
Each implementation is tested with four different sets of scans with 361 points each over an arc of 180 degrees. All distinct areas in the RoboCup Rescue Virtual Robot league are represented. Four situations, which were also encountered in the competition, are included. Figure 7.1 shows the scan data in detail. Figure 7.2 shows the area around the scans in the simulated environment.

1. **Indoor, many features:** A scan of a lobby in a simulated office building. It includes many features such as legs of benches, flower pots and doors. This should be a rather easy problem for a scan matcher. The difficult factor is that the distance between scans is about 1.2 meters, with rotation.
2. **Indoor, few features:** A scan of a corridor in a simulated office building. The length of the corridor gives rise to many local optima. However, there are some features, like doorways, which make a correct alignment possible.
3. **Outdoor, very few features:** A scan of a wooded area. This scan includes many "infinity" values, and only very few and small features. These are mainly tree trunks. The main difficulty is the complete lack of structure.
4. **Best case, like 1.:** Same scan as the first situation, but the scan matcher tries to match on scan to itself. Since the scans are completely identical, this test shows the best case match, or an upper bound for a certain method.

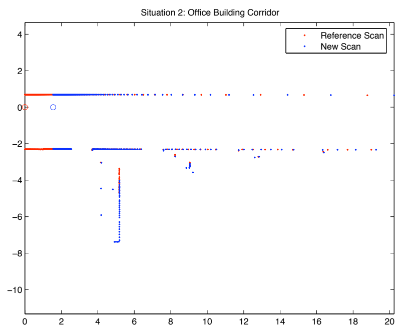
For each situation, 1275 trials were conducted with different added noise. 17 different translational perturbations are included: No perturbation, 8 positions each 45 degrees on a circle with a radius of 25 cm, and 8 positions each 45 degrees on a circle with a radius of 50 cm. On top of translational noise, rotational noise is added as well. Here, we include 5 different levels: from -30 to +30 degrees in steps of 15 degrees. That is 85 different perturbations, each repeated 15 times.

These noise levels are very high, however, we want to establish a worst-case scenario. In practice, the robot does not travel much between two consecutive observations from the laser scanner, but we need to know what kind of error a specific method is able to correct and how fast.

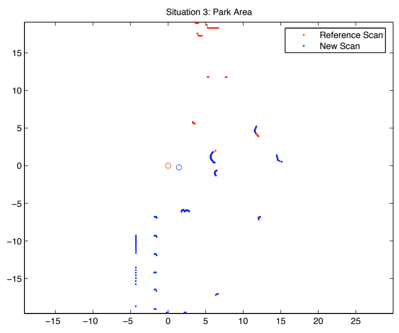
In each trial, the disturbance is recorded, as well as the initial guess for the trial. We also record the final answer from the scan matcher, and how many iterations were needed to converge. Settings, such as the maximum number of iterations remain unchanged in the individual implementations. In general, we make no distinction between converging or reaching the maximum number of iterations allowed. Our goal is to evaluate the methods by their best possible guess given reasonable settings, which we assume the defaults to be.



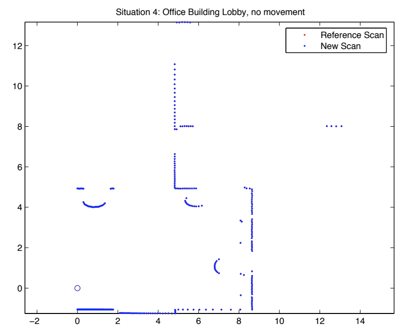
(a) Situation 1



(b) Situation 2



(c) Situation 3



(d) Situation 4

Figure 7.1: The scans used for each experimental setup

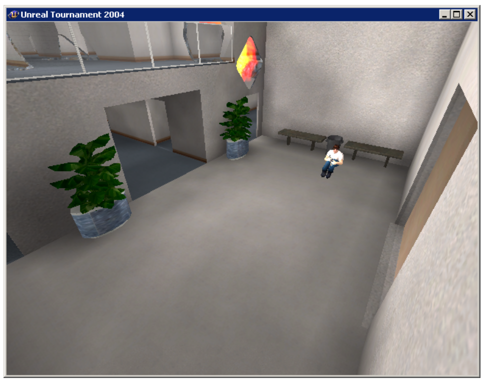
7.1.3 Results

In each following section, we will explain the results on the above situations from each specific method. As a distance metric for two poses, we used the metric proposed by Minguez et al. [36]:

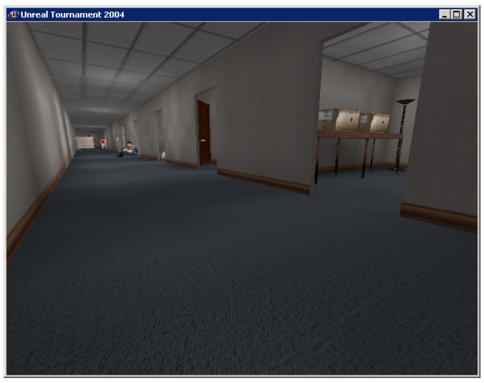
$$distance(a,b) = \sqrt{(a_x - b_x)^2 + (a_y - b_y)^2 + L^2 \cdot (a_\theta - b_\theta)^2}$$

As described by Minguez et al. [36], L is analogous to a length (the length of one radian compared to meters). Just as recommended in the paper, we also chose $L = 3$ for these evaluations.

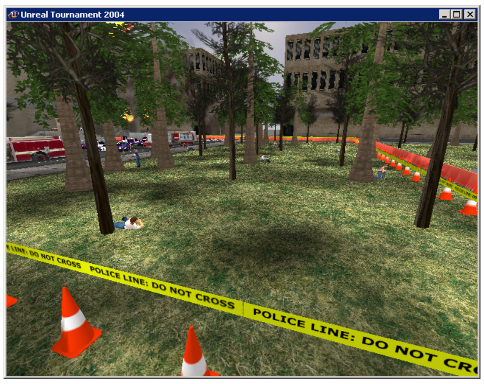
With the induced noise in the test trials, this results in the following baseline errors. Here, we assume the *worst-case* scan matcher which always returns the initial guess:



(a) Situation 1 and 4



(b) Situation 2



(c) Situation 3

Figure 7.2: Locations in the simulated world where the experimental scans were taken

Rotation / Translation	**0**	**.25**	**.5**
0 deg	0.000000	0.250000	0.500000
+/- 15 deg	0.785398	0.824227	0.931048
+/- 30 deg	1.570796	1.590566	1.648454

In the mean, the baseline error is 1.1751. However, the largest errors appear in the cells with high translational and high rotational error, so the mean is not very representative.

Any scan matcher should be able to improve on these figures or at least not return a result which is worse than the initial estimate. Please note that slight deviations from this baseline are acceptable. The exact optimum of the specific method can differ a little from the ground-truth because of the point sampling of the laser range finder. The reason is the same as given in the introduction of the *Correspondence Error* in Section 4.1. Only in the last situation are the points identical, thus the true minimum should be exactly at the ground-truth value. This is especially important in the first case where the baseline error is zero.

Situation 1: Indoor, many features

In this situation, the feature-rich office building lobby, we assume that it is the best case for scan matchers: It is a dense scan, with many distinguishing features such as doorways and flower pots. As seen in Figure 7.1a, there is still a lot of overlap between the two scans even though the difference between the poses is quite large. However, there are still significant portions of the scans that do not have a corresponding part in the other scan. This is because the second scan is taken at an angle of around 30^o and after a translation of about 1.2 meters.

Alg.	$0m/0^o$	$.25m/0^o$	$.5m/0^o$	$0m/15^o$	$.25m/15^o$	$.5m/15^o$	$0m/30^o$	$.25m/30^o$	$.5m/30^o$	Mean	# It.
base	0.000000	0.250000	0.500000	0.785398	0.824227	0.931048	1.570796	1.590566	1.648454	1.1751	n/a
NDT	0.001094	0.325732	0.522488	0.785398	1.019792	0.905721	1.570796	1.592676	1.682743	1.211989	2.117647
IDC	0.006687	0.007594	0.007845	0.007873	0.007873	0.007873	4.185263	3.931223	3.831785	2.466790	24.717647
MbICP	0.010804	0.010489	0.011106	0.011206	0.010933	0.010854	0.011329	0.010947	1.216617	0.527934	20.435294
WSM	0.009132	0.009132	0.009132	0.009132	0.009132	0.009132	1.227053	2.013077	1.046770	1.002273	16.223529

In the above comparison of the performance of the four different algorithms, *NDT* seems to be the best in the first column, meaning the one which has the closest optimum to the ground-truth, the other noise settings show how close its error is to the baseline. This, along with the very low iteration count, is an indication that it never significantly changes the initial estimate it is given. This behavior of the *NDT* algorithm can also be observed in the rest of the experiments described later.

The comparison shows that *IDC* is the best for small noise, closely followed by the *WSM* algorithm. In the cases of extreme translational noise, the rule to find correspondences of the *MbICP* algorithm seems to work much better than the one used in the *WSM* or *IDC* methods. However, the *WSM* method converges in the least number of iterations on average.

In this trial, we would prefer the *WSM* method because it displays very good performance under small rotational noise, and reasonable performance under high rotational noise while requiring a small number of iterations.

Situation 2: Indoor, few features

In this situation, the main difficulty is the lack of distinguishing features. It should be rather easy for any method to find the correct rotation because the corridor itself is a very big clue in this case. However, finding the translation in the direction of the corridor is hard because not many unique features exist. There are only four doorways, and the scan was only able to record very little of the first room. With high rotational and translational noise, it might be possible for methods to misattribute the walls and infer that there are actually two parallel corridors with a very thin wall between them. In our specific run, that does not seem to be the case since the recorded errors are too small.

