Graz University of Technology

Institute for Computer Graphics and Vision

Master Thesis

Simultaneous localisation and mapping for mobile robots with recent sensor technologies

---

Elmar A. Rückert

Graz, Austria, December 2009

Thesis supervisors

Univ.-Prof. Dipl.-Ing. Dr.techn. Horst Bischof

Dipl.-Ing. Matthias Rüther

Deutsche Fassung:
Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008
Genehmigung des Senates am 1.12.2008

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Graz, am ……. ……..
(Unterschrift)

Englische Fassung:

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

……. ……..
date (signature)

Abstract

Autonomous mobile robots need a map of the environment for navigation. *Simultaneous Localisation and Mapping* (SLAM) is essential for autonomous navigation, path planning and obstacle avoidance. SLAM describes a process of building a map of an unknown environment and computing at the same time the current robot position. Both steps depend on each other. A good map is necessary to compute the robot position and on the other hand just an accurate position estimate yields to a correct map. Several popular SLAM packages, like *DP-SLAM*, *GMapping* or *GridSLAM* are available for research purposes and allow a not yet available and meaningful comparison between sensors and algorithms. The aim of this work is to find a robust method to generate 2D or 3D maps with recent sensor technologies. We compare a grid based method with two implementations of geometric feature based SLAM algorithms. All methods rely on a probabilistic estimate of the robot state realised with a *Particle Filter*. Recent sensor technologies: Laser range finders, sonar sensors and time of flight cameras are evaluated with respect to accuracy and robustness. Laser beam based sensors yield to the most exact results and are commonly used. Because of the low price of sonar sensors, ambitious efforts are being made to build cheap household robots. The last sensor technology, listed, is the newest and allows 3D scans of the environment. The experiments take place in indoor environments and a quantitative evaluation of the results is performed with the recently published *RawSeeds* datasets.

Keywords. SLAM, particle filter, time of flight, laser scanner, sonar sensor, sensor fusion, line features, corner features, player/stage, rawseeds

Kurzfassung

Autonome mobile Roboter benötigen eine Karte der Umgebung zum Navigieren. Das gleichzeitige Erstellen dieser Karte, ohne Wissen über die aktuelle Roboterposition wird im Englischen *Simultaneous Localisation and Mapping* (SLAM) genannt und ist essentiell für selbstständiges Navigieren, Pfad Planung und Kollisionsvermeidung. Dieser Prozess ist komplex, da eine gute Karte vonnöten ist um die Roboterposition zu bestimmen und umgekehrt führt nur eine genaue Position zu einer korrekten Karte. Mehrere bekannte Softwarepakete wie *DP-SLAM*, *GMapping* oder *GridSLAM* sind für Forschungszwecke frei verfügbar und ermöglichen einen noch nicht da gewesenen und aussagekräftigen Vergleich zwischen Sensoren und Algorithmen für SLAM. Das Ziel dieser Arbeit ist es, robuste Methoden zu finden um 2D oder 3D Karten mit gängigen Sensortechnologien zu erstellen. Wir vergleichen eine Zellen basierende Methode (*GMapping*) mit zwei auf geometrischen Strukturen basierenden Implementierungen. Alle Methoden beruhen auf einem statistischen Schätzen der Roboterposition mit einem *Particle Filter*. Die gängigen Messtechnologien: Lasermesssysteme, Sonar Sensoren und Tiefenbildkameras werden auf Genauigkeit und Robustheit untersucht. Lasermesssysteme führen zu sehr genauen Ergebnissen und werden häufig für SLAM verwendet. Da Sonar Sensoren günstiger sind, wird intensiv daran geforscht diese Technologie in Haushaltsrobotern einzusetzen. Die zuletzt erwähnte Sensortechnologie ist die neueste und generiert ein 3D Distanzbild der Umgebung. Alle Experimente finden im Innenbereich statt und eine quantitative Auswertung der Ergebnisse wird mit dem kürzlich veröffentlichten *RawSeeds* Datensatz durchgeführt.

Danksagung

Bei Herrn Univ.-Prof. Dipl.-Ing. Dr.techn. Horst Bischof bedanke ich mich für die Vergabe und Betreuung der Diplomarbeit. Besonderen Dank schulde ich Herrn Dipl.-Ing. Matthias Rüther für die Betreuung der Arbeit. Katrin Pirker danke ich für stete Diskussionsbereitschaft, geduldige Hilfestellung und vielseitigen Denkanstöße. Ich danke meinen Freunden Marian Kepesi, Gregor Perner, Rene Ranftl, Robert Beierheimer, Hannes Grünbacher, Marijan Stjepanovic, meiner lieben Schwester Claudia und meinem Bruder el Profeta Paolo für ihre spontane Bereitschaft dem Werk den letzten Schliff zu verpassen. Nicht zuletzt möchte ich meinen Eltern für den Rückhalt und die liebevolle Betreuung meines Hundes Aki in arbeitsintensiven Zeiten danken.

Diese Arbeit wurde im Rahmen des Research Studios Austria: "Machine Vision Meets Mobility - Positionsbestimmung für autonome Fahrzeuge" durchgeführt und von der österreichischen Forschungsförderungsgesellschaft (FFG) sowie dem Bundesministeriums für Wirtschaft, Familie und Jugend (BMWFJ) gefördert.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Contents

'Everything that is new or uncommon raises a pleasure in the imagination, because it fills the soul with an agreeable surprise, gratifies its curiosity, and gives it an idea of which it was not before possessed.', Joseph Addison.

(a) (b)

Figure 1.1: The DARPA Urban Challenge (2007), [12]: Two examples of autonomous vehicles driving in traffic.

Mobile robots have become more and more present in our daily life. These helpers clean our flat or mow the lawn while we are in the office. In some years, a machine may be cooking your favourite dishes or a robot nanny is taking care of your children while

(a) (b) (c)

Figure 1.2: Fraunhofer-Institute IPA, Museum for Communication, [21]: Three entertainment robots.

they are doing their homework. The content of this thesis, *Simultaneous Localisation And Mapping* (SLAM), is essential for all these tasks and many more: autonomous navigation, path planning and obstacle avoidance. In the following some popular research projects are described.
The DARPA Urban Challenge competitions have made it their goal to build autonomous vehicles capable of driving in traffic, see Figure 1.1. *Fraunhofer Institute for Manufacturing Engineering and Automation* (IPA) [21] are developing mobile autonomous guides, see Figure 1.2. The three robots are guiding visitors in a museum and provide additional audio-visual information. The success of these applications depend highly on the accuracy and robustness of their SLAM implementation.

A robot system can be structured the following way: First of all, the hardware needs to meet the requirements to solve a group of tasks, like exploring the environment or picking up various objects. A driver separates the algorithmic layer from the hardware layer, which means the same algorithm can be used for many different actuators or sensors as long as they implement the same functionality. The algorithm layer contains basic software modules to solve a group of tasks like *Simultaneous Localisation And Mapping* (SLAM), face recognition or object grasping. Finally, the networking layer, on the top of this structure, connects different software modules from

the algorithmic layer, as shown in Figure 1.3.

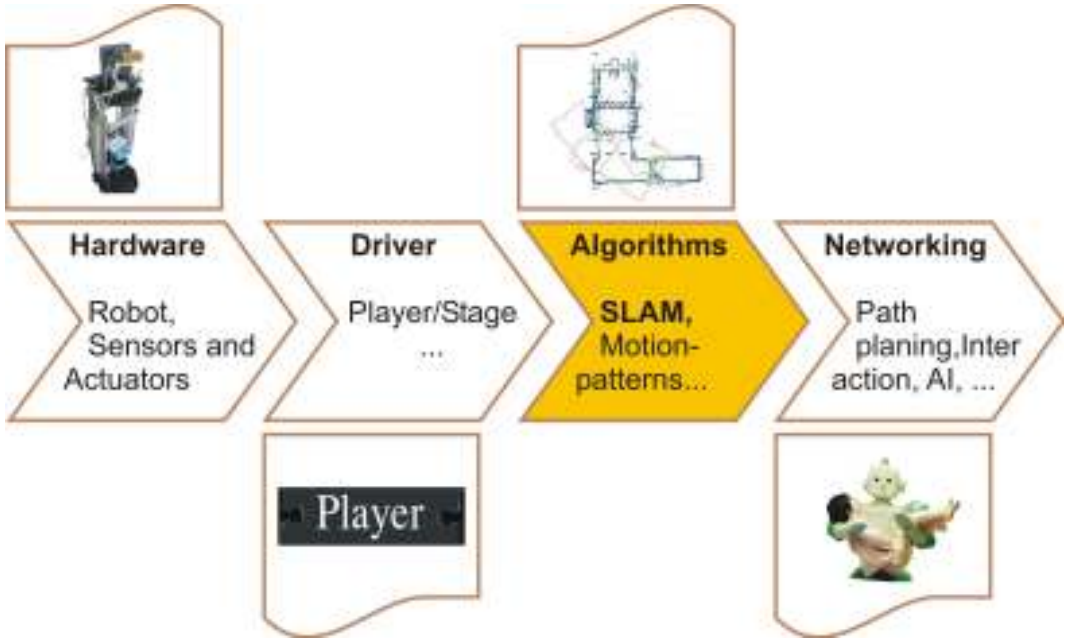


Figure 1.3: Hierarchical robot model: A robust SLAM algorithm is the base for many more complex tasks. The ambition of this work is to create indoor maps and discover sensor technologies and algorithms.

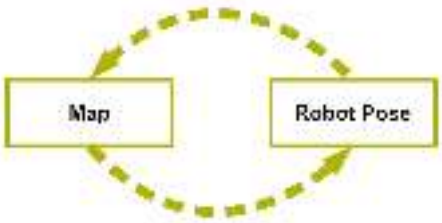


Figure 1.4: Interaction between the map and the robot for SLAM. A good map is necessary to compute the robot position, but only an accurate position estimate yields to a correct map.

Simultaneous Localisation And Mapping (SLAM) describes the process of building a map of an unknown environment and computing at the same time the robot position with the constructed map. Both steps depend on each other. A good map is necessary to compute the robot position, but only an accurate position estimate yields to a correct map, see Figure 1.4. This is often referred as the *chicken or the egg* causality dilemma.

Much research work on this topic has been contributed over the past decades, see Section 2. Hugh Durrant-Whyte et al. [16] states that a solution to the SLAM problem is the 'holy grail' for the mobile robotics community. A robust method would make a robot truly autonomous.

Despite considerable progress over the past decades, the SLAM problem is not solved and some issues are still remaining. Existing SLAM methods are limited to specialised robot platforms, small environments and certain sensor technologies. It is mandatory to find solutions, which work for a large variety of robots without altering the underlying concept. This is important to reuse an existing algorithm for a more general class of mobile robots. Another issue is to build accurate and large maps of dynamic environments. The methods should run in real time with the available memory, even for large maps. Finally, it is desirable to solve the SLAM problem with low-cost sensors, like sonars.
To find proper solutions to the above mentioned problems, different SLAM methodologies are discussed and evaluated in this work. The *GMapping* grid SLAM algorithm is compared with two feature based implementations. The first implementation uses corner features, while the second method is based on edge information. Both implementations are designed for laser range measurements. In contrast, grid SLAM algorithms have also been successful applied to noisy sonar readings. Hereby, maps generated with a laser range finder, sonars and time of flight cameras are compared in Section 6.

Another problem is the evaluation of SLAM methods. Due to the lack of ground truth data in the past, many authors evaluated their methods based on qualitative impressions of the resulting maps. In this context, often large maps with several loops were evaluated to prove the robustness and correctness. Although this methodology is suitable to compare different approaches, it is hard to compare the algorithms. Only a few weeks ago, indoor and outdoor SLAM datasets with ground truth measurements have been published by *The RawSeeds project* [6]. This is the first effort in this direction and will probably benchmark the results of the robotics community in future. This master thesis is one of the first with quantitative comparable SLAM results.

1.1 Motivation

It is an open question, which sensor technology suits well for state of the art SLAM methods. Furthermore, it would also be exciting to known which sensor technologies can be combined to enhance the map accuracy. Concerning these questions, this work has two main ambitions. First, different SLAM methods for building large and accurate maps are evaluated. This topic is settled in the algorithmic layer of Figure 1.3. The *GMapping* grid SLAM algorithm is compared with two feature based implementations. The first implementation uses corner features, while the second method extracts edges. Additionally to the comparison of SLAM methods, an evaluation of state of the art sensor technologies is given. SLAM is a complex task, because of large and dynamic environments, noisy sensors or occlusions. Therefore, most methods are designed for highly accurate laser range finders, which are relatively expensive. An ambition of this thesis is to evaluate different sensor technologies with respect to accuracy and robustness. New consumer household robots need to be cheap and are often equipped with noisy sensors. Hereby, a static ground truth dataset from the *RawSeeds* [6] project is used to evaluate methodical differences, while a small indoor dataset from our lab is applied to compare maps created with different sensor technologies. The evaluated time-of-flight camera [24] is a rather new sensor technology and as far as we know, nobody has tried to solve the SLAM problem with it. Finally, we also consider a sensor fusion approach to combine the different sensor technologies.

1.2 Overview

The thesis is structured as follows: Section 2 describes the recent work in the field of mobile robotics for the SLAM problem. The theoretical background can be found in Section 3. Properties of the evaluated sensor technologies are summarised in Section 4. The general SLAM problem and realisation considerations are described in Section 5. Section 6 considers the experiments done within this work. Finally, the work is concluded with a summary and an outlook in Section 7.

Chapter 2

Related work

Contents

Simultaneous localisation and mapping (SLAM) has been studied for more than twenty years. Hugh Durrant-Whyte et al. [16] states that the 'solution' of the SLAM problem has been one of the notable successes of the robotics community. The topic is still not completed and reasons therefore are discussed in Section 5.
To understand the bandwidth of the developed approaches, a chronological description of the progress is first provided in this chapter, followed by a description of state of the art SLAM algorithms. In this work a new sensor fusion approach is proposed and therefore existing sensor fusion techniques are discussed. Finally, this chapter discusses evaluation methods and existing ground truth datasets.

2.1 SLAM Methods

The statistical basis of the SLAM problem was first described by Smith et al. [47] and Durrant-Whyte [15]. These key papers provide probabilistic estimation techniques for correlations between features and robot poses. Furthermore, they have been the first, who proposed methods to refine geometrical uncertainties with continuous observations. A successive update process of the uncertainties, described by uncertainty ellipses is illustrated in Figure 2.1.

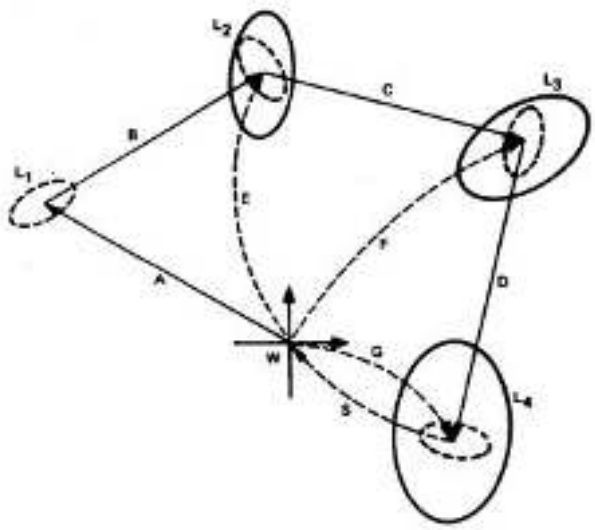


Figure 2.1: Landmark uncertainties from Smith et al. [47]: The successive update process of the uncertainties is described by uncertainty ellipses. A new observation results in decreasing uncertainties for all previously seen landmarks.

A key finding of their work is that there must be a high correlation between different map features and that indeed these correlations are growing with successive observations.
Crowley [10] and Leonard et al. [34] proposed SLAM methods using line segments extracted from ultrasonic data. Other researchers like Vandorpe et al. [50] and Gonzalez et al. [26] were using laser data instead. At the same time Ayache and Faugeras [3] developed the earliest work in visual navigation and mapping.
A consistent solution to the SLAM problem would require a joint state vector with the robot pose and all landmarks. This huge state vector needs to be updated after each observation, which leads to a high computational effort.
Leonard et al. [33] tried to reduce computational effort by splitting the state vector into local sub parts. This concept was later skipped, because in the year 1995, a break-through came with the realisation that the SLAM problem is convergent and the huge state vector is essential. The more the correlations between features grow, the better the solution is.
So far, the robot pose was described by a pose and a certainty like for the landmarks. Murphy [38] introduced *Particle Filters*, which are in general discretised representations of probability density functions. With *Particle Filters* the robot pose is represented by a set of discrete states. In contrast to the previously used feature based map representations, Murphy proposed an occupancy grid map, which is a quantisation of the world in blocks

or cells.

Later on, Montemerlo et al. [36] gave an extension to feature based maps (*FastSLAM*). The map is estimated with a *Kalman Filter*, while the robot pose is represented by a *Particle Filter*. Montemerlo demonstrated that just a few features are sufficient to navigate in a map with a size of several square kilometres, see Figure 2.2.

Figure 2.2: Outdoor FastSLAM evaluation from Montemerlo et al. [36] (2003): The yellow path is the estimated path of the mobile robot. The blue dashed line represents the GPS ground truth data. Features or landmarks extracted from visual data are illustrated by yellow circles.

2.2 Algorithms for SLAM

The main advantage of a *Particle Filter* is the ability to represent multi-modal probability functions. Many authors like Eliazar et al. [18] , Grisetti et al. [29] and Cyclic et al. [30] use a combination of *Particle Filters* and grid based map representations to address the SLAM problem with large maps under dynamic conditions, see Figure 2.3, Figure 2.4 and Figure 2.5.

GMapping (see Figure 2.3) was developed by Grisetti et al. [29]. It uses a *Particle Filter* in combination with an occupancy grid. Hähnel et al. [31] published an algorithm called *GridSlam* (see Figure 2.4) and Eliazar et al. [17] proposed a SLAM method called *DP-*

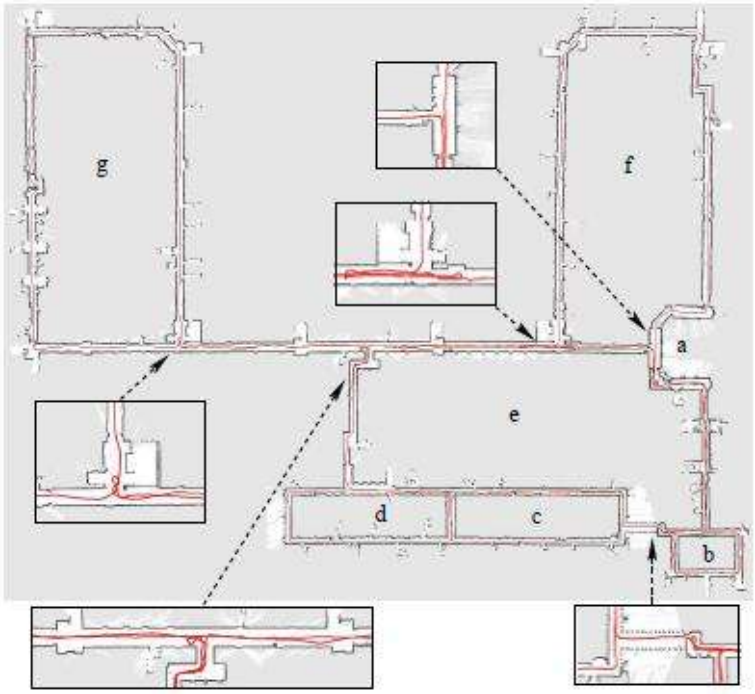


Figure 2.3: High quality indoor map, GMapping from Grisetti et al. [29] (2007): This indoor map was created with the GMapping algorithm and 80 particles. The robot travelled 1.9km without losing the correct vehicle pose.

SLAM (see Figure 2.5). All of the three algorithms use a *Particle Filter* for the robot pose estimation and a grid based map representation. The minor differences exist in the way they try to decrease the computational cost while maintaining a high level of robustness. *GMapping* and *GridSlam* reduce the number of particles, while *DP-SLAM* works with *sub maps* and an effective tree-based data structure.
These state of the art algorithms are freely available on the internet, see *The OpenSLAM Project* [11]. The next section gives a review about sensor fusion techniques and sensor comparison publications.

2.3 Sensor Fusion Methods

Two main types of sensor fusion techniques can be distinguished. A *centralised* sensor fusion approach works on a single computer. Whereas, a *decentralised* method runs on a distributed system with a team of robots. Only the *centralised* sensor fusion model is considered in this thesis. For further information concerning the *decentralised* model, we refer to the works of Rosencrantz et al. [44] and Makarenko et al. [35].

Figure 2.4: High quality indoor map, GridSlam from Hähnel et al. [31] (2003): This indoor map was created with the GridSlam algorithm and 500 particles.

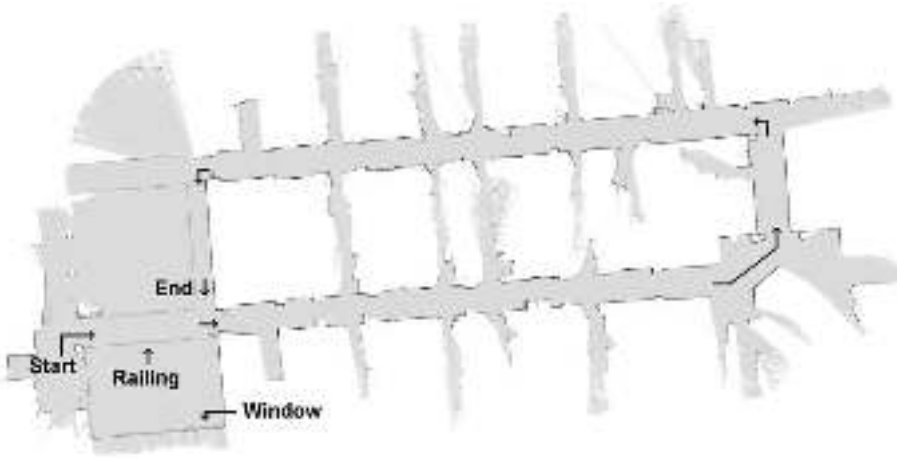


Figure 2.5: High quality indoor map, DP-SLAM 2.0 from Eliazar et al. [18] (2004): This indoor map was created with the DP-SLAM 2.0 algorithm.

