Drucksachenkategorie

Simultaneous 2D Localization and 3D Mapping on a mobile Robot with Time-of-Flight Sensors

Maximilian Eck

DLR

Institut für Robotik und Mechatronik	BJ.: 2013 IB.Nr.: 572-2013/19

MASTERARBEIT

SIMULTANEOUS 2D LOCALIZATION AND 3D MAPPING ON A MOBILE ROBOT WITH TIME-OF-FLIGHT SENSORS

Freigabe:	Der Bearbeiter:	Unterschriften
	Maximilian Eck	Maximilian Eck
	Betreuer:	
	Daniel Seth	Daniel Seth
	Der Institutsdirektor	
	Dr. Alin Albu-Schäffer	

Dieser Bericht enthält 68 Seiten, 21 Abbildungen und 2 Tabellen

Ort: Oberpfaffenhofen	Datum: 27.11.2013	Bearbeiter: Maximilian Eck	Zeichen:

SIMULTANEOUS 2D LOCALIZATION AND 3D MAPPING ON A MOBILE ROBOT WITH TIME-OF-FLIGHT SENSORS

Master's thesis

Master of Mathematics in Finance and Life Science
University of Applied Sciences Koblenz, RheinAhrCampus Remagen

presented by
Maximilian Eck
born on April 18th, 1988 in *Lindlar*

performed at
German Aerospace Center, Oberpfaffenhofen, **Germany**

First referee:	Prof. Dr. Maik Kschischo
Second referee:	Prof. Dr. Uwe Jaekel
External referees:	Dipl.-Math. techn. Daniel Seth
	Dipl.-Math. Christian Rink

Munich, October 29, 2013

Abstract

The problem of building consistent maps of unknown environments is one greatest importance within the mobile robot community. Since the first successful attempts, the variety of solutions has grown larger. One of the most famous approaches, namely the use of a Rao-Blackwellized Particle Filter(RBPF), was introduced by *Murphy et al.* It relies on sampling from the distribution over robot poses and map parameters. Amongst the large number of succeeding publications, a couple of them teamed the RBPF with some scan matching procedure. Acting on that idea, this thesis describes an algorithm, which is based upon the combination of the RBPF and a form of the *Iterative Closest Point*(ICP) algorithm. In different way from most established methods, this procedure manages with a much smaller number of samples. It aims to calculate a 3D grid-based map of environments with planar floors, using Time-of-Flight cameras. This kind of sensors allows to extract 3 dimensional information of the environment efficiently, measuring ranges to obstacles. The robustness of the resulting algorithm was proved by virtual experimental mapping of a laboratory, using an "omniRob" platform.

Zusammenfassung

Das Problem, konsistente Karten zu erstellen ist eines von größter Wichtigkeit in der Geimeinschaft der mobilen Robotik. Seit den ersten erfolgreichen Versuchen, ist die Anzahl an Lösungen deutlich gewachsen. Einer der populärsten Ansätze, die Benutzung eines Rao-Blackwellized Particle Filter(RBPF), wurde von *Murphy et al.* entwickelt. Die Grundidee ist es, aus der gemeinsamen Verteilung der Roboterlagen und den Kartenparametern zu samplen. Unter der Vielzahl der darauf folgenden Veröffentlichungen befinden sich einige, die den RBPF mit einer sogenannten Scan-Matching-Prozedur kombinieren. Diese Idee aufgreifend, beschreibt diese Thesis einen Algorithmus, der auf der Kombination des RBPFs und einer Form des *Iterative-Closest-Point*(ICP)-Algorithmus. Anders als die meisten bestehenden Methoden, kommt dieser Ansatz mit einer kleinen Anzahl an Samples aus. Das Ziel ist es, eine 3 dimensionale voxel-basierte Karte einer Umgebung zu erstellen, die einen ebenen Untergrund besitzt. Sensorischer Input soll dabei durch Time-of-Flight-cameras generiert werden. Diese Art von Sensoren erlaubt es, 3 dimensionale Informationen über die Umgebung effizient zu gewinnen, indem sie die Enfernung zu Hindernissen messen. Die Stabilität des resultierenden Algorithmus wurde in virtuellen Experimenten demonstriert. Dabei wurde ein virtuelles Labor mittels einer “omniRob” Plattform kartiert.

Contents

List of Figures **xi**

List of Tables **xiii**

1 Introduction **1**

2 Related Work **5**

3 Overall structure of probabilistic SLAM **7**
3.1 Notation and assumptions . . . 7
3.2 Mathematical formulation . . . 10

4 Mapping environments **13**
4.1 EKF-SLAM . . . 13
4.2 Graph-based SLAM . . . 14
4.3 Mapping with Rao-Blackwellized Particle filter . . . 16

5 Initial settings **21**
5.1 Hardware and Environment . . . 21
5.2 Software . . . 23

6 The algorithm **27**
6.1 ICP-algorithm . . . 27
6.2 Robot Motion . . . 30
6.3 Particle weighting and resampling . . . 31
6.4 Navigation . . . 33

7 Implementation Structure **35**
7.1 Overall structure . . . 35
7.2 Important implementational aspects . . . 37

8 Results **41**
8.1 Mapping results . . . 41
8.2 Misalignments caused by ICP . . . 46
8.3 Performance . . . 46

9 Review and Outlook **49**

Appendix **51**

Bibliography **53**

List of Figures

1.1 Noisy map and good reference map 2

3.1 Illustration of the pose variables 8
3.2 SLAM process as Dynamic Bayesian Network 9
3.3 Exact and approximative motion model 12

4.1 Schematic representation of graphSLAM 15
4.2 Strong and weak links in graphSLAM 15
4.3 Illustration of Sampling Importance Resampling 18

5.1 OmniRob-platform and Time-of-Flight camera 22
5.2 Environment to be mapped 24
5.3 Storage of space inside an octree 26

6.1 Pose correction via ICP registration 28
6.2 Scheme of “Low Variance Sampling” 32
6.3 Overview on exploration process 34

7.1 UML-class diagram 36

8.1 Best possible Outcome 42
8.2 Result without pose correction 43
8.3 Result with moderate odometry and sensor measurement noise 44
8.4 Result with moderate odometry- and strong sensor-noise 44
8.5 Result with strong odometry- and moderate sensor-noise 45
8.6 Result with strong odometry- and sensor-noise 46
8.7 Misaligned map, caused by ICP-failure 47

List of Tables

6.1 Parameters of the ICP-algorithm 30

8.1 Variance parameters 41

1 Introduction

Biologist, author and former Harvard Professor Edward O. Wilson once stated in an interview: *"Our brain is mapping the world. Often that map is distorted, but it's a map with constant immediate sensory input."* Whether this is an attestation of the brilliance of the human brain shall be discussed elsewhere, but it is remarkable, that it requires a robot's best efforts to manage the same task.

Robots today receive positions of great importance in several areas of society. They are used as highly efficient or precise tools for industrial purposes, but are unable to perform some of the easiest tasks we can imagine. However, to enable a robot to master at least some of those challenges, it has to be qualified to process information, relying on which it is able to make a decision on what to do next.

This thesis deals with the problem, known as *Simultaneous Localization and Mapping* (SLAM) [5,9,14,30,33,44]. It is the task of placing a mobile robot in an unknown environment and to enable it to build a consistent map of its surroundings [9, 14]. Building maps is one of the most fundamental challenges in modern robotics, since it provides important means to make a robot truely autonomous [1, 41]. According to [14] and [9] a solution has even been seen as the "holy grail" for the mobile robot community. Potential applications would include search and rescue operations, as well as space, underwater and subterranean exploration [2, 15].

The high complexity of SLAM arises from its concurrent structure, which is sometimes compared to a chicken-egg problem [43]. More in detail, an accurate map is needed for precise localization, while, at the same time, a good pose estimate is neccessary to build a consistent map. Probably for that reason, SLAM is also known as *Concurrent Mapping and Localization* [2, 6, 44].

Actually, a robot, that performed perfectly noise free motion, could easily map its environment by putting together a sequence of scans. Basically, since the robot's global position estimate becomes increasingly uncertain, due to accumulating motion errors, the resulting map will do so, too [33]. Figure 1.1 clarifies the problem in form of an example. The map on the left side was constructed from pure odometry measurements, that is measurements of the executed motions. In absence of any pose correction, the result is a highly inconsistent representation of the environment. Alongside, a map is shown as a reference, which was constructed under favorable conditions, using a working SLAM implementation.

For solving the given task, the robot has to be equipped with some kind of sensorial

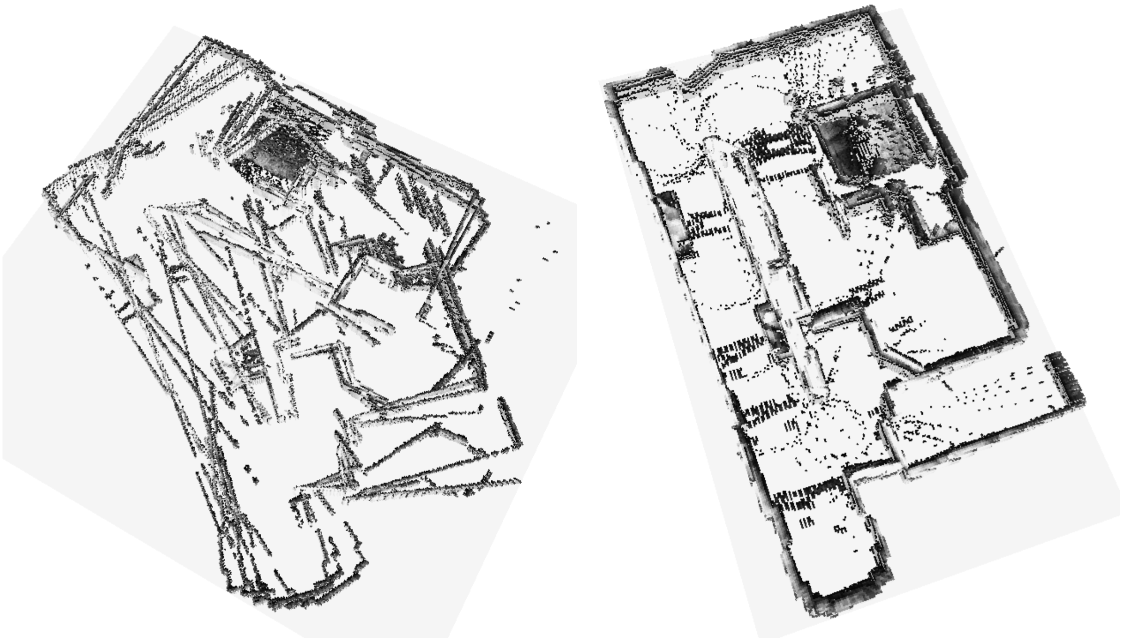


Figure 1.1: Left: Map, constructed from pure odometry measurements. The misalignments, that result from accumulating motion errors are highly visible and clear shapes are unrecognizable.
Right: Good map, which was constructed with a working SLAM-algorithm.

system, by which it extracts information about its environment. Most commonly, those systems are either laser-based, sonar-based or vision-based and are sources of certain errors [2]. Since control actions are often executed with a substantial amount of noise and sensor precision is limited by resolution and scope, it is nesseccary to take uncertainty into account [44]. Additional noise usually occurs by unmodeled effects of the robot and its environment. Probabilistic methods are therefore often involved in the solution of certain robotic challenges and so they are in SLAM.

The aim of this thesis is to overview the problem of *Simultaneous Localization and Mapping* and state-of-the-art approaches on the first hand, and to adapt and implement a robust 3D-indoor application on the other hand. Since the robot is suspected to move on a planar floor, the localization task shall be restricted to two dimensions. Extraction of 3 dimensional environment data is performed by so-called *Time-of-Flight* cameras. Such sensors observe their surrounding by measuring the range to obstacles at different points and hence allow for recording of 3 dimensional data structure.

To date, the question of how to handle the SLAM problem has been answered in a number of different approaches. Chapter 3 introduces the general mathematical formulation of the SLAM problem, that forms the basis of most working probabilistic methods. Afterwards, chapter 4 deals with some established solution statements, which have proved to be successful in this context. Special attention will be given to a method called *Rao-Blackwellized Particle Filter* [36], which will be the technique of choice in the later algorithm. The subsequent chapter aims to give a general survey over the initial situation. This involves parts of the hardware, including its motion components and sensors, as well as some crucial elements of the software library. Unfortunately, the solid robot did not get fit for service in time. For that reason, results were generated by a virtual reproduction within a virtual environment. Robot motion will be concretized and the most important algorithmic elements explained in detail. Putting the previously depicted theories into practice was certainly the most time-consuming part of this thesis. In chapter 7 the program structure is clarified at the level of the overall class interactions and important implementational details respectively. Finally the section “Results” gives a demonstration of the softwares reliability. Next to consistent maps as the main results, there will be some statement on performance, too. This work concludes with a short review and outlook on future challenges.

2 Related Work

In 1987 *Cheeseman et al.* [7] were the first who presented a solution to the SLAM-problem, that is based upon the use of the *Extended Kalman Filter*(EKF) [48]. They represented the map by a finite number of landmarks, that is to say significant points in space, that are easy to identify and have to be extracted from observations first. Their general idea was to store robot poses and landmark locations in a combined, Gaussian distributed vector and perform EKF updates at each time step. *Weingarten et al.* [46] developed an extension to that approach, that can even build 3-dimensional landmark-based maps. In 2001 *Dissayanake et al.* [9] proved important convergence properties of the EKF-SLAM algorithm for the 2D case of linear motion and measurement models. Later, *Huang et Dissayanake* [24] were able to generalize some of these convergence evidences to the non-linear case.

Lu et Milos [30] presented an iterative approach, in which a number of spatial constraints between the robot poses is derived and summarized in an objective function. Their general idea was to define *weak* constraints, whenever motion measurements between two poses are present and *strong* constraints, where two poses observe a common part of the environment [30]. Maximization of the objective function can be done efficiently and special procedures exist for graph construction and optimization respectively(see e.g. [17]). Later publications refered to this proceeding as *Graph-based SLAM* [17, 26] or *graphSLAM* [44].

In 2002, *Montemerlo et al.* presented *FastSLAM* [34], an algorithm, that is based on a Monte-Carlo method, known as *Rao-Blackwellized Particle Filter*(RBPF) [36]. As will be shown in chapter 4, this technique reduces the sample space dimension by the amount of map parameters, making the use of an ordinary particle filter tractable. Later, the authors extended their previous work by developing an improved proposal distribution, that allows for a more effective use of *Importance Sampling* [16] and named the new version of the algorithm *FastSLAM 2.0* [33]. Unfortunately they restricted their work on landmark-based SLAM and did not face the problem of grid mapping. Furthermore, their approach refers to the building of 2D-maps.

Eliazar et Pall [15] were the first, who overcame the former restriction of a landmark-based map representation, but resumed the idea of Rao-Blackwellized particle filtering. They even presented their own extension of the FastSLAM-algorithm and called their solution *Distributed Particle Mapping.* Their approach however, was outperformed by *Haehnel et al.* [21], who combined laser-based FastSLAM with a 2D scan matching procedure. The registration algorithm they used was previously described in [22]. A similar proceeding was performed by *Grisetti et al.* [18, 19], who faced the task of reducing the number of samples additionally. Moreover they introduced a variant of so-called *Selective Resampling* [12] to reduce the problem of *particle depletion* [45]. Their considerations, relies on the assumption, that sensor information is significantly more precise than odometry measurements. The challenge of 3D mapping remains unfaced in each of those works.