Alg.	$0m/0^o$	$.25m/0^o$	$.5m/0^o$	$0m/15^o$	$.25m/15^o$	$.5m/15^o$	$0m/30^o$	$.25m/30^o$	$.5m/30^o$	Mean	# It.
base	0.000000	0.250000	0.500000	0.785398	0.824227	0.931048	1.570796	1.590566	1.648454	1.1751	n/a
NDT	0.000000	0.253762	0.519781	0.785398	0.843109	0.980248	1.356286	1.673229	1.677732	1.208638	1.894118
IDC	0.041257	0.063864	0.310380	0.069467	0.320919	0.389518	1.607720	1.586968	1.576797	1.029767	20.705882
MbICP	0.006066	0.119723	0.129221	0.037217	0.120134	0.135755	0.076706	0.661858	0.672813	0.420649	30.352941
WSM	0.011013	0.011527	0.152342	0.011694	0.052939	0.169485	0.012337	0.094252	1.103251	0.488799	17.482353

Just as in the first situation, *NDT* does not converge at all. The results are only included for completeness.

In this case, *WSM* and *MbICP* perform very similarly. However, we can see that *WSM* needs significantly fewer iterations on average to converge to its solution. As seen previously, *MbICP*'s correspondence rule also seems to work better for this situation. It is notable that the difference is not so significant this time as both methods seem to deal very well with matching to the corridor in general. The specific translation along the corridor however is exactly what gives rise to the *Correspondence Error*, which is explicitly dealt with in the *WSM* method. This is what might speed up that specific method so significantly. Accuracy, however, does not improve very significantly. *WSM* is more accurate than *MbICP* 5 out of 9 times. Though it is visible that *WSM* can be 2-10 times better than *MbICP*, while, conversely, *MbICP* can only be at most twice as good as *WSM*.

As a conclusion, *WSM* also shows to be best in this situation. It converges significantly faster than other methods, and displays either comparable or much better accuracy than its closest competitor.

Situation 3: Outdoor, very few features

This specific situation is very hard for any method because it displays such a small number of features. Since it is in an outdoor setting, both scans show many "out of range" values. This reduces the number of potential correspondences significantly. Also, the translation and rotation between both scans reduces the overlap, which again decreases the potential correspondences. Figure 7.1c clearly displays this difficulty.

Additionally, the very localized features create very significant local optima. With high noise, methods are more and more likely to find those solutions rather than the correct one.

Alg.	$0m/0^o$	$.25m/0^o$	$.5m/0^o$	$0m/15^o$	$.25m/15^o$	$.5m/15^o$	$0m/30^o$	$.25m/30^o$	$.5m/30^o$	Mean	# It.
base	0.000000	0.250000	0.500000	0.785398	0.824227	0.931048	1.570796	1.590566	1.648454	1.1751	n/a
NDT	0.000000	1.078081	0.489488	0.786170	0.831364	0.925397	1.570796	1.590607	1.709810	1.233846	2.176471
IDC	0.012821	0.012821	0.012821	0.936856	1.043303	0.967307	3.472407	2.858219	2.845654	1.935837	21.882353
MbICP	0.003299	0.006904	0.007047	0.006979	0.006708	0.073947	0.006394	1.545696	1.531521	0.944613	16.741176
WSM	0.001959	0.001959	0.001934	0.001926	4.378479	4.377351	1.198689	3.403532	7.742021	4.551092	17.329412

Again, the *WSM* and *MbICP* algorithms perform similarly in the lower noise levels. As soon as rotational noise is introduced, however, *WSM* performs very badly. Since its correspondence rule does not include a specific way to deal with rotation (unlike *MbICP*), it can not find good correspondences and gets stuck in local optima. Here, the advantage of *MbICP* is especially noticeable. It can quickly find reasonable correspondences, even with large rotational noise. *IDC* also seems more robust than *WSM* when dealing with rotational noise. This is because *IDC* also employs a special rule to identify rotational error.

In this situation, *MbICP* performs the best. It is just as fast as *WSM*, but much more accurate.

Situation 4: Indoor, many features, no translation

In this test situation, the scan of Situation 1 is used again. However, the second scan is discarded and the first is used in its place. This gives the impression that the robot did not move at all between both scans. All points in the scan thus correspond to each other exactly. Such an unlikely situation should facilitate the performance of point-based scan matchers significantly. Therefore, this particular situation serves as a best-case scenario. It will show how well each scan matcher can perform at all.

Alg.	$0m/0^o$	$.25m/0^o$	$.5m/0^o$	$0m/15^o$	$.25m/15^o$	$.5m/15^o$	$0m/30^o$	$.25m/30^o$	$.5m/30^o$	Mean	# It.
base	0.000000	0.250000	0.500000	0.785398	0.824227	0.931048	1.570796	1.590566	1.648454	1.1751	n/a
NDT	0.000121	0.191923	0.719698	0.785398	0.794804	1.004130	1.575583	1.601607	1.617779	1.187160	2.294118
IDC	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	51.941176
MbICP	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.470635	0.472485	0.289337	29.988235
WSM	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	24.494118

It is obvious that all point-based methods perform very well. *NDT* shows very bad performance again, but that is probably not because it is not point-based. In this particular case, *MbICP*'s correspondence rule does not seem to work as well as the one used in *IDC* or *WSM*. With high rotational noise, *MbICP* performs much worse than the others (yet still very well). A possible cause for this might be that the modified correspondence rule does not necessarily find the right correspondences. Wrong points may seem closer (meaning more likely to be the right corresponding point), which gives rise to local minima. The two best algorithms only differ in the number of iterations necessary to converge. *WSM* clearly outperforms *IDC* in this respect.

Clearly, *WSM* is generally better than all other methods in this best-case scenario. I performs perfectly, and in the least amount of iterations.

7.1.4 Implications

The *NDT* scan matcher does not seem to perform very well, it is always very close to the baseline. This is because it never does many iterations, most probably because or problems with the convergence criteria and the optimization step. This observation was already predicted in the introductory section and is visible consistently throughout all trials.

In very extreme situations, *MbICP* performs very well. However, in most settings, *WSM* performs best. It is consistently the fastest or one of the fastest methods, and always very accurate. However, it does not deal well with very sparse scans, like in Situation 3. Sparse scans give rise to many local optima which can be reported instead of the true global optimum.

We are mainly interested in fast and accurate methods, which work well with dense scans and minimal noise on the initial estimate. Before the competition, we were not aware of extensive outdoor areas where very sparse scans might be generated. With this in mind, we chose the *WSM* algorithm as our scan matcher.

7.2 ManifoldSLAM Evaluation

Our *ManifoldSLAM* implementation has been thoroughly tried and tested at the RoboCup of 2006. The results acquired in the Virtual Robots League will be discussed in detail in Section 7.2.1.

After the RoboCup, we also investigated the applicability of our system on real-world data. For this purpose we conducted a set of experiments where we apply *ManifoldSLAM* on the Cogniron data-set available on Radish [46]. In Section 7.2.2 we discuss the results obtained this data-set from [69].

7.2.1 RoboCup 2006 Results

The competition runs in the Virtual Robots League took place in simulated arenas with simulated robots and sensors. In this controlled setting, the exact ground truth can be made available. As a result, our results acquired at RoboCup 2006 can be compared to those obtained by our fellow competitors in detail.

In the following sections, the results obtained by our *ManifoldSLAM* implementation are compared to those of the other award winners. They are:

- First place: 'Rescue Robots Freiburg' from the University of Freiburg, Germany, from now on referred to as 'Freiburg';
- Second place: 'Virtual IUB' from the International University of Bremen, Germany, from now on referred to as 'IUB';
- Best Human-Computer Interface Award: 'Steel Team' from the University of Pittsburgh, USA, from now on referred to as 'Steel'.

Following this convention we will refer to our team in the diagrams with 'UvA'. We won two awards:

- We won the third place in the competition.
- We received the *Best Mapping Award*.

The scoring function, that was introduced in Section 2.1.2 and is also further explained in [1, 2], is used as the basis for our results analysis. All results are plotted per competition run, where we denote the three qualification runs with **Q1**, **Q2**, and **Q3** and the two semifinals with **Semi1** and **Semi2**. Where applicable, the number of robots deployed by our system is indicated in the diagrams between parentheses.

Overview

Like us, IUB and Freiburg had designed a fully autonomous system that was capable of supporting multiple robots. Steel also employed a multi-agent system, but in their approach an operator was guiding the team of robots. With the operator, Steel performed comparably or better on all individual rewards. However, the penalty for having an operator kept Steel below the other three teams.

In Figure 7.3 each team's total score per run is plotted. As can be observed, all teams have improved during the competition, but as it turns out this did not cause many changes in the daily rankings. The only time Freiburg did not rank first was in the first semifinals run, during which IUB and UvA were able to outperform them. In the second semifinals run, however, Freiburg performed best again while both IUB and UvA showed a drop in their scores. Steel showed a slow but steady rate of improvement throughout the competition and was able to surpass UvA in **Q1** and **Semi2**.

Figure 7.3 only plots the total scores acquired. In the following sections we further discuss the individual rewards: exploration, mapping and victims found.