Sensor data from the same type of sensor can be easily combined. For example, the laser readings from two laser range finders (LRFs) can be fused if the pose of each sensor is given. This technique is applied on the data of the two Sick LRFs from the indoor datasets obtained from the RawSeeds project [6]. One could name this approach *single modality sensor fusion*, comparable to registration methods from the medical image analysis field. The modality describes the image acquisition technique. (*Computer Tomography (CT)*, *Magnetic Resonance (MR)* ...).
In terms of this nomenclature, *multi modality sensor fusion* denotes the combination of observations from two different types of sensor technology. Yang et al. [53] describes an elegant way to cope with wrong laser measurements caused by mirrors or glass panes. His approach and the method from Diosi [13] combine a laser range finder with sonar readings.
Moravec H. P. [37] first published an occupancy grid based sensor fusion method, which combines sonar data with features extracted from a stereo vision system. The reliability of the grid map is increased by merging occupied cells of the stereo vision grid map and the sonar grid map.
Castellanos and Tardos [7] describe a fusion of laser data and monocular vision data. The corners, extracted with the vision system, reduces the covariances of the previously observed laser corners.

2.4 Evaluation Methods

This section describes evaluation strategies, especially for indoor datasets.
Due to the lack of ground truth data, many authors like Balaguer et al. [4], Grisetti et al. [29], Haehnel et al. [31] or Eliazar et al. [17] evaluated their methods based on qualitative impressions of the resulting maps. In this context, often large maps with several loops were evaluated to prove the robustness and correctness.
Balaguer et al. [4] published an evaluation of simulation environments, and also a comparison of the algorithms *GMapping*, *GridSlam* and *DP-SLAM*. He states:

> 'The three algorithms seem to be equally and reasonably robust to noise in the SICK laser.'

Sometimes a blue print of a building is available and can be used for the evaluation, see Nguyen et al. [40]. But even in this case, every publication is evaluated with a different dataset and a direct comparison is impossible. To overcome these issues, the *The RawSeeds project* was founded.

The RawSeeds project [6], funded under the *European Union's Sixth Framework Programme (FP6)*, is the first effort to provide ground truth (GT) data to evaluate SLAM algorithms, see Figure 2.6. The GT data is a fusion of stationary laser range finder distance measurements and observations of a multi camera system. The final GT data uncertainty is less than a few centimetres and enables a quantitative comparison of SLAM algorithms.

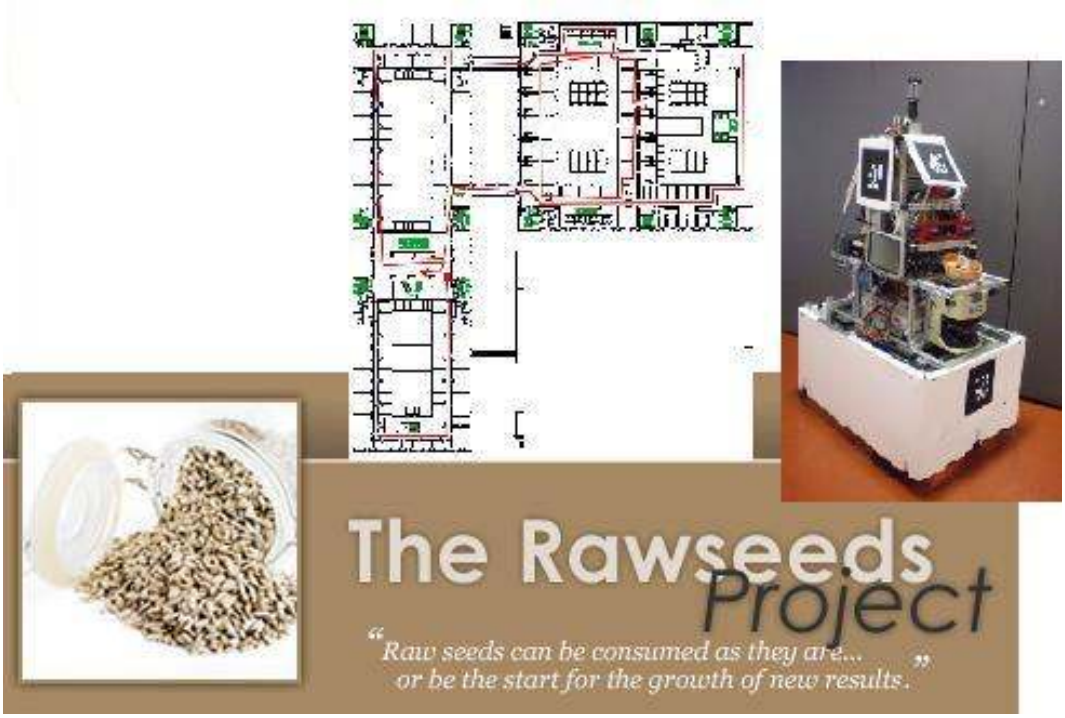


Figure 2.6: The RawSeeds project [6] is the first effort to provide ground truth data to evaluate SLAM algorithms. `http://www.rawseeds.org`

Chapter 3

Theory and Background

Contents

This chapter provides the theoretical basics, required to understand the content of this master thesis. Different SLAM approaches are founded on different basic filter concepts, which are explained in detail in this section.

Starting with a brief description of the notation, the mathematical basics of the *Kalman Filter*, the *Extended Kalman Filter*, the *Particle Filter* and the *Rao-Blackwellised Particle Filter* are reviewed.

There exist many more algorithms related to robotics and navigation, like the *Unscented Kalman Filter* and the *Information Filter* for example, see [49], but the provided selection is founded by the fact that our implementations rely on the *Rao-Blackwellised Particle Filter*, which can be seen as a combination of *Extended Kalman Filters* and a *Particle Filter*.

At the end of this section, two motion models are discussed in detail, namely the *Particle Motion Model* and the *Gaussian Probabilistic Motion Model*. Motion models, describing the kinematics of a mobile robot, play an important role for developing robust and fast SLAM algorithms.

3.1 Notation

The notation is similar to the book of Thrun et al. [49]. Vectors are denoted by lower case letters with subscripts (x_t or z_t for instance) and upper case letters are used for matrices (H or K for example). Constants are indicated by lower case Greek letters, like α_1 for a motion model noise control parameter. Finally, scalars are denoted by lower case letters without subscripts (the x coordinate of the robot pose for instance) .
A robot state at time step t is denoted by x_t and, in the two dimensional case, it is a vector consisting of a position x, y and a heading angle θ. For simplicity, it is assumed that only one observation is made at each time step, which is denoted by z_t.
In the description of filters, one often needs to distinguish between vectors before and after the filter process. Therefore the a priori vectors are denoted by a minus as superscript (x^- for instance), and the a posteriori vector is denoted by x^+ for this example.
At time step t, the filter estimate is denoted by $\tilde{x}_t$. In contrast, the true state is x_t. The non-linear extension of a state x_t, is denoted by $\hat{x}_t$. A superscript with brackets, $x_t^{[i]}$ denotes one sample of a list X_t.
The proposed notation is used consistently throughout this work. Only in some special cases a different notation is used, if a declaration is established in common literature (see the coefficient N_{eff} in Equation 3.32 for example).

3.2 Probabilistic Filters

What is filtering in principal? This is discussed next and the chapter concludes with a practical example.
All algorithms discussed in this chapter are based on the *Bayes Rule*. The simplest time-discrete example is denoted by Equation 3.1 for conditional probabilities.

$$P(A|B) = \frac{P(A) \cdot P(B|A)}{P(B)}, \tag{3.1}$$

where $P(A|B)$ is the posterior probability, $P(A)$ is the prior probability, $P(B|A)$ is the conditional probability of the random variable B and $P(B)$ acts as a normalisation constant. The simple example can be extended by an arbitrary conditional random variable z and this results in:

$$P(A|B,z) = \frac{P(A|z) \cdot P(B|A,z)}{P(B|z)}. \tag{3.2}$$

We are interested in computing the probability $P(x_t|z_{1:t}, u_{1:t})$, where x denotes the robot state. In the two dimensional case, the vector x consists for example of a cartesian coordinate and a heading angle. The set $z_{1:t}$ represents all observations and $u_{1:t}$ are control commands. Control commands for a simple velocity motion model consist of a translational velocity v_t and a rotational velocity ω_t. More complex odometry motion models are discussed in Section 3.6.

At the current time step, we are interested in estimating the state x_t with the last state x_{t-1}, the observation z_t and the control input u_t. Earlier states, observations or control inputs do not provide additional information and this leads to the simplification:

$$P(x_t|z_{1:t}, u_{1:t}) = P(x_t|x_{t-1}, z_t, u_t). \tag{3.3}$$

Two assumptions are made in computing x_t. First, the states follow a *first-order Markov process*, which means the current state depends only on the last state, $P(x_t|x_{1:t-1}) = P(x_t|x_{t-1})$. Secondly, the current observation z_t is independent of previous observations $z_{1:t-1}$. In the following, the mathematical derivation applying the *Bayes Rule* is shown:

$$\begin{aligned} P(x_t|z_{1:t}, u_{1:t}) &= \frac{P(z_t|x_t, z_{1:t-1}, u_{1:t}) \cdot P(x_t|z_{1:t-1}, u_{1:t})}{P(z_t|z_{1:t-1}, u_{1:t})} \\ &= \frac{P(z_t|x_t) \cdot P(x_t|z_{1:t-1}, u_{1:t})}{P(z_t|z_{1:t-1}, u_{1:t})} \\ &= \eta * P(z_t|x_t) \cdot P(x_t|z_{1:t-1}, u_{1:t}), \end{aligned}$$

where $\eta = 1/P(z_t|z_{1:t-1}, u_{1:t})$ is a normalisation term, which can often be neglected, if likelihoods are sufficient. The remaining two probabilistic terms are essential for filtering and inference.
The probability $P(z_t|x_t)$ is the prediction of an observation, known as *Measurement Model*. Probability $P(z_t|z_{1:t-1}, u_{1:t})$ represents the *Motion or Transition Model* and implies knowledge about the robot kinematics. Further information can be found in Chen's publication [8] or in Thrun's book [49].

As a practical example for a filtering process, imagine a person trying to take a turn at a dangerous crossing. His view is occluded by some parking cars and therefore, he can see the moving vehicles just for some seconds and has to guess when

the road is clear. This guessing or prediction step is one of many examples for a filter process.

Based on the basic knowledge about Bayes filtering processes in mobile robotics, the next section discusses the linear *Kalman Filter*.

3.3 Kalman Filter

In the year 1960, R. E. Kalman [32] proposed a novel recursive filter technique. He provided an efficient solution to estimate the state of a static or dynamic process. This filter can estimate the past, the present or the future state, even if parts of the model are unknown. Many filtering algorithms, especially for navigation tasks are based on the *Kalman Filter*.
Welch et al. [51] gives a detailed introduction of the *Discrete Kalman Filter* and to the *Extended Kalman Filter* and he concludes with a simple intuitive example. In this work, the same structure is followed. First the *Discrete Kalman Filter* is reviewed, followed by the *Extended Kalman Filter*. The interested reader find an intuitive example in [51].

3.3.1 Discrete Kalman Filter

A *Kalman Filter* estimates an n dimensional state x_t at time step t by applying the stochastic differential Equation:

$$x_t = A \cdot x_{t-1} + B \cdot u_{t-1} + w_{t-1}, \tag{3.4}$$

where the matrix A relates the state at the previous time step x_{t-1} to the current state x_t. B denotes the influence of the optional control input u to x_t and w_{t-1} indicates the Gaussian transition noise with zero-mean.
An observation z_t is expressed by:

$$z_t = H \cdot x_t + v_t, \tag{3.5}$$

where the system Jacobian H is calculated by the observation model and v_t is a random variable introducing zero mean Gaussian noise to the filter.
We define the a priori state $\tilde{x}^-$ to be the estimated state before the filter update and the a posteriori state $\tilde{x}^+$ to be the estimate state after the update step. The a priori filter

error is therefore $e_t^- = x_t - \tilde{x}_t^-$, where x_t denotes the true value and the corresponding covariance is given by $P_t^- = E[e_t^- \cdot e_t^{-T}]$ (E indicates the expected value).
We are now able to describe a *Kalman Filter*, which estimates the a posteriori state $\tilde{x}_t^+$ with the a priori state $\tilde{x}_t^-$ and a weighted difference between the current observation z_t and the predicted observation $H \cdot \tilde{x}_t^-$:

$$\tilde{x}_t^+ = \tilde{x}_t^- + K \cdot (z_t - H \cdot \tilde{x}_t^-), \tag{3.6}$$

where K is the *Kalman Gain*, which minimises the a posteriori error covariance P_t^+. The term $z_t - H \cdot \tilde{x}_t^-$ is called residual or measurement innovation and expresses the difference between the predicted and the actual measurement. There are various ways to determine the *Kalman Gain*. A common definition of K is:

$$K = \frac{P_t^- H^T}{H P_t^- H^T + R}. \tag{3.7}$$

If the observation error covariance R decreases, the actual observation z_t is treated to be more accurate than the prediction $H \cdot \tilde{x}_t^-$. On the other hand, a smaller a priori error covariance P_t^- leads to a less important actual measurement z_t and weights the prediction more.
In summary, a discrete recursive *Kalman Filter* consists of two alternating steps. The next state and the new error covariance matrix P_t^- is computed in the prediction step:

$$\tilde{x}_t^- = A \cdot \tilde{x}_{t-1}^+ + B \cdot u_{t-1} \tag{3.8}$$

$$P_t^- = A \cdot P_{t-1}^+ \cdot A^T + Q, \tag{3.9}$$

where Q is the transition noise variance of the random variable w.
In the correction or update step, the *Kalman Gain* K is computed as well as the a posteriori state $\tilde{x}_t^+$ and the a posteriori error covariance P_t^+:

$$K_t = \frac{P_t^- H^T}{H P_t^- H^T + R} \tag{3.10}$$

$$\tilde{x}_t^+ = \tilde{x}_t^- + K \cdot (z_t - H \cdot \tilde{x}_t^-) \tag{3.11}$$

$$P_t^+ = (I - K_t \cdot H) \cdot P_t^-. \tag{3.12}$$

Note that Equations (3.10) and (3.11) are the same as (3.7) and 3.8 and they are listed here for completeness.

The prediction and the correction step of the *Kalman Filter* are performed alternating. The filter is very popular because of the computational efficiency, for small state matrices A, and the simple implementation.
Regarding the computational complexity, the *Kalman Filter* is quite efficient. The most time consuming part is the matrix inversion in Equation 3.10, $(HP_t^- H^T + R)^{-1}$. State of the art algorithms come up with a complexity of approximately $O(d^{2.4})$, where the matrix is of the size d x d.
One problem of the *Kalman Filter* is the initial choice of the parameters. For further information the reader is refered to the book of Grewal et al. [28]. A mathematical derivation of the *Kalman Filter* based on the Equation 3.4 can be found in the book of Thrun et al. [49].
The *Extended Kalman Filter*, considered in the next section, is able to cover non-linear models, which is important because often the motion kinematics of mobile robots are non-linear problems.

3.3.2 Extended Kalman Filter

In contrast to the *Discrete Kalman Filter*, the extended variant is able to model a non-linear system, by extending the basic Equations (3.4) and (3.5). A Taylor series expansion is used, which is indicated by the functions f and h. The function f represents the system model and function h the measurement model:

$$\hat{x}_t = f(x_{t-1}, u_{t-1}) + w_{t-1} \tag{3.13}$$

$$\hat{z}_t = h(x_t) + v_t. \tag{3.14}$$

The full recursive estimation equations are then:

$$x_t \approx \hat{x}_t + A \cdot (x_{t-1} - \tilde{x}_{t-1}^+) + W \cdot w_{t-1} \tag{3.15}$$

$$z_t \approx \hat{z}_t + H \cdot (x_t - \hat{x}_t) + V \cdot v_t, \tag{3.16}$$

where $\approx$ denotes the approximation of the true state x_t and the true observation z_t. The Jacobian matrices A and W are partial derivatives of the transition function f (3.13), with respect to the state x respectively to the random variable w. H and V are partial

derivatives of the measurement function h (3.14), with regard to x and v respectively:

$$A[i,j] = \frac{\partial f[i]}{\partial x[j]}(\tilde{x}^+_{t-1}, u_{t-1}) \tag{3.17}$$

$$W[i,j] = \frac{\partial f[i]}{\partial w[j]}(\tilde{x}^+_{t-1}, u_{t-1}) \tag{3.18}$$

$$H[i,j] = \frac{\partial h[i]}{\partial x[j]}(\hat{x}_t) \tag{3.19}$$

$$V[i,j] = \frac{\partial h[i]}{\partial v[j]}(\hat{x}_t). \tag{3.20}$$

All Jacobian matrices A, W, H and V are time dependent. The subscript t was omitted to keep the formulas readable.
The following two equations are evaluated during the update or prediction step. In contrast to the *Discrete Kalman Filter* (3.8), the covariance matrix Q is multiplied with the Jacobian W:

$$\tilde{x}^-_t = f(x_{t-1}, u_{t-1}) + w_{t-1} \tag{3.21}$$

$$P^-_t = A \cdot P^+_{t-1} \cdot A^T + W \cdot Q_{t-1} \cdot W^T_t. \tag{3.22}$$

The correction process is described by the following Equations, which are very similar to (3.10), (3.11) and (3.12). The measurement noise covariance R is multiplied with the observation Jacobian V:

$$K_t = \frac{P^-_t H^T}{H P^-_t H^T + V \cdot R \cdot V^T} \tag{3.23}$$

$$\tilde{x}^+_t = \tilde{x}^-_t + K \cdot (z_t - h(\tilde{x}^-_t)) \tag{3.24}$$

$$P^+_t = (I - K_t \cdot H) \cdot P^-_t. \tag{3.25}$$

Again, matrices H and V are time dependent. A more detailed mathematical derivation of the *Extended Kalman Filter* based on (3.4) can be found in the book of Thrun et al. [49].
Two drawbacks remain with the *Extended Kalman Filter*. First, the filter can model only unimodal, Gaussian probability distributions of x and the filter diverges easily in the case of strong non-linearities in the state transitions [16].
In contrast, the *Particle Filter*, explained in the following section, is able to model multimodal distributions.

3.4 Particle Filter

A *Particle Filter* approximates an optimal Bayesian filter [27] by representing the robot position through an arbitrary, multimodal probability distribution, using a set of particles:

$$X_t = \{x_t^{[1]}, x_t^{[2]}, x_t^{[3]}, ..., x_t^{[M]}\}, \tag{3.26}$$

where M is the number of particles. A single state, or one particle, $x_t^{[i]}$ is associated with an importance factor or weight:

$$< x_t^{[i]}, w_t^{[i]} >= P(x_t^{[i]}|z_{1:t}, u_{1:t}), \tag{3.27}$$

The weight reflects the probability of the particle and is updated after each new observation:

$$P(x_t|z_{1:t}, u_{1:t}) = \eta * P(z_t|x_t) * P(x_t|z_{1:t-1}, u_{1:t}). \tag{3.28}$$

Based on (3.28), the a priori estimate $P(x_t \mid z_{1:t-1}, u_{1:t})$ is recursively computed by:

$$P(x_t \mid z_{1:t-1}, u_{1:t}) = \int P(x_t \mid x_{t-1}, z_{1:t-1}, u_{1:t}) \cdot P(x_{t-1} \mid z_{1:t-1}, u_{1:t}))dx_{t-1} \tag{3.29}$$

$$= \int P(x_t \mid x_{t-1}, u_t) \cdot P(x_{t-1} \mid z_{1:t-1}, u_{1:t}))dx_{t-1}, \tag{3.30}$$

which is valid, because previous observations and control inputs do not provide any additional information if x_{t-1} is known: $P(x_t \mid x_{t-1}, z_{1:t-1}, u_{1:t}) = P(x_t \mid x_{t-1}, u_t)$.
The idea behind a *Particle Filter* is that the probability given by (3.28) can be approximated by a set of particles (3.26), where a single state is approximated by:

$$x_t^{[m]} \approx P(x_t|z_{1:t}, u_{1:t}). \tag{3.31}$$

The *Particle Filter* constructs the actual belief of the state x in a recursive way, which implies that the current particle set X_t (3.26) is recursively calculated from X_{t-1}. In Table 3.1 the most basic variant of a *Particle Filter* is illustrated. The temporary variable $\bar{X}_t$ holds a list of particles and their correspondent likelihood, denoted by the weights w_t. The third line shows the generation of a hypothetical state $x_t^{[m]}$ based on the previous particle $x_{t-1}^{[m]}$ and the control input u_t. This is done by the transition or motion model, which is discussed in Chapter 3.6.
The probability of a sampled hypothesis is proportional to the measurement or observa-

	$X_t = \text{filter}(X_{t-1}, u_t, z_t)$
1:	$\bar{X}_t = X_t = 0$
2:	for m = 1 to M
3:	sample $x_t^{[m]} \sim P(x_t \mid u_t, x_{t-1}^{[m]})$
4:	$w_t^{[m]} = P(z_t \mid x_t^{[m]})$
5:	$\bar{X}_t = \bar{X}_t + < x_t^{[m]}, w_t^{[m]} >$
6:	endfor
7:	for m = 1 to M
8:	draw i with probability $\propto w_t^{[m]}$ from $\bar{X}_t$
9:	add $x_t^{[i]}$ to X_t
10:	endfor
11:	return X_t

Table 3.1: Particle Filter pseudo code

tion probability $P(z_t \mid x_t^{[m]})$. In literature, w_t is often referred to as the importance factor. For a navigation task in robotics, the observation probability is based on the difference between the current observation and the predicted observation according to the stored map of the sampled particle $x_t^{[m]}$.

Lines seven to ten in Table 3.1 describe the resampling step. This step creates a new particle set X_t out of the current one $\bar{X}_t$ to reduce the variance of the underlying distribution. Particles with a higher weight will appear more often in the new list than ones with lower likelihoods. In other words, good hypotheses of robot poses will remain in the non-parametric representation of the state, while others vanish.

Several resampling techniques are known (e. g. *Multinomial Resampling*, *Residual Resampling*, *Stratified Resampling* and *Systematic Resampling*). For detailed information and implementation issues, the reader is referred to [49] and a resampling strategy comparison was published by Douc et al. [14].

Particle Filters are powerful tools and used for many filtering, tracking and navigation tasks, but sometimes the performance is lower compared to *Kalman Filters*. Usually, *Kalman Filters* are preferred for small state matrices and small non-linearities, which is not the case for localisation and mapping tasks with mobile robots.