Nüchter et al. presented a non-probabilistic approach, using 3D laser range images, computed from sequential 2D observations [37]. The work focuses on the matter of accurate relocalization via a variant of the *Iterative Closest Point* algorithm. However, their approach is limited for the application in SLAM with 6 degrees of freedom and can not be applied to robots in planar environments. *May et al.* adapted that approach for the use of *Time-of-Flight* cameras, but did not refer to 2D-localization either [31]. None of them tried to combine the ICP-algorithm with the use of a RBPF, as it will be the case in this work.

3 Overall structure of probabilistic SLAM

This chapter deals with the principles, being at the basis of a variety of SLAM-approaches. Its common probabilistic form traces back to the *IEEE Robotics and Automation Conference* in 1986, where it became subject to intensive research for the first time [14]. One noticed, that the SLAM problem could be faced by modelling some sources of noise, that cause inconsistent mapping results, explicitely. The resulting algorithms tend to be more robust, as one knows today [44].

In the next subsection the notation is introduced and general assumptions will be explained and justified. Afterwards, the mathematical backgrounds and requirements will be stated.

3.1 Notation and assumptions

The underlying situation is the one of a mobile robot, that explores a room and comes to a halt at discrete time steps $t_1, \ldots, t_k$. After reaching a particular pose, an observation is made, what requires the robot to be equipped with at least one kind of sensory system. Subsequently, some kind of control action causes the platform to move further on, aiming for a new pose. Depending on the type of algorithm, that is used, map building occurs either incrementally during the movement or by processing a batch of collected data afterwards. Remembering this, the designations of variables can now be introduced. One usually distinguishes between *data variables* and *hidden states* [15,17], where the latter ones are not measureable and have to be estimated therefore. For reasons of traceability, the notation in this work conforms with the one used in most of the papers, that refer to SLAM.

Hidden states

x_k is a vector, that represents the robot pose at time k relative to a global external coordinate frame. In general, such poses comprise 6 degrees of freedom, namely the 3 cartesian coordinates, and 3 angular orientation quantities, called *pitch*, *roll* and *yaw* [44]. Naturally, for robots in planar environments, it is sufficient to have 2 coordinates and one single angle. Since this case is the one to deal with in this thesis, the reference frame, as it is used, is shown in Figure 3.1. Denoting the cartesian coordinates with x and y, the robot's heading direction

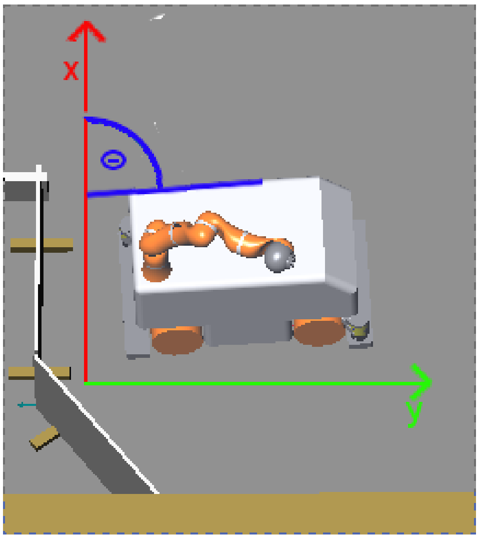


Figure 3.1: Reference frame for the robot's pose. The cartesian coordinates are denoted by x and y. The heading direction of the robot is measured in terms of an angle Θ, that describes the rotational difference to the x-axis.

is described as an angle Θ, which is measured with respect to the x-axis. The pose x_k can be considered as the vector (x, y, Θ) then.

m is a finite vector, representing the map in a reasonable way. Estimating this variable as accurate as possible is the actual aim of any SLAM approach. The concrete representation of the map depends on the sensor, the specific problem and the algorithm, that is used [17]. This work aims to represent the environment in form of a three-dimensional probabilistic *Grid Map* [44], what means to discretize the mapping space into equally sized voxels and label each voxel with its probability to be occupied.

Observable variables

u_k refers to some kind of control, executed at time step k to move the robot and resulting in a new pose x_{k+1}. It is also common, to consider u_k to be an odometry measurement. Odometers register the revolution of the robot's wheels and, in so doing, provide information about the change of the pose [44]. In the following u_k will exclusively denote odometry measurements of the form [44]:

$$u_k = \begin{pmatrix} \bar{x}_k \\ \bar{x}_{k+1} \end{pmatrix}, \tag{3.1}$$

where $\bar{x}_k$ and $\bar{x}_{k+1}$ refer to poses before and after the movement, respectively, measured in the same robot internal coordinate frame.

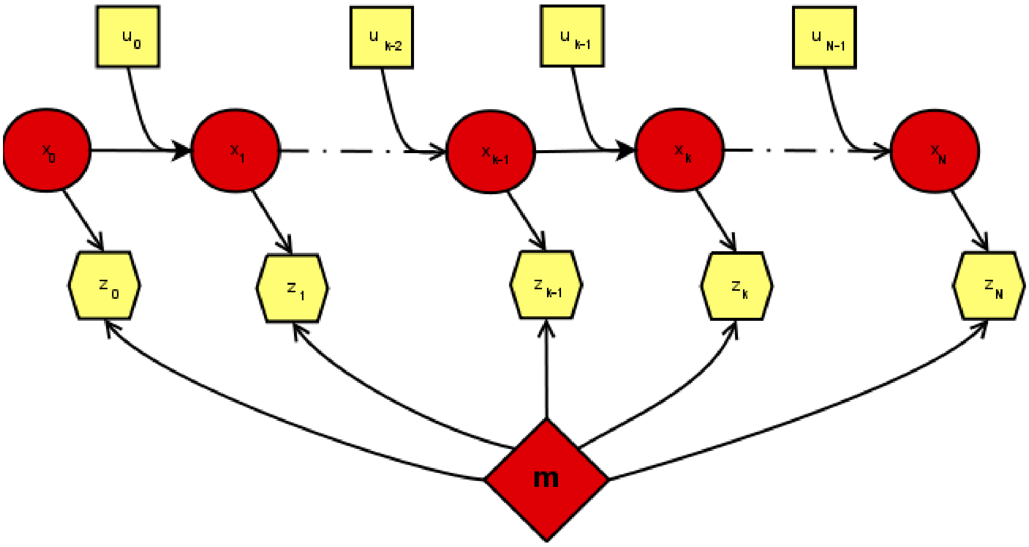


Figure 3.2: The SLAM process, considered as Dynamic Bayesian network(DBN). Hidden states are highlighted in dark red, while observations and controls are marked in light yellow. The conditional dependencies are represented by solid arrows. Thus, poses x_k depend on their precursor pose x_{k-1} and the control u_{k-1}, as described by the motion model. Observations z_k depend on the map m and the pose x_k, what is declared by the observation model.

z_k represents an observation, taken by the robot's sensory system at time k. Often those measurements are laser- or vision-based and, depending on the sensor, refer to two- or three-dimensional data. In the algorithm used here, z_k will be refered to a collection of 8 depth images, taken with Time-of-Flight cameras.

In the context of the underlying problem, it is often useful to have a short notation for a sequence of poses. Therefore

$$x_{t_1:t_2} = \{x_{t_1}, x_{t_1+1}, \dots, x_{t_2-1}, x_{t_2}\}$$

shall serve as shortcut. The variables $u_{t_1:t_2}$ and $z_{t_1:t_2}$ are defined in this way as well. A sequence of the form $x_{1:k}$ will be called a *trajectory* in the following.

A common way to describe the dependencies of the variables in SLAM is to consider its structure as a *Dynamic Bayesian Network*(DBN) as depicted in Figure 3.2. The graphical illustration chosen here expresses conditional dependecies between nodes as solid arrows. Hidden states are distinguished in dark red, while observations and controls are marked in light yellow. For a start, this graphic shall serve as an overview, but it will be refered to later in this chapter.

For the purpose of SLAM, the hidden state x_k is mostly assumed to be *complete*. This means, according to [44], that no additional information from past states, observations or control actions, provokes any improvement in future pose predictions, given x_k. In other words, x_k unifies the information, that can be used to make inferences on future poses. Temporal processes, that achieve this requirement are called *Markov chains* [44]. The most meaningful consequence of that presumption, as will become clear in section 3.2, can be expressed as follows:

$$p(x_k|x_{k-1}, z_{0:k-1}, u_{0:k-1}) = p(x_k|x_{1:k-1}, u_{k-1}) \tag{3.2}$$

It is essential to notice, that z_{k-1} provides information to specify x_{k-1}, while u_{k-1} is executed after the specification of x_{k-1} and therefore provides additional information about x_k very well.

One more assumption, namely the *static world assumption* [17] shall be made here. It expects the environment to stay unchanged during mapping. Actually, this excludes moving people to be in the room as well. Graphic 3.2 includes this assumption, thus, in contradiction to the other states, m is stated only once. The problem of SLAM with non-static environments is faced e.g. in [38] and [40].

3.2 Mathematical formulation

Having introduced the notation and done the assumptions, the SLAM problem can now be formulated. Next to the overall formulation, this subsection introduces the motion- and measurement model.

3.2.1 Full- and Online-SLAM

In probabilistic form, SLAM occurs in one of the following two alternatives [44].

Online SLAM: Estimate the joined posterior over the map, along with the robots current pose at time k:

$$p(x_k, m|z_{0:k}, u_{0:k-1}, x_0)$$

Full SLAM: Estimate the joined posterior over the map, along with all past poses:

$$p(x_{1:k}, m|z_{0:k}, u_{0:k-1}, x_0)$$

Many online SLAM algorithms work incrementally and abolish observations and controls, that have already been processed [44]. By contrast, full SLAM solutions usually do not integrate sensor data permanently or incrementally, but re-estimate the whole map in each iteration step [30] and have to store the overall data for that reason. Both forms are of practical relevance and are connected via [44]:

$$p(x_k, m|z_{0:k}, u_{0:k}, x_0) = \int\int\dots\int p(x_{1:k}, m|z_{0:k}, u_{0:k-1}, x_0)dx_1\ dx_2\dots dx_{k-1}$$

Having at least one of these distributions allows to recover the map, using e.g. maximum likelihood techniques. However, adressing the problem analytically usually ends up in intractable integrals [11]. Further serious problems arise from the high dimensionality of the parameter space, from which the map is described [44].

3.2.2 Motion- and Measurement Model

The SLAM-problem to be well-defined requires the specification of two more elements, as becomes clear by the factorizations

$$\begin{aligned} p(x_k, m|z_{0:k}, u_{0:k-1}) =& \eta \cdot \underbrace{p(z_k|x_{1:k}, z_{0:k-1}, u_{0:k-1}, m)}_{\text{Measurement Model}} \\ & \cdot \int \underbrace{p(x_k|x_{k-1}, u_{k-1}, m)}_{\text{Motion Model}} \cdot p(x_{k-1}, m|z_{0:k-1}, u_{0:k-1})\, dx_{k-1} \end{aligned}$$

and

$$\begin{aligned} p(x_{1:k}, m|z_{0:k}, u_{0:k-1}) =& \eta \cdot p(x_{1:k-1}, m|z_{0:k-1}, u_{0:k-2}) \\ & \cdot \underbrace{p(z_k|x_{1:k}, z_{0:k-1}, u_{0:k-1}, m)}_{\text{Measurement Model}} \cdot \underbrace{p(x_k|x_{k-1}, u_{k-1}, m)}_{\text{Motion Model}} \end{aligned}$$

respectively. Both equations take use of the Markov assumption and equation (3.2) in particular. The constant value η serves as normalizer. These recursive formulations of the SLAM problem reveal the lack of concreteness and itemize the missing probabilities.

1. The term $p(x_k|x_{k-1}, u_{k-1}, m)$, is refered to the so-called *motion model* [14, 34, 44]. It represents the probability of being at pose x_k after executing control u_{k-1} at the starting pose x_{k-1}. Obviously, the motion model comprises knowledge about the map. Analytical computation of such a probability distribution is quite complex, since it has to take the robot's path into account. Neglecting the path leads to an approximative model, in which the robot is allowed to pass through walls [44], as is demonstrated by Figure 3.3. Thus, information about the map is usually ignored totally [44] and collision avoidance performed by different means. Expressing this assumption in formula leads to

   $$p(x_k|x_{k-1}, u_{k-1}, m) = p(x_k|x_{k-1}, u_{k-1}).$$

   This map independent motion model will be used subsequently. It matches the case, that is illustrated in Figure 3.2, since the poses are not supposed to be conditionally dependend on m. Because mapping takes place relative to the robots start pose x_0, the latter one can be chosen arbitrarily and a suitable

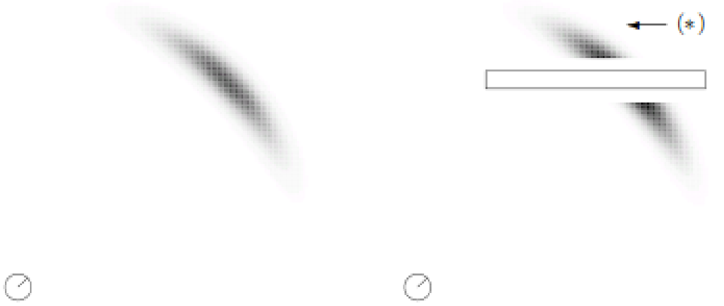


Figure 3.3: Motion model without the map information(left) and conditioned on the map (right). Dark shading implies high likelihood of the robots next pose. The white bar on the right-hand side represents an obstacle, as it could be encoded by a map. Since the area marked with (∗) is blocked, all likeli poses should be located on the other side of the obstacle. Computing a closed form respecting that issue is difficult, thus the approximative model allows the robot to pass through walls(source: [44]).

representation for $p(x_0)$ would be given by a Dirac-distribution therefore [17]. In the following, the start pose will be neglected in the formulation.

2. The probability $p(z_k|x_{1:k}, z_{0:k-1}, u_{0:k-1}, m)$ describes the likelihood of a certain observation z_k, given the vehicle state and available informations on the map. It is easily seen from Figure 3.2, that observations are assumed to be independent from all other variables but the current pose and the map. Intuitively this seem reasonable and, expressed mathematically, leads to the following useful simplification.

$$p(z_k|x_{1:k}, z_{0:k-1}, u_{0:k-1}, m) = p(z_k|x_k, m),$$

   which is spoken of as *observation model* [9,14] or *measurement model* [34]. Once the pose and map are defined, observations are assumed to be independent additionally [14].

Motion- and sensor- models are robot and sensor specific respectively. Suggestions for practical implementations are submitted e.g. in [44].

4 Mapping environments

To date, a large number of publications describe methods to adress and solve the SLAM problem successfully. Most commonly, they are either based on the use of the Extended Kalman Filter, Rao-Blackwellized Particle Filter or are refered to a Maximum Likelihood technique, called graph-based SLAM. Therefore, this chapter aims to give a short overview of the general ideas at the basis of these three approaches. Special attention will be given to the Rao-Blackwellized Particle Filter, since it plays a decisive role in the algorithm developed in this thesis.