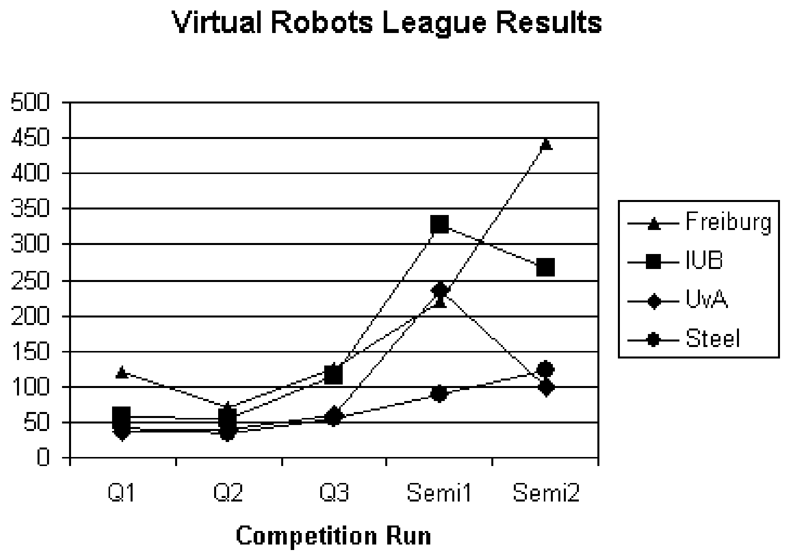
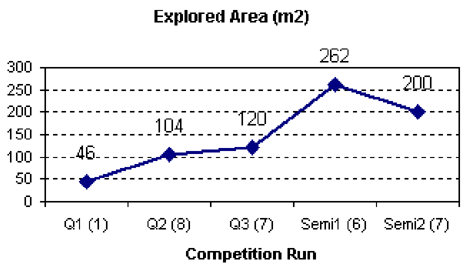


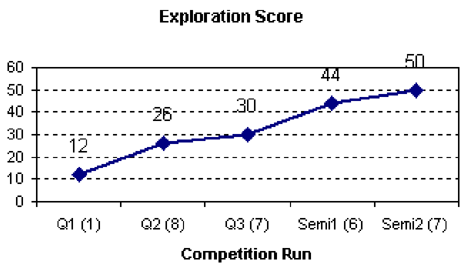
Figure 7.3: Total scores during the competition.

Exploration

On the first day, at the first qualification round, we did not support multiple agents yet. So we only deployed a single agent in this round. In the second qualification round we *did* have support for multiple agents and since then we have always deployed 6 to 8 robots.



(a) Explored Area



(b) Exploration Score

Figure 7.4: Exploration Results.

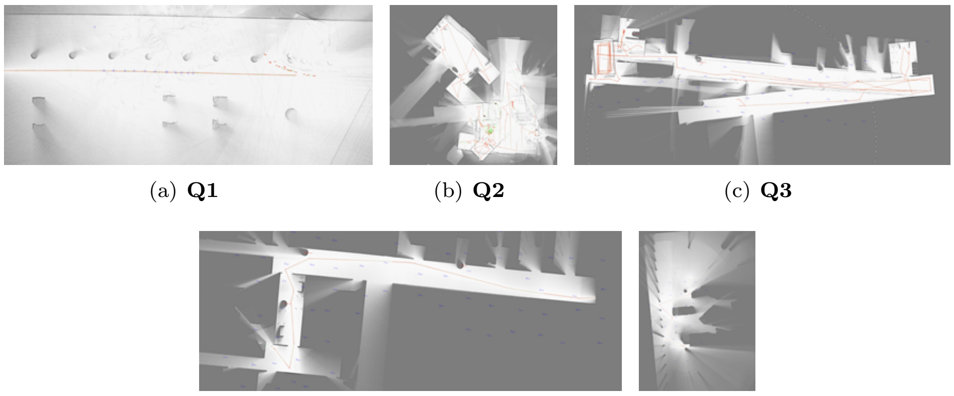
Note that for each run a target area of explored space was set. If that area or more was explored then the maximum score of 50 points was rewarded. This target area was much higher for the first semifinals run than it was for the second one. Therefore, even though we explored significantly more area in **Semi1**, the score was still lower than in **Semi2**.

The plots in Figure 7.4 show that the exploration as conducted by our team of robots improved in each single round. The increase between **Q1** and **Q2** is due to the jump in the number of robots deployed. After **Q2**, we have continuously further fine-tuned our behavior control and motion control specifications which caused the exploration to keep improving until we acquired the maximum score of 50 points in the second semifinals run.

Mapping

The mapping score was computed by taking the product of the automatically computed metric score and a manually assessed topological score. The metric score was based on the single-shot RFID report and would result in a multiplier between 0 and 1 that was used to scale the manual topological score. As the name indicates, the latter was assessed manually by a jury and would result in a score ranging from 0 to 50 points showing how well and to what quality the map represented the environment.

All the maps that we have submitted during RoboCup 2006 are listed in Figure 7.5.



(a) **Q1** (b) **Q2** (c) **Q3**

(d) **Semi1** (e) **Semi2**

Figure 7.5: The maps produced during the competition.

In Table 7.1, the acquired mapping scores based on these maps are displayed in detail. In the first qualification round no metric quality was computed, so the multiplier was fixed to 1 for all participants. The jury score is of particular interest as this earned us the *Best Mapping Award*, more details about this will follow. The mapping score is the product of the former two scores and institutes the mapping-related reward. The localization error was only computed after the qualification rounds and describes how far off the RFID position estimates were on average.

When reviewing Figure 7.5 in conjunction with Table 7.1 several observations can be made. To start with, the map of **Q1** is not too big as the robot only crossed the street from left to right. However, the map captures a lot of detail and does so at high precision.

	Metric Score	Jury Score	Mapping Score	Localization Error
Q1	(1.0)	24	24	-
Q2	0.4	21	8	-
Q3	-	32	-	-
Semi1	0.95	44	42	0.2 meter
Semi2	0.18	50	9	0.02 meter

Table 7.1: Map score details

Note that the laser range sensor is mounted at such a height that only the wheels of cars are observed. In **Q2** parts of the map are clearly misplaced, which caused the low metric score. In **Q3** there was a problem with our RFID report, so the metric score could not be computed. Although the map for **Q3** was not too bad, due to the missing multiplier we got 0 mapping points for that round. In **Semi1** we produced a very good map with only 20 centimeters localization error. And in the second semifinal we even had only 2 centimeters localization error, but because the map only covers a small fraction of the explored area, the metric score was only 0,18.

In Figure 7.6 we focus on the part of the mapping score that was assessed by the jury.

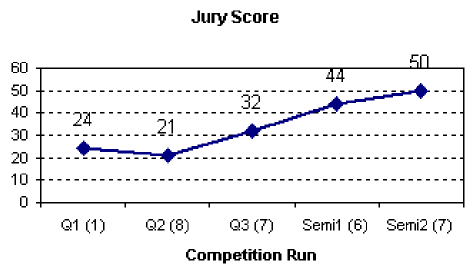
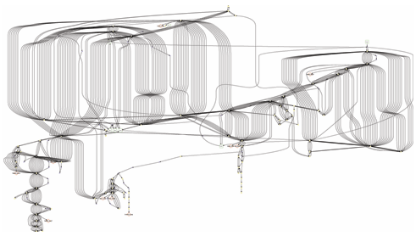


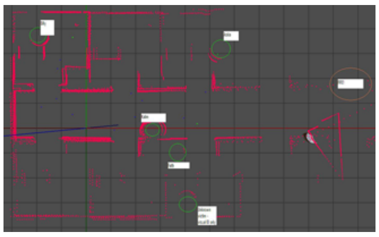
Figure 7.6: Jury assessment of our maps.

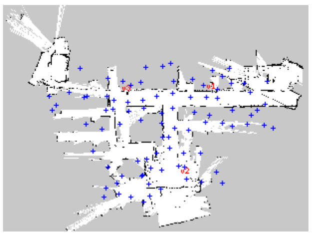
The diagram shows a steady increase in the evaluation of our maps by the jury. After **Q1**, a slight drop in **Q2** can be observed, but from **Q3** onward the jury continuously gave higher rewards for our maps in each run until we acquired the maximum score in **Semi2**. Balakirsky et al. [1] discuss the factors that have been taken into account when assessing a map. Among these are: *Accuracy* and *augmentation*.

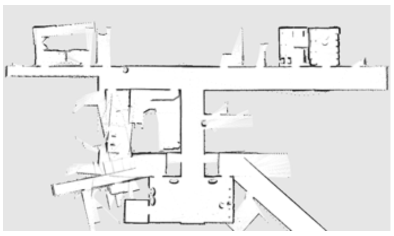
Mapping - Accuracy

According to [1] *accurate* maps should ease the job of human rescuers that want to use the map by accurately displaying features and augmented information. The maps produced by *ManifoldSLAM* exceed all other maps by far on this aspect. This difference was made possible mainly by supporting a grid resolution in our map that is an order of magnitude higher than those of our competitors. Therefore, the visualizations produced by *ManifoldSLAM* display what is known about the environment's geometry at an unmatched level of detail.