The computational effort is proportional to the number of particles and the most time consuming part is the resampling step (lines seven to ten in Table 3.1), because this step creates copies of the particles to re-use. For navigation tasks with mobile robots, it is not necessary to resample if the robot stops or if no observations are made. Other methods like

GMapping [29] or *DP-SLAM* [17] resample only, if the particle weight variance is above a certain threshold. Equation 3.32 describes a measure for the particle weight variance:

$$N_{eff} = 1/\sum_{m=1}^{M}[(w_t^{[m]})^2]. \tag{3.32}$$

The coefficient N_{eff} is maximal for equal weights and resampling would not reduce the variance of the represented probability distribution.
Resampling could also be dangerous and could lead to the *deprivation* or *depletion* problem. The *deprivation* problem describes the case if no particle exists in the vicinity of the correct state. This problem is triggered by the resampling step and it may happen that even good samples are replaced and the final particle distribution loses track of the correct state. Mostly, the problem occurs when M is too small.

3.5 Rao-Blackwellised Particle Filter

As stated at the beginning of this chapter, the *Rao-Blackwellised Particle Filter* can be seen as a combination of a *Particle Filter* and an *Extended Kalman Filter*.
The created map of the environment consists of features, which could be corners or edges for instance. A *Particle Filter* models the robot pose, and each map feature is represented by an *Extended Kalman Filter*. This leads to an efficient solution of the SLAM problem and is also referred as *FastSLAM* [49]. The *Kalman Filter* needs to update the full system Jacobian (3.5), while an update here affects only one feature. Therefore, the *Extended Kalman Filter* scales between linearly and quadratically with the number of dimensions of the estimation problem, while the *Particle Filter* scales exponentially. A more detailed discussion can be found in Section 7.

Each particle represents one hypothesis of a robot pose and contains its own set of map features describing the map, see (3.33). A map feature position is denoted by μ and the certainty is denoted by Σ.
Note that the covariance of the *Extended Kalman Filter* P_t^- (3.22) is denoted by $\Sigma_{n,t-1}^{[i]}$ for one particle (in *Particle Filter* notation). The mean is labelled with $\mu_{n,t}$ in contrast to $\tilde{x}_t^+$ in (3.24):

$$x_t^{[i]} =< [(x, y, \theta)^T]^{[i]}, \quad \mu_{1,t}^{[i]}, \Sigma_{1,t}^{[i]}, \quad \mu_{2,t}^{[i]}, \Sigma_{2,t}^{[i]}, \quad ..., \quad \mu_{N,t}^{[i]}, \Sigma_{N,t}^{[i]} >, \tag{3.33}$$

where the vector $(x, y, \theta)^T$ represents one robot pose hypothesis, i is the particle index and t denotes the point in time. The list of μ's and Σ's represents the map.
Table 3.2 illustrates the *Rao-Blackwellised Particle Filter*. Note the similarity between this algorithm and the *Particle Filter* example (Table 3.1).
The correspondence variables are denoted with c_t and are computed by a data association method. The data association process is responsible to correlate observations with a stored map. In the case of a feature map (see Section 5.2), a simple matching strategy could be a nearest neighbour approach with a defined distance measure. More details about data association strategies can be found in the book of Thrun et al. [49].
For simplicity it was assumed that only one observation z_t was measured at each point in time t. The third line shows the generation of a hypothetical state $x_t^{[m]}$ based on the previous particle $x_{t-1}^{[m]}$ and the control input u_t.
Lines 10 to 17 describe the *Extended Kalman Filter* map update procedure for each landmark or feature.
For new features, only the parameters $\mu_{j,t}^{[m]}$ and $\Sigma_{j,t}^{[m]}$, representing a map feature, are initialised. This is shown in lines 6 to 8. The function h^{-1} denotes a transformation from a local (robot) frame to a global (world) coordinate system and H is the Jacobian of h^{-1} with respect to the local observation z_t.
The 'else' path describes the necessary calculations for re-observed features. Vector $\tilde{z}$ denotes the measurement prediction and is used to update the mean $\mu_{j,t}^{[m]}$ in line 14 and to calculate the importance factor $w^{[m]}$ (line 16). The transformation from a world frame to a local coordinate is performed by the function h. Matrix H in Line 11 of Table 3.2 is the corresponding Jacobian, with respect to the global map feature $\mu_{j,t}^{[m]}$.
The resampling procedure is illustrated by lines 20 to 24 and we refer to the description in the *Particle Filter* chapter 3.4.
The *Rao-Blackwellised Particle Filter* is one of the easiest SLAM algorithms to implement and combines the advantages of a *Particle filter* and *Extended Kalman Filters*. The interested reader will find a more detailed description and a mathematical derivation in the book of Thrun et al. [49]. There, an optimised variant of this algorithm, referred as *FastSLAM 2.0*, is also discussed. This optimisation includes a different distribution $x_t^{[m]} \sim P(x_t \mid u_t, x_{t-1}^{[m]}, z_{1:t})$, which takes the current measurement into account, see line 3 in Table 3.2.

	$X_t = \text{filter}(X_{t-1}, u_t, z_t, c_t)$
1:	$\bar{X}_t = X_t = 0$
2:	for m = 1 to M
3:	sample $x_t^{[m]} \sim P(x_t \mid u_t, x_{t-1}^{[m]})$
4:	$j = c_t$
5:	if feature j never seen before
6:	$\mu_{j,t}^{[m]} = h^{-1}(z_t, x_t^{[m]})$
7:	$H = h^{-1'}(x_t^{[m]}, \mu_{j,t}^{[m]})$
8:	$\Sigma_{j,t}^{[m]} = HQ_tH^T$
9:	else
10:	$\tilde{z} = h(\mu_{j,t-1}^{[m]}, x_t^{[m]})$
11:	$H = h'(x_t^{[m]}, \mu_{j,t}^{[m]})$
12:	$Q = H\Sigma_{j,t-1}^{[m]}H^T + Q_t$
13:	$K = \Sigma_{j,t-1}^{[m]}H^TQ^{-1}$
14:	$\mu_{j,t}^{[m]} = \mu_{j,t-1}^{[m]} + K(z_t - \tilde{z})$
15:	$\Sigma_{j,t}^{[m]} = (I - KH)\Sigma_{j,t-1}^{[m]}$
16:	$w^{[m]} = \frac{1}{\sqrt{\lvert 2\pi Q \rvert}} exp\{-\frac{1}{2}(z_t - \tilde{z})^TQ^{-1}(z_t - \tilde{z})\}$
17:	endif
18:	$\bar{X}_t = \bar{X}_t + < x_t^{[m]}, w_t^{[m]} >$
19:	endfor
20:	for m = 1 to M
21:	draw i with probability $\propto w_t^{[m]}$ from $\bar{X}_t$
22:	add $x_t^{[i]}$ to X_t
23:	endfor
24:	return X_t

Table 3.2: Rao-Blackwellised Particle Filter pseudo code.

3.6 Motion Models

Motion models describe the kinematics of a mobile robot and play an important role for developing robust and fast SLAM algorithms.

For a *Rao-Blackwellised Particle Filter* for instance, only a wise population of new particles ensure a good estimation of the real pose. Improper motion models need many more particles to cover all possible robot states and finally, the computational complexity grows with the number of particles.

In the two dimensional case, three parameters are used to describe the state of a mobile robot (x, y and the heading or bearing direction θ). For robots operating in the three dimensional space, six parameters (x, y, z, roll, pitch and yaw angle) are necessary. This section considers only the 2D case and all models describe kinematics of two wheeled robots with differential drives. Thrun et al. [49] distinguishes between velocity and odometry motion models. The two models analysed in this chapter are odometry based and the interested reader will find a good introduction to velocity models in [49].

Introduction

A probabilistic motion model describes the transition between the previous (x_{t-1}) and the current robot pose (x_t) and can be expressed by the conditional density:

$$p(x_t|u_t, x_{t-1}), \tag{3.34}$$

where parameter u_t denotes the motion commands. The used nomenclature for the motion models is summarised in Table 3.3:

Parameter	Description
$\bar{x}_{t-1} = (\bar{x}, \bar{y}, \bar{\theta})$	odometry pose of the previous state
$\bar{x}_t = (\bar{x}', \bar{y}', \bar{\theta}')$	odometry pose of the current state
$x_{t-1} = (x, y, \theta)$	particle estimate of the previous state
$x_t = (x', y', \theta')$	modified or moved pose estimate
$\bar{x}_t - \bar{x}_{t-1} = (\delta\bar{x}, \delta\bar{y}, \delta\bar{\theta})$	relative odometry movement

Table 3.3: Nomenclature of motion model parameters

The models should be able to deal with the following odometry error sources:

- different tire pressures (in general: different wheel diameters)
- ground unevenness
- inaccuracy of the wheel attachment
- drift.

Authors like Fulgenzi et al. [22] often do not mention which motion model they use. In many cases the motion model is neglected and each robot pose is simply shifted by the distance travelled. Afterwards, the bearing angle is corrected and finally Gaussian noise is added to model odometry errors. This simplified model linearises the true robot motion. The rotation is performed before the translation and finally the heading angle is updated. The model is only valid if rotation and translation are performed independently, which is in reality hardly ever the case. The linearisation is still useful if the odometry updates arrive within small time intervals.
In the next two sections, the *Particle Motion Model* from Thrun et al. [49] and the *Gaussian Probabilistic Motion Model* from Eliazar et al. [19] are explained. A discussion of these two models is given in Section 7.2.

Particle Motion Model

The motion model described by Thrun et al. [49] consists of three parameters (δ_{rot_1}, δ_{rot_2}, δ_{trans}), see Figure 3.1. Two parameters denote a pre- and a post-rotation, whereas δ_{trans} describes the translational part. After the last pose x_{t-1} is moved to the current position x_t, the bearing angle is updated. The following equations describe the motion model update process:

$$x^{'} = x + \delta_{trans} \cdot cos(\theta + \delta_{rot_1}) \tag{3.35}$$

$$y^{'} = y + \delta_{trans} \cdot sin(\theta + \delta_{rot_1}) \tag{3.36}$$

$$\theta^{'} = (\theta + \delta_{rot_1} + \delta_{rot_2})\ mod\ 2\pi. \tag{3.37}$$

Zero mean Gaussian noise is added to the parameters δ_{rot_1}, δ_{rot_2} and δ_{trans}, to model odometry errors:

$$\sigma_{rot_1} = \alpha_1 \cdot |\delta_{rot_1}| + \alpha_2 \cdot \delta_{trans} \tag{3.38}$$

$$\sigma_{rot_2} = \alpha_1 \cdot |\delta_{rot_2}| + \alpha_2 \cdot \delta_{trans} \tag{3.39}$$

$$\sigma_{trans} = \alpha_4 \cdot (|\delta_{rot_1}| + |\delta_{rot_2}|) + \alpha_3 \cdot \delta_{trans}. \tag{3.40}$$

Gaussian Probabilistic Motion Model

Eliazar et al. [19] introduce a new parameter C, modelling an orthogonal robot shift. In Figure 3.2 the sequence of operations for the transition is shown. The distance travelled is labelled with D, and T denotes the relative angle $\delta\bar{\theta}$ between the previous and the

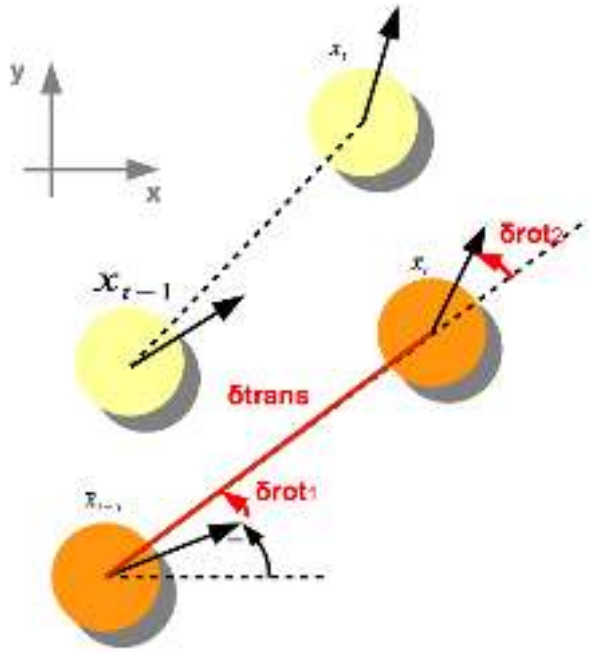


Figure 3.1: Particle Motion Model: The model consists of three parameters (δ_{rot_1}, δ_{rot_2}, δ_{trans}). Two parameters are for a pre- and post rotation and one parameter is the translational part. First the last pose x_{t-1} is moved to the current position x_t then the bearing angle is updated.

current robot pose. First, the pose x_{t-1} is rotated by $T/2$ and then translated with D. The parameter C approximates a drift along the orthogonal direction. This drift cannot be modelled with the *Particle Motion Model*. The mean of C is usually zero because on odometry sensor is unable to measure the drift. The following Equations denote the odometry update rules:

$$x^{'} = x + D \cdot cos(\theta + \frac{T}{2}) + C \cdot cos(\theta + \frac{T+\pi}{2}) \tag{3.41}$$

$$y^{'} = y + D \cdot sin(\theta + \frac{T}{2}) + C \cdot sin(\theta + \frac{T+\pi}{2}) \tag{3.42}$$

$$\theta^{'} = (\theta + T) \ mod \ 2\pi. \tag{3.43}$$

Like for the *Particle Motion Model*, the added noise follows a zero mean Gaussian distribution:

$$\sigma_D = \alpha_1 \cdot D + \alpha_4 \cdot |T| \tag{3.44}$$

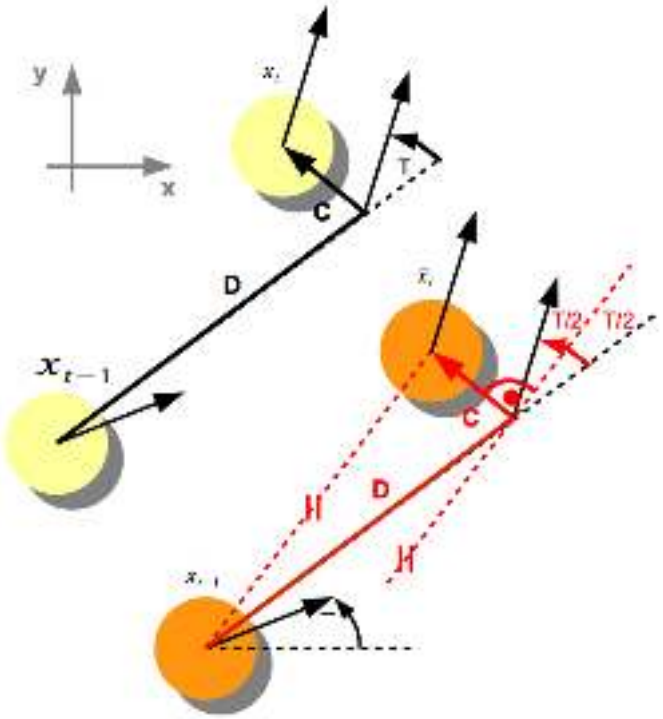


Figure 3.2: Gaussian Probabilistic Motion Model: The distance travelled is labelled with D and T denotes the relative angle $\delta\bar{\theta}$ between the previous and the current robot pose. First the pose x_{t-1} is rotated by $T/2$ and then translated with D. Now the new parameter C approximates a drift along the orthogonal direction.

$$\sigma_C = trans_{varC} \cdot D + rot_{varC} \cdot |T| \tag{3.45}$$

$$\sigma_T = \alpha_3 \cdot |T| + \alpha_2 \cdot D. \tag{3.46}$$

The motion models discussed in this section describe different motion transition sequences. In Section 7.2, the differences are discussed.

Chapter 4

Sensor Technologies

Contents

Simultaneous Localisation and Mapping (SLAM) algorithms estimate the actual robot pose based on observations of the environment. These observations are gathered by sensors and could be distance measurements, camera images, radio frequency identification (RFID) signals, etc.

The robustness and accuracy of SLAM methods is highly related to the sensor technology used. This is the motivation for this chapter, which describes three sensor technologies with respect to their measuring characteristics and their performance. We have chosen sonars, laser range finders and time of flight cameras because they are all distance sensors, which allows a comparison, and their popularity in mobile robotics. The description is a brief introduction and may not be seen as a detailed physical description of the hardware. More details about sensors in mobile robotics can be found in the book of Everett [20] and an evaluation of the sensor accuracy can be found in Section 6.1. Examples of sensor observations, made in a typical office like environment, can be found in the Appendix Section D.

4.1 Introduction

The measuring principle of the described sensors is the same and is based on the *time of flight* method. First, a signal is emitted from the sensor. This signal is reflected by an object and received again. The sensor device measures the time between the signal emission and the signal reception. This principle can also be found in nature. Dolphins use ultrasonic waves to locate objects under water, while bats are able to navigate at night. The mathematical description of the *time of flight* principle is denoted by:

$$d = c \cdot t, \tag{4.1}$$

where d denotes the distance travelled (usually round-trip), c expresses the speed of wave propagation and the time of flight is labelled with t.

In the following, sonar sensors, laser range finders and time of flight cameras are described with respect to their measuring characteristics and their problems.

4.2 Sonar

Sonar (Sound Navigation and Ranging) sensors were chosen because of their popularity and the low costs. The measuring accuracy and quality is low compared to a laser range finder. Common ultrasonic transducers have an operating range between 15 cm and 5 m and have a variance of up to 20 cm per meter. According to specification sheets 10 observations per second are typical for common sonars. In recent years, many researchers like Pandey et al. [42] or Schröter et al. [45] addressed the SLAM problem with sonar sensors and large maps. This is challenging because of the wide cone shaped wave propagation beam of a sonar.

Measuring Characteristics

An ultrasonic sensor transmits a sound wave and measures the time until the reflected wave is received, see Figure 4.1. The detection threshold of the receiving device is increased during the measurement, because of the attenuation of the signal strength with increasing distance. When the reflected signal strength exceeds the detection threshold, the receiver is triggered and the distance is evaluated.
In a sonar belt (Figure 4.2) only one transducer is active at the same time (multiplexing).

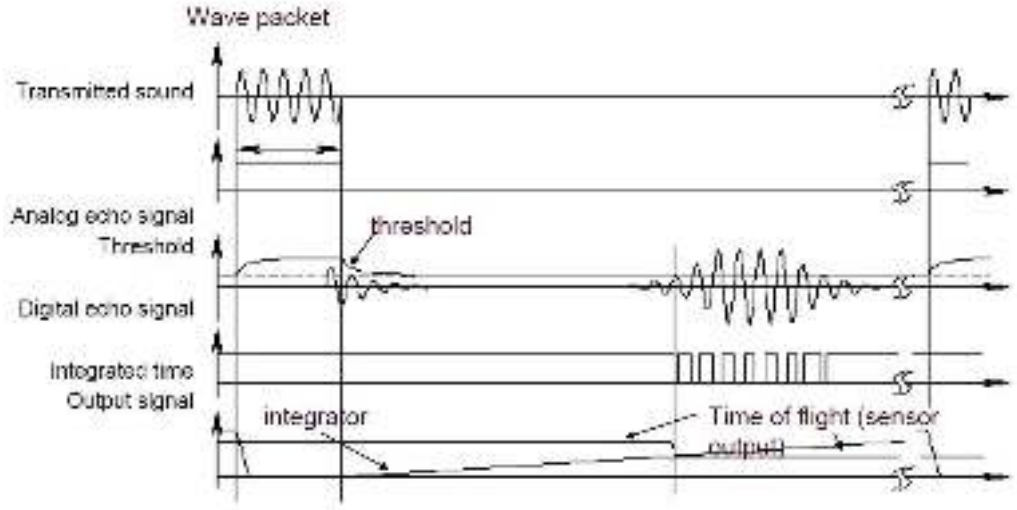


Figure 4.1: Ultrasonic transducer measuring principle, Stiegwart ETH Zürich [46]: The first row shows an emitted burst and the last row indicates the increasing sensitivity of the receiving device. When the reflected signal strength exceeds a threshold, the receiver is triggered and the distance is evaluated.

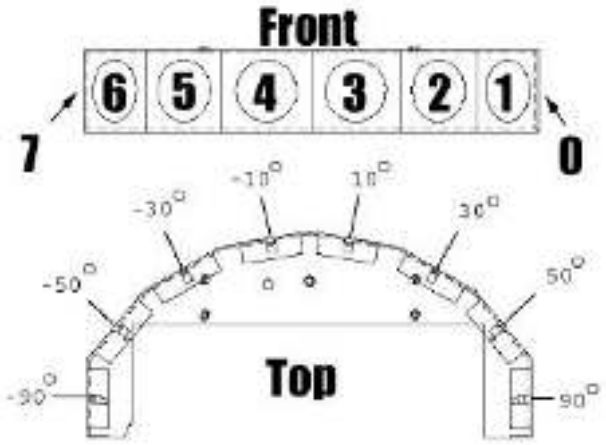


Figure 4.2: Sonar belt from ActivMedia's PeopleBot [1]: The transducers of the sonar belt are individually activated by a multiplexer.

Without multiplexing, all sensors of the belt would measure a response simultaneously. This is one of the reasons for the low response time compared to a laser range finder. The response time or measurement interval is important, because frequent map updates increase the SLAM algorithm robustness.

Problems

Incoming sonar beams could be misinterpreted because of reflections caused by low impact beam angles, noise and maximum range measurements, see Figure 4.3. Any obstacle in the field of view (FOV) of an ultrasonic transducer triggers the distance measuring system. The real distance is often longer than measured. On the contrary, maximum range measurements occur when the obstacle or the wall is out of range.

The next section describes the more accurate laser range finder technology. Most problems of sonar sensors are caused by the wide cone shaped sound propagation. In contrast, the light beam of a laser range finder is a sharp ray.