4.1 EKF-SLAM

The use of Extended Kalman Filters to solve the SLAM problem traces back to a paper of *Smith, Self and Cheeseman* [7]. Due to the incremental nature of the map building process, it is refered to a solution of the online SLAM problem. Its application is restricted to landmark-based maps, where landmarks refer to significant shapes, that may be artificial flags or natural obstacles, like trees or rocks. Knowledge about these landmarks locations enables the robot to localize itself within its previously unknown environment. In EKF-SLAM, the map is therefore represented by a collection of landmark positions and their features when indicated, and denoted by $m = \{m_1, \dots, m_N\}$. The well-known *Bayes-Filter* [44] then provides the general framework for the solution [14]

$$\begin{aligned} \overline{bel}(x_k, m) &\coloneqq p(x_k, m | z_{0:k-1}, u_{0:k-1}) \\ &= \int p(x_k | u_{k-1}, x_{k-1}) \cdot bel(x_{k-1}, m)\, dx_{k-1} \\ bel(x_k, m) &\coloneqq p(x_k, m | z_{0:k}, u_{0:k-1}) \\ &= \eta \cdot p(z_k | x_k, m) \cdot \overline{bel}(x_k, m), \end{aligned}$$

where η serves as a normalizing constant again.

An easy implementation is given by the *Kalman Filter* [48], that assumes both probability distributions to be Gaussian. In addition, it requires to have observation-

and motion- models of the following form:

$$p(x_k|u_k, x_{k-1}) \Leftrightarrow x_k = g(x_{k-1}, u_k) + \epsilon_k \qquad \epsilon_k \sim \mathcal{N}(\mathbf{0}, R_k)$$
$$p(z_k|x_k) \Leftrightarrow z_k = h(x_k) + \delta_k \qquad \delta_k \sim \mathcal{N}(\mathbf{0}, Q_k)$$

with linear functions g and h. However, SLAM is a non-linear problem in general and the Kalman Filter not applicable therefore. To overcome this restriction, one usually enforces g and h to be linear by first order Taylor expansion, resulting in the *Extended Kalman Filter*-algorithm for SLAM(EKF-SLAM) [14,44]. The mathematical formulation is space wasting, but the basic update step can be looked up in the appendix.

Unfortunately, the computational effort in EKF-SLAM grows quadratically with the number of landmarks, since the covariance matrix does so [14]. Its application is therefore restricted to environments with a few hundred of landmarks at most [34]. The overall algorithm has to cope with additional challenges, such as correctly identifying observed landmarks, or how to deal with newly observed features. However, the aim of this thesis is to put the general idea across, thus it shall be pointed to [7] and [44] for more detailed information.

4.2 Graph-based SLAM

A popular technique for the solution of the full SLAM problem is refered to a Maximum-Likelihood approach, in which the robot's trajectory is optimized iteratively. It is based on a graph, in which each node is labeled with a robot pose and the corresponding observation [17], hence the name *graph-based* SLAM. An exemplaric situation is illustrated in Figure 4.1, in which equally shaped figures, labeled with z_k, represent observations of the very same feature. The criterion to measure the quality of a pose configuration can be derived by a number of spatial constraints between poses, which are represented as edges inside the graph. One usually subdivides those links between poses into *weak* and *strong* links [30]:

Weak links occur between successive poses and represent information, that was extracted from odometry readings or control variables. In Figure 4.1 a weak link is available, wherever a solid arrow connects two poses.

Strong links arise, where parts of the environment have been observed from at least two different poses. In this particular example, poses from whom the same landmark was observed, are linked in this way.

The existance of weak and strong constraints in the particular example from Figure 4.1 is shown in Figure 4.2.

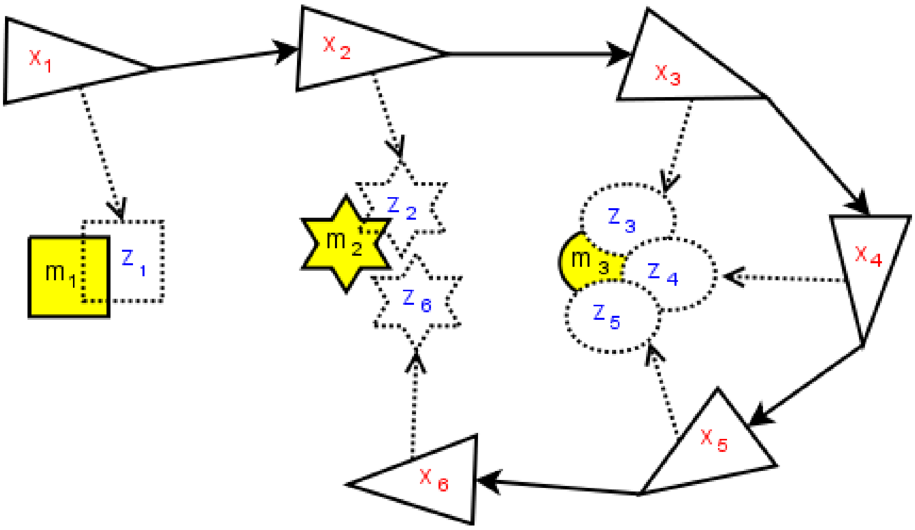
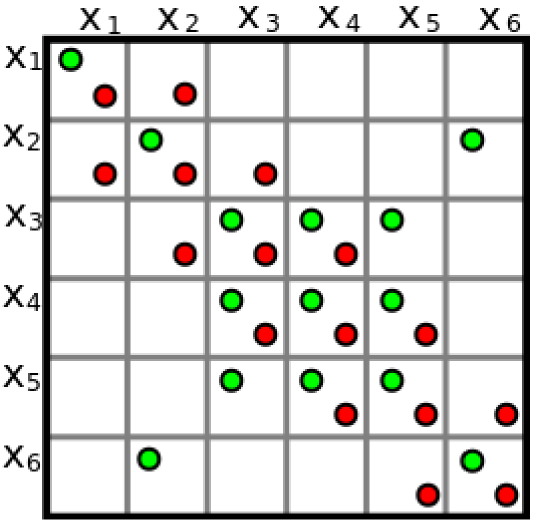


Figure 4.1: Schematic representation of a graph and the resulting constraints. Solid arrows occur between successive poses and indicate a weak link. Equally shaped measurements z_k are refered to an observation of the same landmark. Those common observation indicate a strong link between the corresponding poses.



	x1	x2	x3	x4	x5	x6
x1	🟩 🟥	🟥				
x2	🟥	🟩 🟥	🟥			🟩
x3		🟥	🟩 🟥	🟩 🟥	🟩	
x4			🟩 🟥	🟩 🟥	🟩 🟥	
x5			🟩	🟩 🟥	🟩 🟥	🟥
x6		🟩			🟥	🟩 🟥

Figure 4.2: Strong and weak links, as they occur in the example above. Green marks refer to a strong link, red ones to a weak link. Weak links occur only between succeding poses, strong links, whenever a common part of the environment has been observed from two different poses. Each pose is linked to itself via a strong and a weak link.

In general, graph-based SLAM is refered to a recursive two-step procedure. Defining and integrating the constraints is subject to the so-called *front-end* [17, 26] and leads to an objective function of the following form:

$$F(x_1, \ldots, x_k) = \sum_{(i,j)} e_{ij}^T \Omega_{ij} e_{ij} \tag{4.1}$$

where Ω_{ij} is a known information matrix and e_{ij} some difference between estimated and observed relative pose values. Optimization of equation (4.1) is adressed by the *back-end* step [17, 26]. A number of special algorithms exist for both steps [17, 26, 30, 44].

Due to the fact, that graph-based SLAM re-estimates the map in each step, it overcomes the restriction of updating individual parts of the map independently, what is particularly beneficial in the context of loop-closure [30]. A major disadvantages lies in its high memory allocation, that occurs because observations have to be stored and can not be rejected, as it is the case e.g. in EKF-SLAM [44]. Furthermore, a bad initial trajectory estimate may cause the algorithm to converge slow and to stuck in local minima [39].

For purposes of easier visualization and explanation, the map representation has been chosen to be landmark-based here. Basically, graphSLAM is able to deal with different kinds of maps as well. Examples on how to use it on grid-based maps can be find, amongs others in [26] and [30].

4.3 Mapping with Rao-Blackwellized Particle filter

Amongst the different approaches, that have been developed so far to solve the SLAM-problem, the use of a Rao-Blackwellized Particle Filter is probably the most common proceeding to generate an accurate and consistent map of an unknown environment. In contradiction to other popular proceedings like EKF-SLAM, the RBPF neither depends on the existence of predefined landmarks, nor on linear or linearized motion or measurement models [44]. As the name suggests, the proposed technique is a special application of ordinary particle filters, which will be described in short in the next section. An explanation of the difficulties and characteristics, when using it on the SLAM problem, will be given afterwards.

4.3.1 Particle Filter and Sampling

In contradiction to a variety of popular filters, the particle filter approximates a target distribution not by a different distribution, but in form of a number of samples $(x_1, \ldots, x_k)$ drawn from that distribution [44]. Important inferences can be made on the basis of a number of those so-called *particles* [16, 44] then.

Unfortunately, a sampling procedure may not be available for a particular distribution p. One often uses the service of a method called *Importance Sampling* therefore [13, 16, 44]. It allows to compute e.g. the expectation of p, by sampling from a different distribution π, which fulfills some requirements. π is called a *proposal* distribution [13, 18, 19, 44] or an *envelope* [16]. The crucial step is to associate a weight $w^{(i)}$ to each sample x_i via [13]:

$$w^{(i)} = \frac{p(x_i)}{\pi(x_i)}.$$

Particles, that are likeli according to p receive high weights, what provides information about the target distribution.

Although Importance Sampling is sufficient to estimate important properties of p, it is not a proper representation of the distribution p itself [13]. One therefore performs a resampling step, what means re-drawing the samples with replacement and with probabilities proportional to their weights. The resulting procedure is known as *Sampling Importance Resampling*(SIR) [16]. According to [16], samples, that have been drawn via the SIR procedure are distributed according to the target distribution p. An illustration of the SIR algorithm is shown in Figure 4.3.

A well known procedure for sampling sequences of values $(x_1, \dots, x_k)$ is given by the so-called *Sequential Importance Sampling*(SIS) [13, 16]. It relies on the factorization

$$\pi(x_1, \dots, x_k) = \pi(x_k|x_{k-1}, \dots, x_1) \cdot \pi(x_{k-1}|x_{k-2}, \dots, x_1) \cdot \dots \cdot \pi(x_1)$$

and allows for more effective drawing, since each sequence $(x_1^{(i)}, \dots, x_{k-1}^{(i)})$ can be extended by simply drawing [13, 16]

$$x_k^{(i)} \sim \pi(x_k|x_{k-1}^{(i)}, \dots, x_1^{(i)}).$$

The time complexity is therefore assured to be constant at each time step and does not grow linearly, as it would be the case for drawing whole sequences [13]. SIR and SIS can be integrated easily, when a particle filter is used.

4.3.2 SLAM with RBPF

In general, particle filters could be applied in a straight forward way to solve the SLAM problem, by sampling from the full-SLAM distribution

$$p(x_{1:k}, m|z_{0:k}, u_{0:k-1}, x_0)$$

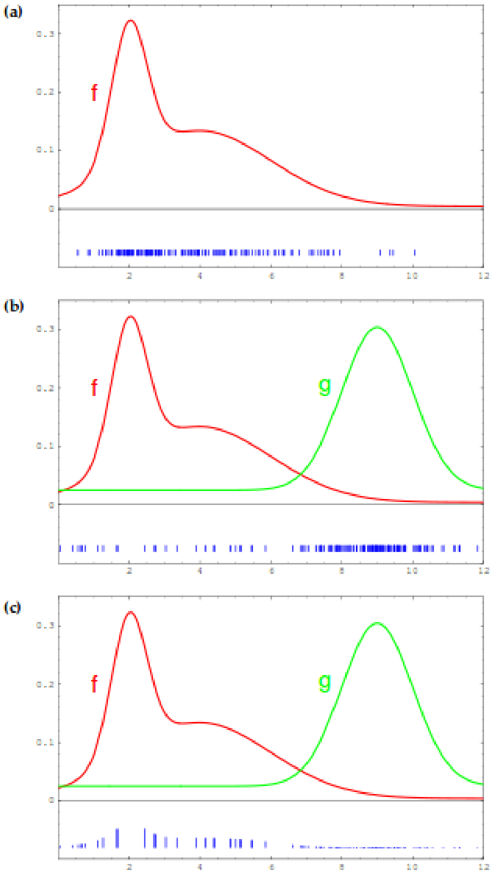


Figure 4.3: Illustration of Sampling Importance Resampling. The aim is to get samples, that are distributed according to f, what is shown exemplarily by blue dashes in (a). Since there is no way to sample from f directly, g is used as a proposal distribution. The weighted and unweighted samples are shown in (b) and (c). Resampling the particles in (c), with a probability proportional to the weights, leads to a sample set similar to the one in (a).(source: [44])

at each time step k. Unfortunately, this proceeding is intractable, due to the high dimensionality of the sample space and the limited computational capacities [14]. In particular the amount of variables, that are neccessary to describe the map is likely to reach extremely high numbers. Even the moderate case of an occupancy grid map with a resolution of 10cm leads to a binary map space of the dimension 1000 per m^3, that is to say 2^{1000} possible samples. To overcome this important limitation, Murphy and colleagues [35] utilized so-called *Rao-Blackwellization* [16] to improve particle filtering. In the underlying context of SLAM, this technique relies on the following factorization:

$$p(x_{1:k}, m | z_{1:k}, u_{1:k-1}) = \underbrace{p(x_{1:k}|z_{1:k}, u_{1:k-1})}_{\text{particle filter}} \underbrace{p(m|x_{1:k}, z_{1:k})}_{\text{analytical computation}} \quad . \tag{4.2}$$

The essential thought is, that the map distribution can be estimated analytically, once the robot poses are known. Thus, the problem reduces to sampling trajectories, what can be done with reasonable computational effort. The resulting algorithm is known as *Rao-Blackwellized particle filter* [35, 36].

Due to the step-by-step progression of the robot, it seems reasonable to adopt the SIS principle for generating new trajectories. Let

$$\pi(x_{1:k}|z_{0:k}, u_{0:k-1})$$

the proposal distribution, that refers to the first term on the right side of equation (4.2). Following the suggestion in [45] it shall be assumed, that each particular pose estimate is independent on successive observations and controls. This allows for the recursive formulation

$$\begin{aligned} \pi(x_{1:k}|z_{0:k}, u_{0:k-1}, x_0) &= \pi(x_k|z_k, x_{k-1}, u_{k-1})\pi(x_{1:k-1}|z_{0:k}, u_{0:k-1}, x_0) \\ &\overset{assum.}{=} \pi(x_k|z_k, x_{k-1}, u_{k-1})\pi(x_{1:k-1}|z_{0:k-1}, u_{0:k-2}, x_0) \end{aligned}$$

and provides an efficient sampling procedure, since established trajectories can be reused. Thus, it is sufficient to draw a single value

$$x_k^{(i)} \sim \pi(x_k|z_k, x_{k-1}^{(i)}, u_{k-1})$$

to extend the i-th trajectory and get $x_{1:k}^{(i)}$. This special structure makes SLAM with RBPF the only known SLAM approach, which solves both, the online- and full- SLAM problem [44]. Defining a suitable proposal distribution π, which allows proceeding like this, is non-trivial and therefore subject to a plenty of publications [19, 20, 33].