(a) Freiburg


(b) GROK


(c) IUB


(d) SPQR

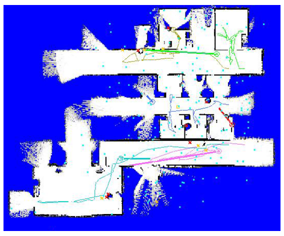

(e) Steel

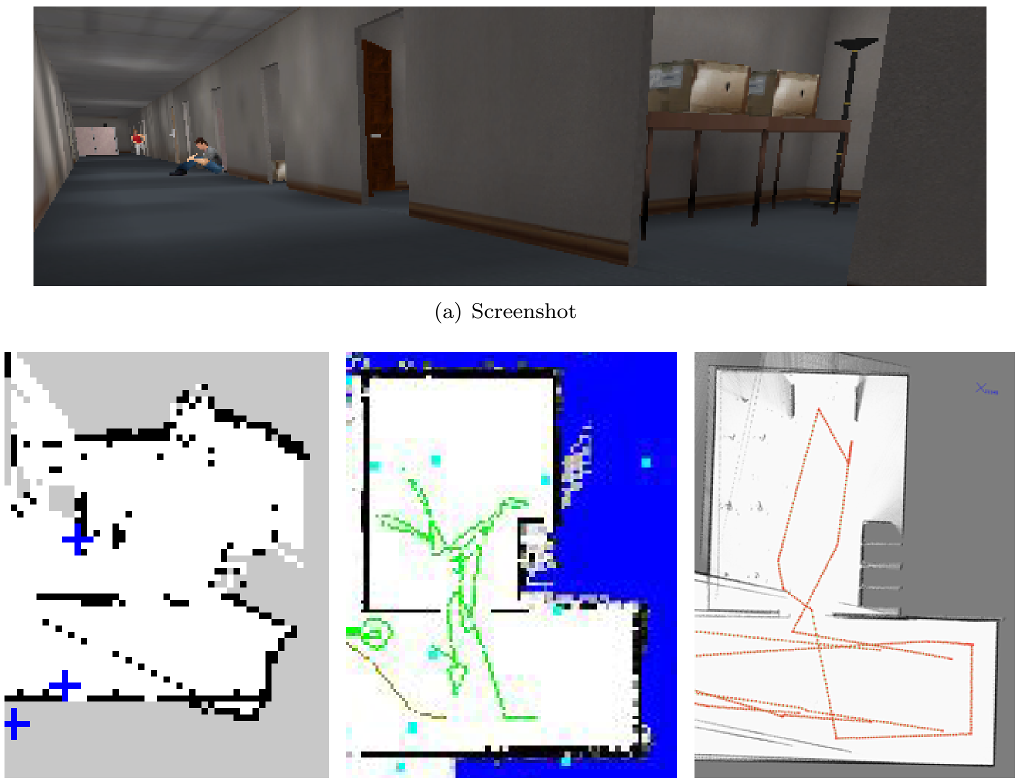
Figure 7.7: Typical maps produced by competitors.

Freiburg reported the environment's connectedness using a graph but did not report any geometric detail. GROK's binary occupancy grid was already much more capable of visualizing the environment's geometry, but did not distinguish between area that is known to be free and area that is yet unexplored. The occupancy grids from IUB, SPQR and Steel *do* make this distinction but still none of them provides the amount of detailed information that *ManifoldSLAM* is able to deliver.

The difference in accuracy is most clearly displayed in Figure 7.8. The same room has been mapped by IUB, Steel and UvA but only the map of UvA displays the table legs and the lamp post.

Mapping - Augmentation

Augmentation refers to the amount of information that is conveyed in the map in addition to geometry. Typical pieces of information to include in the map are victim locations and names, RFID tag positions and IDs, as well as traversed robot paths. Figure 7.9 shows the attributes included in maps by the different teams. SPQR did not augment their maps with any additional information, as can also be observed in Figure 7.7. All other teams report victims in the map, but only Freiburg, GROK and UvA plot their full names. All teams but GROK also marked observed RFID tags in the map, but only Freiburg and UvA also


(a) Screenshot

(b) IUB (c) Steel (d) UvA

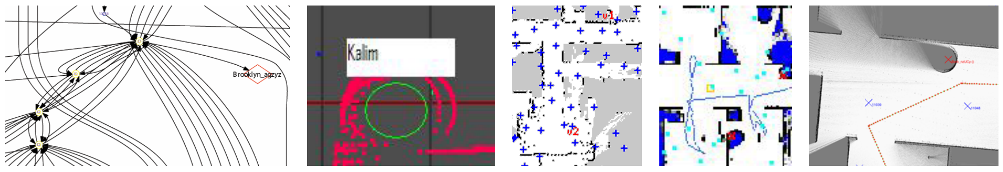
Figure 7.8: Map accuracy.

included the ID in the map. The robot trajectories were plotted only by Steel and UvA. All teams but SQPR applied some augmentation of information, but only our (UvA) maps featured *all* mentioned information.

Finding Victims

In Figure 7.4 a steady increase in exploration is illustrated. Naturally, with increasing amounts of explored areas, a team of robots is also likely to find and report more victims. In Figure 7.10 our acquired victims scores are presented.

In **Q1**, we found no victims and in **Q2** we found only one but also bumped into it. Therefore, we received a bump-penalty. This was addressed and after **Q2**, we never bumped into victims again. In **Q3** and **Semi1**, we found three victims, but in **Semi1** we also reported victim statuses and we received the full localization accuracy bonus for each victim. In **Semi2**, we found two victims which were reported with moderate localization accuracy.



(a) Freiburg (b) GROK (c) IUB (d) Steel (e) UvA

Figure 7.9: Map Augmentation.

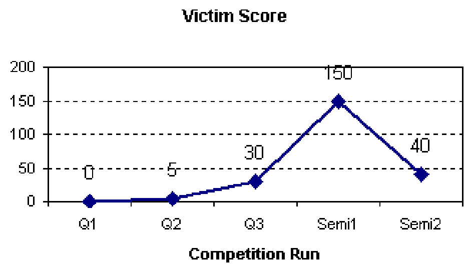


Figure 7.10: Victim score.

Cumulative Results

In 7.11 the individual rewards are plotted with the acquired total scores, where each total score is simply the sum over the rewards.

In a cumulative analysis we observe that the following aspects have been key to our achievements:

- The full autonomy of our system prevented the operator-penalty from being applied.
- The steady improvements in exploration directly contributed to our rankings and are primarily the result from the capability of our system to support up to 8 robots.
- The robust behavior control guided the exploration efforts and ensured that our agents have bumped into a victim only once in 29 individual trajectories that together have a running length of 9 hours and 40 minutes.
- The high accuracy and rich augmentation of our maps received increasing jury scores and earned us the *Best Mapping Award*.
- The maps also facilitated accurate localization of victims in the semifinals and caused a significant increase in the victim score.
- The combined effects of improving exploration and increasingly better maps earned us the third place in the competition.

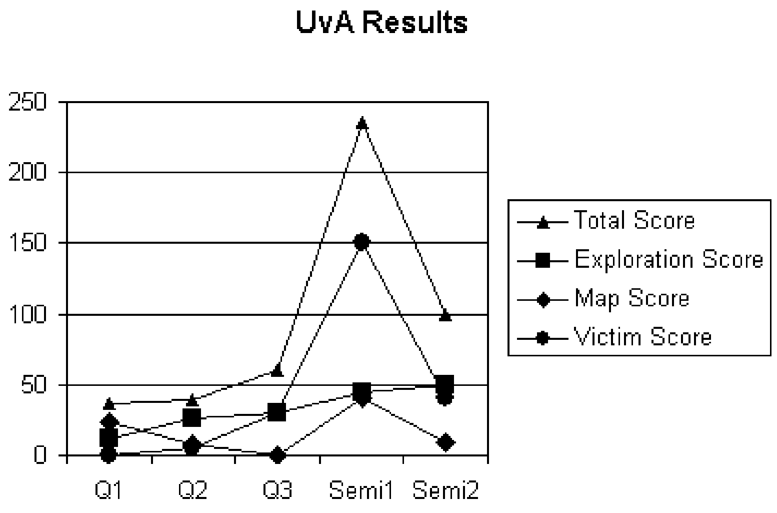


Figure 7.11: UvA scores during the competition.

7.2.2 Cogniron Results

Ben Kröse et al. published the Cogniron data-set as part of the IROS 2006 Workshop: 'From sensors to human spatial concepts' [69]. This data-set is also available on the Radish website [46].

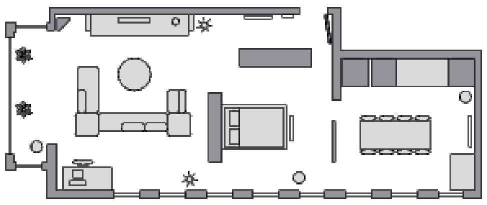
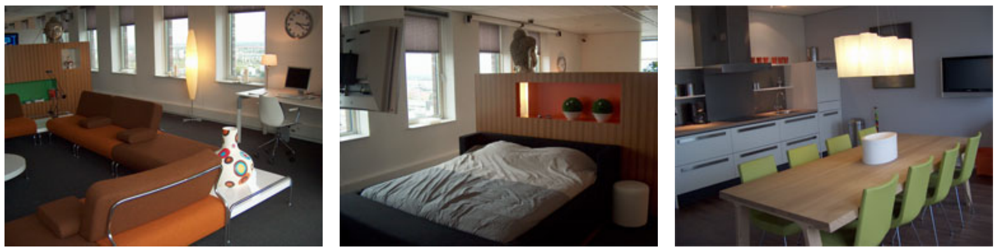


Figure 7.12: Schematic map of the Cogniron home. Courtesy of [69]

The data-set was acquired in a home environment as depicted in Figure 7.12. A Nomad Scout robot moved around through the environment while sensor data was recorded on a mounted laptop. The maps produced by *ManifoldSLAM* for this data-set are based exclusively on the laser range data. Range scans were recorded using a SICK LMS-200 laser scanner. The LMS-200 collected measurements with a field of view of 180 degrees while emitting a laser beam at every 0.5 degree, which resulted in 361 measurements per range scan. The sensor was mounted at a height of approximately 57 centimeters and had a maximum range of approximately 8 meters.