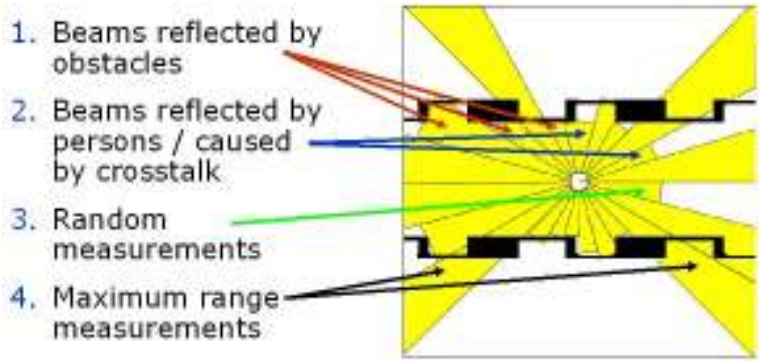


Figure 4.3: Typical sonar measurement errors, Gaurav S. Sukhatme [48]: Sonar sensors have a limited range (approximately 5m) and are very noisy. Most errors arise because of the cone-shaped wave propagation.

4.3 Laser Range Finder

Laser range finders are the most popular sensors for navigation and map building tasks in the robotics community. They can have a range up to 200m, a resolution as low as 0.1° and an accuracy of more than 1 cm per meter. Examples are the *Sick LMS 200*[*], *Ibeo ALASCA XT*[†] or the cheap *HOKUYO URG-04LX*[‡], see Figure 4.3. The high accuracy and the big FOV are the main advantages compared to other sensor technologies. Drawbacks are the energy consumption and the high costs compared to sonar sensors. According to the specification sheets, the measurement interval lays between 5 and 10 observations per

[*]http://www.sick.com
[†]http://www.ibeo-as.com
[‡]http://www.hokuyo-aut.jp

second.

(a) (b) (c)

Figure 4.4: Laser Range Finder samples: The *Sick LMS 200* laser scanner (a) is very accurate and often used for industrial tasks. This applies also for the *Ibeo ALASCA XT* sensor (b), which will be used for obstacle detection in cars. The *HOKUYO URG-04LX* (c) is the cheapest one and only used for indoor tasks.

Measuring Characteristics

The sensor has a rotating mirror which reflects the emitting laser beam. The incoming beam is again reflected by the mirror to a lens and then measured by a photo receiver device, see Figure 4.5. The rotating mirror and the housing is often the restricting factor for the field of view. *Sensor Intelligence*[§] has recently introduced a 3D sensor developed for the automotive industry, see Figure 4.6. This sensor uses the same principle like the one in Figure 4.5, but consists of four parallel laser diodes and achieves a range of 200m. The horizontal field of view is 240° and the vertical 3.2°.

Problems

Since the laser range finder works with light, the main problems are mirrors and windows, see Figure 4.7. The measurement fails, if the beam is not reflected. Therefore, Yang et al. [53] describes a method to combine sonar measurements with laser scans to overcome these issues.

The laser range finder is a pure 2D sensor. Recent SLAM publications, see Section 2, discuss the generation of 3D maps with rotated laser range finders. First, the sensor is rotated by a motor and thereafter, the distances are measured. Since the rotation and the

[§]http://www.sick.com

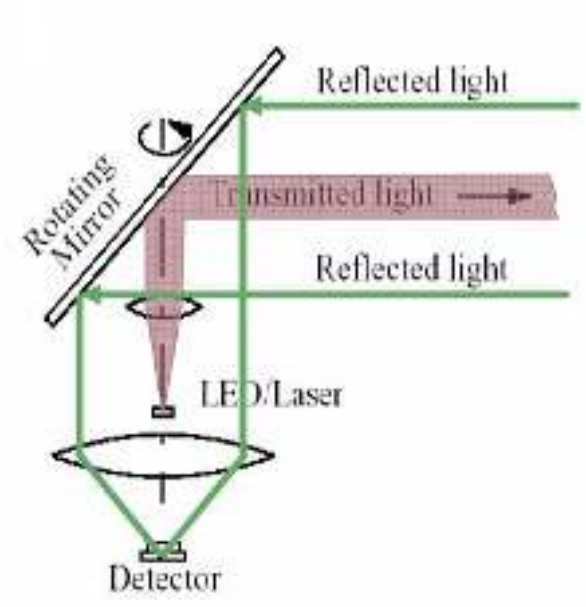


Figure 4.5: Laser range finder measuring principle, Siegwart ETH Zürich [46]: This principle works with a rotating mirror which reflects the emitting laser beam. The incoming beam is again projected by the mirror to a lens and then measured by a photo receiver device.

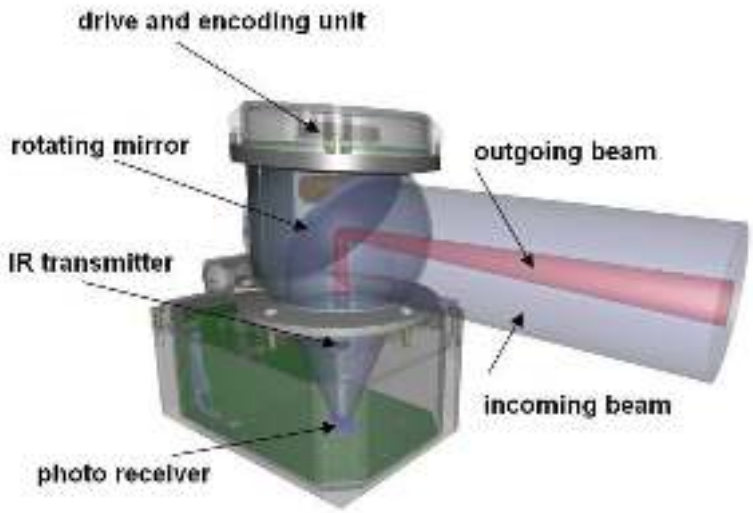


Figure 4.6: 3D Laser range finder measuring principle, IBEO-Alasca XT: This sensor uses the same principle like the one in Figure 4.5, but consists of four parallel working laser diodes and achieves a range of 200m. The horizontal field of view is 240° and the vertical 3.2°.

scan takes some time, the mobile robot may not drive fast or even needs to stop for a new measurement. Furthermore, the rotation of the device is error prone.
In the following, a time of flight camera is introduced, which directly generates a 3D distance image without any mechanical rotating parts.

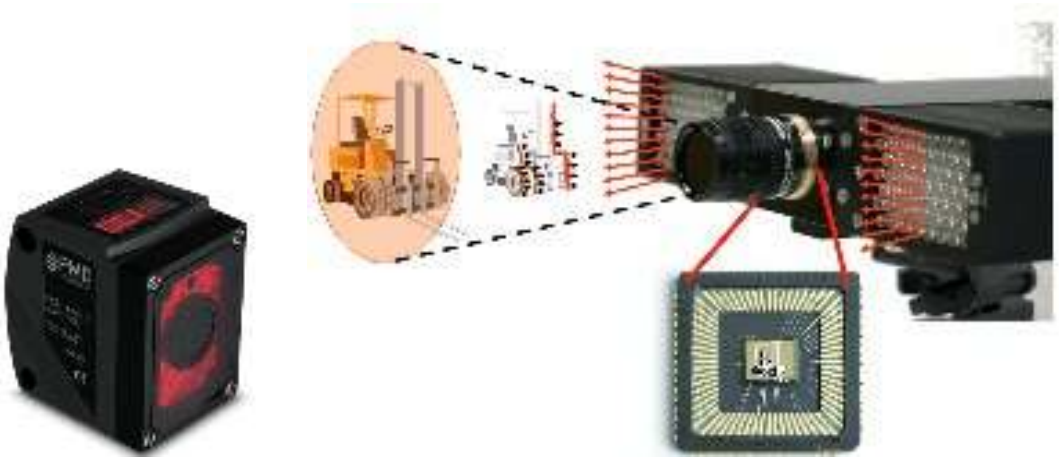
Figure 4.7: Laser range finder problems with window panes, Yang et al. [53]: Laser range finders have problems with mirrors and windows.

4.4 Time of Flight Camera

A time of flight (TOF) camera is a promising 3D sensor, which creates a contrast and a depth image simultaneously, see Figure 4.9. This sensor can be used for many applications, like object detection in a storehouse, where a depth sensor is mounted on a fork truck, see Figure 4.4 (b). In this work, a TOF camera is used to address the SLAM problem.

TOF cameras are very fast (up to 60 fps) compared to the previously discussed technologies, which have measurement intervals of about 10 fps. Burak et al. [25] states that their camera has a depth resolution of a few millimetres, dependent on the distance and the amount of reflected light of the target object. An accuracy evaluation of a TOF camera from PMDTec can be found in Section 6.1.
Despite these advantages, the image resolution is low and the calibration step tends to be challenging. Another drawback to address the SLAM problem with current TOF cameras is the small field of view (FOV). For the camera from PMDTec, see Figure 4.4 (a), the FOV is in the order of 40° horizontally and 30° vertically. Multiple cameras could cover a bigger FOV, but this is still an open issue because of light interferences. The maximum measurement range for this camera is 7.5 metres, which restricts the usage for indoor SLAM problems. Furthermore, sun light causes wrong measurements.

This can be reviewed in the experimental Section 6.1. Because of these issues, the TOF camera is often used as a helping sensor for obstacle avoidance and not for SLAM.



(a) (b)

Figure 4.8: PMDTechnologies GmbH [24] time of flight sensor: (a) represents the current PMD[Vision]® O3 Sensor and (b) illustrates one example application, where a depth sensor is mounted on a fork truck.

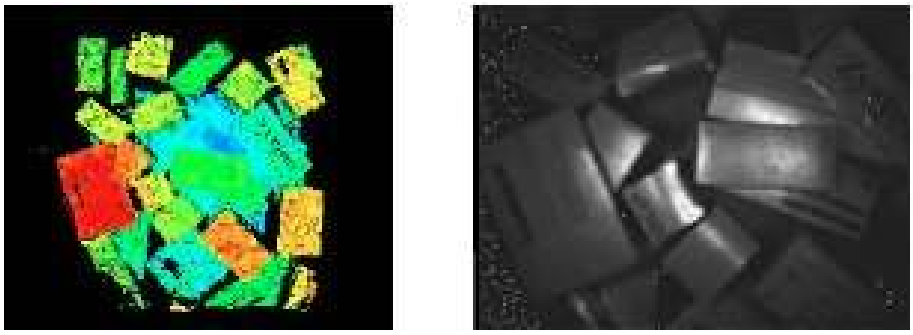


Figure 4.9: Time of flight camera depth images from PMDTec [24]: Depth image on the left and intensity image on the right.

Measuring Characteristics

A TOF camera consists of a modulated light source (LED or laser diode), a CMOS or CCD pixel array and an optical system to focus the incoming light onto the sensor. The phase shift between the emitted light and the received light is calculated for each pixel simultaneously. This is why the sensor is relatively fast compared to other technologies

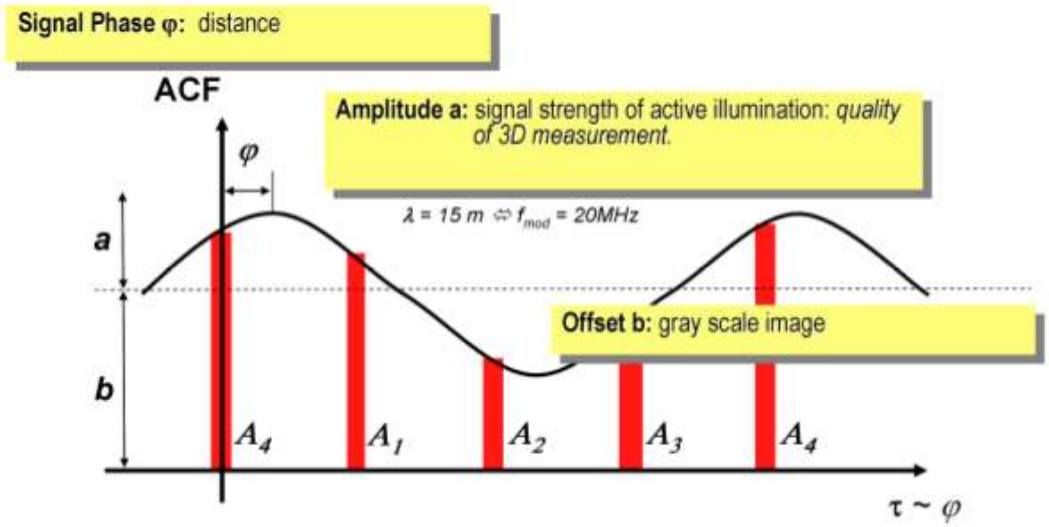


Figure 4.10: Time of flight camera measuring principle [24]: A_1 to A_4 denotes four amplitude values sampled with constant sample time. These four values are used to calculate the phase shift between the emitted and the received light. a denotes the received signal amplitude and b represents the pixel gray value.

like a laser range finder. Equation 4.2 denotes the calculation step for the distance to the target object. The phase shift ϕ can be calculated from four amplitude values:

$$d = \frac{c \cdot \phi}{4 \cdot \pi \cdot f_m}, \tag{4.2}$$

where c is the speed of light, ϕ is the measured phase shift and f_m is the modulation frequency of the emitted light:

$$\phi = arctan(\frac{A_1 - A_3}{A_2 - A_4}), \tag{4.3}$$

where A_1 to A_4 denotes four amplitude values sampled at constant time intervals, see Figure 4.10.

The maximum sensor range is given by the modulation frequency f_m and the speed of light c:

$$d_{max} = \frac{c}{2 \cdot f_m}. \tag{4.4}$$

For a more detailed description we refer to PMDTec [24] and Canesta et al. [25].

Problems

The problem with recent TOF cameras is the limited FOV. This issue is discussed in the evaluation part of this work in Section 6. The 3D sensor used (PMD[Vision]® O3 from PMDTec [24]) has a distance variance of approximately 10 cm per meter, dependent on the target distance and the amount of reflected light. The reflection coefficient of the target object plays an important role regarding the measurement accuracy. Similar to laser range finders, window panes and mirrors are crucial factors. Finally, the camera calibration tends to be difficult because of the low sensor resolution (64x48 pixel for a PMD[Vision]® O3).

Finally, a comparison highlights the main differences according to information obtained from specification sheets.

Comparison

The robustness and the accuracy of SLAM algorithms depend on many sensor features. Most important are the field of view (FOV), the observation interval (speed), the measurement accuracy and the sensor range, see Table 4.1 for specification sheet information. Sonars and laser range finder are pure 2D sensors. In contrast, a TOF camera generates 3D depth images of the scene.

Sensor name	FOV	Speed	Accuracy per m	Range	Price
PeopleBot Sonar belt	360°	100 ms	20 cm	5 m	300 euros
Sick LMS 200	180°	200 ms	1 cm	30 m	4500 euros
HOKUYO URG-04LX	240°	100 ms	5 cm	4 m	900 euros
PMD[Vision]® O3	40°	130 ms	10 cm	7m	1800 euros

Table 4.1: Sensor comparison of distance. Accuracy figures are given according to specification sheets

The sensor technologies discussed in this chapter are evaluated in the experimental Section 6.

Chapter 5

Simultaneous Localisation and Mapping

Contents

Simultaneous localisation and mapping (SLAM) is an essential task in robotics. The knowledge of the environment and the robot pose is mandatory for a mobile vehicle to navigate autonomously.

A brief introduction and a description about the general nature of the SLAM problem is given first to understand the difficulties. The strengths and weaknesses of state of the art solutions are discussed and a categorisation based on pose estimation techniques and map representations is introduced.

Furthermore, the SLAM methods used in this work are described, including the mathematical background to implement the algorithms. A simple sensor fusion approach is proposed to increase robustness. It is possible to merge different types of SLAM algorithms, like a feature based approach and an occupancy grid SLAM method, or to fuse different sensor technologies. For instance, sonar readings can balance the sensitivity of laser range finders to window panes or mirrors. Finally, a conclusion discusses the proposed approaches.

5.1 Introduction

Map building and localisation is essential for all these tasks and many more: autonomous navigation, path planning and obstacle avoidance.
Simultaneous Localisation and Mapping (SLAM) describes the process of building a map of an unknown environment and computing at the same time the robot position with the constructed map, see Chapter 1. Both steps depend on each other. A good map is necessary to compute the robot position and on the other hand just an accurate position estimate yields to a correct map.

Much research work on this topic has been undertaken over the past decades, see Section 2. Hugh Durrant-Whyte et al. [16] states that a solution to the SLAM problem is the 'holy grail' for the mobile robotics community. A robust method would make a robot truly autonomous. Today, several SLAM algorithms are available on the internet for science research. State of the art methods are discussed in Section 3.
Despite considerable progress over the past decades, the SLAM problem is not solved and some issues are still remaining. Existing SLAM methods are limited to specialised robot platforms, small environments and certain sensor technologies. It is mandatory to find robust SLAM solutions, which work for a large variety of robots without altering the model. This is important to reuse an existing algorithm for a more general class of mobile robots. Another issue is to build accurate, large maps of dynamic environments. The methods should run in real time and need to get by with the available memory, even for large maps. Finally, it is desirable to solve the SLAM problem with low-cost sensors, like sonars.

To find proper solutions to the above mentioned problems, different SLAM methodologies are discussed here and an evaluation is given in Section 6. A SLAM algorithm consists of two alternating steps. The prediction, or motion model, determines a new robot pose from the previous pose, possibly taking motion sensor readings as input. Inertial measurement units (IMU) or global positioning systems (GPS) are commonly used. The update, or observation model, corrects the predicted robot pose based on observations of the environment. Sensors like laser range finders, stereo cameras, or time of flight cameras are used to create a map, which could be an occupancy grid map, a feature map or a topological map. All SLAM methodologies discussed in this work rely on the same basic concept, illustrated in Figure 5.1.

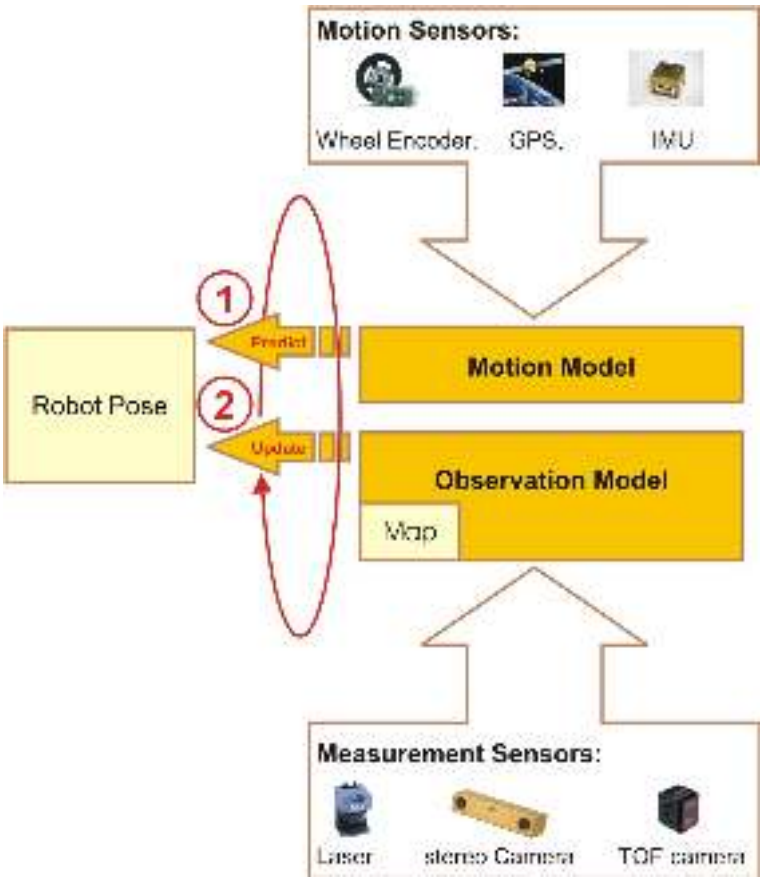


Figure 5.1: SLAM algorithm procedure: (1) The prediction, or motion model, determines a new robot pose based on motion sensors as input. (2) The update, or observation model, corrects the predicted robot pose based on observed features or grid cells.

SLAM algorithms build maps of the environment to estimate the true robot pose. These maps could be in general occupancy grid maps, feature maps or topological maps. There exist more exotic map representations, and the interested reader is referred to the book of Thrun et al. [49].

To evaluate the quality of a SLAM algorithm, in this chapter the differences between occupancy grid based SLAM methods and feature based methods are described. Topological map based algorithms were neglected, because this would go beyond the scope of this master thesis.

Hereby, an existing grid map based algorithm, named *GMapping* is compared to two feature based approaches. *GMapping* is well documented, freely available and uses a laser range finder as sensory input. In contrast, a freely available feature based SLAM implementation could not be found for research purposes and therefore a detailed

description of two feature based approaches of the SLAM problem is given in Section 5.3.

5.2 The SLAM Problem

SLAM algorithms can be roughly classified by their estimation techniques and their map representations. Both aspects are considered in detail in this section.

As mentioned before, a typical SLAM method consists of prediction and an update step, see Figure 5.2. In practice, the true robot position is unknown. A SLAM algorithm estimates the true trajectory (solid line) with observations (dark stars) of the environment. The prediction step leads to an initial guess of the vehicle pose, denoted by the thick, dashed-dotted line, and is furthermore corrected in the update step. Note that the robot surroundings (map) and the robot path are estimated simultaneously. This is the real challenge and makes the problem difficult.
Parts of the map could be either new observations or re-observed. The latter are used for the vehicle pose and feature correction process. New parts are added to the map and initialised with a default uncertainty.
In the following, a categorisation based on pose estimation techniques and map representations is introduced. The difference between a *Kalman Filter* and a *Particle Filter* is discussed. The theoretical background can be found in Section 3.
Another way to categorise SLAM algorithms is based on the map representation. This section highlights the strengths and weaknesses between occupancy grid maps, which are used by the *GMapping* method, and feature maps.