It was stated before, that importance weights are calculated as the quotient of the target distribution p and the proposal distribution π, evaluated at the sampled value. Given the motion- and measurement- model, the weight for the i-th particle may be updated via the following recursion [19]:

$$\begin{aligned} w_k^{(i)} &= \frac{p(x_{1:k}^{(i)}|z_{1:k}, u_{1:k-1})}{\pi(x_{1:k}^{(i)}|z_{1:k}, u_{1:k-1})} \\ &= \frac{\eta \cdot p(z_k|x_{1:k}^{(i)}, z_{1:k-1}) \cdot p(x_k^{(i)}|x_{k-1}^{(i)}, u_{k-1})}{\pi(x_k^{(i)}|x_{1:k-1}^{(i)}, z_{1:k}, u_{1:k-1})} \cdot \frac{p(x_{1:k-1}^{(i)}|z_{1:k-1}, u_{1:k-2})}{\pi(x_{1:k-1}^{(i)}|z_{1:k-1}, u_{1:k-2})} \\ &\propto \frac{p(z_k|x_{1:k}^{(i)}, z_{1:k-1}) \cdot p(x_k^{(i)}|x_{k-1}^{(i)}, u_{k-1})}{\pi(x_k^{(i)}|x_{1:k-1}^{(i)}, z_{1:k}, u_{1:k-1})} \cdot w_{k-1}^{(i)} \end{aligned} \tag{4.3}$$

Resampling has to be performed with care, since it causes loss of information and reduction of sample variety. In particular the loss of particles, that are close to the true solution, is known to cause divergence and is famous as the *particle depletion* problem [18, 19, 41].

So far, the question of how to update a map when using this method has not been answered. In [34] and [14] single landmarks of feature maps were updated by means of the Extended Kalman Filter. Many published implementations of RBPF-based SLAM however, associate a seperate map to each particle, resulting in a number of candidate maps [18–21, 40, 42, 50].

5 Initial settings

The aim of this section is to overview the situation that is at the basis of this work. It will treat of the robot and its hardware and environment, as experimental backgrounds, as well as the software foundations, upon which the algorithm was build.

5.1 Hardware and Environment

First of all, the characteristics of the mobile platform will be pointed out and the most important equipment explained. Its *Time-of-Flight* sensors will be dealt with seperately afterwards, due to their importance in this matter. Finally, the environment, which was used to generate maps, will be introduced.

5.1.1 The "omniRob" platform

It already mentioned at the very beginning of this work, the robot, that was supposed to execute the SLAM-algorithm was indispensable. The virtual reproduction however, aims to get as close as possible to the real robot performance. "omniRob", as the platform is called, is a product of the *Kuka Roboter GmbH.* It has a size of 1.20m × 0.71m × 0.65m, weights about 270kg and reaches a speed up to $1.5\frac{\mathrm{m}}{\mathrm{s}}$. A picture of it is attached by Figure 5.1.

The platform offers omnidirectional wheels and is equipped with several sensor systems [10]. Most importantly to mention, in the context of this work, are the 8 *Time-of-Flight*(ToF) cameras, which have been installed by the DLR-RM-institute. These allow for an almost panoramic view and perceive the environment in depth values. Its equipment further embraces four I7 processor boards, which allow for on board computation and makes the robot independent from any infrastructure [10].

5.1.2 Time-of-Flight sensors

It is obvious, that building 3D map requires the extraction of 3-dimensional data structures. In the past, several 3D-SLAM applications have collected data by means of rotating 2D-laser-scanners or stereo cameras, what comes at the price of slow performance [25, 27, 28, 37]. Nowadays, cheap Time-of-Flight cameras are available

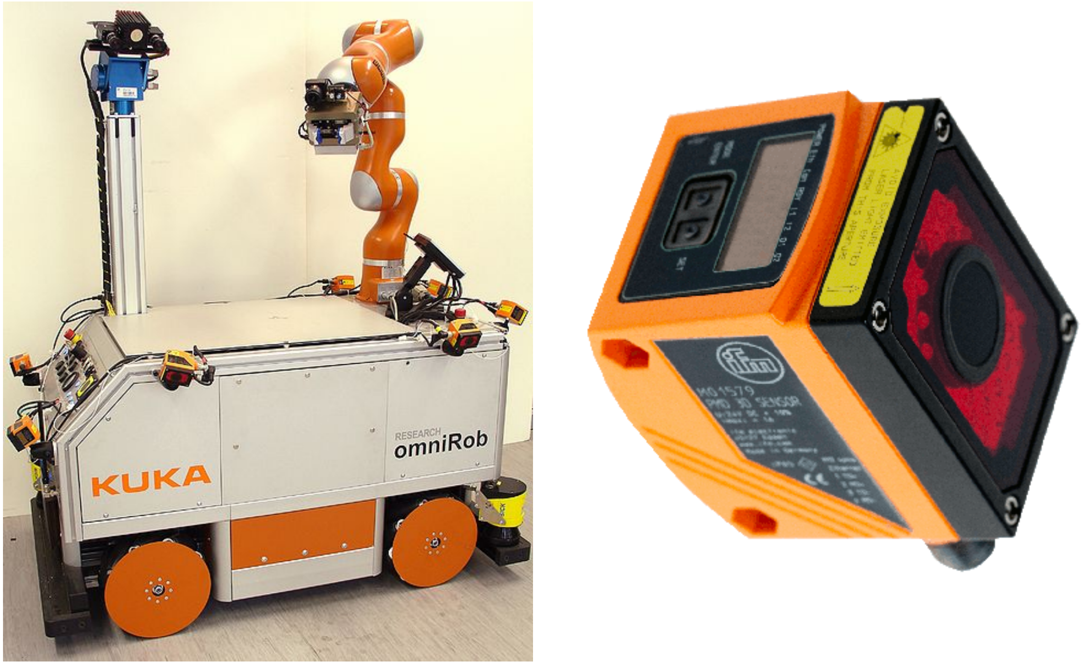


Figure 5.1: Left: The “omniRob”-platform, as it is used at the DLR-Institute Oberpfaffenhofen. Some of its ToF cameras are visible alongside its top and in its front. In addition it carries a *light weight robot*(LWR) arm and a *pan-tilt* unit.
Right: A Time-of-Flight camera of the type IFM O3D100, as it is used by the robot. (source: DLR-intern)

and offer the possibility to record 3D images with reasonable speed, which makes it attractive in the matter of real-time tasks [47].

Lange and Seitz explained the general idea of the operating mode by comparing it with a bat's ultrasonic system, applied on light waves [28]. The performance, however suffer from large systematic and non-systematic errors, what requires data-preprocession and camera calibrationen to make ToF-cameras suitable for robotic applications. [31, 32].

In general, cameras with resolutions up to 20 tsd. pixels are available by now [32]. However, in this work sensors with relatively low resolution of 50×64 pixels were used.

5.1.3 Environment

The object to be mapped is the mobile robot laboratory of the DLR-institute in Oberpfaffenhofen, which is about 11.7m × 5m in length and width. More precisely, it is about a virtual analog of that room, stored in form of a grid map. An adjacent hallway, that is reachable through 2 doors shall be mapped as well and provides additional area of around 13.7m × 1.8m. The floor of the whole complex is flat, as it is required for 2D mobile robotic applications. Two different views of the scene are shown in Figure 5.2.

5.2 Software

At the beginning of that work a remarkable amount of software was already available and has allocated useful functionality, without whom the implementation of a SLAM algorithm would not have been possible, given the short time span. In the following some information about the initial software situation will be provided. Most importantly, the MRE-project will be introduced and the *DynamOctreeSpace*-representation explained.

5.2.1 Lib3D and MRE-project

The software basis, upon which the SLAM-implementation has been established, is given by an intern C++-library called *Lib3D*(L3D). This library comprises a multitude of functions, used in two and three dimensional robotic applications and comprises about 525 classes. Amongst others, it embraces a class for representing a robot, that allows to perform the most basic steps, like moving or measuring, with little effort.

Also important to mention in the context of this work is the *MRE*-project, which is based upon Lib3D. It includes several implementations of mobile robot tasks, some of the most important ones listed above.

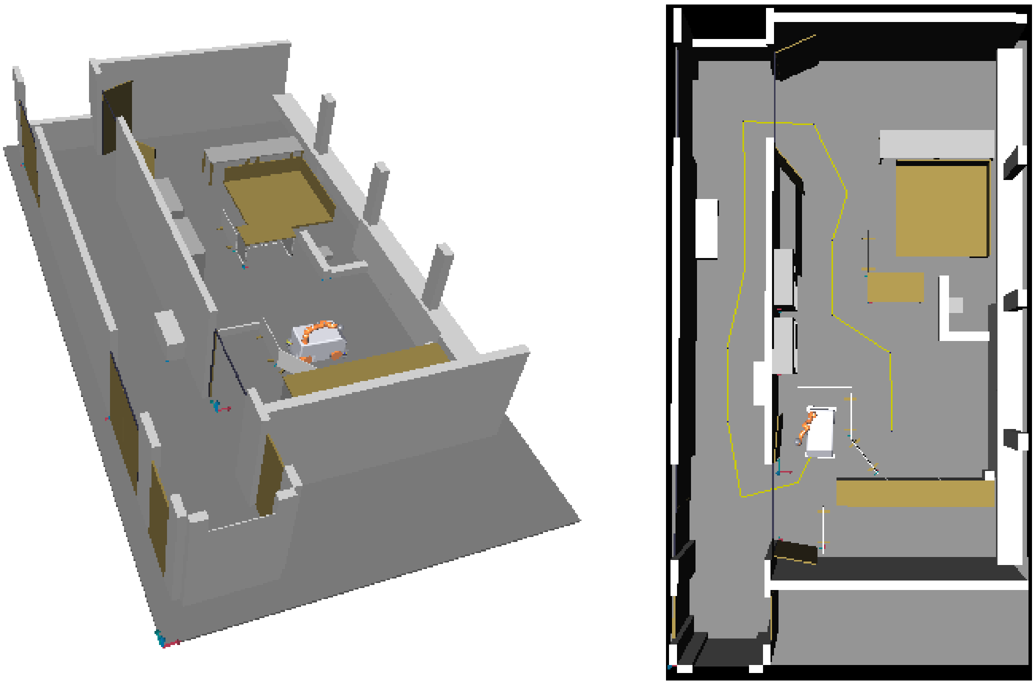


Figure 5.2: Virtual replication of the mobile robot laboratory and the adjacent hallway, which were used as experimental environment. The path, that was used to generate the maps is also shown.

active localization is the task of determining the robot's pose by means of sensor observations and concerted movements

passive updating improves the robot's pose believe via an additional observation, but without moving autonomous

passing obstacle localizes the robot, computes a path to a given destination and moves to the aim along that path

exploration guides the robot through unknown environment and computes the most promising aim for mapping at each step in time (still incomplete)

These operations are connected to SLAM in some way, since they could be inserted as useful tools. The implementation of a mapping algorithm forms an additional feature, that expands the functionality of the MRE-project.

In addition, Lib3D embraces the corresponding project *MREVisualization*, which allows for a graphical display during the execution of the listed applications. The virtual environment from Figure 5.2 was visualized with the functionality of this project.

5.2.2 DynamOctreeSpace

A particlar important part of the Lib3D-library refers to the map storage. Since SLAM is a complex task, it makes high representational demands, from which the most crucial ones are listed in the following:

Compactness: The environment should be memorized parsimoniously. In particular applications in large scale environments, or those, that require the generation of multiple maps are dependend on that property.

Flexibility: Extension of the space to be observed is unknown in general. Thus, the space should be able to grow automatially.

Completeness: Decisions have to be make not just between occupied cells and those who are free. There also has to be a label for cells, who are still unknown. This is, not at last, important to plan the robots path efficiently [23].

An obvious way to model environments is discretizing the mapping space, using a grid of equally sized voxels. The demand for completeness is easily fulfilled, since each voxel allows even for a probabilistic representation of occupancy. However, the consumption of memory resources is unneccessarily high within such models [23,49]. *Wurm et al.* described a structure, named *OctoMap*, to face the challenges listed above [49]. Their approach is based on the use of octrees, which represent cubical spaces, able to subdevide recursively into 8 equally spaced subvolumes, until a predefined minimum voxel size is reached [23, 49]. Economy in memory is achieved by

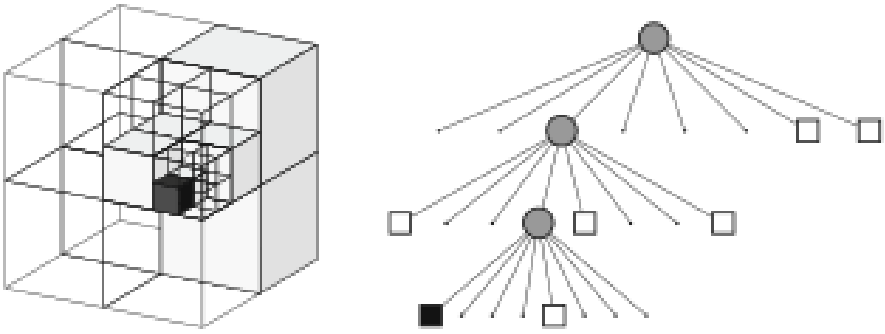


Figure 5.3: Schematic representation of the storage of space via an octree. Grey-marked cubes store a single value. Volumes which are supposed to represent multiple values subdivide, as indicated by the right upper cube in the front level. On the right-hand side, the tree structure that arises from such an octree is illustrated.(source: [23])

saving large voxels, whenever all its subvolumes exhibit the same value. A graphical demonstration of that proceeding is given by Figure 5.3. Finally, flexibility is reached by techniques, applied at the DLR-Institute, which arrange multiple octrees within a grid, to form a, in general non-cubic, space. Extension of that map is perfectly possible by inserting additional octrees. Such a representation is internally known as *DynamOctreeSpace*. An efficient implementation is part of the Lib3d-library and was used for integrating mapping result.

6 The algorithm

It was mentioned before, that the algorithm, whose implementation was a major subject to this thesis, is based upon a Rao-Blackwellized Particle Filter. The drawbacks of EKF-SLAM are obvious, given the restriction to landmark-based maps, which might not be sufficient for future tasks of the robot. Graph-based SLAM on the other hand needs an initial trajectory estimate and can only be applied after recording the data. Hence it needs a full exploration of the environment, before a different task can be executed. What is more, the best initial estimate, that would be available, is given by the relatively uncertain odometry readings. It was stated in [39], that the algorithm is likely to converge slow and to a local minimum therefore. That is why the decision was to use a Rao-Blackwellized Particle Filter.

Following the suggestion of most implementations on grid-based maps, a seperate map shall be associtated to each particle. A crucial point in this algorithm is the additional application of the so-called *Iterative Closest Point*(ICP) algorithm [4] on each particle. *Nüchter et al.* [37] presented the ICP algorithm as a non-probabilistic approach for solving the SLAM problem with 6 degrees of freedom. Their idea was to compute and apply a linear transformation and align each new measurement to the map, build so far. In so doing they received an estimate on the localization mismatch, what is illustrated in Figure 6.1. The process of matching point clouds is commonly known as *registration* [29]. According to [31], this proceeding lacks of robustness, since it is likeli to converge to local minima. Indeed, as will be shown in chapter 8, using ICP-registration in the case of 2D-localization showed significant misalignments in some of the maps. The general idea of combining ICP alignment with a small number of particles was to develope a robust algorithm, that removes errors in registration through the resampling step. In the broader sense, this proceeding has to be seen as a 3-dimensional generalization of the work presented in [21], since it combines the use of Rao-Blackwellized particles with a scan matching procedure. However, the algorithm presented there still requires up to 100 particles.