Three separate runs have been published and we show maps generated by *ManifoldSLAM* for each of them:



(a) living (b) bed (c) kitchen

Figure 7.13: Some pictures taken in the Cogniron home, from http://www.unet.nl

1. The first run is a relatively 'clean' run where no people were walking in the robot's field of view.

2. The second run was intentionally perturbed by having people walk in front of the robot. This run involves a number of loops.

3. The third run was acquired during a 'home tour' where someone took the robot on a guided tour through the room. Thus the guide is almost continuously in the robot's field of view.

With each of our maps we also present a visualization of the raw data that was published with the data-set. These visualizations are based on the raw odometry and laser range measurements. Note that *ManifoldSLAM* did not use the odometry data when constructing the maps. It should also be noted that our loop closure algorithm only works with victim observations that could be acquired in the USARSim simulator. We have not implemented any detection algorithm for other landmarks, so all the maps presented here were generated without explicitly closing loops.

No fine-tuning specific to one particular run was done, so the configuration of our system was identical for all runs. This enables a direct comparison of the generated maps.

Run 1 - Static Obstacles Only

The map captures a very large amount of detail. The couches in the living room, the chairs in the kitchen and the observable part of the bed are all clearly and sharply visualized. Also, much smaller details are neatly preserved, like the legs of the kitchen table, the decorated vase, the trunks of the plants in the living room and also the base of the lamp halfway at the bottom (see Figure 7.13).

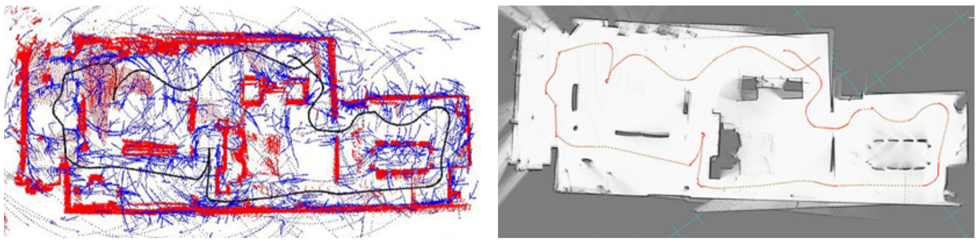

(a) raw data (b) map

Figure 7.14: Cogniron run 1.

Run 2 - Three Loops

The robot started in the kitchen, then traversed the smaller loop in the living room followed by the loop through the kitchen and ends with one and a half loop around the living room and bed together. Despite these loops, our system is still able to build a highly consistent map. Some ghosting of the exterior wall can be observed on the far left and in the bottom-left corner of the living room. Also one of the couches is displayed twice and the piece of furniture above the bed shows some ghosting.

However, other than that the map is of comparable quality to the one of the first run. The kitchen, the bed and the bottom couch are visualized sharply and a great amount of additional detail is still accurately preserved. For example the legs of the kitchen table and of the computer desk in the bottom-left corner and also the base of the lamp at the bottom-center are clearly visible.

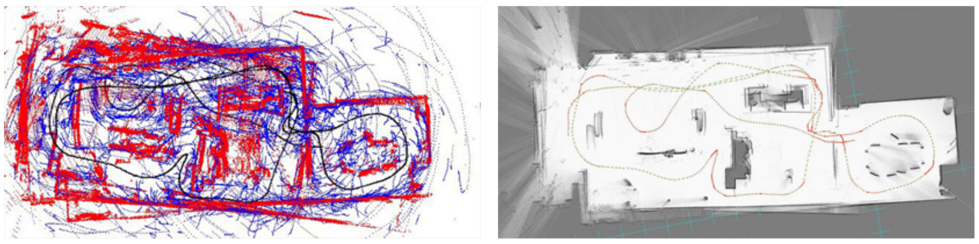

(a) raw data (b) map

Figure 7.15: Cogniron run 2.

Run 3 - Home Tour

This run was the most challenging one as a dynamic obstacle, i.e. the guide, is continuously affecting the range scan measurements. The robot started at the top and made one big loop in counter-clockwise direction. It can be observed that our system finds a local optimum only once, somewhere near the upper-left corner of the map. Obviously, the upper wall and also several measurements of the cupboard above the bed and measurements of the couches are rotated by several degrees.

The remainder of the map however, from the upper-left corner onward, is very consistent. The bottom wall is almost perfectly straight and certainly the straightest of all runs. Additionally, similar crispness and detail as observed in the previous maps is also exhibited here.

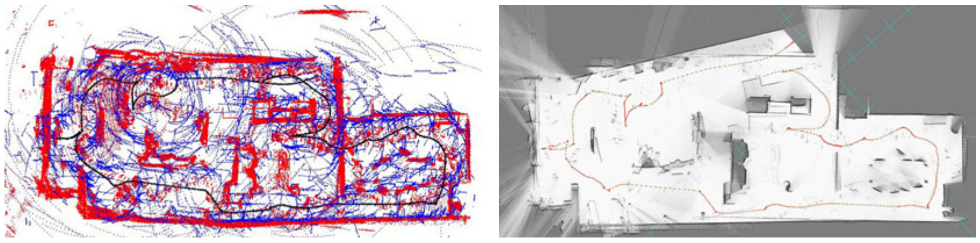

(a) raw data (b) map

Figure 7.16: Cogniron run 3.

Analysis

In this set of experiments we applied *ManifoldSLAM* on data that was acquired on a real robot using real sensors that were affected by real noise. The applicability of our system to such real-world data is illustrated with results acquired on three different runs. Each run posed different challenges to overcome, but regardless of these our system produced consistent, accurate and detailed maps for each of them. Note that no explicit loop closing was performed to generate these maps.

The maps are topologically consistent and clearly indicate walls and doorways. The maps acquired for the first two runs match the schematic map displayed in 7.12 quite well, despite the number of loops in the second one. In the map of the third run, a small part of the living room is rotationally off, but the majority of the map's topology compares to the former two.

The kitchen with the chairs is very clearly rendered in all three maps. Also the couches and the observable part of the bed are often rendered without ghosting. When observed, smaller details are usually preserved very sharply.

All three maps preserve a large amount of detail, like individual table legs and the trunks of the plants in the living room.

Chapter 8

Related Work

8.1 Comparison to the State of the Art

The SLAM approach presented in this paper shows a certain resemblance to the Atlas framework that was originally presented by Bosse et al. [7] and applied in [8] and has been discussed in Section 3.4.2. Both Atlas and *ManifoldSLAM* use a graph structure to decompose the global map in small-scale local maps. Thus, both approaches share several benefits like the low update costs, the capability to map large environments and the fact that the online effects of error accumulation are limited to the extent of a local map only. Note though that in *ManifoldSLAM* a local sub-map is a dynamic concept whereas in Atlas the extent of a local map remains fixed once defined. A much more significant difference can be observed in the way potential false data associations are addressed. Atlas employs an elaborate strategy that tracks multiple concurrent hypotheses for candidate closure points but still closes loops in the map as soon as they are discovered. *ManifoldSLAM* on the other hand simply refrains from loop-closing until enough certainty is acquired. Both approaches work fine, but we believe that the multi-hypothesis tracking in Atlas has a significant impact on the online performance of the SLAM algorithm and that this is the key motivation for Atlas to only use a limited amount of features to match observations. *ManifoldSLAM* does not suffer from this limitation and has been implemented using range scans containing 361 'point features', to use Atlas terminology. The key advantage is that *ManifoldSLAM* is not biased towards particular environments, while Atlas is strongly biased to the particular features employed in the local maps. For example, Bosse et al. [7] propose to use line segments which would strongly bias Atlas to regular-shaped indoor environments. On the other hand, the Weighted Scan Matcher, which *ManifoldSLAM* uses, has been shown to operate indoors as well as outdoors when matching point clouds. Another aspect to note is that *ManifoldSLAM* inherently supports mapping multi-floor environments because of the way the graph structure is employed. In Atlas, this feature depends on the kind of used features and local maps. The consequence is that *ManifoldSLAM* enables mapping multiple floors while still using inherently two-dimensional algorithms, while in Atlas, the map matching and hypothesis tracking algorithms would have to take all spatial dimensions into account explicitly. Also it is unclear how Atlas would scale to support multiple agents because the map matching seems to rely on *known* relative positions.

The design of FastSLAM [38] is driven by the observation that, given the robot's path, the landmark observations are conditionally independent, see also Section 3.4.1. In effect, this decomposes the joint space of landmark estimates in path estimates. *ManifoldSLAM* exhibits a similar decomposition as the Manifold data structure stores all observations as independent estimates with respect to the patches, i.e. the robots' trajectories. However, while the Manifold is a hybrid approach that really decomposes the map in small-scale maps, FastSLAM remains a holistic approach and the decomposition is only the result of a factorized formulation of the global map probability. As a result, although FastSLAM has been extended in a number of papers [40, 37, 42, 25, 22], it remains an approach that assumes planar environments. A direct extension to incorporate the capability to map multi-floor environments like *ManifoldSLAM* would result in having the particles of the particle filter maintain the robot trajectory and landmark positions in 3D. The performance impact of such an extension is an open question. Also, FastSLAM has yet to be applied in a multi-agent setting. It is unclear how multiple agents whose relative position is unknown would contribute to the same map. They would probably maintain individual instances but then an appropriate merging operation remains to be defined.