Robot Pose Estimation Techniques

Since the 1990s, probabilistic methods like *Kalman Filters (KFs)*, *Particle Filters (PFs)* and *Expectation Maximisation Methods (EMs)* have become popular approaches for the SLAM problem. The theory behind *Kalman Filters (KFs)* and *Particle Filters (PFs)* has been discussed in detail in Section 3. Here, the differences between the *Kalman Filter* and *Particle Filter* are discussed.
A *Kalman Filter* is a linear recursive filter. The assumption of linearity does not hold for motion sequences of typical mobile robots and therefore some extensions have been developed. An *Extended Kalman Filter* uses a Taylor series expansion to handle non-

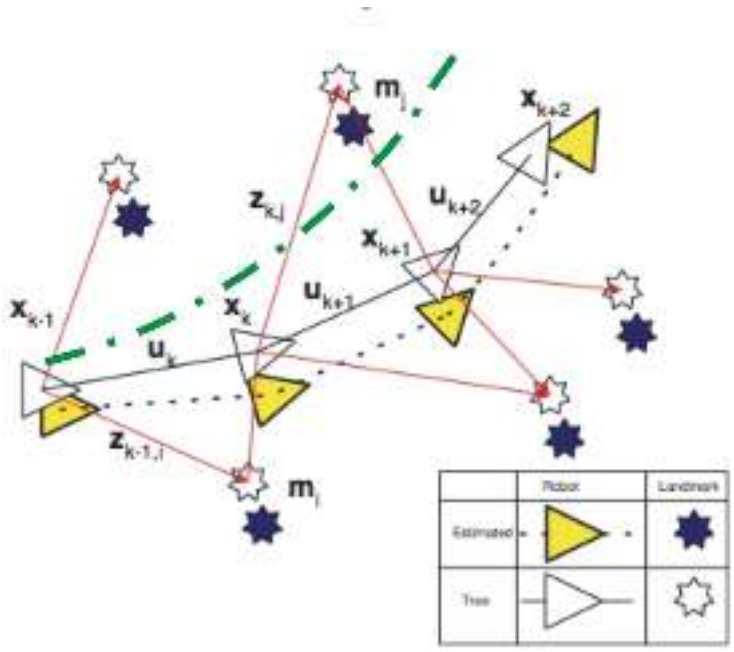


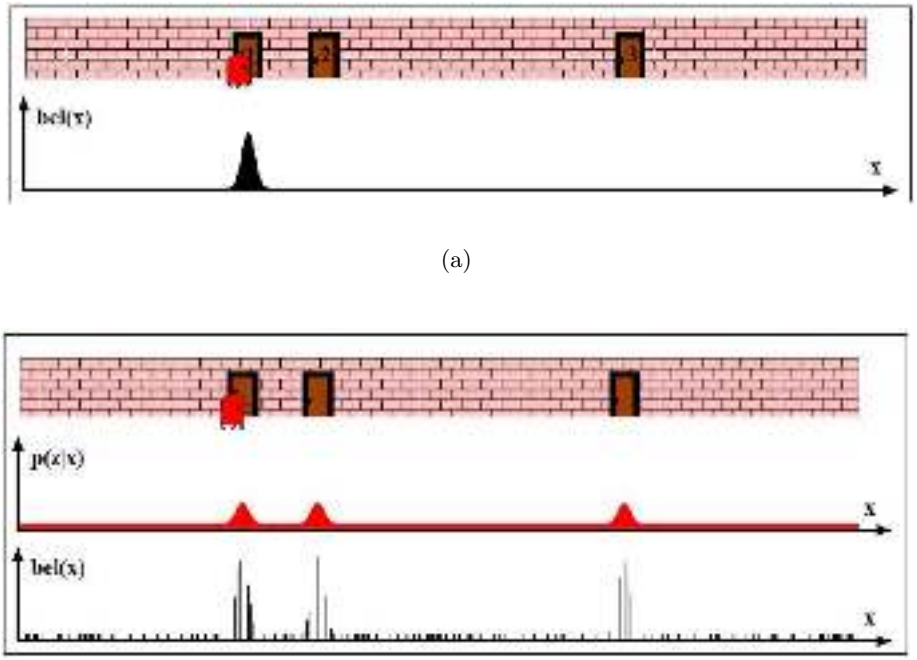
Figure 5.2: The essential SLAM problem, Hugh Durrant-Whyte et al. [16]: The robot trajectory and the landmarks are estimated simultaneously. The true robot path x_k (solid line) and the true landmarks m_i (dark stars) are unknown. Control commands u_k are used for an initial prediction and the observations $z_{k,j}$ are necessary for the estimation process.

linearities, see Section 3.3.2. The *Compressed Kalman Filter* is similar to the *Extended Kalman Filter* but reduces the computational complexity by updating only a local subset of the map features. There exist many more extensions and the reader is referred to the comprehensive work of Thrun et al. [49].

A *Particle Filter* approximates a Bayes filter with a set of discrete states. The likelihood of a discrete state hypothesis is given by a weight. New observations lead to weight changes and continuous resampling prevents degeneration of the estimated robot pose.

To illustrate the differences between a *Kalman Filter* and a *Particle Filter*, one imagines an localisation example. The map is known and the task is to estimate the robot pose. First, a mobile robot is equipped with a specific door detector and travels through a corridor, see Figure 5.3 (a). The doors are numbered and the estimator knows exactly its location at each detected door.

In the case of a *Kalman Filter*, the robot pose distribution has one peak around the door location. When no observation is made, the peak is shifted by the odometry information and stretched because of the higher uncertainty. But what happens, if the door detector cannot distinguish between the three doors? In this case the *Kalman Filter* fails, because



(a)

(b)

Figure 5.3: *Kalman Filter* and *Particle Filter* representation for localisation tasks, Thrun et al. [49]: (a) A *Kalman Filter* represents the pose of a robot with a uni-modal Gaussian density function. b) A *Particle Filter* can represent arbitrary density functions with discrete states.

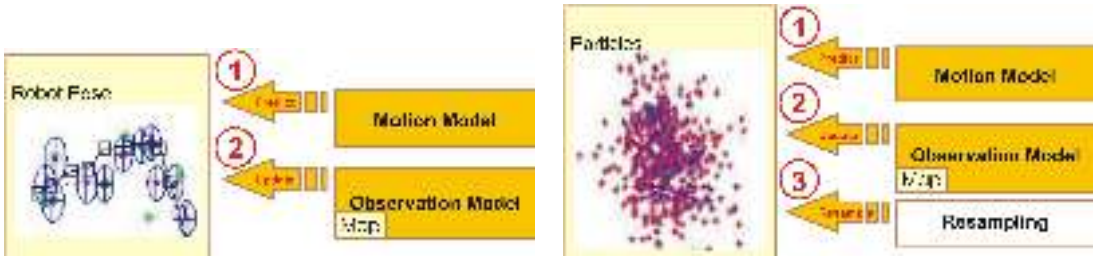
it is averaging over the three observations.

In contrast, a *Particle Filter* is able to represent arbitrary density functions with a set of discrete weights. In the first case, the doors are numbered and the exact position is known. The particles with the highest weights are located at the door position and the pose representation is a discrete version of the parametric representation, used by the *Kalman Filter*. In the second case, when the door number is unknown the particle weights are higher at all three observations, see Figure 5.3 (b). A *Particle Filter* is able to track multiple hypotheses simultaneously and is able to recover from a lost track of its location.

In the higher dimensional case, a *Kalman Filter* represents a robot pose by a state vector and a corresponding uncertainty, often denoted by the covariance ellipses, see Figure 5.4 (a). The discrete state vector of a *Particle Filter* is shown in Figure 5.4 (b).

For a more detailed comparison between different filter algorithms see Section 7.1.

In addition to the pose estimation technique, SLAM approaches can also be categorised



(a) (b)

Figure 5.4: *Kalman Filter* and *Particle Filter* procedure: (a) A *Kalman Filter* represents a robot pose by a state vector and a corresponding uncertainty, often denoted by the ellipses. (b) A *Particle Filter* represents the pose by an arbitrary density function with discrete states.

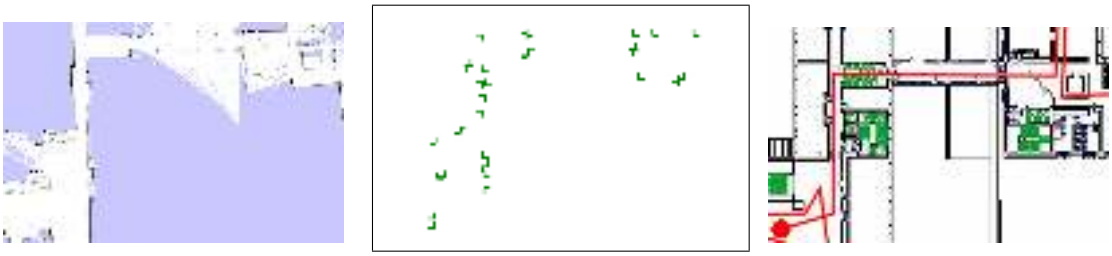
by their map representation. This is done in the next section.

Map Representation

For SLAM algorithms, a map can be in general an occupancy grid map, a feature map or a topological map. This section covers the details of occupancy grid maps and feature maps. Topology maps are discussed in [49].

An occupancy grid quantises the world in small blocks and uses raw data observations directly to estimate the robot trajectory. Each block (cell) of the map could be either labelled as free or occupied cell in the binary case, or obtain a real value, describing the uncertainty (0 - free, 0.5 - uncertain and 1 occupied). No assumption about the environment is made and therefore arbitrary object structures can be modelled, see Figure 5.5 (a). The map can consist of corners, edges, circles, chairs, etc. as long as the block resolution is small enough.
Occupancy grid maps are quite robust to outliers (e.g. moving persons in dynamic environments), because an observation often leads to several hundred block updates, which is a lot compared to a few block outliers.
Large maps could lead to huge memory consumptions. One optimisation is to store only blocks in the map where a laser beam, for instance, ends. More details and efficient implementations can be found in the work of Grisetti et al. [29] and in the publication of Eliazar et al. [18].
Grid maps are mostly limited to two dimensions. Grid SLAM algorithms are also

limited to geometry information obtained from distance sensors. In contrast, feature SLAM methods can handle arbitrary multidimensional features (geometrical features like corners, SIFTs, barcodes ...).



(a) (b) (c)

Figure 5.5: Map representation examples: (a) An occupancy grid based approach quantises the world in small blocks and uses raw data observations directly to estimate the robot trajectory. (b) Feature maps represent specific geometrical objects of the environment. Here, corners were extracted from laser readings. (c) Corresponding detail blue-print of environment.

Feature maps represent specific objects of the environment. Corners and edges are typical features extracted from laser readings, see Figure 5.5 (b).
Regarding the computational time necessary for the map update and the data association step (see Section 3.3.2), feature based maps are very efficient. The main drawback is that feature extraction methods are sensitive to noise and outliers. The second problem is the sensitivity of the SLAM algorithms to incorrect data association. *Kalman Filters* may easily diverge, whereas *Particle Filters* are more robust because of the multiple hypotheses.
It depends on the specific task and the sensor technology, which map representation is the best choice. Occupancy grid maps tend to become the state of the art map representation for sonar and laser readings, see [29] and [18]. In vision based SLAM algorithms, more often feature based maps are used, see [49].

5.3 SLAM Methodologies

In this section, different SLAM algorithms are discussed. The chosen occupancy grid map based algorithm is GMapping. It is well documented ([49], [29] and [11]), freely available and uses a laser range finder as sensory input.
A freely available feature based SLAM implementation could not be found for research

purposes. Therefore, this section describes a corner based SLAM method and an edge based SLAM approach. The motivation behind this geometrical feature choice, relies on the fact that these are easy to extract from laser readings and there exist a lot of reference publications (Vandorpe et al. [50] and Gonzalez et al. [26]), see Section 2.

Both feature SLAM methods are based on *Rao-Blackwellized Particle Filters* (see Section 3.5), which is a state of the art recursive filter. In principle, both methods are designed for a laser range finder and wheel encoders, see Figure 5.6.

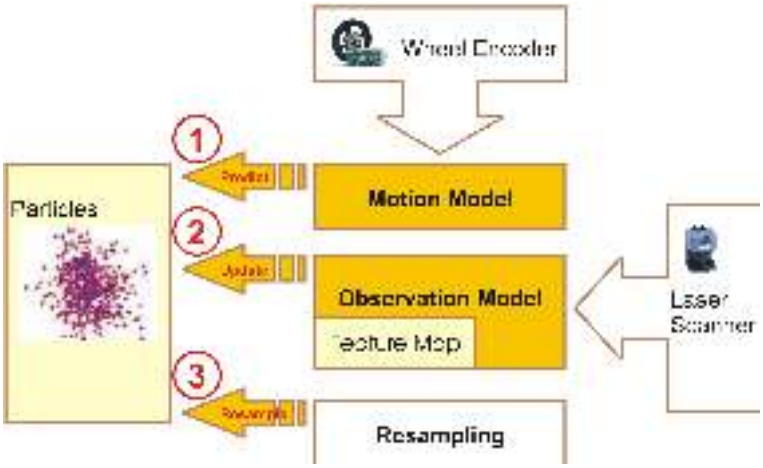


Figure 5.6: SLAM procedure of our realisation: Our choice was a feature based algorithm with an efficient implementation of a *Particle Filter*. A laser range finder (Sick LMS 200) is used as measurement input and the wheel encoders of the *PeopleBot* are the source for the motion model.

The first method relies on corner features, while edges are used by the second SLAM algorithm. In both cases, an assumption of orthogonal geometric relationships between the features like in the work of Nguyen et al. [40] and Fulgenzi et al. [22] is applied. This assumption is useful to restrict geometrical feature search space, especially for indoor office like environments.
In both proposed algorithms, the feature extraction procedure was inspired by the work of Xavier et al. [52]. He has published an efficient method to extract arcs, lines and legs from laser readings. The geometrical relationships between the two-dimensional laser points are used for a fast feature extraction, see Figure 5.7. Starting from an arbitrary center, a local, growing neighbourhood is evaluated to test if the angular values are within a defined range. This relationship between neighbouring points is used to find center points which build within their neighbours a geometrical structure, like a corner

or an edge.

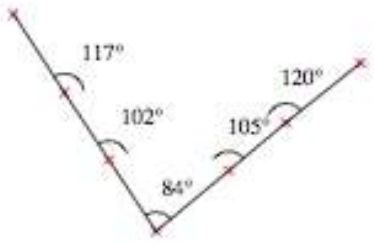


Figure 5.7: Laser point relationships, Xavier et al. [52]: The edge extraction procedure uses the relationships between local laser points. Starting from an arbitrary center, a local, growing neighbourhood is evaluated to test if the angular values are within a defined range.

In the following, a detailed description of a corner SLAM method and an edge based SLAM algorithm is proposed. To conclude this section, a comparison between these two feature based approaches is given, highlighting their strengths and weaknesses under certain environmental conditions.

5.3.1 Corner Feature based SLAM

Corner features can be easily extracted from laser data and are used in [40], [22] and [2]. Altermatt et al. [2] uses a relative corner map approach in combination with a *Kalman Filter*. Instead of the relative map, the following technique uses a geometric constraint on the orientation of the extracted corners and a *Rao-Blackwellized Particle Filter* (Section 3.5).

Corner Representation

To apply the orthogonality constraint, a two dimensional corner position (μ^x, μ^y) is extended by a corner orientation μ^α, see Figure 5.8.
A local (robot) coordinate of a corner position is denoted by the two dimensional vector $(z^r, z^\theta)^T$ and is observed in the local frame indicated by x^L and y^L. Map corners are stored as global features, belonging to the global frame, which is indicated by x^W and y^W.

Corner Extraction

As stated in the introductory part of this chapter, a local relationship between neighbouring two dimensional laser points is used to find center points which build a geometrical structure, like a corner, see Figure 5.9 (a).

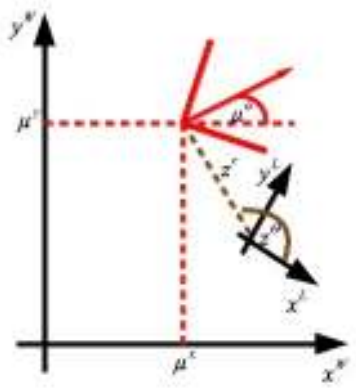


Figure 5.8: Corner feature representation: The two dimensional corner position (μ^x, μ^y) has been extended by a corner orientation μ^α. The local coordinate frame is denoted by x^L and y^L and the global frame is labelled with x^W and y^W.

For robustness, corners with less than four neighbour points lying on the rectangular triangle are rejected (b). This simple constraint removes bad corners caused by noisy observations.

The orthogonality constraint restricts the possible corner space to a smaller sub set and is calculated with respect to the coordinate system x-axis, see Figure 5.9 (c). This can increase the robustness of the algorithm for some special environments. In a typical office-like indoor environment, corners are often orthogonal to walls, but the assumption may not hold outdoors. Further discussions can be found in Section 5.3.3.

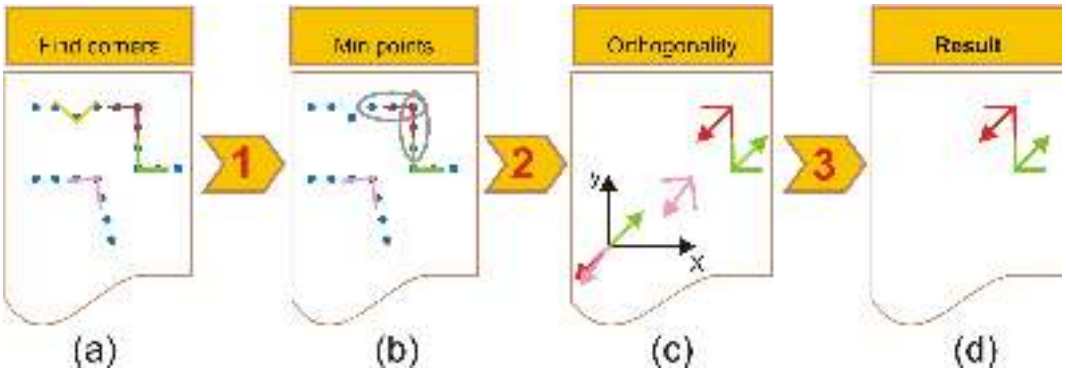


Figure 5.9: Corner extraction procedure: (a) A local relationship between two dimensional laser points is used to find center points which build a rectangular triangle. (b) Corners with less than four neighbour points, lying on the rectangular triangle are rejected. The orthogonality constraint restricts the possible corner space to a smaller sub set (c). The orientation is calculated with respect to the coordinate system x-axis. (d) illustrates the final result with two remaining orthogonal corners.

Corner Association

In principle, the data association process is responsible for connecting an observation to correspondences in the stored map representation (see Section 3.5). An observation is extracted in a local frame and the goal is to find a match in the hole map. For a corner SLAM algorithm, a simple data association technique could be performed by a nearest neighbour approach, as shown in Figure 5.11. This technique is also used by Altermatt et. al. [2]. The distance measure used, is a combination of a Euclidean distance between two corner positions and the difference of their orientations, see Equation 5.1. Map corners are denoted by the red arrows (1 and 3). The dashed green arrow (2) illustrates an observation. Due to the influence of the corner orientation, the association is correct. A purely Euclidean distance measure would associate the other map corner (3) with the observation.(2). The corner distance measure is:

$$d = k_1 \cdot \sqrt{(x - \tilde{x})^2 + (y - \tilde{y})^2} + k_2 \cdot |sinh(\theta - \tilde{\theta})|, \tag{5.1}$$

where the first corner is given by the vector $[x, y, \theta]$ and the second one by $[\tilde{x}, \tilde{y}, \tilde{\theta}]$. The weighting parameters k_1 and k_2 balance the influence of the position and the angular difference. In this work, the values for k_1 and k_2 were empirically set to $k_1 = 5$ and $k_2 = 1/5$. The *sinh* function was chosen because $|sinh(x)|$ is symmetric in the range $[-\pi, \pi]$, see Figure 5.10.

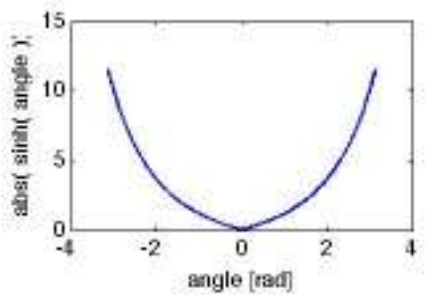


Figure 5.10: Corner distance measure angular part: The *sinh* function was chosen because $|sinh(x)|$ is symmetric in the range $[-\pi, \pi]$.

An association between an observation and a map feature leads to a map feature update. The mathematical description can be found in Section 3.3.1 and the following section describes the computation of the necessary Jacobian matrices for a complete SLAM algorithm implementation.

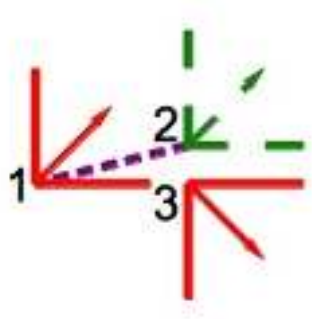


Figure 5.11: Corner association with orientation: The map corners are denoted by the red arrows (1 and 3). The dashed green arrow (2) illustrates an observation. Due to the influence of the corner orientation, the association is correct for this example. The Euclidean distance measure would associate the other map corner (3) with the observation.(2)

Transition Function and System Jacobian

By now, it has been described how corners are associated with map features. For a complete implementation, see Section 3.5, a mapping from local features to global features and vice versa is needed. This is mapping described in the following.

New Features The features are extracted in a local frame, transformed with h^{-1} to a global feature, see Equation 5.3, and finally added to the map. An example code of the *Rao-Blackwellised Particle Filter* has been presented in Table 3.2, where the mean of a new landmark is computed in Line 6. Given a particle robot pose $x_t^{[m]}$ and a local feature z_t (see Figure 5.8), the global feature $\mu^{[m]}$ are computed by:

$$\mu^{[m]} = h^{-1}(z_t, x_t^{[m]}) \tag{5.2}$$

$$= \begin{pmatrix} \mu^x \\ \mu^y \\ \mu^\alpha \end{pmatrix} = \begin{pmatrix} x^{[m]} + z^r \cdot cos(\theta^{[m]} + z^\theta) \\ y^{[m]} + z^r \cdot sin(\theta^{[m]} + z^\theta) \\ \theta^{[m]} + z^\theta \end{pmatrix}. \tag{5.3}$$

The mapping function h^{-1} describes the transformation from a locally extracted feature to a global or absolute landmark.