In the following, the different elements, that were used will be described in detail.

6.1 ICP-algorithm

The accuracy of the map and the preciseness of the robot's pose estimate are highly interdependend. The ICP algorithm is therefore used to find a linear transfor-

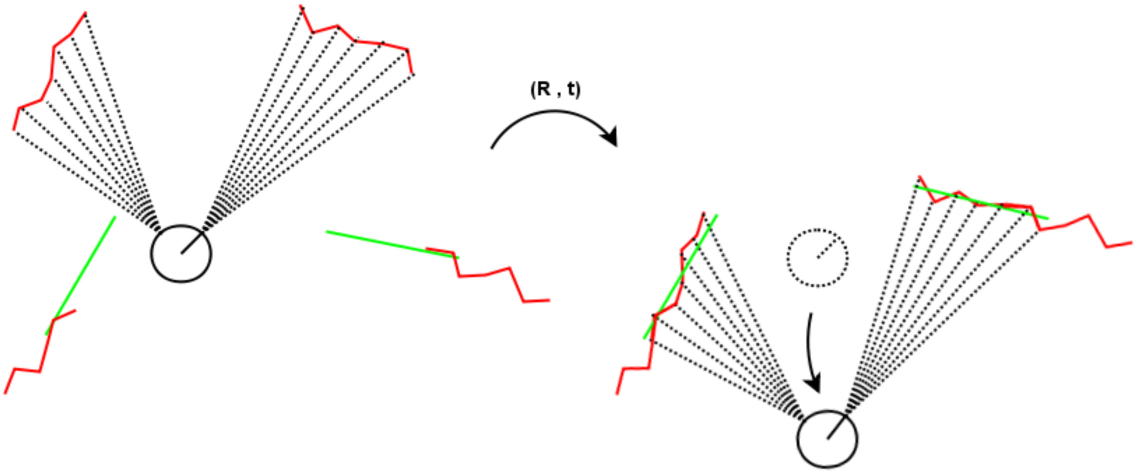


Figure 6.1: Pose correction, performed by ICP registration. The red shapes represent noisy measurements of the environment. Aligning the red curves allows to estimate the localization mismatch and integrate new data at the correct place, that is, the green colored obstacle.

mation, that aligns two given point sets best and in so doing, estimates the localization mismatch. Denoting these two point sets by $M = \{m_1, \ldots, m_{N_m}\}$ and $D = \{d_1, \ldots, d_{N_d}\}$, the quantity of this mismatch can be expressed by the following cost function E:

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

Here, the linear transformation is represented in terms of a rotation matrix R and a translation vector t. Additionally,

$$w_{i,j} = \begin{cases} 1 & \text{if } \; m_i \mathrel{\hat{=}} d_j \\ 0 & \text{else} \end{cases}$$

indicates, whether m_i and d_j are supposed to represent the same point in space. Following the approach in [37], the search for corresponding points is reduced to the problem of searching closest points. More precisely, each point in D is associated to the nearest point from M within a maximum search radius. Let $C = \{\langle m'_1, d'_1 \rangle, \ldots, \langle m'_N, d'_N \rangle$ denote the set of correspondences, thus the cost function reduces to

$$E(R,t) = \sum_{i=1}^{N} \|\mathbf{m}'_i - (\mathbf{R}\mathbf{d}'_i + \mathbf{t})\|^2$$

and refers to an ordinary *Least Square*(LS) approach obviously. The minimization step requires the definition of the matrix

$$\mathbf{H} = \sum_{i=1}^{N} (\mathbf{m}_i' - \bar{\mathbf{m}})(\mathbf{d}_i' - \bar{\mathbf{d}})^T = \begin{pmatrix} S_{xx} & S_{xy} & S_{xz} \\ S_{yx} & S_{yy} & S_{yz} \\ S_{zx} & S_{zy} & S_{zz} \end{pmatrix}$$

where $\bar{\mathbf{m}}$ and $\bar{\mathbf{d}}$ denote the centroids of the data sets M and D, respectively. Let now $\mathbf{H} = \mathbf{U}\Lambda\mathbf{V}^T$ the singular value decomposition(SVD) of $\mathbf{H}$. $\mathbf{U}$ and $\mathbf{V}$ therefore fulfill the property of orthonormality, while $\boldsymbol{\Lambda}$ is diagonal with non-negative elements [8]. As prooved e.g in [37], the optimal transformation arises out of that factorization and is given by:

$$\mathbf{R} = \mathbf{V}\mathbf{U}^T \qquad\qquad \mathbf{t} = \bar{\mathbf{m}} - \mathbf{R}\bar{\mathbf{d}}$$

Under the circumstances of this thesis, the common ICP-registration requires a slight modification. More precisely, since the case of only 3 degrees of freedom is considered here, the fixation of the remaining ones has to be ensured. In terms of the rotation R, it is restricted to an occurance within the x-y-plane. To reach this aim, the matrix H was modified and any dependencies with the third cartesian coordinate, denoted by z, set to 0:

$$H = \begin{pmatrix} S_{xx} & S_{xy} & 0 \\ S_{yx} & S_{yy} & 0 \\ 0 & 0 & S_{zz} \end{pmatrix}. \tag{6.1}$$

This step seems reasonable, since H represents covariances and the underlying situation suggests independence to the z-component. Application of the ICP, as stated above results in a valid transformation then, what is proved with the following theorem.

Theorem 1. *Let H be the matrix from equation* (6.1) *and $H = U\Lambda V^T$ its singular value decomposition. The matrix $R = VU^T$ is orthogonal and of the form*

$$R = \begin{pmatrix} * & * & 0 \\ * & * & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

what adds up to a linear rotation, that does not change the z-coordinate.

Proof. First of all, the orthogonality follows from the general proof in [37]. Let now

$$H = \begin{pmatrix} S_{xx} & S_{xy} & 0 \\ S_{yx} & S_{yy} & 0 \\ 0 & 0 & S_{zz} \end{pmatrix} =: \begin{pmatrix} \multicolumn{2}{c}{\mathrm{B}} & \begin{matrix} 0 \\ 0 \end{matrix} \\ 0 \; 0 & & S_{zz} \end{pmatrix}.$$

parameter	meaning	value
$maxRad$	initial maximum radius, in which a closest point will be searched	200mm
$radFac$	scaling factor, that is used to scale down $maxRad$ under specified conditions	20
$maxIterNum$	maximum number of iteration steps	500
$convErrRatio$	treshold, that defines a stop criterion and refers to the quotient of two succeeding error terms	0.9999
$convErrTh$	treshold, that defines a stop criterion and refers to the absolute error term	1

Table 6.1: Parameter explanation and the chosen values for the ICP registration

Since a singular value decomposition exists for any arbitrary real matrix [8], it is

$$B = \tilde{U}\tilde{\Lambda}\tilde{V}^T$$

in particular, where $\tilde{U}$ and $\tilde{V}$ are othogonal matrices and $\tilde{\Lambda}$ is diagonal with non-negative elements. Since $S_{zz} \geqslant 0$ it follows

$$H = \begin{pmatrix} \multicolumn{2}{c}{\multirow{2}{*}{B}} & 0 \\ & & 0 \\ 0 & 0 & S_{zz} \end{pmatrix} = \underbrace{\begin{pmatrix} \multicolumn{2}{c}{\tilde{U}} & 0 \\ & & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{U} \cdot \underbrace{\begin{pmatrix} \multicolumn{2}{c}{\tilde{\Lambda}} & 0 \\ & & 0 \\ 0 & 0 & S_{zz} \end{pmatrix}}_{\Lambda} \cdot \underbrace{\begin{pmatrix} \multicolumn{2}{c}{\tilde{V}} & 0 \\ & & 0 \\ 0 & 0 & 1 \end{pmatrix}^T}_{V^T} .$$

Obviously $H = U\Lambda V^T$ is a valid singular value decomposition, thus the statement follows by calculating R

$$R = VU^T = \begin{pmatrix} \multicolumn{2}{c}{\tilde{V}\tilde{U}^T} & 0 \\ & & 0 \\ 0 & 0 & 1 \end{pmatrix},$$

which has the form of a rotation matrix, that does not manipulate the z-component.

□

In practical application a number of parameters are required for ICP registration. Table 6.1 explains these parameters and specifies the values, that were chosen here.

6.2 Robot Motion

In [44] there are two different models specified, namely the *velocity model*, which assumes, that the robot can be controlled by a translational and a rotational velocity, and the *odometry model*, which involves odometry readings instead of control commands. In this work, the latter one will be described, since the robot, that was employed, provides odometry measurements, that are usually more accurate [44]. The input is of the form, that was stated in equation (3.1) and describes a relative state inside a robot-intern coordinate frame, from which a 3 step sequence of movement uniquely can be derived [44].

1. an initial turn by a specific angle, denoted by δ_{rot1}
2. a translational component, describing a straight on movement δ_{trans}
3. a final rotation, aligning the robot towards its new heading direction δ_{rot2}

The motion model used for the “omnirob” platform is close to the one described above. A difference originates from the omnidirectional wheels, which redundantize an initial rotation. Furthermore, the remaining rotation (now denoted by δ_{rot}) and translation are executed simultaneously, resuming the movement along a straight line. Experiments with a former platform indicated, that proceeding like this yields to more accurate movements. For reasons of simplicity, errors for translation and rotation are assumed to be independent. Let $\hat{\delta}_{trans}$ and $\hat{\delta}_{rot}$ denote the measured values for translation and rotation, respectively. The following error model is supposed to be at the basis of the odometry measurements:

$$\begin{aligned} \hat{\delta}_{trans} &= \alpha \cdot (|0.05 \cdot \delta_{trans}| + 1) \cdot X & X &\sim \mathcal{N}(0, I_2) \\ \hat{\delta}_{rot} &= \beta \cdot (\delta_{rot} + 1) \cdot U & U &\sim \mathcal{U}([-0.5; 0.5]), \end{aligned}$$

where I_2 denotes the 2×2 identity matrix. The model provides errors with dependence to the degree of the movement, but involves an offset as minimal error. In the end, the values of the model were chosen heuristically, but have never been estimated properly.

6.3 Particle weighting and resampling

A straightforward way to evaluate particles recursively was given in equation (4.3). Following the suggestion in [34], the proposal distribution π was chosen as the motion model

$$\pi(x_k | x_{1:k-1}, z_{1:k}, u_{1:k-1}) = p(x_k | x_{k-1}, u_{k-1}),$$

what leads to a compact weight update:

$$w_k^{(i)} = p(z_k | x_k) \cdot w_{k-1}^{(i)}$$

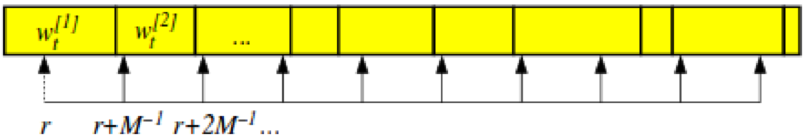


Figure 6.2: Scheme of "Low Variance Sampling". By sampling a random number r and adding M^{-1} repeatedly, one gets M samples again, each corresponding to a single particle.

Assuming Gaussian measurement noise and taking independence between measurements into account, this would lead to an expression like the following one:

$$\begin{aligned} w_k^{(i)} &= w_{k-1}^{(i)} \cdot \prod_i \exp\{(z_k^{(i)} - \hat{z}_k^{(i)})^T Q_k^{-1} (z_k^{(i)} - \hat{z}_k^{(i)})\} \\ &= w_{k-1}^{(i)} \cdot \exp\{\sum_i (z_k^{(i)} - \hat{z}_k^{(i)})^T Q_k^{-1} (z_k^{(i)} - \hat{z}_k^{(i)})\} \end{aligned} \tag{6.2}$$

with some covariance matrices Q_k and estimated observation $\hat{z}_k^{(i)}$. However, implementing weighting this way has proved to be extremely difficult, since incompletenesses within the partially build map made an observation estimate $\hat{z}_k^{(i)}$ a complex task. To keep the implementational effort small, an approximative alternative had to be used. The sum in the latter term of equation (6.2) has therefore been substituted by the error, that results from the ICP registration. The new update recursion is given by:

$$\begin{aligned} w_k &= w_{k-1} \cdot \exp\{E(R,t)\} \\ &= w_{k-1} \cdot \exp\{\sum_i \|m_i' - Rd_i'\|\} \end{aligned}$$

Since misalignments often appear, because the ICP algorithm converges to a local minimum instead of the global one, this substitution is meaningful in terms of the map's quality. Proceeding like this makes the whole algorithm come along without the need for a measurement model.

The resampling procedure follows a Quasi Monte Carlo approach, called *Low Variance Sampling* [44]. It aims to cover the sample space in a more systematic way, by generating a single random number and computing the remaining ones deterministically. Without loss of generality, the sample weights sum up to 1. Let M the absolute number of particles and r a random number generated from a uniform distribution $\mathcal{U}([0; M^{-1}])$. By refering each weight to a specific range inside the interval $[0; 1]$, one gets M samples again. The proceeding is illustrated in Figure 6.2. Using Low Variance Sampling conserves particle variety better than sampling independently [44]. This is of particular importance, since the number of samples is small. In addition, maps with above average weight are guaranteed to remain in the set.

6.4 Navigation

Autonomeous navigation of robots is a feature, that might be of good use in SLAM. As shown e.g. in [42], the choice of an adequate destination, potentially raises the map quality significantly. Several algorithms for efficient exploration of unknown environments exist. However, most of them aim to explore the environment as fast as possible and do not face the problem of consistent mapping [3]. On the other hand, common SLAM approaches do not deal with the problem of sensor placement. Combining both, exploration and SLAM-techniques, leads to the so-called *integrated approaches* [42]. The subject of this thesis however, was not to implement such an algorithm, but to adapt an already existing exploration implementation for the use in SLAM. The general procedure, which is at the basis of the implementation, is close to the *Next-Best-View*(NBV) algorithm presented in [3] and can be subsumed into 2 steps.

calculate potential aims: Since the aim of an exploration strategie is to discover its environment as fast and efficiently as possible, the general idea is to look for destination points, where the expected amount of newly observable voxels is large. On the other hand, navigation has to be save and each new destination has to be in known area. Thus potential aims turn out to be the collection of reachable and already observed points. In general such a set could look like the one in Figure 6.3(b), with border voxels, as potential destinations, marked in red. However, because the robots own prolongation has to be taken into account as well, the set of reachable points is reduced by those voxels, that are close to an obstacle or unknown area. The reduced set is shown in Figure 6.3(c). Finally, a single destination per border is determined, as is also shown in (c).

evaluate potential aims: After discovering several candidate positions, the most promising one has to be figured out. Exploration strategies pursue the two conflicting objectives of detecting as many unknown voxels as possible on the one hand and of keeping the costs in time and distance low on the other hand. It is therefore common proceeding to calculate a score that reflects a compromise between the costs and benefits. Let q denote an arbitrary potential destination, $A(q)$ the maximal number of unobserved but from q observable voxels and $B(q)$ the distance to travel. Following the approach in [3], the formula for the score

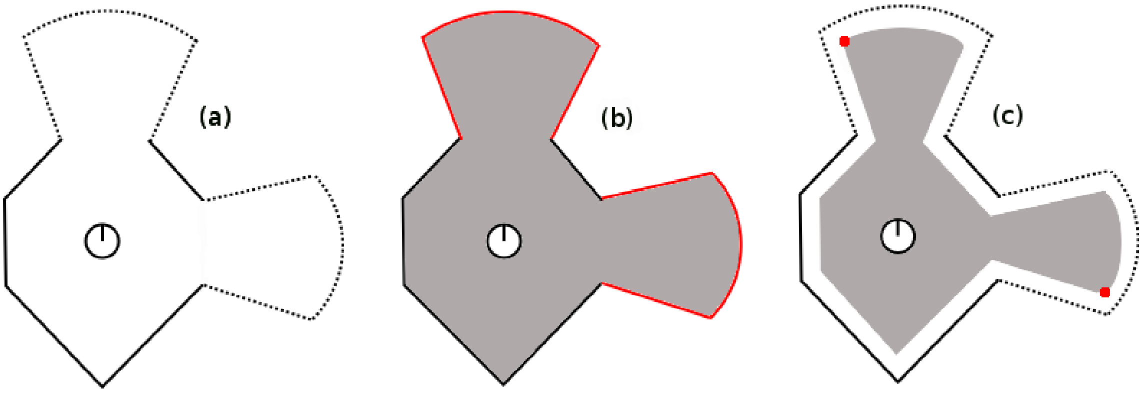


Figure 6.3: Exploration process as it is implemented.
(a) The solid lines represent obstacles, that have been observed, the dashed lines borders between known and unknown terrain.
(b) Reachable voxels are grey marked, two borders exist and are colored in red.
(c) The set of reachable voxels has been reduced to consider the robots own prolongation and a potential aim was determined per border.

of a pose q can be specified through

$$S(q) = A(q) \cdot e^{-\lambda \cdot B(q)}$$

The parameter λ determines the meaning of the costs. The higher λ, the more expensive it is to move the robot.