The concept of the Manifold data structure was inspired by Howard et al. [27]. In our approach, the same benefits and strengths can be observed. On the implementation level though, we might indicate a minor difference as we have decided to store uncertainty information as covariance matrices on the relations, while Howard et al. [27] seem to hint at a single information matrix that stores all uncertainty information globally. A much more significant difference though concerns the scalability of the *ManifoldSLAM* implementation versus the implementation discussed by Howard et al.. In their paper, the authors illustrate an experiment that was conducted with 4 robots. The algorithm was run online, but as commented in [27], Section V - Experiments:

> *"During island merging or loop closure, however, it is necessary to halt the robots for some tens of seconds to allow the mapping process to 'catch up'."*

and in Section VI - Conclusion and Further Work:

> *"The algorithm, as presented ... scales poorly with increasing team size (the largest experiment conducted to date employed just four robots)."*

ManifoldSLAM also uses a centralized process, the Mapping Server (see Section 6.3.2 for details), and as emphasized in [2] the system has run several competition runs with up to 8 agents. This is most probably due to the use of the Weighted Scan Matcher of Pfister et al. [44] in our approach. Howard et al. on the other hand refer to work by Lu and Milios [32] which builds on the IDC framework presented in [33] and is also discussed in Section 4.3.2. In an extensive set of experiments that are specific to our challenge (see Section 7.1), we have reproduced similar findings to those reported by Pfister et al. [44]. In short, the Weighted Scan Matcher has proven to be more accurate and to perform faster than the IDC algorithm of Lu and Milios. This led to our observation that we can distinguish between local accuracy and global accuracy, see also Section 5.5. This was translated into a decomposition of the loop-closing and island-merging algorithms where the map-refitting is postponed until visualizations are requested.

8.2 Competitors at the RoboCup

The RescueRobots Freiburg team from the University of Freiburg acquired first place in our league. The aspect of their system that clearly made the difference was the employed exploration strategy based on RFID tags that the robots deployed by themselves [29]. Following the multi-agent frontier-based exploration strategy discussed in [68] and [70], the Freiburg team had designed a system that showed superior exploration performance during the competition. Good exploration naturally leads to finding a lot of victims as well. However, the energy invested in exploration was apparently lost on the mapping part. Only in the second semifinals run and in the finals did their system produce maps. Until then their ranking had been based solely on exploration and victim reports.

Virtual IUB of the International University of Bremen was the first runner up. Their system also employed a frontier-based exploration strategy. However, their implementation did not coordinate the joint actions of the team, but each agent would evaluate the frontier independently and execute an individual plan. Especially when multiple agents were deployed from the same entrance point they would take the same greedy decisions, move to the same destinations and become dynamic obstacles to each other. The frontiers were constructed on a grid-map that was produced by an off-the-shelf SLAM approach [22] that is basically an extension to FastSLAM. Note that although Virtual IUB employed a multi-agent system, the mapping and localization did not truly support multiple agents. Each robot worked independently of the others and after competition runs, map merging was done by simply overlaying maps from individual robots based on the known starting position of each robot.

The primary focus of the Steel team from the University of Pittsburgh was on human computer interaction. Their systems was equipped with an extensive user interface that allowed a single operator to control a complete team of rescue robots. During the competition runs the operator was usually able to find a lot of victims and the team also showed good exploration and mapping scores. However, the applied penalty for manually controlling the robots caused the team to rank lower than the top three which were all using fully autonomous systems. The Steel team won the special award for having the best human-computer interface.

When we compare our system with those of the other prize winners, we obviously had superior mapping software, since we won the special award in this field. The comparison to maps of all other teams is further detailed in Section 7.2.1. The RescueRobots Freiburg team only delivered maps that were strictly topological and did not display any geometric features. Virtual IUB and the Steel Team used occupancy grids for online SLAM. Consequently, they had to constrain the resolution of the map to what the software could handle during online execution. Our system on the other hand only produced a complete occupancy grid in a post-processing step, so we could optimize the resolution to the maximum of what our rendering algorithm could handle in the 10 minutes time limit. Usually we would deliver maps with a resolution that was an order of magnitude higher than those of our competitors.

Interestingly, of all teams in our league, only Freiburg's approach and our approach appear to be ready for multi-floor environments. Freiburg's approach relies on a graph structure which makes no assumptions about planarity. All team-coordination is based on this graph and the frontiers are only defined locally on the graph nodes. The team from Bremen uses a system that is based on work by Grisetti et al. [22] and the Steel Team's mapping software is based on Carmen [39]. Carmen includes a mapping module that builds on [24]. Both systems thus rely on occupancy grids that is inherently planar.

The other aspects of our system that enabled us to win the third position in our league is the fact that it operates fully autonomously and can handle a large number of agents. With the former feature, we avoided the operator penalty that for example the Steel team suffered from. With the latter one, we were able to still explore reasonably well and therefore find a good number of victims. The key limitation of our system concerns the behavior control. Although our behavior control has proven to be very robust, the map-based greedy behavior as employed by the teams from Freiburg and Bremen significantly outperformed our strictly reactive behavior.

Chapter 9

Conclusions and Future Work

9.1 Concluding Remarks

An award-winning SLAM system is presented in this thesis. We designed and implemented a multi-agent system that successfully addresses the challenges faced by rescue robots in an Urban Search and Rescue setting. Using the presented system, we participated in the Rescue Virtual Robots league in the RoboCup World Championships, held in Bremen in June 2006. We won third place in our league and also received the *Best Mapping Award*.

Based on the Manifold concept presented by Howard et al. [27], we implemented a sophisticated data structure that we describe in Section 5.3. This data structure supports online localization and mapping, loop closing and island merging. In Chapter 3, we survey the state of the art in SLAM research and we discuss the individual strengths and weaknesses of the different approaches. We based our work on [27] as we intended our approach to address the challenges faced by rescue robots in the best possible way. This choice is supported by a detailed explanation in Section 5.2. In Chapter 8, we compare and contrast our approach with those reviewed in Section 3.4 and those of our competitors in the RoboCup Rescue Virtual Robots League. By combining the strengths of topological and metric maps in a layered data structure, our system is able to deal effectively with the challenges we envision in Chapter 2 and meet the objectives presented in Section 1.2.

The mapping and localization algorithms in our system rely on the Weighted Scan Matcher (WSM) that was introduced by Pfister et al. [44]. In their paper the authors show that WSM outperforms the well-known IDC scan matcher from Lu and Milios [34, 32]. We conducted an extensive set of experiments designed specifically for our field of application (the simulated USAR setting) with a broader scope than the ones performed in [44]. These experiments also included the recent MbICP scan matcher from Minguez et al. [36]

and the Normal Distribution Transform (NDT) by Biber and Straßer [5]. In Section 7.1, we conclude that in our domain, where dense range scans are usually available, the WSM outperforms IDC, MbICP and NDT in both speed and accuracy. This implies that the WSM algorithm is the best choice for us and we thus also confirm the findings in [44].

By combining the best scan matcher for our domain with a hybrid data structure in *ManifoldSLAM*, we have constructed a state-of-the-art SLAM approach with which we successfully competed in the RoboCup Rescue Virtual Robots League. As can be verified in Section 7.2.1, the following aspects of our system were key to our achievements: The highly accurate and detailed mapping capabilities, the system's scalability, and the fully autonomous and robust behavior control implementation.

The maps produced by *ManifoldSLAM* compare favorably to those delivered by our competitors at RoboCup, see also Section 7.2.1. Our system does not impose limitations on the amount of data preserved in our maps during competition runs. This is contrary to the grid-based SLAM approaches used by the teams from University of Pittsburgh and International University Bremen, for example. Therefore, we were able to capture an amount of detail in our maps that was unparalleled in our league. Basically, we were able to provide maps at arbitrary high resolutions and we were only limited by the maximum resolution that our software could handle in the 10 minutes time limit.

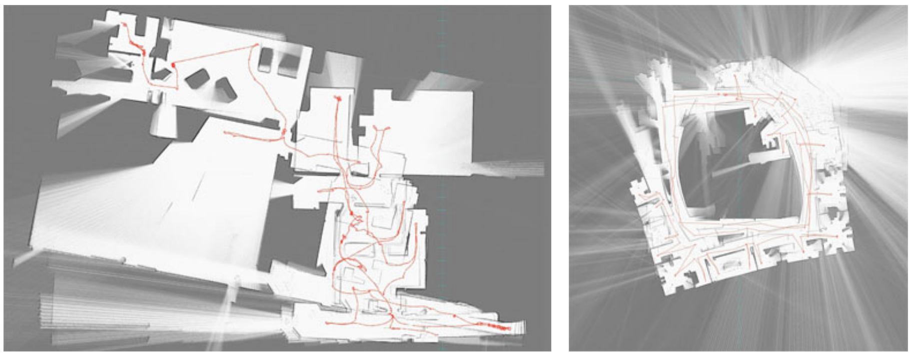
In Section 5.5, we describe the decomposition of the loop-closing and island-merging algorithms in an online part and an offline part implemented in *ManifoldSLAM*. This significantly increased the online performance and hence the scalability of our system. *ManifoldSLAM* has demonstrated support for up to 8 robots that simultaneously explored the arena and searched for victims. In Section 7.2.1, we show that the exploration exposed by our robot team improved every day during the competition until the maximum score was achieved in the second semi-final. Thus, the exploratory capabilities of our system served as a constant source of rewards for our scoring.

The behavior control that steers the exploration efforts and our robots also proves to be of great value in other respects. In Section 7.2.1, we display our victim finding rewards and also the contribution of individual rewards to our total scores for every run. This shows that especially in the semi finals the received victim finding rewards ensured us a third place in the competition. Our behavior control has done an excellent job in securing us from penalties, which was of great value throughout the competition. In 29 individual runs of 20 minutes each, our robots only bumped into a victim once. Moreover, the full autonomy of our robot team ensured that any rewards were not discounted by the severe operator penalty.

While *ManifoldSLAM* has proven to be able to produce high quality maps under the simulated conditions at the RoboCup, in Section 7.2.2 we show that our system can also be applied on real-world data. We demonstrate that *ManifoldSLAM* can produce maps from raw laser range data that suffers from real-world odometric error and sensor noise which exhibit similar levels of accuracy and detail.