Line 8 of Table 3.2 describes the computation of the initial landmark

covariance $\Sigma_{j,t}^{[m]} = HQ_tH^T$. The involved Jacobian H of h^{-1} is:

$$\begin{aligned} H &= h^{-1'}(x_t^{[m]}, \mu_{j,t}^{[m]}) && (5.4)\\ &= \begin{pmatrix} \frac{\partial \mu_1}{\partial z^r} & \frac{\partial \mu_1}{\partial z^\theta} \\ \frac{\partial \mu_2}{\partial z^r} & \frac{\partial \mu_2}{\partial z^\theta} \end{pmatrix} && (5.5)\\ &= \begin{pmatrix} cos(\theta^{[m]} + z^\theta) & -z^r \cdot sin(\theta^{[m]} + z^\theta) \\ sin(\theta^{[m]} + z^\theta) & z^r \cdot cos(\theta^{[m]} + z^\theta) \end{pmatrix}. && (5.6) \end{aligned}$$

Note that the corner orientation z^α is not involved in the computation, because it is only used for an orthogonal filter pre-processing step.

Re-observed Features In contrast to a new feature, a re-observed feature $\mu_{j,t-1}^{[m]}$ is already stored in a map, where t denotes the time index and the associated feature number is labelled by j. Map features before the update procedure are labelled by the index $t-1$ (see Line 14 in Table 3.2).

In Line 10 in Table 3.2, the local feature $\tilde{z} = h(\mu_{j,t-1}^{[m]}, x_t^{[m]})$ with the stored absolute landmark $\mu_{j,t-1}^{[m]}$ and the robot pose $x_t^{[m]}$ is estimated. The measurement function h is defined as a transformation from a global coordinate frame to the local robot coordinate system and the estimated landmark is:

$$\begin{aligned} \tilde{z} &= h(\mu_{j,t-1}^{[m]}, x_t^{[m]}) && (5.7)\\ &= \begin{pmatrix} \sqrt{q} \\ tan^{-1}(\frac{\mu^y - y^{[m]}}{\mu^x - x^{[m]}}) \end{pmatrix}, && (5.8) \end{aligned}$$

where the distance q is:

$$q = (\mu^x - x^{[m]})^2 + (\mu^y - y^{[m]})^2. \quad (5.9)$$

The Jacobian H of function h (Line 11 in Table 3.2) is calculated with respect to the map feature:

$$H = \begin{pmatrix} \frac{\partial \tilde{z}_1}{\partial \mu^x} & \frac{\partial \tilde{z}_1}{\partial \mu^y} \\ \frac{\partial \tilde{z}_2}{\partial \mu^x} & \frac{\partial \tilde{z}_2}{\partial \mu^y} \end{pmatrix} = \begin{pmatrix} \frac{\mu^x - x^{[m]}}{\sqrt{q}} & \frac{\mu^y - y^{[m]}}{\sqrt{q}} \\ -\frac{\mu^y - y^{[m]}}{q} & \frac{\mu^x - x^{[m]}}{q} \end{pmatrix}, \quad (5.10)$$

where q is defined in (5.9).
The next Section describes an edge based SLAM algorithm and follows the same structure like this chapter.

5.3.2 Edge Feature based SLAM

We propose a method similar to the work of Nguyen et al. [40]. There, edge features, extracted from laser data are used. Recently, a three dimensional variant of the method was also published [41]. Compared to this laser based technique, Choi et al. [9] extracts edge features from sparse and noisy sonar readings. Both authors use geometrical constraints to restrict the feature space and to increase the robustness.
Regarding the SLAM method of Nguyen et al. [40], we use a different feature extraction procedure and a simplified data association technique. A clustering step based on angles between neighbouring points is performed to reduce the computational costs. Furthermore a different distance measure is proposed.
Like for the corner based SLAM method, first the feature representation is discussed. The next section describes the feature extraction procedure and is followed by an association technique description. The algorithm section is concluded with a mathematical description about the transition functions and the system Jacobians.

Edge Representation

An edge, extracted in a local coordinate frame, is denoted by the two dimensional vector $z_t = (z^r, z^\theta)^T$, where r represents the perpendicular distance from an infinite line to the origin of a reference frame and θ denotes the orientation with respect to the x-axis, see Figure 5.12. This representation was also used by Leonard et al. [33] and introduced as the *Plane Target Model*.

Edge Extraction

As stated in Section 5.3, a local relationship between neighbouring two dimensional laser points is used to find center points which build a geometrical structure.
At first a clustering is performed. The angles between neighbouring points are evaluated. Valid, consecutive points are grouped, which is indicated by the ellipses in Figure 5.13 (a). To increase the feature robustness, all clusters containing less points than a given threshold are rejected. Furthermore, a least squares method is used to obtain the edge parameters (r, θ).

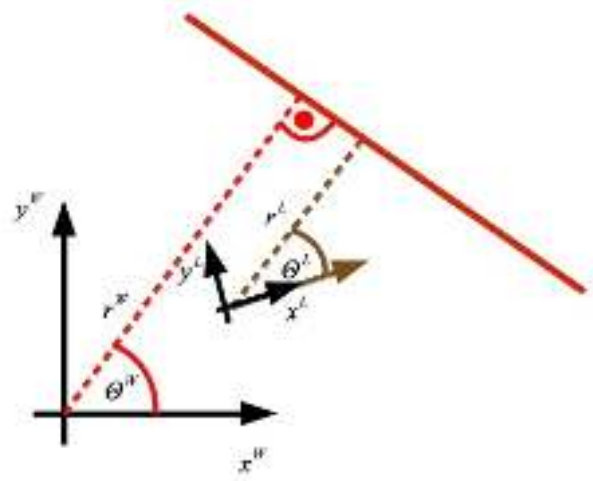


Figure 5.12: Edge feature transformation: The red line denotes one edge feature, which is represented by the vector $(r^W, \theta^W)^T$ in world coordinates. The local representation is denoted by $(r^L, \theta^L)^T$ and the transformation is given by (5.13).

Finally the orthogonality constraint is applied to the remaining edges. The edge angle is evaluated with respect to the x-axis of the world coordinate system and compared with a previously extracted reference angle (extracted from the first edge). This constraint is useful for some special environments. In a typical office-like indoor environment, edges are characterised by walls, which are usually orthogonal to each other. The assumption may not hold outdoors. Further discussions on this topic can be found in Section 5.3.3.

Edge Association

In principle, the data association process is responsible for connecting an observation to correspondences in the stored map representation (see Section 3.5). An observation is extracted in a local frame and the goal is to find a match in the hole map.
Like for the corner SLAM algorithm, a simple edge feature data association technique could be performed by a nearest neighbour approach. The proposed distance measure:

$$d = r^2 + \tilde{r}^2 - 2 \cdot r \cdot \tilde{r} \cdot cos(\theta - \tilde{\theta}), \tag{5.11}$$

where the first edge is given by the vector $[\theta, r]$ and the second one by $[\tilde{\theta}, \tilde{r}]$. Note that the distance measurement is the Euclidean distance between two polar coordinates, which are intersection points of the edge feature and the perpendicular lines from the origin (next to the dot of Figure 5.12). A small variation of θ has a big influence on

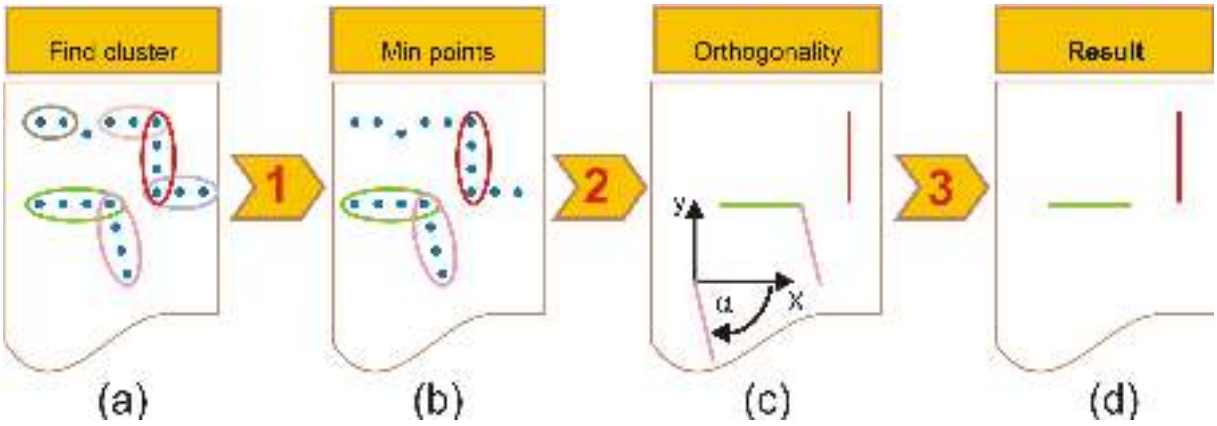


Figure 5.13: Edge extraction procedure: (a) At first, the angles between neighbouring points are evaluated. Valid, consecutive points are grouped to clusters. (b) To increase the feature robustness, all clusters containing less points than a given threshold are rejected. (c) illustrates how the orthogonality constraint affects the remaining edges. The edge angle is evaluated with respect to the x-axis of the world coordinate system.

edges, far from the coordinate origin. In contrast, edges close to the origin are not affected that much. This local dependency is not desirable and more details can be found in the discussion Section 5.3.3.

An association between an observation and a map feature leads to a map feature update. The mathematical description can be found in Section 3.3.1 and the following section describes the computation of the necessary Jacobian matrices for a complete SLAM algorithm implementation.

Transition Function and System Jacobian

By now, it has been described how edges are associated with map features. For a complete implementation, a mapping from local features to global features and vice versa is needed. This is mapping described in the following.

New Features The mapping from a local to a global feature, denoted by the transition function h^{-1} was rather intuitive and simple for corner landmarks (see Equation 5.3). We will have a closer look to the geometrical relations of a function which maps a local edge to a global one in Figure 5.12.
The red line denotes one edge feature, which is represented by the vector $(r^W, \theta^W)^T$ in

world coordinates. The local representation is denoted by $(r^L, \theta^L)^T$ and the transformation is given by:

$$\mu^{[m]} = h^{-1}(z_t, x_t^{[m]}) \tag{5.12}$$

$$= \begin{pmatrix} r^W \\ \theta^W \end{pmatrix} = \begin{pmatrix} r^L + x^{[m]} \cdot cos(\theta^L + \theta^{[m]}) + y^{[m]} \cdot sin(\theta^L + \theta^{[m]}) \\ \theta^L + \theta^{[m]} \end{pmatrix}. \tag{5.13}$$

The parameter θ^W can be used for the orthogonal filter pre-processing step. Per definition, r^W may not be negative and in the case of a negative distance, the corrections $r^W = -1 \cdot r^W$ and $\theta^W = \theta^W + \pi$ are applied.

The computation of the system Jacobian H (Line 8 of Table 3.2) is necessary to calculate an initial landmark covariance $\Sigma_{j,t}^{[m]} = HQ_tH^T$. Equation 5.16 shows the estimation of the involved Jacobian H of h^{-1}.

$$H = h^{-1'}(x_t^{[m]}, \mu_{j,t}^{[m]}) \tag{5.14}$$

$$= \begin{pmatrix} \frac{\delta \mu_1}{\delta r^L} & \frac{\delta \mu_1}{\delta \theta^L} \\ \frac{\delta \mu_2}{\delta r^L} & \frac{\delta \mu_2}{\delta \theta^L} \end{pmatrix} \tag{5.15}$$

$$= \begin{pmatrix} 1 & -x^{[m]} \cdot sin(\theta^W + \theta^{[m]}) + y^{[m]} \cdot cos(\theta^W + \theta^{[m]}) \\ 0 & 1 \end{pmatrix}. \tag{5.16}$$

Re-observed Features In contrast to a new edge, a re-observed feature $\mu_{j,t-1}^{[m]}$ is already stored in a map, where t denotes the time index and the associated feature number is labelled by j. Map features before the update procedure are labelled by the index $t-1$ (see Line 14 in Table 3.2).

A local landmark $\tilde{z}$ is estimated with a robot hypothesis $x^{[m]}$ and a stored world feature $\mu_{j,t-1}^{[m]}$. The transformation from a world feature $\mu_{j,t-1}^{[m]}$ to a local estimate $\tilde{z}$ is given by:

$$\tilde{z} = h(\mu_{j,t-1}^{[m]}, x_t^{[m]}) \tag{5.17}$$

$$= \begin{pmatrix} r^L \\ \theta^L \end{pmatrix} = \begin{pmatrix} r^W - x^{[m]} \cdot cos(\theta^W) - y^{[m]} \cdot sin(\theta^W) \\ \theta^W - \theta^{[m]} \end{pmatrix}, \tag{5.18}$$

where the vector $\mu_{j,t-1}^{[m]}$ is computed with (5.13). The parameter r^L may not be negative and in the case of a negative distance, the corrections $r^L = -1 \cdot r^L$ and $\theta^L = \theta^L - \pi$ are applied.

The involved Jacobian H (see Line 15 in Table 3.2) of the function h for edge features can

be calculated with respect to the world feature $\mu_{j,t-1}^{[m]}$ the following way:

$$H = \begin{pmatrix} \frac{\delta \tilde{z}_1}{\delta r^W} & \frac{\delta \tilde{z}_1}{\delta \theta^W} \\ \frac{\delta \tilde{z}_2}{\delta r^W} & \frac{\delta \tilde{z}_2}{\delta \theta^W} \end{pmatrix} \tag{5.19}$$

$$= \begin{pmatrix} 1 & x^{[m]} \cdot sin(\theta^W) - y^{[m]} \cdot cos(\theta^W) \\ 0 & 1 \end{pmatrix}. \tag{5.20}$$

Two feature based SLAM methods were proposed until now and the next section concludes with a discussion about the geometric feature choice and useful applications.

5.3.3 Discussion

The quality of a feature based SLAM algorithm depends heavily on the number of re-observed features. The more often a corner or an edge is updated the better is its quality and the better is the current robot pose estimate. Clearly, the algorithm fails if no observations are made at all.

Concerning the frequency of occurrence, corners and edges could be a proper choice in a typical office-like environment. Note that for *Particle Filters*, continuous updates are important to prevent degeneration of the estimated robot pose.

The edge feature representation, discussed in this section, is sensitive to the distance to the coordinate origin. A small variation of θ has a big influence on edges, far from the coordinate origin. Therefore, a relative map approach is used by Nguyen et al. [40].

Corners could be extracted from laser readings more accurate, but are also easily occluded. In contrast, edges could be observed more frequently than corners in long straight corridors. For more details concerning different geometrical features we refer to the book of Castellanos et al. [7].

The motivation behind our feature type selection relies on the nature of the evaluated indoor datasets. The first recording is a static indoor dataset from *RawSeeds* [6] and the second dataset was generated at our university, which has the structure of a typical office like environment.

Therefore, the observed results (see Section 6) are only valid for special indoor environments.

5.4 Sensor Fusion

Sensor fusion is desired to build more robust and more accurate algorithms based on multiple sensors. As mentioned in the previous section, SLAM algorithms may fail, if no observations are made over a longer time period. This could occur for specific geometrical features, like corners, or for different sensor technologies. For instance, laser range finders are sensitive to window panes or mirrors (see Section 4) and a vision based system may fail, in the case of homogeneous regions, like a white wall.
This section describes a simple SLAM framework to take advantages of different sensor technologies and different SLAM algorithms. The influence of a combination of different sensor technologies is evaluated in Section 6.
Our fusion framework is based on *Particle Filters*, which were discussed in Section 3.4. The described approach relies on separate particle weight updates for each SLAM method and each sensor technology.
Each sensor type or algorithm builds its own map and updates for an observation all particle weights. These weight updates can be performed independent of the arrival time of the different observations, see lines 4 to 6 in Table 5.1.
For simplicity, only one motion model was implemented, (see Section 3.6 for details) denoted by Line 3, where a robot hypothesis $x_t^{[m]}$ is sampled from $P(x_t \mid u_t, x_{t-1}^{[m]})$.

5.5 Conclusion

In this section, the SLAM problem it self and possible SLAM methodologies were discussed. *GMapping* is a well documented, freely available occupancy grid SLAM method. A freely available feature based SLAM implementation could not be found for research purposes and therefore a detailed description of two feature based approaches of the SLAM problem was given.
Finally, the reader was introduced to a new sensor fusion framework based on a *Particle Filter*. This framework takes advantages of a combination of different SLAM algorithms and of different sensor technologies.
In future, the fusion framework can be the basic concept to integrate new sensor technologies. By now, the discussed maps were two dimensional. More research has to be undertaken to evaluate the capabilities of three dimensional maps. This map could be generated by a time of flight camera (see Section 4.4), for instance.

	$X_t = \text{filter}(X_{t-1}, u_t, z_t)$
1:	$\bar{X}_t = X_t = 0$
2:	for m = 1 to M
3:	sample $x_t^{[m]} \sim P(x_t \mid u_t, x_{t-1}^{[m]})$
4:	$w_t^{[m]} = w_t^{[m]} \cdot P_{Sonar}(z_t \mid x_t^{[m]})$
5:	$w_t^{[m]} = w_t^{[m]} \cdot P_{Laser}(z_t \mid x_t^{[m]})$
6:	$w_t^{[m]} = w_t^{[m]} \cdot P_{Vision}(z_t \mid x_t^{[m]})$
7:	$\bar{X}_t = \bar{X}_t + < x_t^{[m]}, w_t^{[m]} >$
8:	endfor
9:	for m = 1 to M
10:	draw i with probability $\propto w_t^{[m]}$ from $\bar{X}_t$
11:	add $x_t^{[i]}$ to X_t
12:	endfor
13:	return X_t

Table 5.1: Sensor Fusion with a Particle Filter example code: Each sensor (P_{Sonar}, P_{Laser} and P_{Vision}) updates the particle weights of the current population independently and reduces the final variance of the scattered particle positions.

Chapter 6

Experiments

Contents

The main question of this thesis is, which sensor technologies and combinations of them, suit well for state of the art SLAM methods. For this reason, two datasets are used for the experiments, which are both indoor datasets. The datasets were chosen, because of the limited measurement range of the evaluated sensors. The laser range finder can be used outside too, but sonars and the time of flight (TOF) camers are restricted to a range of about five meters.

In Section 4 the sensor noise, which is a crucial factor for SLAM, was presented according to the specification sheets. To prove this specifications, an evaluation of the sensors is given at the beginning.

The *RawSeeds* dataset, "Bicocca_2009-02-25b", is a static indoor dataset with artificial lighting, which is, compared to other *RawSeeds* indoor datasets, often referred by other contributions of the *RawSeeds* web page. This dataset is used to evaluate different SLAM methods and for a sensor fusion between laser measurements and sonars. The generated maps have a size of 200 times 200 metres and contain several big loops.

For the sensor comparison between TOF readings, sonars and laser measurements in general, a new dataset was created, because TOF measurements are not included in the *RawSeeds* dataset. The recordings were made at the institute for Computer Graphics and Vision (ICG), Graz University of Technology. A small loop with a perimeter of about 40

metres is used for the evaluation.
Only the *RawSeeds* dataset provides ground truth (GT) measurements and the evaluation with the dataset from ICG must be performed based on a comparison of the resulting maps.
On the algorithmic side, the *GMapping* method, a corner feature based SLAM implementation and an edge feature based SLAM implementation have been reviewed in Section 5.3, which are evaluated with the *RawSeeds* dataset. A fusion of sonars and time of flight camera measurements is evaluated on the ICG indoor dataset.

6.1 Distance Sensor Evaluation

Sensor specific problems have been reviewed theoretically in Section 4. A practical discussion is given in the following. The accuracy of the evaluated distance sensors depends on the environmental structures and in general, the distance measure fails, if the sonar or the light ray is not reflected back to the sensor. This occurs, dependent on the coarseness of the material surface, if the angle between the obstacle and the ray is too small. In addition, laser range finders and TOF cameras are sensitive to transparent or reflective materials, like window panes and mirrors. Sonars suffer from reflections in narrow floors or corners.

Distance Accuracy Evaluation

Laser measurements are used as ground truth (GT), to evaluate the accuracy of the used sonar transducers and the TOF camera from PMDTec, see Table 6.1. The experiment took place in front of a non-reflecting concrete wall. Only two sonar transducers at front of the mobile robot were evaluated. For the experiment, the robot was placed in front of a concrete wall and was not moving. The maximum error of the sonars is -0.1499m, whereas the TOF camera maximum error is 0.0532m.
The experimentally evaluated working range for the sonar transducers, mounted on the *PeopleBot*, is 0.18m to 4.8m. In contrast the working range of the TOF camera from PMDTec is 0.1m to 5.5m.

Environmental Characteristics

In this chapter, four environmental structures are considered: Concrete walls, metal doors, window panes and metal fences, see Figure 6.1. For each of these structures, samples taken

Sonar Distance Measurements with Laser Readings as GT in [m]

GT	mean	variance	mean error	number of observations
0.5497	0.5315	0.0001	**0.0182**	310
1.0582	1.2081	0.6	**-0.1499**	232
2.0743	2.0783	0	**-0.004**	192
4.0821	4.1511	0	**-0.0690**	186

TOF Distance Measurements with Laser Readings as GT in [m]

GT	mean	variance	mean error	number of observations
0.5497	0.5304	0.0001	**0.0193**	6336
1.0582	1.0048	0.0040	**0.0534**	4992
2.0743	2.0391	0.0022	**0.0352**	4032
4.0821	4.0800	0.0114	**0.0021**	3904
5.0800	5.0532	0.0215	**0.0268**	3581

Table 6.1: Distance Sensor Accuracy evaluation in [m]: For the experiment, the robot was placed in front of a concrete wall and was not moving. The maximum error of the sonars is -0.1499m, whereas the TOF maximum error is 0.0532m. Laser measurements are used as ground truth (GT).

from the side and the front are discussed. The sensor observations can be found in the Appendix Section D, where the wrong measurements are marked red.

(a) (b) (c)

Figure 6.1: ICG indoor environment characteristics:(a) Concrete wall and metal fence. (b) Reflecting metal door. (c) Transparent window pane.

Sun light reflections on concrete walls can lead to wrong measurements for TOF cameras, see Figure D.1 and Figure D.2. In contrast, the Sick LMS200 laser range finder is not sensitive to environmental light. Some sonar beams are wrong because of reflections.
Metal doors lead to reflections, which causes error prone distance measurements of the sonar transducers and the TOF camera. With the side view, Figure D.3 (b), half of the

TOF measurements are reflected.
Window panes are error sources for vision based sensors. Laser readings and the TOF measurements are corrupted. Sonar distances are correct, despite the large noise, see Figure D.4.
The metal fences, present at ICG, are fine (2cmx2cm) and therefore, the distance measurements imprecisions can be led back to the sensor noise, see Figure D.5.