Having this, the new pose can be determined easily as the potential aim with the highest score. Similar to the approach in [42], only the particle with the highest weight is used for planning the next destination.

However, the implemented version can not be considered to be working, since it does calculate a new point of destination, but does not provide a reasonable path planning tool to get to that point. What is more, the proceeding as stated here neglects the subject of revisiting areas actively, what might reduce the maps quality especially in cyclic environments [41]. Finally, the use of exploration techniques in a SLAM-approach as the one provided here, requires sufficient overlap between two subsequent scans, since the scan matching tends to fail otherwise. Although there exist methods to ensure mutual ranges of the required size [3, 42], the established implementation does not provide that functionality. Thus the results of this thesis were generated via predefined poses. This poses were won via a path planning algorithm, that requires a given map. Figure 5.2 includes a graphical representation of the path, that was used.

7 Implementation Structure

Certainly the most challenging and time-consuming part of this thesis was the practical realization of the algorithm described previously. Before presenting the final results, this section therefore takes a look at some implementational aspects. Like all applications in the L3D-library, the SLAM procedure was realized within the programming language *C++*. The first step in explaining the structure will be a bird's eye view of the overall design, which aims to highlight the most important classes and the way, they are linked to each other. Afterwards some important elements will be explained more detailed to point out some of the challenges, that had to be overcome.

7.1 Overall structure

A summary of the program design is given by Figure 7.1. The UML-class-diagram shown there demonstrates the most important classes and highlights new established classes in green color, while already existing classes are blue- or red-marked, depending on whether they have been modified or not. At the root of the application is the MRE-project, which provides the main-method and comes with an object of the class **CommandLoop**. Instances of that class serve as steering elements of the MRE-functionality and as such are at the basis of the SLAM-application as well. For better understanding, it is helpful to have the diagram in mind while reading this section.

It seems reasonable to start the explanation with the representation of a single Rao-Blackwellized particle, which is given by an object of the class **MapSample**. These MapSample objects provide per-particle functionality and each holds its seperate map in form of a **ProbabilisticSpace**, which is, as a matter of fact, a **DynamOctreeSpace** object, that stores probabilistic values. Furthermore, it contains an object, to represent the current robot pose and a weight, that represents its likelihood. The collection of MapSample objects is administrated by the **SLAM** class, which has to be seen as the core of the program, since it provides the superior functionality of the algorithm.

A representation of the robot is given by the **GenericMobileRobot**(GMR) class. This class executes some of the most basic tasks inside the algorithm, like moving, observing or delivering information about the robots states and properties. It shall

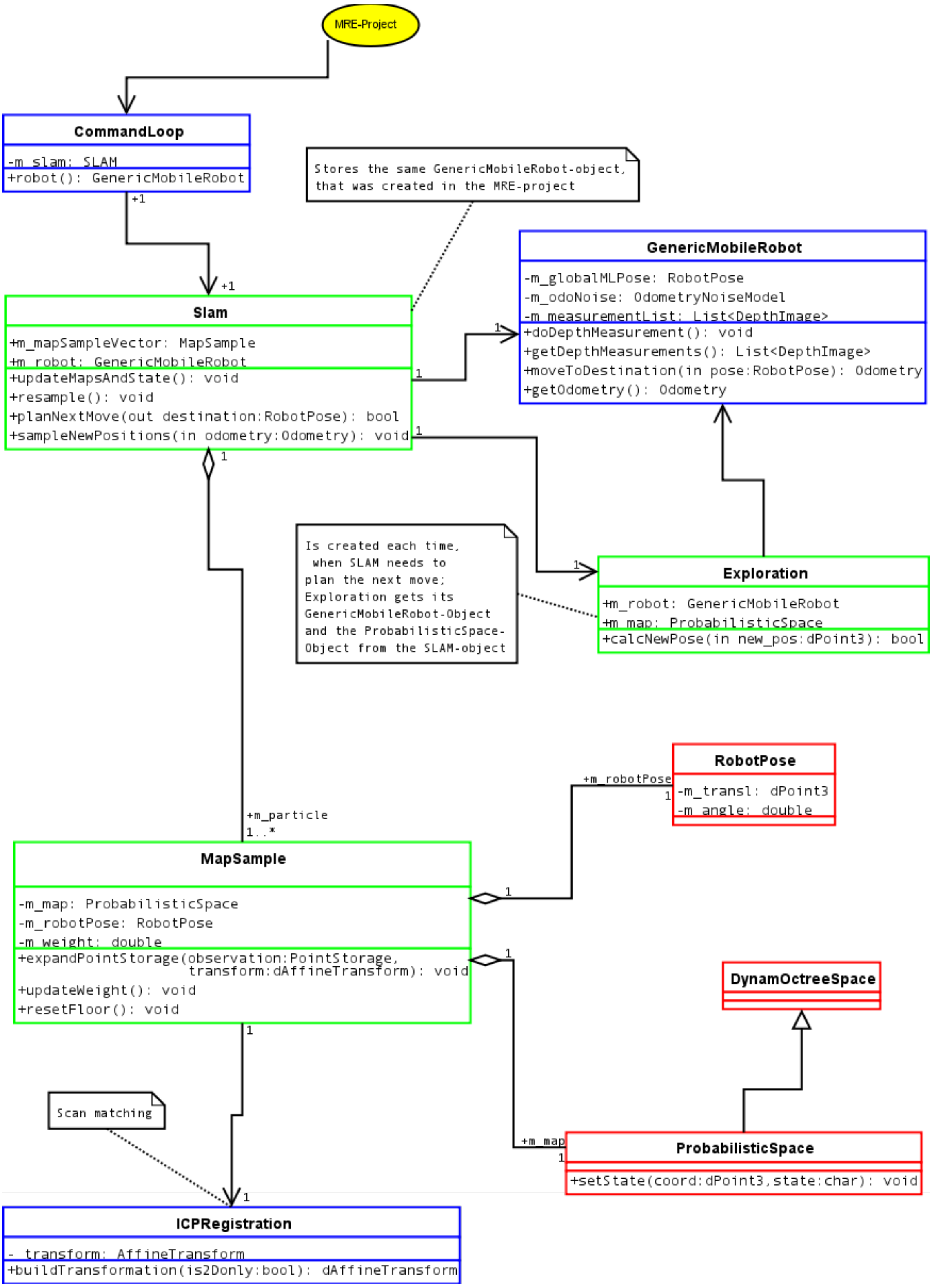

MRE-Project
CommandLoop
-m_slam: SLAM
+robot(): GenericMobileRobot
+1
Stores the same GenericMobileRobot-object,
that was created in the MRE-project
+1
Slam
+m_mapSampleVector: MapSample
+m_robot: GenericMobileRobot
+updateMapsAndState(): void
+resample(): void
+planNextMove(out destination:RobotPose): bool
+sampleNewPositions(in odometry:Odometry): void
1
1
1
GenericMobileRobot
-m_globalMLPose: RobotPose
-m_odoNoise: OdometryNoiseModel
-m_measurementList: List<DepthImage>
+doDepthMeasurement(): void
+getDepthMeasurements(): List<DepthImage>
+moveToDestination(in pose:RobotPose): Odometry
+getOdometry(): Odometry
Is created each time,
when SLAM needs to
plan the next move;
Exploration gets its
GenericMobileRobot-Object
and the ProbabilisticSpace-
Object from the SLAM-object
1
Exploration
+m_robot: GenericMobileRobot
+m_map: ProbabilisticSpace
+calcNewPose(in new_pos:dPoint3): bool
1
RobotPose
-m_transl: dPoint3
-m_angle: double
+m_robotPose
1
+m_particle
1..*
MapSample
-m_map: ProbabilisticSpace
-m_robotPose: RobotPose
-m_weight: double
+expandPointStorage(observation:PointStorage,
transform:dAffineTransform): void
+updateWeight(): void
+resetFloor(): void
1
1
DynamOctreeSpace
1
Scan matching
+m_map
1
ProbabilisticSpace
+setState(coord:dPoint3,state:char): void
1
ICPRegistration
-_transform: AffineTransform
+buildTransformation(is2Donly:bool): dAffineTransform

Figure 7.1: UML-class diagram. New classes are green colored, blue classes were modified and red ones were used without modification. The central classes are the *Slam* class, which executes the superior functionality of the SLAM algorithm and the class *MapSample*, that represents a particle within each object.

be pointed out, that SLAM objects hold a GMR object, that was already created in the MRE-project before and passed by reference. Both GenericMobileRobot objects used here are therefore memorized at the same adress in memory. This is important, since changes of its states or additional knowledge, are supposed to be known or might even be essential in different functions of the MRE-project.

Special attention has to be given to the **Exploration**-class. Although most of its functionality was already implemented, it was stored in the CommandLoop class, which lost its slimness and clearness due to the excessive accommodation of source code. The elements of exploration have therefore been split up into small subroutines and outsourced, following the principles of object-orientated programming. To plan the best next position, an exploration object requires a GMR object, which again is the same as the one in the Slam class and a map, that is taken from the most likely MapSample object.

Furthermore, attention shall be given to the class **ICPRegistration**, that is used to determine a transformation, consisting of a translation and a rotation and describing the suggested pose correction in scan matching.

7.2 Important implementational aspects

The description of the most important implementation details will now be done in top down action. The major control loop of the algorithm, as most superior unit, is shown as algorithm 1. This short piece of pseudo source code summarizes the algorithm in terms of the most crucial steps and is implemented in this way in the routine `doSlam()` of the CommandLoop class.

The initial poses were chosen to be $(x, y, \Theta) = (0, 0, 0)$, what means location at the origin and heading in x-direction.

7.2.1 Matching frames

When searching for an optimal alignment in a grid-based map one comes across the problem of discretization errors. Additional to the noise, that is caused by the sensors, the accuracy in voxel representation is limited by the resolution. Therefore, there is need for an alternative representation, which is fulfilled by storing the measurements into point clouds. Lib3D allocates those point clouds in form of a container class called *PointStorage*. Each particle comprises a permanent PointStorage object, containing past measurements, and a temporal point cloud, storing the actual measurement in form of points in space. However, while depth values declare distances relative to the camera, point storages need to be known in global coordinates for registration. Hence, conversion of measurements into PointStorage objects requires the comprehension of an appropriate transformation. Given two global point

```
set initial poses;
while true do
    if not at start pose anymore then
        for each MapSample object do
            update robot position;
        end
    end
    for each MapSample object do
        update map;
        update weight;
    end
    resample according to the weights;
    calc new destination;
    if no reasonable destination available then
        STOP;
    end
    move robot to destination;
    for each MapSample object do
        sample new position;
    end
end
```

Algorithm 1: Control loop of the Slam-algorithm

storages, application of scan-matching can be done in a straight-forward fashion. The modification of the procedure was described in 6.1 and did not require high implementational effort.

7.2.2 Updating maps

Most of the former implemented functionings of the MRE-project take use of a construct, known as the local map, which reproduces the immediate vicinity of the robot. The established function `updateLocalMap()` for integrating new observations into that formation has much in common with the procedure, that is nessessary to build a general global map incrementally. Since avoidance of redundant code is a principles of object-orientated programming, the established routine has been split up and a common usable part was outsourced in the novel and more general function `updateMap(...)`. The newly defined function `updateWorldMap(...)` was equipped with some more specific source code and, this way, enabled to update an arbitrary map based on an arbitrary pose. The general operating mode is made clear in the following few lines of source code.

```
updateLocalMap(){
  specific code
  updateMap(local map)
  specific code
}
```

```
updateWorldMap(global map){
  specific code
  updateMap(global map)
  specific code
}
```

Listing 7.1: Updating local and global map after the modification. The established method has been split up and a common usable part was outsourced as the function 'updateMap()'

For reasons of higher accuracy, the robot turns by 26 degrees to its left and right respectively and updates the map at three different poses, taking new camera measurements at each pose.

Early versions of this implementation were bound to choose an initial size for the environment to be mapped. In practice one might find situations where a parsimonious estimation of this quantity is not available. Later implementations overcame this restriction by using the point cloud, that was already used for ICP registration and simply extended the space sufficiently to embrace each of the corrected points. Recently the update procedure itself was enabled to extend the map while inserting new measurements, what provokes the step just described to be redundant.

7.2.3 Exploring the environment

It has already been stated before, that the exploration-functionality in the MRE-project can not be regarded as fully developed. A major drawback occurs from map building, where the previously introduced local map has been enlarged sufficiently to cover the entire mapping area and has been updating incrementally then. This proceeding is neither elegant, nor suitable for application in the SLAM algorithm, since the size of a room has to be known initially. Thus, this nuisance was revised and the Exploration object enabled to deal with arbitrary maps, which are dynamic in their size.

In addition, a change within the evaluation of potential destinations was carried out. More precisely, the calculation of the benefit component was substituted, since the established one was unrealistic in a sense. The old version as well as the modernised one based on sending beams through the space and count the number of intersecting unobserved voxels. However, while originally the beams all based in a single point and there direction was determined via spherical coordinates, the recent version is somehow more realistic. It calculates the camera positions of the potential poses and therefore can simulate beams in a more accurate manner.

Whether this modification effects better results has not been prooved, but it can't be denied, that the reengineered version is closer to reality.

8 Results

This section finally presents the results, that were produced by the implemented algorithm. The number of particles was chosen by 5, what resulted in 5 different maps, from which the best one was taken. Each map was supposed to have a minimum voxel size of 40mm. The noise in measurements was supposed to be Gaussian and to grow stronger with increasing distances d, following the model

$$var(d) = (\gamma \cdot d^n)^2,$$

where var denotes the variance.