9.2 Future Work

We are in the process of acquiring more real-world results in order to further investigate and improve the applicability of *ManifoldSLAM* on real-world data. In an upcoming special issue of the Elsevier Robotics and Autonomous Systems Journal about the 2006 RoboCup Rescue Virtual Robots competition, we intend to demonstrate the applicability of our system to real-world settings by publishing results on several popular data-sets that are available on the Radish website [46]. Below are some preliminary results acquired on the 'AP Hill' and 'Intel Lab' data-sets.



(a) AP Hill (b) Intel Lab

Figure 9.1: Preliminary results on real-world data-sets

Note that both maps involve a lot of loops which we are closed as we currently only support loop closure based on victim observations in the simulator. Considering the accurate visualization of individual rooms in 'AP Hill' (Figure 9.1a) and the relatively small accumulation of error in the 'Intel Lab' (Figure 9.1b) over the three large consecutive loops, the results are very encouraging.

In the context of future RoboCup competitions, several desirable improvements can be listed. First and foremost, the key limitation that prevented us from achieving a higher ranking this year was the lack of a greedy exploration strategy. The patches of the Manifold lend themselves excellently for a frontier-based exploration strategy as presented in [68, 70] or a RFID-based exploration strategy as presented in [29].

A similar improvement, but from a broader perspective, would be to implement planning on the Manifold. This idea builds on the observation that SLAM and exploration are opposing forces: For better maps it is usually preferable to acquire more certainty about the same environment through additional observations while exploration aims at observing yet unobserved parts of the environment. Reinforcement Learning techniques like Markov Decision Processes (MDPs) [49, 3] seem to be the ideal framework for planning on the Manifold on as they integrate nicely with its graph-based map structure. Localization within the Manifold to a specific patch allows for state observations. Actions such as *move to a neighboring patch* could be formulated. This gives rise to a very intuitive definition of planning on the Manifold as an MDP. There exist efficient methods to solve such Reinforcement Learning problems which can also be applied online [49]. In addition, with further examination of patches it would be possible to identify frontiers within them. Such features, along with potential victim observations and other sensory inputs, can then be used to define a reward function. For example: A positive reward would be given if frontiers would become observable as they might lead to an increase in the amount of explored area. Moving too close to a victim might imply a negative reward. This way we could formulate the problem of the RoboCup Rescue Virtual Robots League in a few simple positive and negative goals, and the planning algorithms would control the robot such that it pursues or avoids these goals. This behavior would be vastly superior to reactive approaches since it would be able to take the current environment and possible futures into account.

Another straight forward extension of the system would be to add the third dimension in our map. A 3D scan matcher would be all that is needed to compute 3D maps, all other parts of *ManifoldSLAM* can be reused without changes. The RoboCup Rescue Virtual Robots simulator, USARSim, already includes a 3D range scanner sensor that could be used for this purpose. This sensor is modeled after existing sensors such as the Swiss Ranger 3D camera and 2D range scanners which are mounted in a way to tilt up and down. Including the third dimension in maps will be necessary in future competitions as the environments are bound to get more complex and unstructured. Proper obstacle avoidance may have to include obstacles that extend from the ceiling, which are significant bumps on the floor, or that do not have straight sides. Consequently, maps will have to capture such complex environments properly. Also, Balakirsky et al. [1] mention a possible future requirement for participants to include an extensive 'data sheet' in the victims report that enumerates all hazards encountered en route. It is yet unclear what kind of hazards are meant here, but the organizers of our league have proven be very creative. This would add another argument in favor of incorporating more detailed and potentially three-dimensional obstacle information in the map.

Another beneficial extension would be to implement the refitting of the Manifold as a global optimization instead of the current piece-wise locally optimal implementation. A suitable function which represents the fit of the complete map can be constructed from the scan matching results that are stored on the graph links. Such a function would take all patch poses as free parameters and evaluate the quality of fit, possibly on the basis of a Gaussian function for which we already have a covariance matrix. This way, it would be straightforward to compensate for errors in directions where the initial scan matching process was uncertain as the Gaussian would indicate those directions. Since closed form solutions for first and second derivatives from Gaussian functions are readily available, efficient optimization methods similar to the NDT scan matching method discussed in Section 4.3.1 could be used.

Alternatively, loop closure, island merging and related operations can be done using a different method of global optimization. Olson et al. [43] present a nonlinear optimization method for pose graphs using exactly the same uncertainty representation as in *Manifold-SLAM*. Their work is motivated by work on the Atlas SLAM framework (described in Section 3.4.2). Their method seems very efficient and accurate, although implementing it may be laborious. It would be a nice task for the future as it would presumably significantly improve map alignment and achieve loop-closing and island-merging in one operation.

Bibliography

[1] S. Balakirsky, C. Scrapper, S. Carpin, and M. Lewis. Usarsim: providing a framework for multi-robot performance evaluation. In *Proceedings of PerMIS 2006*, 2006.

[2] Stephen Balakirsky, Stefano Carpin, Alexander Kleiner, Michael Lewis, Arnoud Visser, Jijun Wang, and Vittorio Amos Ziparo. Towards heterogeneous robot teams for disaster mitigation: Results and performance metrics from robocup rescue. *Journal of Field Robotics*, Special Issue on Teamwork in Field Robotics, (submitted).

[3] D. P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 2nd, vols. i and ii edition, 2001.

[4] Paul J. Besl and Neil D. Mckay. A method for registration of 3-D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, February 1992. ISSN 0162-8828. doi: 10.1109/34.121791. URL `http://dx.doi.org/10.1109/34.121791`.

[5] Peter Biber and Wolfgang Straßer. The normal distributions transform: A new approach to laser scan matching. In *IEEE/RJS International Conference on Intelligent Robots and Systems*, 2003.

[6] Andreas Birk, Holger Kenn, Stefano Carpin, and Max Pfingsthorn. Toward autonomous rescuerobots. In *First International Workshop on Synthetic Simulation and Robotics to Mitigate Earthquake Disasters*, 2003.

[7] M. Bosse, P. Newman, J. Leonard, and S. Teller. An atlas framework for scalable mapping. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2003.

[8] M. C. Bosse, P. M. Newman, J. J. Leonard, and S. Teller. Simultaneous Localization and Map Building in Large-Scale Cyclic Environments Using the Atlas Framework. *The International Journal of Robotics Research*, 23(12):1113–1139, 2004.

[9] S. Carpin, J. Wang, M. Lewis, A. Birk, and A. Jacoff. High fidelity tools for rescue robotics: Results and perspectives. In *Proceedings of the 2005 RoboCup Symposium*, 2005.

[10] S. Carpin, M. Lewis, J. Wang, S. Balakirsky, and C. Scrapper. Bridging the gap between simulation and reality in urban search and rescue. In *RoboCup 2006: Robot Soccer World Cup X*, LNAI. Springer, 2006.

[11] J.A. Castellanos and J.D. Tardos. *Mobile Robot Localization and Map Building: A Multisensor Fusion Approach.* Kluwer Academic Publishers, Boston, MA, 2000.

[12] Howie Choset and Keiji Nagatani. Topological simultaneous localization and mapping (slam): Toward exact localization without explicit localization. *IEEE Transactions on Robotics and Automation*, 17:125–137, 2001.

[13] Thomas M. Cover and Joy A. Thomas. *Elements of information theory.* Wiley-Interscience, New York, NY, USA, 1991. ISBN 0-471-06259-6.

[14] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Matematik*, 1:269–271, 1959.

[15] G. Dissanayake, P. Newman, S. Clark, H.F. Durrant-Whyte, and M. Csorba. A solution to the simultaneous localisation and map building (slam) problem. *IEEE Transactions of Robotics and Automation*, 2001.

[16] Mauro Dragone, Ruadhan O'Donoghue, John J. Leonard, Gregory O'Hare, Brian Duffy, Andrew Patrikalakis, and Jacques Leederkerken. Robot soccer anywhere: Achieving persistent autonomous navigation and mapping and object vision tracking in dynamic environments. In *Proceedings of SPIE Opto Ireland*, Dublin, Ireland, April 2005.

[17] H. Durrant-Whyte, S. Majumder, S. Thrun, M. de Battista, and S. Scheding. A bayesian algorithm for simultaneous localization and map building. In *Proceedings of the 10th International Symposium of Robotics Research (ISRR01)*, 2001.

[18] A. Elfes. Sonar-based real-world mapping and navigation. In *IEEE Transactions on Robotics and Automation*, pages 249–465, 1987.

[19] D. Ferguson, A. Morris, D. Hähnel, C. Baker, Z. Omohundro, C. Reverte, S. Thayer, W. Whittaker, W. Whittaker, W. Burgard, and S. Thrun. An autonomous robotic system for mapping abandoned mines. In S. Thrun, L. Saul, and B. Schölkopf, editors, *Proceedings of Conference on Neural Information Processing Systems (NIPS).* MIT Press, 2003.

[20] U. Frese. A discussion of simultaneous localization and mapping. *Autonomous Robots*, 20(1):25–42, 2006.

[21] Erich Gamma, Richard Helm, and Ralph Johnson. *Design Patterns. Elements of Reusable Object-Oriented Software.* Addison-Wesley Professional Computing Series. Addison-Wesley, 1995. GAM e 95:1 1.Ex.

[22] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2005.

[23] J. Gutmann and K. Konolige. Incremental mapping of large cyclic environments. In *Proceedingsof the IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA)*, pages 318–325, Monterey, California, 1999.