6.2 RawSeeds Indoor

The RawSeeds project [6] funded under the *European Union's Sixth Framework Programme (FP6)*, is the first effort to provide ground truth (GT) data to evaluate SLAM algorithms, see Section 2.4. For the chosen static indoor dataset, "Bicocca_2009-02-25b", the GT data is a fusion of stationary laser range finder distance measurements and observations of a multi camera system. The final GT data uncertainty is less than a few centimetres and enables a quantitative comparison of SLAM algorithms. The GT measurements are only available for one place in the environment, see yellow region in Figure 6.2. This is sufficient for an evaluation, because the GT region is crossed two times during the experiment. For more information concerning the GT measurements, see [6].

6.2.1 Dataset and Setup

The *RawSeeds* dataset contains several sensor measurements:

- Ultrasound transducers
- Inertial measurement unit (IMU)
- Onboard camera systems
  - Binocular and trinocular black-and-white (B/W) vision
  - Normal perspective, colour and B/W cameras
  - Omnidirectional colour vision with hyperbolic mirror
- Laser range finders (LRFs)
  - Two short-range (4m range, less at low reflectivity) Hokuyo URG-04LX LRFs
  - Two Sick LRFs: LMS200 and LMS291

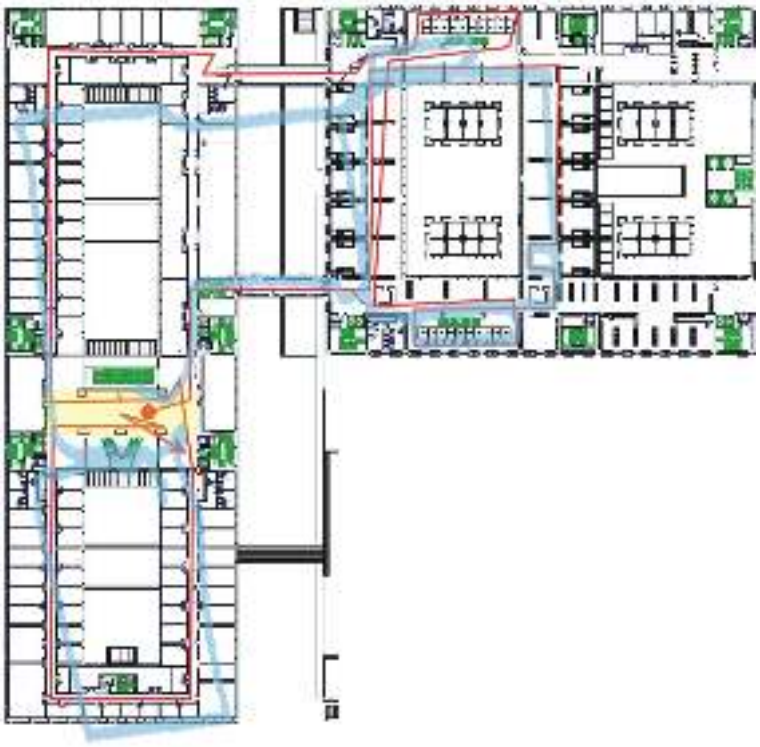


Figure 6.2: RawSeeds static indoor dataset (2009-02-25b): This figure illustrates the blueprint of the environment. The red trajectory indicates the travelled path and was manually annotated. The odometry is denoted by the blue lines and the yellow region shows the only place of the environment where the ground truth information is extracted.

This dataset is used to evaluate the performance of the *GMapping* method, a corner feature based SLAM implementation and an edge feature based SLAM implementation. In addition, the performance gain, obtained by a fusion of laser measurements and sonars is evaluated. Therefore, the sensors used are the front Sick laser range finder (LMS291) and the 12 sonar transducers, see Figure 6.3. The travelled trajectory is 823m long and was recorded in about 25 minutes.

6.2.2 Laser Range Finder

The *GMapping* grid SLAM method, the corner feature based SLAM implementation and the edge feature based SLAM implementation are evaluated, see Figure 6.4. These methods are compared with respect to their accuracy, their robustness and their computational time. The odometry is denoted by the red line, while the estimated trajectory is blue.

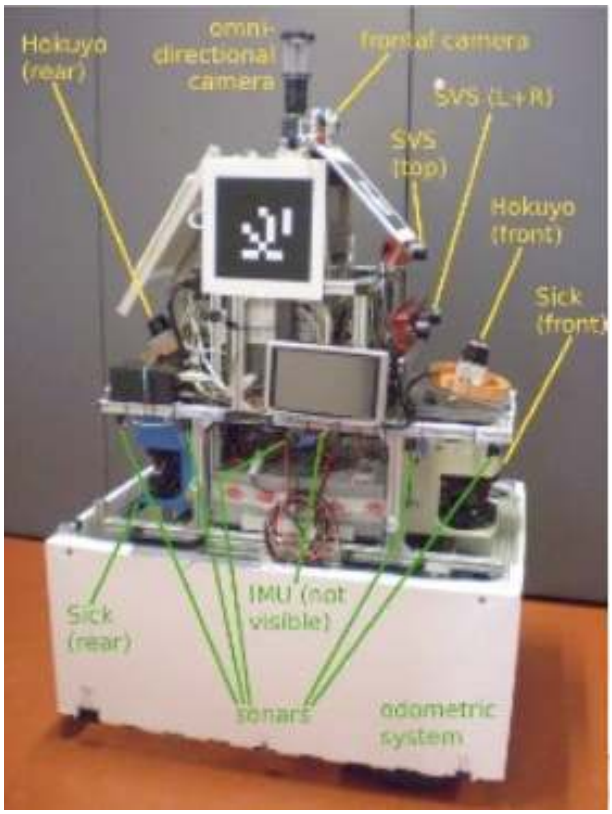


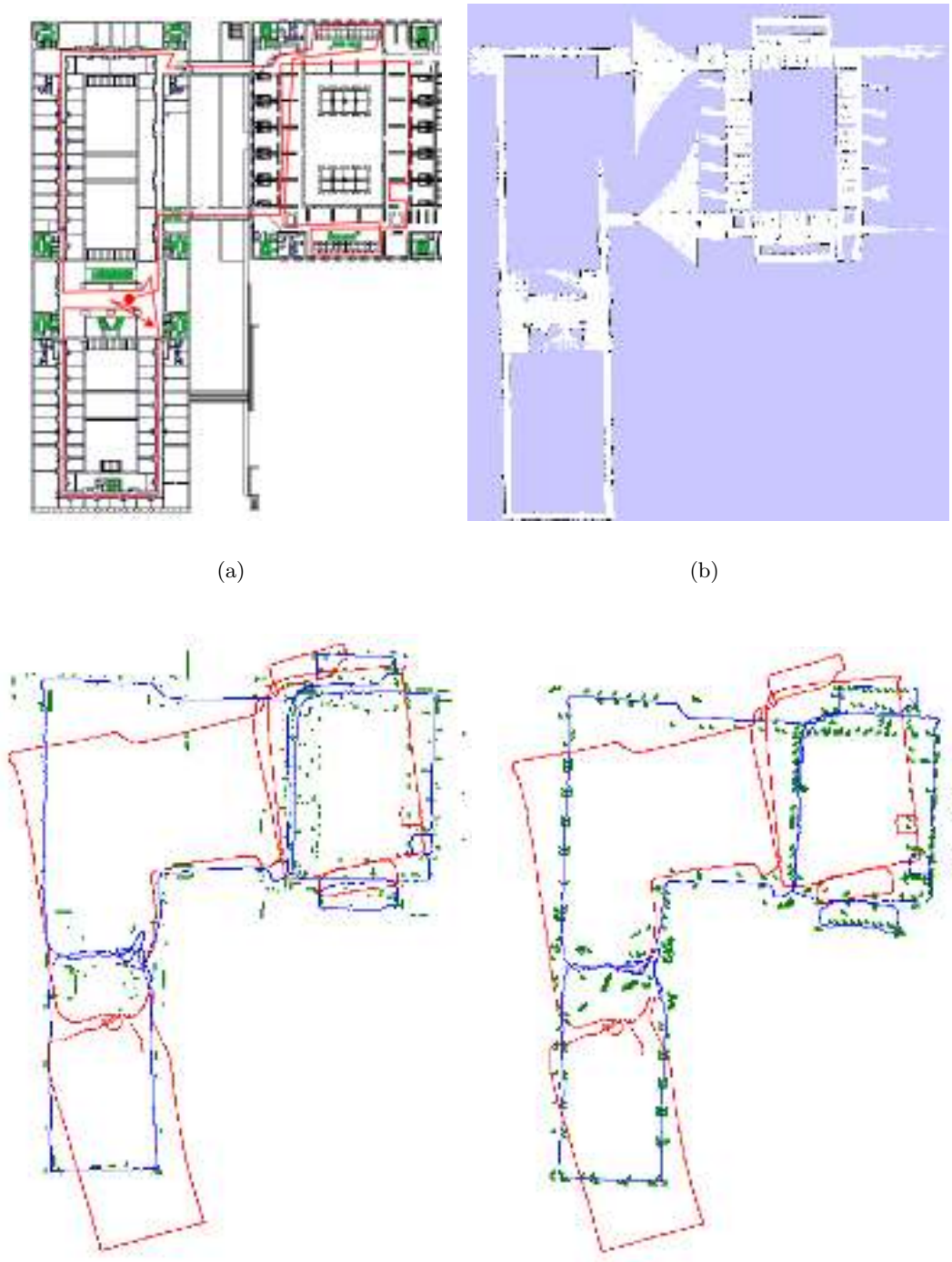
Figure 6.3: Mobile indoor robot of the RawSeeds project [6]: This robot is equipped with four laser range finders, four camera systems, sonar sensors and with an inertial measurement unit.

SLAM Methods Accuracy

The accuracy is evaluated based on the estimated trajectory. The mean squared error is used to compute the difference between the absolute ground truth pose $x_t = [x_i, y_i]$ and the estimated pose $\tilde{x}_t = [\tilde{x}_i, \tilde{y}_i]$. This measurement over all ground truth points in time is referred as the absolute trajectory error (ATE) and is given by:

$$error = \frac{1}{N_{GT}} \cdot \sum_{i=1}^{N_{GT}} \sqrt{(x_i - \tilde{x_i})^2 + (y_i - \tilde{y_i})^2}, \tag{6.1}$$

where N_{GT} denotes the number of ground truth measurements. The heading angle θ_i is not involved, because a wrong angle would automatically result in a position displacement. The estimated poses have different points in time compared to the ground truth measurements. Therefore, an interpolation of the estimate pose $\tilde{x}_t = [\tilde{x}_i, \tilde{y}_i]$ between two estimates $\tilde{x}_{\tilde{t}-1}$ and $\tilde{x}_{\tilde{t}}$ around the ground truth position x_t is necessary, see Figure 6.5.



(a) (b)

(c) (d)

Figure 6.4: SLAM methods evaluation example maps: (a) RawSeeds static indoor dataset "Bicocca_2009-02-25b". (b) GMapping result, obtained with 50 particles. (c) Corner feature based map, generated with 50 particles. (d) Resulting map of the edge feature based SLAM implementation, using 50 particles. The red trajectories denote the odometry paths and the estimated robot paths are indicated by blue lines.

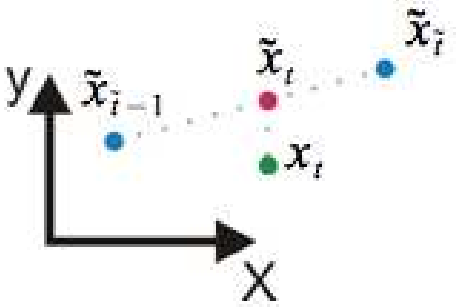


Figure 6.5: Robot pose interpolation for the evaluation: The estimated poses have different points in time compared to the ground truth measurements. Therefore, an interpolation is necessary to compute the absolute trajectory error. $\tilde{x}_{\tilde{t}-1}$ and $\tilde{x}_{\tilde{t}}$ denote the estimated poses, the ground truth position is labelled with x_t and the interpolated pose is denoted by $\tilde{x}_t$.

The *GMapping* grid SLAM method, the corner feature based SLAM implementation and the edge feature based SLAM implementation were evaluated 10 times with 20, 50 and 100 particles, see Table 6.2. For both feature based SLAM implementations, only every third laser reading was used for the update step. The edge feature based SLAM implementation, using 100 particles, has an average ATE of 0.88m. In contrast the ATE of the odometry is 2.25m. The SLAM methods fail sometimes, if not enough particles are in the vicinity of the true robot state. This is indicated by dashes.

A comparison of different SLAM results is published on the *RawSeeds* web page [6], see Figure 6.6. Best results are obtained by the *GraphSLAM* method with an ATE of 0.38m. *GMapping* leads to an ATE of 1.19m, while our implementations have an average error of less than one meter. More particles would result in a lower ATE by the cost of computational complexity and computational time, which are unknown for the published methods on the *RawSeeds* web page. The *GMapping* result used, is the average ATE of our experiments with 100 particles.

SLAM Methods Robustness

There are mainly two indicators for robust SLAM algorithms evaluated with the static indoor dataset "Bicocca_2009-02-25b". First, methods fail more often if the number of particles is too low. This occurs more often with less than 50 particles. Secondly, the methods under test are probabilistic algorithms, were the variance of the ATE indicates

Corner feature based SLAM												
Particles											MEAN	VAR
20	1.32	1.52	0.83	0.89	1.64	0.99	0.83	1.42	1.23	0.96	**1.16**	0.09
50	1.17	-	-	-	0.94	1.22	1.47	0.89	0.98	1.01	**1.1**	0.04
100	1.47	1.31	0.85	0.57	0.68	1.22	0.8	1.06	0.81	0.93	**0.97**	0.08
Edge feature based SLAM												
Particles											MEAN	VAR
20	1.87	1.08	1.97	1.8	1.07	1.16	-	1.63	1.51	2.04	**1.57**	0.15
50	1.7	1.79	-	-	0.67	1.24	-	0.83	0.7	0.96	**0.88**	0.05
100	0.67	1.37	0.82	1.31	0.62	0.64	0.58	0.87	0.61	1.17	**0.87**	0.09
GMapping grid SLAM												
Particles											MEAN	VAR
20	1.21	1.21	-	1.14	0.45	1.77	0.32	2.19	2.18	1.33	**1.31**	0.44
50	2.24	-	-	2.41	0.80	-	0.73	-	0.69	1.07	**1.33**	0.62
100	0.56	0.73	0.52	0.7	1.11	1.97	-	2.09	1.54	0.83	**1.19**	0.37

Table 6.2: SLAM methods evaluation results "Bicocca_2009-02-25b", ATE in [m]: Results of corner feature based SLAM, edge feature based SLAM and *GMapping* evaluated with the "Bicocca_2009-02-25b" dataset.

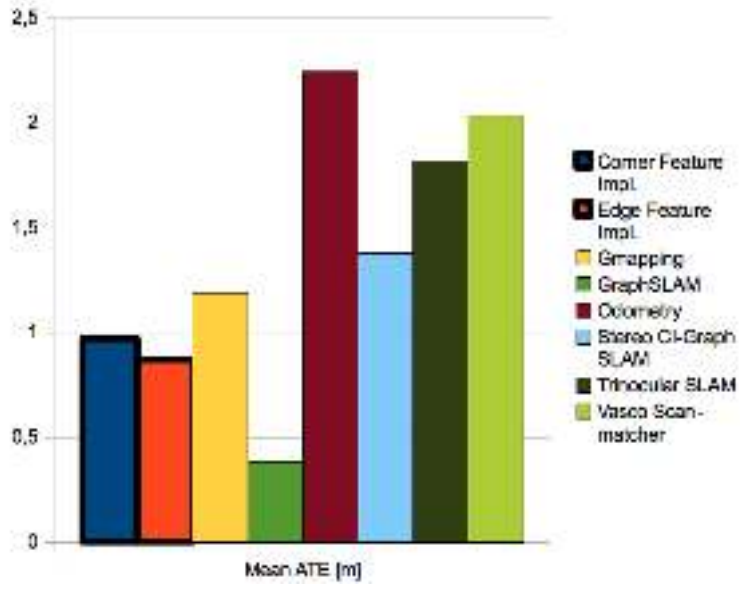


Figure 6.6: SLAM methods evaluation results "Bicocca_2009-02-25b" ATE in [m]: Evaluation results published at *The RawSeeds Project* [6]. Comparison of different SLAM method results for the static indoor dataset.

the repeatability, see Table 6.2. The variance is in general anti-propotional to the number of particles.
Results published at the *RawSeeds* web page [6] provide the standard deviation (StdD) of the ATE, see Figure 6.7. This information is a sign of the smoothness of the estimated trajectory.

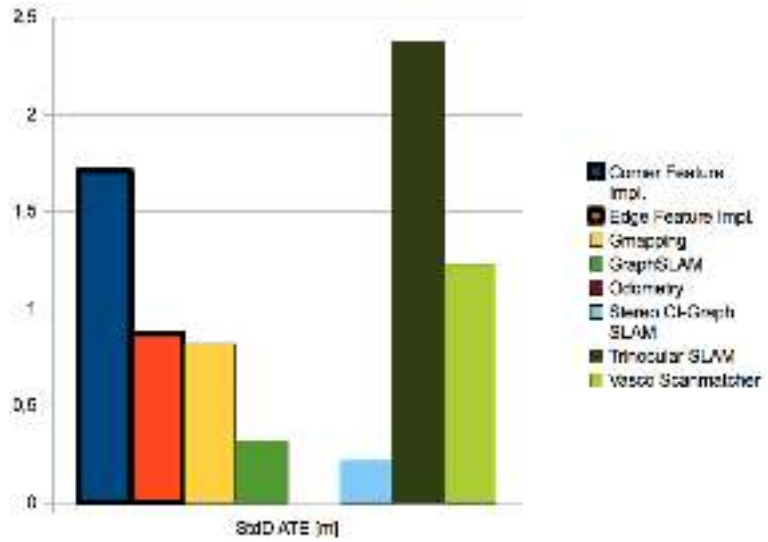


Figure 6.7: SLAM methods evaluation results "Bicocca_2009-02-25b" Standard Deviation in [m]: Evaluation results published at *The RawSeeds Project* [6]. Comparison of different SLAM method results for the static indoor dataset.

Computational Time

For research purposes, our framework was first implemented with MATLAB and later ported to C#. The computational time of our feature based implementations is linear proportional to the number of particles and was evaluated with the C# framework, see Figure 6.8. In contrast, *GMapping* is an efficient C++ implementation with logarithmic time complexity, [29]. The test platform was an Intel 2.6GHz, 2GB Ram machine, running in the *VirtualBox* environment.
The corner feature based implementation is the fastest, but only every third laser reading was used, like for the edge feature based implementation. In the "Bicocca_2009-02-25b" dataset, the mobile robot travelled 823m in 26 minutes.

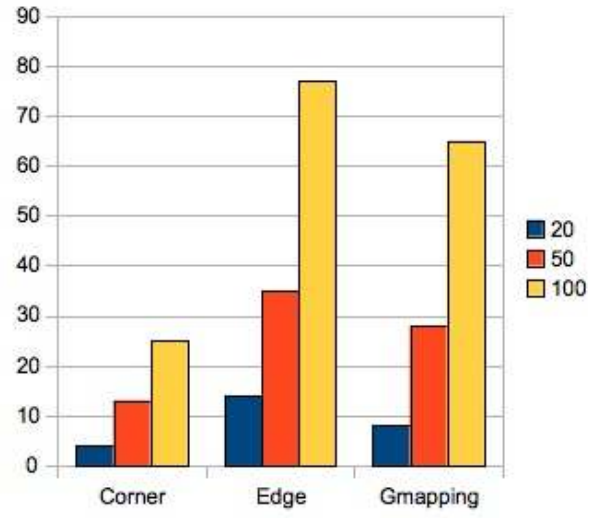


Figure 6.8: SLAM methods computational time evaluation, "Bicocca_2009-02-25b" in [min]: Linear time dependency for all three methods. Test platform: Intel 2.6 Ghz, 2GB Ram, VirtualBox.

6.2.3 Sonar

The "Bicocca_2009-02-25b" dataset contains sonar readings from 12 circular arranged sonar transducers. For the presented experiment, the ATE, generated with 50 particles, is 14.76m, see Figure 6.9. The odometry is denoted by the red line, while the estimated trajectory is blue. A map based error measurement would indicate the improvement of sonar SLAM compared to raw odometry. The upper part of the map shows rectangular structures, but the map in the lower part is corrupted.

6.2.4 Sensor Fusion: Laser and Sonar

The sensor fusion of laser measurements and sonar readings was evaluated 5 times, see Table 6.3. The laser measurements were used by the corner and edge feature based implementation, while sonar readings were used by a grid SLAM method, see Figure 6.10. The odometry is denoted by the red line, while the estimated trajectory is blue. Edges are denoted by the green lines, where the used feature representation is not based on end points, see Section 5.3.2. The results are very similar to the results obtained without sonar readings, see Table 6.2.