First of all, the outcome of the algorithm shall be examined carefully. This will offer the opportunity to demonstrate, why it is useful to combine ICP-based SLAM with the use of a Rao-Blackwellized particle filter, afterwards. Both subjects will be adressed in the following.

8.1 Mapping results

For checking the robustness of the algorithm, it was executed with moderate and strong variance in odometry- and sensor measurements in combination. For the purpose of simulation, the measurement depth values were provided with some zero mean Gaussian noise. The parameter values, that were chosen for the model introduced in section 6.2 and for the measurement variance, are summarized in Table 8.1. It shall be pointed out, that the parameters were chosen in the way, that strong noise in odometry and measurement has four times the variance of moderate noise.

For the purpose of better comparison, two reference maps were created. On the one hand, the environment was mapped in the absence of any odometry or measurement error. The result is pictured by Figure 8.1. On the other hand a map was produced without application of any pose correction technique. A similar result

	α	β	γ	n
moderate noise	1	0.1	0.2	1
strong noise	2	0.2	0.4	1

Table 8.1: Parameters chosen for moderate and strong noise

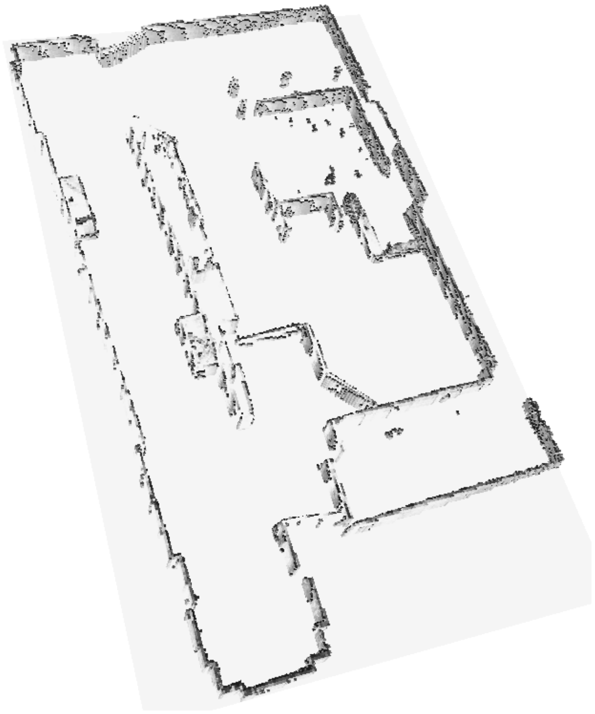


Figure 8.1: Noise-free map, constructed by turning off any odometry- or measurement error. This is the best possible outcome of a SLAM-algorithm. Comparison with Figure 5.2 shows, that table boards have not been mapped properly, since sensors were not able to detect them.

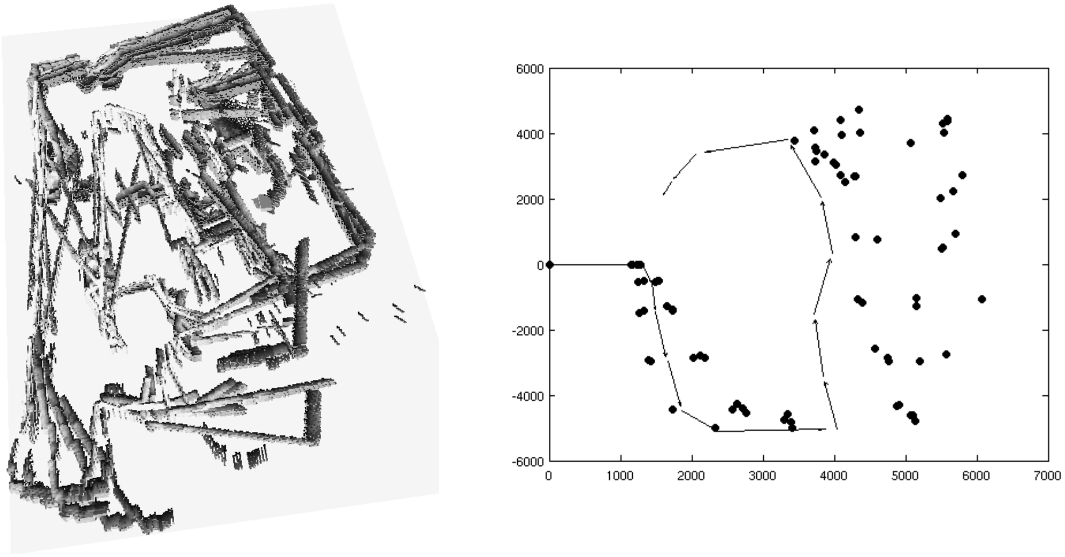


Figure 8.2: Mapping result with moderate noise, but without pose correction. The map is badly misaligned, similar to the one shown in Figure 8.2. The right-hand side shows the divergence of the sample poses.

was already shown in Figure 1.1 to introduce the general problem, that occurs with robotic mapping. Figure 8.2 states the same problem, but the map was created with the particle filter and pictures the sample poses along with the settled robot path additionally.

These two cases can be considered as extremely contradictive, since they result in one perfect and one highly inconsistent map respectively.

The first achievement was computed under the condition of moderate error variance in both, odometry and sensor measurements. In addition to map building, the trajectories were recorded and the position beliefs summarized in a graphic. The results are shown in Figure 8.3. The shapes have clearly been reconstructed from the environment, that is shown in Figure 5.2. Marginal inaccuracies occur at the positions of tables, since those are easy to overlook for visual sensors. In comparison to Figure 8.1, walls look more expanded in some way. That is due to measurement variance, that affects uncertainties about the location of objects. Facing the second graphic, the effect of ICP pose correction becomes obvious. In contradiction to the original distribution from Figure 8.2, most of the position beliefs are close to the robot's true path. This result is particularly suitable to demonstrate the effect of resampling, since one of the sampled trajectories seems to drift away from the true robot path in the beginning. Its sudden disappearance is an obvious sign, that the corresponding particle has been eliminated by resampling.

The second case to consider is the one with increased measurement variance, which is shown in Figure 8.4. Similar to the result before, the environment's shapes

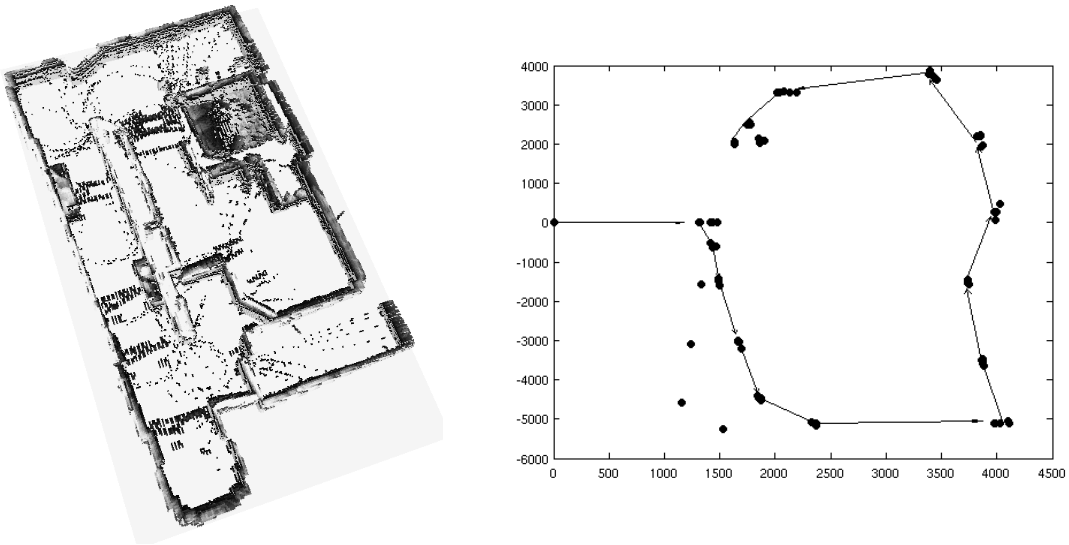


Figure 8.3: Mapping result, constructed under moderate odometry and measurement noise. The scans have been aligned and the map is consistent. Most of the sample poses are focused along the true robot path.

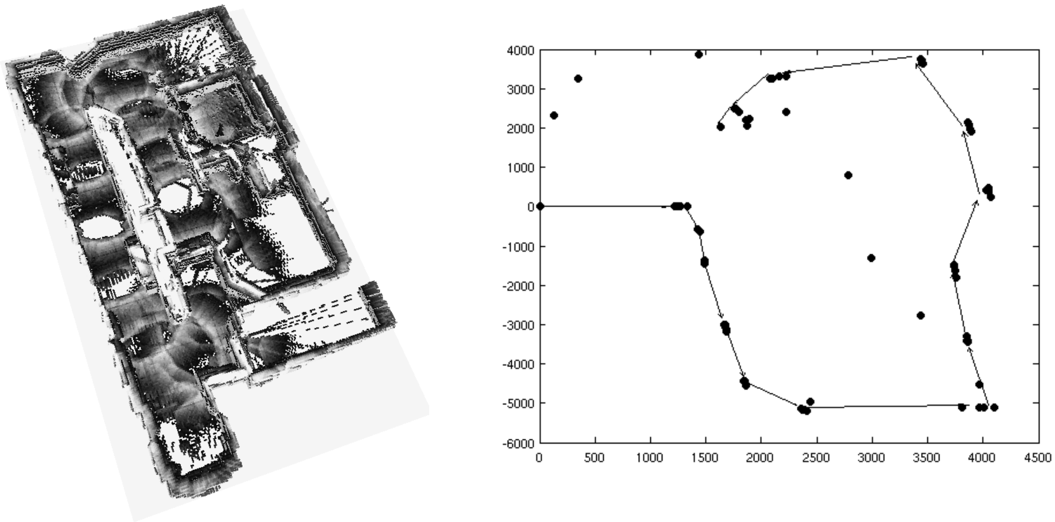


Figure 8.4: Mapping result with moderate odometry- and strong measurement-noise. Although the effects of stronger sensor noise are clearly visible through the dark floor, the alignment worked well. The sample poses on the right-hand side confirm that impression.

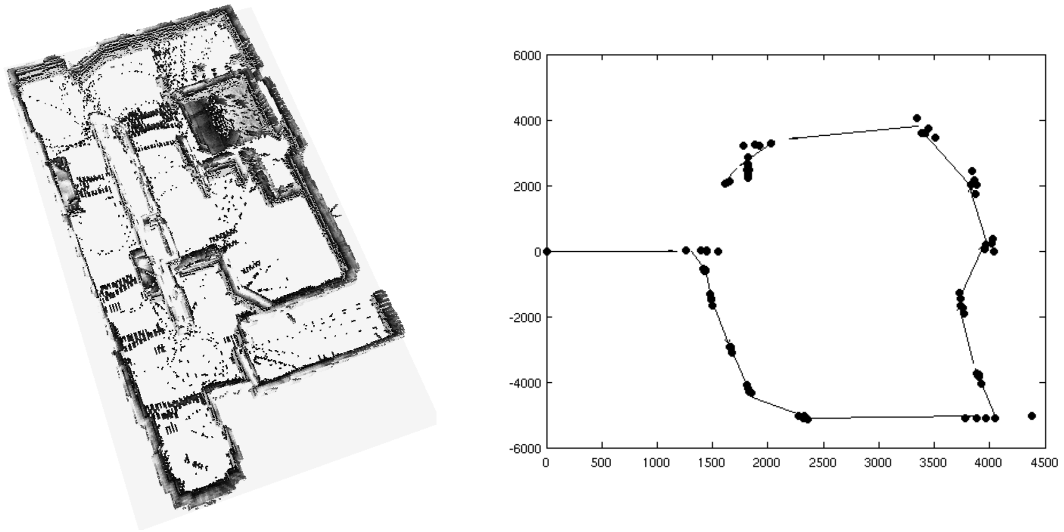


Figure 8.5: Mapping result, computed with strong odometry- and moderate measurement-noise. The map is consistent, although the fraction of samples, that drift away from the true path is significantly higher, than it was with moderate odometry noise.

are recognizable, although the effect of strong noise is clearly visible. The darker appearance of the map is due to the noise, that affects the smoothness of the floor. In path planning, this might lead to problems, when the robot recognizes too many obstacles. However, the alignment of the scans seems to have worked very well. A look at the trajectories registers, that most of the robot poses reside close to the true path again, by chance even better than it was the case with normal measurement noise.

More interesting than increased measurement variances is the case, where the odometry noise is enlarged, since the accumulation of motion error increases the risk of divergence. As can be seen in Figure 8.5, the algorithm proved to be robust even here. However, considering the sample poses clarifies the effect of the increased odometry uncertainty. While at least some of the samples approximate the true path, a significant amount of poses deviate from it, what highlights the necessity for a proper resampling even more.

Finally, the result with increased variance in both, odometry- and sensor- noise, confirms the robustness of the algorithm again. Although the map itself looks quite consistent, it has to be figured out, that even the best poses differ from the true trajectory by the end, as can be seen at the right hand side of Figure 8.6. This indicates slight misalignments during the mapping process, what might be a hint, that the algorithm reaches the limit of its robustness within this scenario.

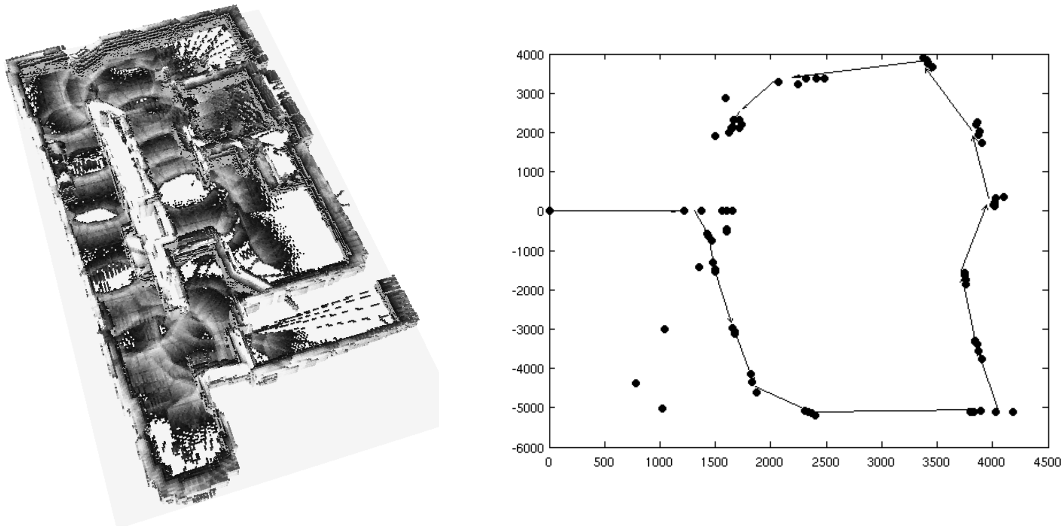


Figure 8.6: Mapping result, where odometry- and sensor- noise are increased. The map looks still consistent, but the sample poses indicate small misalignments at the end of the path.

8.2 Misalignments caused by ICP

Finally, the procedure of combining Rao-Blackwellized particles with utilization of the ICP algorithm is justified. The reader might have noticed from the results, that some of the trajectories differ significantly from the true path. Figure 8.7 shows one particular map, which is subject to such a false trajectory. Even without comparison to the true environment from Figure 5.2, it gets clear, that the map is badly misaligned. Those inconsistencies are caused by registration errors, that have its source in the failure of at least one ICP-step. The use of a small number of Rao-Blackwellized particles in combination with the ICP-algorithm avoids the final result from suffering from misalignments. Scan matching from a slightly different pose might avoid a registration error and resampling removes inconsistent and therefore unlikely particles.