[24] D. Hähnel, D. Schulz, and W. Burgard. Map building with mobile robots in populated environments. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2002.

[25] D. Hähnel, D. Fox, W. Burgard, and S. Thrun. A highly efficient fastslam algorithm for generating cyclic maps of large-scale environments from raw laser range measurements. In *Proceedings of the Conference on Intelligent Robots and Systems (IROS)*, 2003.

[26] Andrew Howard, Lynne E. Parker, and Gaurav S. Sukhatme. Experiments with large heterogeneous mobile robot team: Exploration, mapping, deployment and detection. *International Journal of Robotics Research*, 25(5):431–447, May 2006.

[27] Andrew Howard, Gaurav S. Sukhatme, and Maja J. Matarić. Multi-robot mapping using manifold representations. *Proceedings of the IEEE - Special Issue on Multi-robot Systems*, 2006.

[28] R. E. Kalman. A new approach to linear filtering and prediction problems. *Trans. ASME, Journal of Basic Engineering*, 82:35–45, 1960.

[29] A. Kleiner, J. Prediger, and B. Nebel. Rfid technology-based exploration and slam for search and rescue. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2006 (to appear).

[30] K. Konolige, D. Fox, C. Ortiz, A. Agno, M. Eriksen, B. Limketkai, J. Ko, B. Morisset, D. Schulz, B. Stewart, and R. Vincent. Centibots: Very large scale distributed robotic teams. In *International Symposium on Experimental Robotics (ISER-04)*, 2004.

[31] B. Lisien, D. Morales, D. Silver, G. Kantor, and H. Rekleitis, I. andChoset. The hierarchical atlas. *IEEE Transactions on Robotics and Automation*, 21:473–481, 2005.

[32] F. Lu and E. Milios. Globally consistent range scan alignment for environment mapping. *Autonomous Robots*, 4:333–349, 1997.

[33] Feng Lu and E. Milios. Robot Pose Estimation in Unknown Environments by Matching 2D Range Scans. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 1994. Proceedings CVPR '94., 1994*, pages 935–938, 1994. URL `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=323928`.

[34] Feng Lu and E. Milios. Robot pose estimation in unknown environments by matching 2D range scans. *Journal of Intelligent and Robotic Systems*, 18:249–275, 1997.

[35] M. Maimone, A. Johnson, Y. Cheng, R. Willson, and L. Matthies. Autonomous navigation results from the mars exploration rover (mer) mission. In *9th International Symposium on Experimental Robotics (ISER)*, June 2004.

[36] J. Minguez, L. Montesano, and F. Lamiraux. Metric-based iterative closest point scan matching for sensor displacement estimation. *IEEE Transactions on Robotics*, 22(5): 1047–1054, October 2006. URL `http://webdiis.unizar.es/~jminguez/MbICP_TRO.pdf`.

[37] M. Montemerlo and S. Thrun. Simultaneous localization and mapping with unknown data association using FastSLAM. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 2, pages 1985–1991. IEEE, September 2003.

[38] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. Fastslam: A factored solution to the simultaneous localization and mapping problem. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, Edmonton, Canada, 2002. AAAI.

[39] M. Montemerlo, N. Roy, and S. Thrun. Perspectives on standardization in mobile robot programming: The carnegie mellon navigation (CARMEN) toolkit. In *Proceedings of the Conference on Intelligent Robots and Systems (IROS)*, 2003.

[40] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. Fastslam 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI)*, Acapulco, Mexico, 2003. IJCAI.

[41] Hans Moravec. Sensor fusion in certainty grids for mobile robots. *AI Magazine*, 9: 61–74, 1988.

[42] J. Nieto, J. Guivant, E. Nebot, and S. Thrun. Real time data association for fastslam. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2003.

[43] Edwin Olson, John Leonard, and Seth Teller. Fast iterative optimization of pose graphs with poor initial estimates. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2262–2269, 2006.

[44] Samuel T. Pfister, Kristo L. Kriechbaum, Stergios I. Roumeliotis, and Joel W. Burdick. A weighted range sensor matching algorithm for mobile robot displacement estimation. *IEEE Transactions on Robotics and Automation, submitted for publication*, 2006.

[45] M. E. Pollack, S. Engberg, J. T. Matthews, S. Thrun, L. Brown, D. Colbry, C. Orosz, B. Peintner, S. Ramakrishnan, J. Dunbar-Jacob, C. McCarthy, M. Montemerlo, J. Pineau, and N. Roy. Pearl: A mobile robotic assistant for the elderly. *AAAI Workshop on Automation as Eldercare*, 2002.

[46] Radish. The robotics data set repository, -. URL `http://radish.sourceforge.net`.

[47] R. Smith, M. Self, and P. Cheeseman. Estimating uncertain spatial relationships in robotics. *I.J. Cox and G.T. Wilfong, editors, Autonomous Robot Vehnicles*, pages 167–193, 1990.

[48] R. C. Smith and P. Cheeseman. On the representation and estimation of spatial uncertainty. *International Journal of Robotics Research*, 5:56–68, 1986.

[49] R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA, 1998.

[50] S.Williams, G. Dissanayake, and H.F. Durrant-Whyte. Towards terrain-aided navigation for underwater robotics. *Advanced Robotics*, 15(5), 2001.

[51] S. Thrun. Learning metric-topological maps for indoor mobile robot navigation. *Artificial Intelligence*, 99(1):21–71, 1998.

[52] S. Thrun. Robotic mapping: A survey. In G. Lakemeyer and B. Nebel, editors, *Exploring Artificial Intelligence in the New Millenium.* Morgan Kaufmann, 2002.

[53] S. Thrun, M. Bennewitz, W. Burgard, A.B. Cremers, F. Dellaert, D. Fox, D. Hähnel, C. Rosenberg, N. Roy, J. Schulte, and D. Schulz. MINERVA: A second generation mobile tour-guide robot. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 1999.

[54] S. Thrun, M. Beetz, M. Bennewitz, W. Burgard, A.B. Cremers, F. Dellaert, D. Fox, D. Hähnel, C. Rosenberg, N. Roy, J. Schulte, and D. Schulz. Probabilistic algorithms and the interactive museum tour-guide robot minerva. *International Journal of Robotics Research*, 19(11):972–999, 2000.

[55] S. Thrun, W. Burgard, and D. Fox. A real-time algorithm for mobile robot mapping with applications to multi-robot and 3D mapping. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, San Francisco, CA, 2000. IEEE.

[56] S. Thrun, S. Thayer, W. Whittaker, C. Baker, W. Burgard, D. Ferguson, D. Hähnel, M. Montemerlo, A. Morris, Z. Omohundro, C. Reverte, and W. Whittaker. Autonomous exploration and mapping of abandoned mines. *IEEE Robotics and Automation Magazine*, 2004.

[57] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L.-E. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. van Niekerk, E. Jensen, P. Alessandrini, G. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. Nefian, and P. Mahoney. Winning the darpa grand challenge. *Journal of field Robotics*, 2006. accepted for publication.

[58] Sebastian Thrun and Michael Montemerlo. The graph slam algorithm with applications to large-scale mapping of urban structures. *Int. J. Rob. Res.*, 25(5-6):403–429, 2006. ISSN 0278-3649.

[59] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, September 2005. ISBN 0262201623.

[60] N. Tomatis, I. Nourbakhsh, and R. Siegwart. Hybrid simultaneous localization and map building: Closing the loop with multi-hypotheses tracking. In *Proceedings of the IEEE International Conference on Robotics and Automation, Washington DC, USA, May 11 - 15.*, 2002.

[61] A. Visser, P. van Rossum, J. Westra, J. Sturm, D.A. van Soest, and M. de Greef. Dutch aibo team at robocup 2006. Team description paper for the 10th RoboCup International Competition, June 2006, Bremen, Germany, June 2006.

[62] J. Wang, M. Lewis, and J. Gennari. Usar: A game-based simulation for teleoperation. In *Proceedings of the 47th Annual Meeting of the Human Factors and Ergonomics Society*, pages 493–497, Denver, CO, 2003.

[63] J. Wang, M. Lewis, and J. Gennari. Interactive simulation of the nist usar arenas. In *Proceedings of the 2003 IEEE International Conference on Systems, Mand and Cybernetics*, pages 1350–1354, Washington, DC, USA, October 2003.

[64] J. Wang, M. Lewis, and J. Gennari. A game engine based simulation of the nist usar arenas. In *Proceedings of the 2003 Winter Simulation Conference*, pages 1039–1045, New Orleans, LA, 2003.

[65] Jijun Wang. Usarsim v2.0.2, a game-based simulation of the nist reference arenas. Available on the USARsim website, 2006. URL `http://usarsim.sf.net/`.

[66] S. Williams and I. Mahon. Simultaneous localisation and mapping on the great barrier reef. In *Proceedings of the IEEE International Conference on Robotics and Automation*, Jan 2004.

[67] B. Yamauchi. A frontier based approach for autonomous exploration. In *Proceedings of IEEE International Symposium on Computational Intelligence in Robotics and Automation, Monterey, CA, July 10-11, 1997.*, 1997.

[68] Brian Yamauchi. Frontier-based exploration using multiple robots. In *AGENTS '98: Proceedings of the second international conference on Autonomous agents*, pages 47–53, New York, NY, USA, 1998. ACM Press. ISBN 0-89791-983-1.

[69] Zoran Zivkovic. Ieee/rsj iros 2006 workshop: 'from sensors to human spatial concepts', 2006. URL `http://staff.science.uva.nl/~zivkovic/FS2HSC`.

[70] R. Zlot, A. Stentz, M. Dias, and S. Thayer. Multi-robot exploration controlled by a market economy. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2002.