8.3 Performance

The major drawback of the implementation is given by its time consumption. Building a consistent map with the given configurations leads to an expanditure of time up to half an hour, using a single CPU. The operative point is the complex search for corresponding points within the ICP algorithm. Since this search has to be done not once, but once in each iteration step and in each time step, it sums up to large computation times. A second, but less significantly time-consuming part is the map update step.

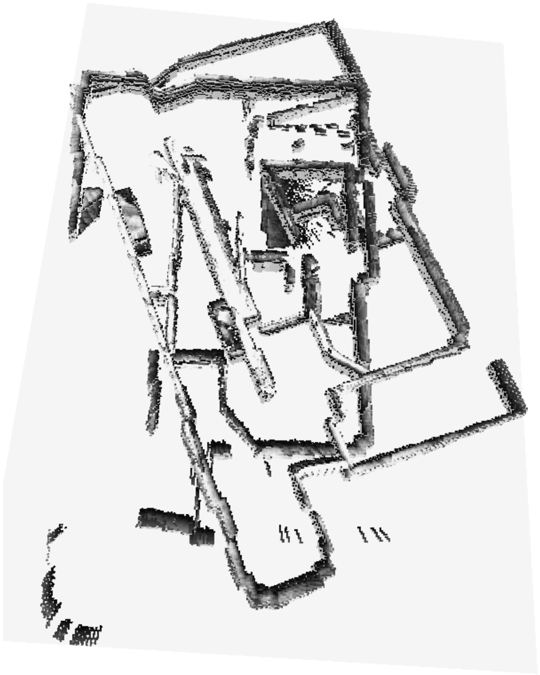


Figure 8.7: Misaligned map, computed by the algorithm with a single sample. The absence of further particles avoided the elimination of the consequence from a failed ICP registration.

All computation have been performed on a Quad Core PC, equipped with the Linux-distribution *CentOS 6.4*.

9 Review and Outlook

The aim of this thesis was to overview the problem of *Simultaneous Localization and Mapping* and develope and implement a robust algorithm for indoor application. Next to the general mathematical formulation, three popular approaches to solve the SLAM problem were presented, namely EKF-SLAM, graph-based SLAM and Rao-Blackwellized Particle filter. The latter one was combined with a registration technique, to build the centerpiece of the indoor algorithm. Furthermore, an exploration technique was integrated, but not applied, since it transpired not to be fully developed. The implemented algorithm performed well and prooved to be robust, even under the influence of generous amount of odometry- and measurement- noise. As the most distracting disadvantage, the time performance has to be mentioned, since it takes up to half an hour to compute a consistent map with 5 samples.

Further works, based on the hitherto existing results should, amongst others, concentrate on improving the exploration. The existing one lacks a good path planning procedure, what makes the algorithm practically infeasible. In addition, it does face the problem of finding a reasonable new position, but neglects the question of the best new facing direction. Secondly, the computationally expensive correspondence search within the ICP-algorithm yields to the high expenditure of time. Since the quality of the results turned out to be very well, an approximative but faster searching procedure might still produce sufficiently accurate results. Moreover, the particle weighting procedure is based on the error result from the ICP-algorithm at present. Although this works very well, it is just an approximation, in that errors are assumed to be distributed according to the measurement model. Embracing this into the implementation might lead to a better performance with even less particles neccessary. Finally, the very difficult topic of loop detection was neglected completely in this work, but has to be taken into account, especially for the use in cyclic environments.

Including all those points might lead to a robost and likewise fast SLAM-algorithm. To prove that speculation is up to subsequent researchers.

Appendix

Update step in EKF-SLAM

$$
\begin{aligned}
y_{k-1} &\coloneqq \begin{pmatrix} x_{k-1} \\ m_1 \\ \vdots \\ m_N \end{pmatrix} \\
F_x &\coloneqq \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ \vdots & \vdots & \vdots \\ 0 & 0 & 0 \end{pmatrix} \\
\tilde{g}(y_{k-1}, u_k) &\coloneqq y_{k-1} + F_x \cdot g(x_{k-1}, u_{k-1}) + \begin{pmatrix} \mathcal{N}(0, G_k) \\ 0 \\ \vdots \\ 0 \end{pmatrix} \\
g_k &= \frac{\partial g}{\partial x_{k-1}}(\mu_{k-1}, u_{k-1}) \\
G_k &= \frac{\partial \tilde{g}}{\partial y_{k-1}}(u_k, \mu_{k-1}) = I + F_x^T g_k F_x \\
\bar{\mu}_k &= \tilde{g}(\mu_{k-1}, u_{k-1}) \\
\bar{\Sigma}_k &= G_k \Sigma_{k-1} G_k^T + F_x^T R_k F_x \\
\hat{z}_k^i &= h(\bar{\mu}_k, c_k^i) \\
H_k &= \frac{\partial h}{\partial y_k}(\bar{\mu}_k, j) \\
C_k &= \bar{\Sigma}_k H_k^T (H_k \bar{\Sigma}_k \cdot H_k^T + Q_k)^{-1} \\
\bar{\mu}_k &= \bar{\mu}_k + C_k(z_k - h(\mu_k)) \\
\bar{\Sigma}_k &= (I - C_k \cdot H_k) \cdot \bar{\Sigma}
\end{aligned}
$$

Algorithm 2: Extended Kalman Filter for SLAM

Bibliography

[1] Vazha Amiranashvili. *Robust Real-Time Localization and Mapping in Single and Multi-Robot Systems*. PhD thesis, Universität Müenchen, 2007.

[2] Josep Aulinas, Yvan Petillot, Joaquim Salvi, and Xavier Lladó. The slam problem: a survey. In *Proceeding of the 2008 conference on Artificial Intelligence Research and Development*, pages 363–371, 2008.

[3] Nicola Basilico, Francesco Amigoni, Letizia Tanca, and Barbara Pernici. *Navigation Strategies for Exploration and Patrolling with Autonomous Mobile Robots*. PhD thesis, Politecnico di Milano, 2010.

[4] Paul J. Besl and Neil D. Mckay. A method for registration of 3-d shapes. *IEEE Transactions PAMI*, 14(2):239–256, February 1992.

[5] José A. Castellanos, JMM Montiel, José Neira, and Juan D. Tardós. The spmap: A probabilistic framework for simultaneous localization and map building. *Robotics and Automation, IEEE Transactions on*, 15(5):948–952, 1999.

[6] José A. Castellanos, José Neira, and Juan D. Tardós. Limits to the consistency of ekf-based slam 1. In *Proceedings of the 5th IFAC Symposium on Intelligent Autonomous Vehicles*. Citeseer, 2004.

[7] Randall Smith Matthew Self Peter Cheeseman, R. Smith, and M. Self. A stochastic map for uncertain spatial relationships. In *4th International Symposium on Robotic Research*, pages 467–474, 1987.

[8] Peter Deuflhard and Hohmann Andreas. *Numerische Mathematik 1*, volume 2. de Gruyter, 1993.

[9] MWM Gamini Dissanayake, Paul Newman, Steve Clark, Hugh F Durrant-Whyte, and Michael Csorba. A solution to the simultaneous localization and map building (slam) problem. *Robotics and Automation, IEEE Transactions on*, 17(3):229–241, 2001.

[10] Andreas Dömel, Simon Kriegel, Manuel Brucker, and Michael Suppa. Autonomous pick and place operations in industrial production environments. In *IEEE ICRA*, Hongkong, China, May 2014. Submitted.

[11] Arnaud Doucet, Nando De Freitas, Kevin Murphy, and Stuart Russell. Rao-blackwellised particle filtering for dynamic bayesian networks. In *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, pages 176–183. Morgan Kaufmann Publishers Inc., 2000.

[12] Arnaud Doucet, Simon Godsill, and Christophe Andrieu. On sequential monte carlo sampling methods for bayesian filtering. *Statistics and computing*, 10(3):197–208, 2000.

[13] Arnaud Doucet and Adam M Johansen. A tutorial on particle filtering and smoothing: Fifteen years later. *Handbook of Nonlinear Filtering*, 12:656–704, 2009.

[14] Hugh Durrant-Whyte and Tim Bailey. Simultaneous Localization and Mapping (SLAM): Part I the essential algorithm. *Robotics & Automation Magazine*, 13(2):99–110, 2006.

[15] Austin Eliazar and Ronald Parr. Dp-slam: Fast, robust simultaneous localization and mapping without predetermined landmarks. In *IJCAI*, volume 3, pages 1135–1142, 2003.

[16] G. H. Givens and J. A. Hoeting. *Computational Statistics.* Wiley, 2005.

[17] Giorgio Grisetti, Rainer Kümmerle, Cyrill Stachniss, and Wolfram Burgard. A tutorial on graph-based slam. *Intelligent Transportation Systems Magazine, IEEE*, 2(4):31–43, 2010.

[18] 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 2005 IEEE International Conference on Robotics and Automation, ICRA 2005*, pages 2432–2437. IEEE, 2005.

[19] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE Transactions on Robotics*, 23:34–46, 2007.

[20] Giorgio Grisetti, Gian Diego Tipaldi, Cyrill Stachniss, Wolfram Burgard, and Daniele Nardi. Fast and accurate slam with rao-blackwellized particle filters. *Robotics and Autonomous Systems*, 55:30–38, 2006.

[21] Dirk Hahnel, Wolfram Burgard, Dieter Fox, and Sebastian Thrun. An efficient fastslam algorithm for generating maps of large-scale cyclic environments from raw laser range measurements. In *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 1, pages 206–211. IEEE, 2003.

[22] Dirk Hahnel, Dirk Schulz, and Wolfram Burgard. Map building with mobile robots in populated environments. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, volume 1, pages 496–501. IEEE, 2002.

[23] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 34(3):189–206, February 2013.

[24] Shoudong Huang and Gamini Dissanayake. Convergence and consistency analysis for extended kalman filter based slam. *Robotics, IEEE Transactions on*, 23(5):1036–1049, 2007.

[25] Fumio Kanehiro, Takahashi Yoshimi, Shuuji Kajita, Mitsuharu Morisawa, Kiyoshi Fujiwara, Kaneko Harada, Kenji Kaneko, Hirohisa Hirukawa, and Fumiaki Tomita. Whole body locomotion planning of humanoid robots based on a 3d grid map. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation, ICRA 2005*, pages 1072–1078. IEEE, 2005.

[26] Henrik Kretzschmar, Giorgio Grisetti, and Cyrill Stachniss. Lifelong map learning for graph-based slam in static environments. *KI-Künstliche Intelligenz*, 24(3):199–206, 2010.

[27] Nosan Kwak, Olivier Stasse, Torea Foissotte, and Kazuhito Yokoi. 3d grid and particle based slam for a humanoid robot. In *Humanoid Robots, 2009. Humanoids 2009. 9th IEEE-RAS International Conference on*, pages 62–67. IEEE, 2009.

[28] Robert Lange and Peter Seitz. Solid-state time-of-flight range camera. *Quantum Electronics, IEEE Journal of*, 37(3):390–397, 2001.

[29] N. Li, P. Cheng, M.A. Sutton, and S.R. McNeill. Three-dimensional point cloud registration by matching surface features with relaxation labeling method. *Experimental Mechanics*, 45(1):71–82, 2005.

[30] Feng Lu and Evangelos Milios. Globally consistent range scan alignment for environment mapping. *Autonomous robots*, 4(4):333–349, 1997.

[31] Stefan May, David Dröschel, Stefan Fuchs, Dirk Holz, and A Nuchter. Robust 3d-mapping with time-of-flight cameras. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 1673–1678. IEEE, 2009.

[32] Stefan May, Bjorn Werner, Hartmut Surmann, and Kai Pervölz. 3d time-of-flight cameras for mobile robotics. In *IROS*, pages 790–795. IEEE, 2006.

[33] 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.

[34] Michael Montemerlo, Sebastian Thrun, Daphne Koller, and Ben Wegbreit. Fastslam: A factored solution to the simultaneous localization and mapping problem. In *AAAI/IAAI*, pages 593–598, 2002.

[35] Kevin Murphy. Bayesian map learning in dynamic environments. *Advances in Neural Information Processing Systems (NIPS)*, 12:1015–1021, 1999.

[36] Kevin Murphy and Stuart Russell. Rao-blackwellised particle filtering for dynamic bayesian networks. In *Sequential Monte Carlo methods in practice*, pages 499–515. Springer, 2001.

[37] Andreas Nüchter, Kai Lingemann, Joachim Hertzberg, and Hartmut Surmann. 6D SLAM with approximate data association. In *Advanced Robotics, 2005. ICAR'05. Proceedings., 12th International Conference on*, pages 242–249. IEEE, 2005.

[38] John G Rogers, Alexander JB Trevor, Carlos Nieto-Granda, and Henrik Iskov Christensen. Slam with expectation maximization for moveable object tracking. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 2077–2082. IEEE, 2010.

[39] Hagit Shatkay and Leslie Pack Kaelbling. Learning topological maps with weak local odometric information. In *IJCAI (2)*, pages 920–929, 1997.

[40] Cyrill Stachniss. *Exploration and mapping with mobile robots.* PhD thesis, Universität Freiburg, 2006.

[41] Cyrill Stachniss, Giorgio Grisetti, and Wolfram Burgard. Recovering particle diversity in a rao-blackwellized particle filter for slam after actively closing loops. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 655–660. IEEE, 2005.

[42] Cyrill Stachniss, Dirk Hahnel, and Wolfram Burgard. Exploration with active loop-closing for FastSLAM. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings.*, volume 2, pages 1505–1510. IEEE, 2004.

[43] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. A real-time algorithm for mobile robot mapping with applications to multi-robot and 3d mapping.

In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 1, pages 321–328. IEEE, 2000.

[44] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. MIT Press, MA, 2005.

[45] Rudolph Van Der Merwe, Arnaud Doucet, Nando De Freitas, and Eric Wan. The unscented particle filter. In *NIPS*, pages 584–590, 2000.

[46] Jan Weingarten and Roland Siegwart. Ekf-based 3d slam for structured environment reconstruction. In *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*, pages 3834–3839. IEEE, 2005.

[47] Jan W Weingarten, Gabriel Gruener, and Roland Siegwart. A state-of-the-art 3d sensor for robot navigation. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 3, pages 2155–2160. IEEE, 2004.

[48] Greg Welch and Gary Bishop. An introduction to the kalman filter, 1995.

[49] K. M. Wurm, A. Hornung, M. Bennewitz, C. Stachniss, and W. Burgard. OctoMap: A probabilistic, flexible, and compact 3D map representation for robotic systems. In *Proc. of the ICRA 2010 Workshop on Best Practice in 3D Perception and Modeling for Mobile Manipulation*, Anchorage, AK, USA, May 2010. Software available at `http://octomap.sf.net/`.

[50] Kai M Wurm, Cyrill Stachniss, Giorgio Grisetti, and Wolfram Burgard. Improved simultaneous localization and mapping using a dual representation of the environment. In *EMCR*, 2007.

DECLARATION

Herewith I declare that I wrote my Master's Thesis without external support and that I did not use other than the quoted sources and auxiliary means.

Munich, October 29, 2013

Maximilian Eck