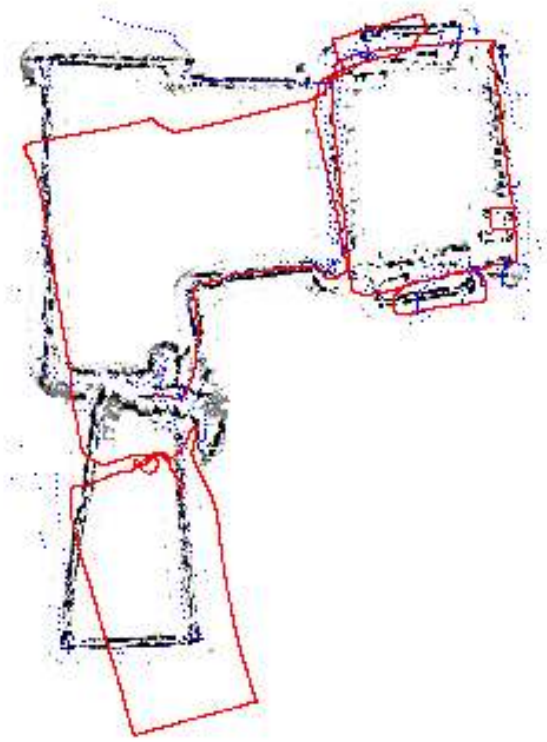
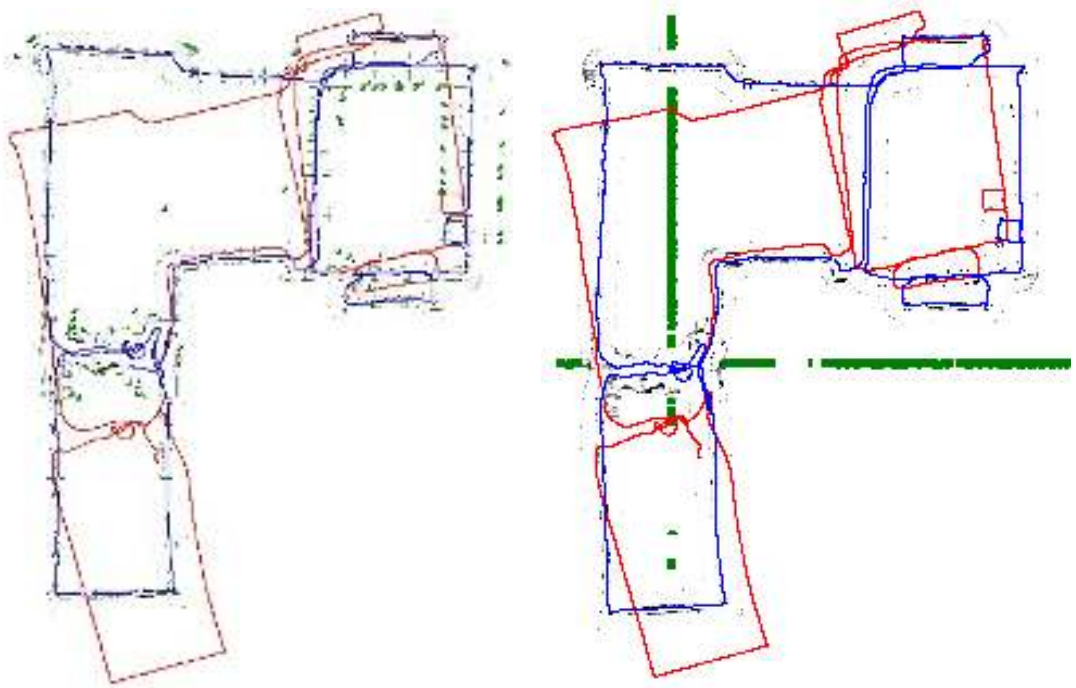


Figure 6.9: Sonar Evaluation on RawSeeds "Bicocca_2009-02-25b" dataset: This map was generated with 12 sonar transducers, a cell size of 20cm and 50 particles. The upper part of the map shows rectangular structures. The red trajectory denote the odometry path and the estimated robot poses are indicated by blue points.

Fusion of Corner feature SLAM Laser and Grid SLAM Sonar							
Particles	1	2	3	4	5	MEAN	VAR
20	0.8	0.76	1.34	1.3	1.3	**1.1**	0.09
Fusion of Edge feature SLAM Laser and Grid SLAM Sonar							
Particles	1	2	3	4	5	MEAN	VAR
20	2.17	2.15	2.28	1.93	1.21	**1.95**	0.19

Table 6.3: Sensor Fusion evaluation results "Bicocca_2009-02-25b", ATE in [m]: Results of corner feature based SLAM combined with sonar grid SLAM. Edge feature based SLAM and sonar grid SLAM evaluated with the "Bicocca_2009-02-25b" dataset.



(a) (b)

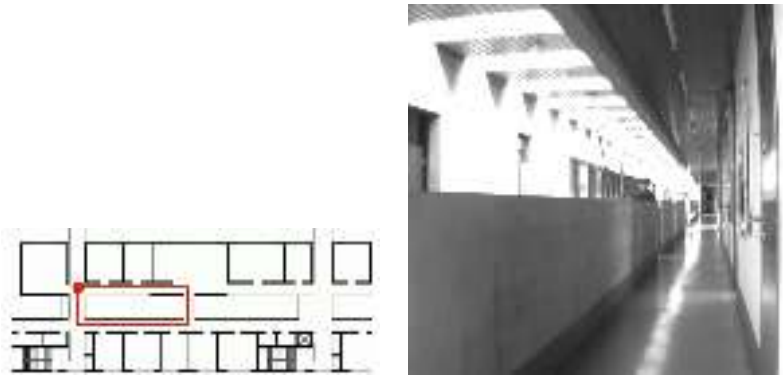
Figure 6.10: Sensor Fusion of Laser and Sonar evaluation results "Bicocca_2009-02-25b": (a) Corner feature based SLAM implementation, using laser readings in combination with a sonar grid SLAM method. The map was generated with 20 particles and results in an ATE of 0.8m. (b) Edge feature based SLAM combined with sonar SLAM leads to an ATE of 1.21m with this run. The red trajectories denote the odometry paths and the estimated robot paths are indicated by blue lines.

6.3 ICG Lab Indoor

This dataset is used to compare a time of flight (TOF) camera with a laser range finder and sonars and was recorded at Graz University of Technology. In general, the ICG dataset consists of four structures: Concrete, metal, glass and fences, which lead to different measurement errors, see Section 4. This chapter provides an evaluation of these effects. Furthermore, the distance accuracy of the sensor technologies is evaluated in an experiment. Finally, maps generated with all three sensor technologies are compared. For this experiment, a small loop is sufficient. Accurate results lead to rectangular maps and are evaluated in a qualitative way, because of the lack of ground truth measurements.

6.3.1 Dataset and Setup

The dataset was recorded at the institute for Computer Graphics and Vision ICG, Graz University of Technology, and is used to compare maps generated with a laser range finder, sonars and a time of flight camera, see Figure 6.12. The travelled trajectory in the office like environment is a small loop of about 40 metres, see Figure 6.11.
For the experiment, the two-wheeled *PeopleBot* from ActivMedia [1] was equipped with a Sick LMS200 laser range finder, 36 sonar transducers and a time of flight (TOF) camera from PMDTec (PMD[Vision]® O3). The laser range finder maximum range was set to 7 metres and the angular resolution to 0.5 degrees. The working range of the *PeopleBot* sonar belt is 0.2 metres to 4 metres, according to the specification sheet. Finally, the TOF camera has a resolution of 64x50 pixels and the maximum range was set to 5 metres. The laser range finder and the TOF camera are mounted in the viewing direction of the robot. Details about the sensor technology specific characteristics can be found in Section 4. The hardware was addressed by a robot software framework, which is described in Appendix B.



(a) (b)

Figure 6.11: ICG static indoor dataset: (a) illustrates the blue-print of the environment at Graz University of Technology. The red trajectory indicates the travelled path and was manually annotated. (b) Picture of the environment.

6.3.2 Laser Range Finder

The ICG dataset and a laser range finder were used to create maps with the *GMapping* method, and a grid SLAM implementation, see Figure 6.13. The odometry is denoted by

Figure 6.12: [Mobile robot used for the experiments at ICG: This mobile robot is equipped with two sonar belts, a Sick LMS 200 laser range finder and time-of-flight camera from PMDTec. The stereo camera was not used for this work. FOV means field of view.

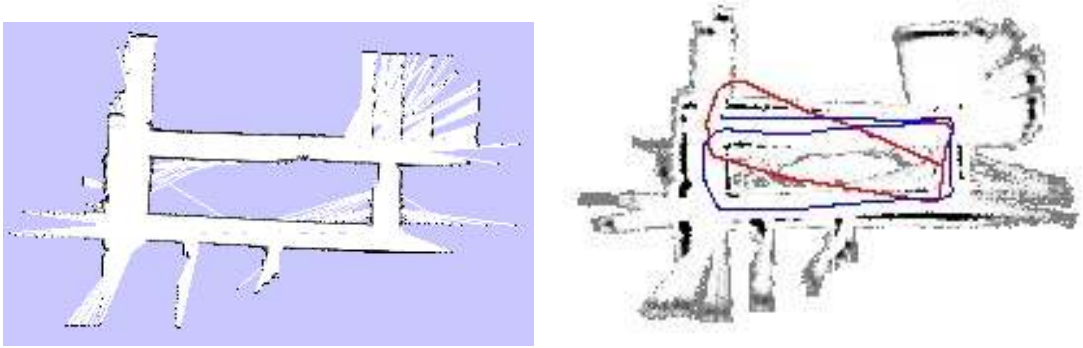
the red line, while the estimated trajectory is blue. The resulting maps of the feature based SLAM implementations are not present, because the geometrical features are hard to visualise. *GMapping* does not provide robot trajectories in the resulting map.

6.3.3 Sonar

The sonar sensors used are noisy, see Section 6.1 and therefore, the only SLAM method used for the evaluation, is a grid SLAM approach, see Figure 6.14. The resulting maps were generated with 100 particles.

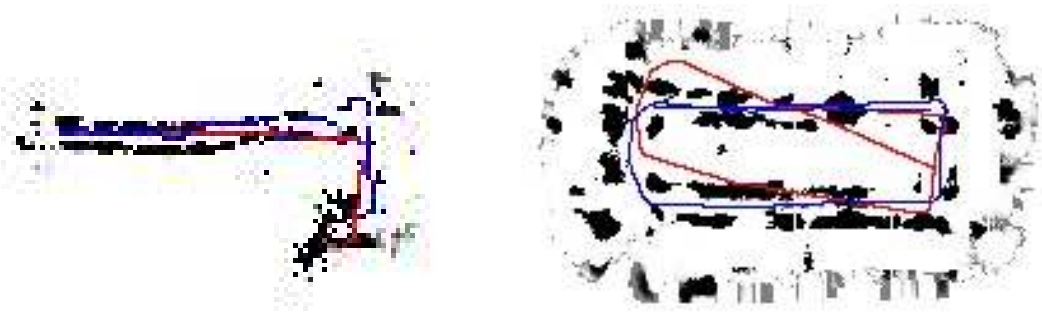
6.3.4 Time of Flight Camera

The ICG indoor dataset was evaluated with a TOF camera, mounted at front, see Figure 6.15. The maps were generated with 50 particles and a maximum range of 2.5 metres reduces the amount of noisy measurements. Because neither corners nor edges could be extracted from the TOF camera readings, only a grid SLAM method was used. The center line of the 3D information, observed by the TOF camera, was used.



(a) (b)

Figure 6.13: Laser Range Finder Evaluation on ICG indoor dataset: (a) GMapping result, obtained with 100 particles. (b) Grid SLAM map created with 50 particles. The red trajectory denote the odometry path and the estimated robot poses are indicated by blue points.

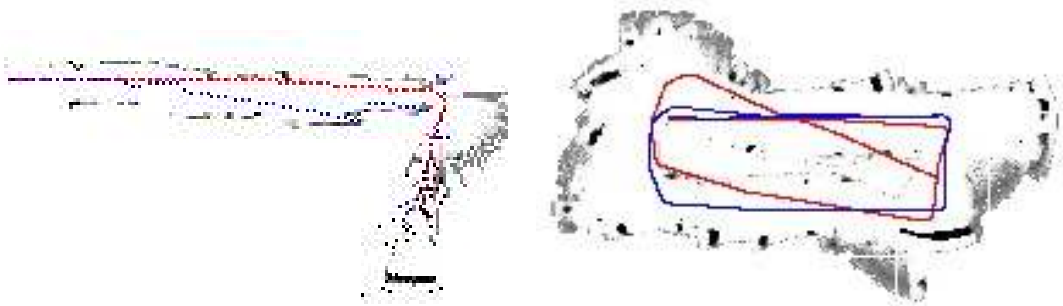


(a) (b)

Figure 6.14: Sonar Evaluation on ICG indoor dataset: (a) Sonar Grid SLAM result after a few iterations. (b) Final map, created with sonar grid SLAM, using 100 particles. The red trajectory denote the odometry path and the estimated robot poses are indicated by blue points.

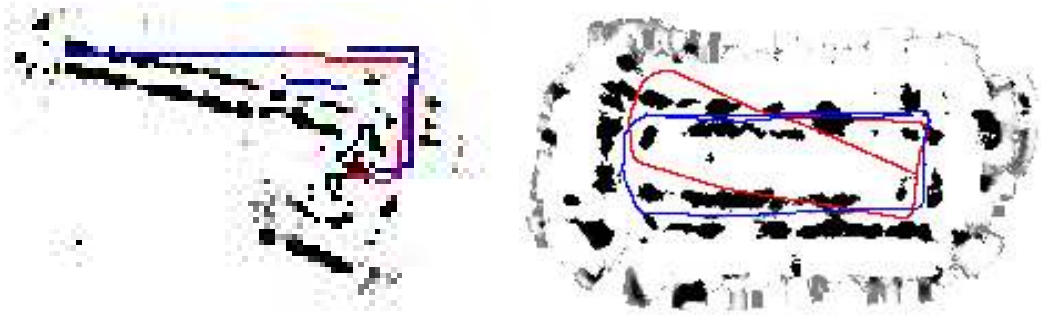
6.3.5 Sensor Fusion: Sonar and Time of Flight

The sensor fusion results of sonar grid SLAM and TOF camera grid SLAM were created with 50 particles, see Figure 6.16.



(a) (b)

Figure 6.15: TOF Evaluation on ICG indoor dataset: (a) TOF grid SLAM result after a few iterations. (b) Final TOF grid SLAM results, using 50 particles. The red trajectories denote the odometry paths and the estimated robot poses are indicated by blue points.



(a) (b)

Figure 6.16: Sonar and TOF Evaluation on ICG indoor dataset: (a) Sensor fusion of sonar and TOF Grid SLAM results after a few iterations. (b) Final result, using 50 particles. The red trajectories denote the odometry paths and the estimated robot poses are indicated by blue points.

Chapter 7

Summary and Outlook

Contents

Within this thesis, it was discovered, which sensor technology suits well for common state of the art SLAM methods. Sensor technologies could be combined to enhance the map accuracy. For an easy sensor comparison, all evaluated devices were distance sensors: A laser range finder, sonars and a time of flight (TOF) camera.
Sensor fusion is desired to build more robust and more accurate methods. The proposed sensor fusion framework relies on separate particle weight updates for each sensor technology. It was shown that robust SLAM approaches can be build with *GMapping* or feature based mapping approaches using an accurate laser range finder. In contrast, the evaluated sonar sensors and TOF measurements are too sparse and noisy to build accurate maps. The fusion of a laser range finder and sonar sensors did not improve either the mapping nor the localisation accuracy. The environmental structures could be represented proper by the laser scanner alone.
In Section 3 basic filter concepts were reviewed. The strengths and weaknesses of the discussed filters with respect to the SLAM problem are highlighted in the following. Furthermore, two motion models, namely the *Particle Motion Model* and the *Gaussian Probabilistic Motion Model* are reviewed. The results of the experiments are discussed and improvements are proposed. This chapter closes with an outlook.

7.1 Comparison of Recursive Filters

In the theoretical part of this work, Section 3, several filters have been reviewed. In this chapter the strengths and weaknesses of the discussed filters are highlighted. Tables 7.1 and 7.2 summarise the features of the discussed algorithms.

Algorithm	State Probability	State Noise	Transition Model
Kalman Filter	unimodal Gaussian	Gaussian	linear
Extended Kalman Filter	unimodal Gaussian	Gaussian	non-linear
Particle Filter	arbitrary multimodal	white	non-linear
Rao-Blackwellised Particle Filter	arbitrary multimodal	white	non-linear

Table 7.1: Filter algorithm comparison. Properties.

The *Kalman Filter* is commonly used for linear, low dimensional tasks and is often the first choice because of the computational efficiency. Large maps with many features lead to huge state matrices and the filter becomes impractical. Non-linear state transitions are possible with the *Extended Kalman Filter*.

Some applications need to track several objects simultaneously or follow multiple hypotheses. This filter nature is achieved by the *Particle Filter*. The *Particle Filter* is more robust against wrong data associations, because of its probabilistic resampling step.
Finally, the *Rao-Blackwellised Particle Filter* uses simple *Extended Kalman Filters* for the feature estimates and a robust *Particle Filter* models the robot state. This leads to an efficient SLAM implementation which scales logarithmically with the number of features.

7.2 Comparison of Motion Models

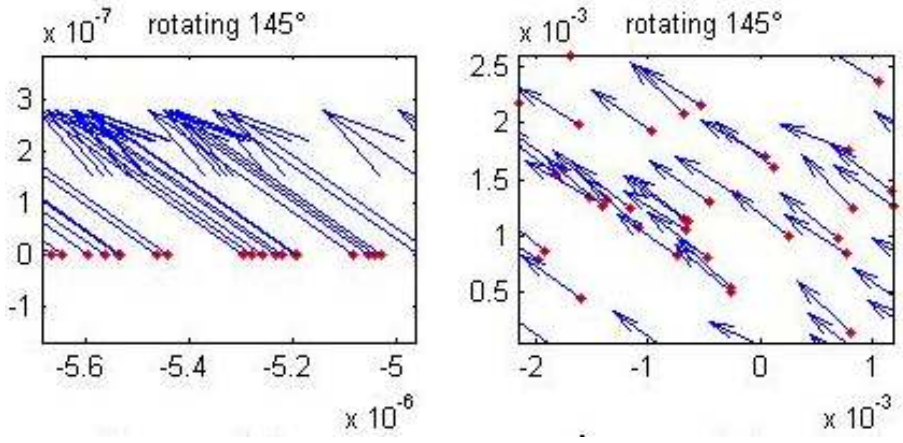
In Section 3 two motion models, namely the *Particle Motion Model* and the *Gaussian Probabilistic Motion Model* were reviewed. This chapter discusses the differences.
It can be said that both models are equivalent, except for a robot motion consisting of a single rotation. In this case, the translational part is zero.

Algorithm	Comment
Kalman Filter	Computational time increases quadratically with the number of features. Mostly not practical to use this filter for SLAM, because of non-linear robot motions.
Extended Kalman Filter	Computational time increases quadratically with the number of features, but the filter can model non-linear state transitions. Fails if the non-linearities are too strong.
Particle Filter	Computational time is proportional to the number of particles. Difficult to find a proper number of particles and the resampling process is crucial.
Rao-Blackwellised Particle Filter	The computational time of efficient implementations of the *FastSLAM* algorithm scales logarithmic with the number of features. The resampling process is crucial.

Table 7.2: Filter algorithm comparison. Comments.

With the *Particle Motion Model*, only the heading angle is affected by Gaussian noise. The particle positions (x, y) stay the same, see Figure 7.1 (a). Note that some minimum noise is added to create the plot.
With the *Gaussian Probabilistic Motion Model*, the robot positions are affected by Gaussian noise, see Figure 7.1 (b). This behaviour is requested to model real robot kinematics. A rotation often results in a small shift of the robot center.

For both motion models it is challenging to find proper noise control parameters (α_1, α_2, α_3 and α_4). The robot kinematic uncertainties depend not only on the robot architecture and motion sensing technology (incremental sensor, IMU ...), but also on environmental conditions. A bumpy or slippery ground floor leads to completely different parameter values. Additionally, the amount of added noise has a direct impact on the variance of the particle weights. One strategy to avoid too frequent resampling, is based on variance of the particle weights, see Section 3.4. Therefore, different noise control parameters result in different filter behaviour.



(a) (b)

Figure 7.1: Example Transition, Rotation by 145 degrees: (a) The *Particle Motion Model* does not implement a drift and all particle positions would stay the same after the rotation. (b) In contrast, with the *Gaussian Probabilistic Motion Model* the robot positions are affected by Gaussian noise, see Figure 7.1 (b).

7.3 Sensors and Sensor Fusion

The sensor distance accuracy was evaluated with a laser range finder as ground truth and scene specific sensor problems, like reflections on metal door were considered in Section 6.1. In the static experiment, the evaluated sonars had a maximum distance error of about 20 cm, whereas the TOF camera had a maximum error of 5.34 cm. Different robot heading angles or a dynamic experiment, where the measurements are made while the robot is moving, would lead to bigger measurement errors. The maximum accuracy is given by the laser range finder, which is used as reference distance sensor.
The evaluated TOF camera is sensitive to sun light and to reflections on metal and glass surfaces, see Figure D.2, Figure D.3 and Figure D.4 . The evaluated sonar sensors of our mobile robot are sensitive to the angle of inclination, see Figure D.2. Only the sonar transducers, orthogonal to obstacles, a concrete wall, a metal fence etc., provide useful measurements, see Figure D.4. Laser beams are reflected by window panes and lead to corrupted measurements, see Figure D.4. A combination of different sensor technologies should balance the weaknesses.

The laser range finder is the most accurate sensor evaluated in this thesis.

The *GMapping*, the corner feature SLAM implementation and the edge feature SLAM implementations lead to accurate trajectories with 100 particles or more, see Table 6.2. Within the *RawSeeds* dataset, the robot made 826m in 26 minutes. The fastest method was the corner based SLAM approach, which runs in realtime.
Sonar grid SLAM did not give good results with the *RawSeeds* dataset. A combination of sonar readings and laser measurements did not reduce the absolute trajectory error (ATE), but the estimated trajectory was smoother, see Figure 6.10. The amount of completely wrong positioned particles was reduced by the sensor fusion. The map, created by the laser range finder, could not be improved by the noisy and sparse sonar readings.

The evaluation of the ICG indoor dataset showed good quality maps with a laser range finder, see Figure 6.13. The laser range finder was robust against reflections on concrete walls, metal doors and metal fences.
Sonar grid SLAM and TOF camera grid SLAM could not cope with the big systematic drift of the odometry trajectory and the environmental difficulties, see Figure 6.15.
A fusion of sonars and TOF camera measurements did not improve the map quality with the evaluated sensors. Sonar and TOF camera readings were only evaluated with a grid SLAM method because no geometrical features could be extracted from the measurements, see Figure 6.16. Either, more sophisticated SLAM methods are necessary to correct the robot trajectory with the sparse and noisy sensor readings or more accurate sensors should be used.

7.4 Outlook

More sophisticated sonars, a TOF camera with a higher resolution or multiple TOF cameras used simultaneously could be a launch for further investigations. Multiple simultaneously operating TOF cameras are still an open issue, because the infrared beams must not interfere with each other. In order to solve this problem, different modulation frequencies of the LED sources could be used. Another solution could be multiplexing, where only one camera is active.
Other sensor technologies can be easily integrated into the developed sensor fusion framework. In the presented experiments the integrated sensors are of the same importance. An adaptive weighting strategy could improve the result. If one is able to detect misleading observations, like reflected TOF camera rays at metal doors, they could be neglected and

only the reliable measurements would be used to correct the robot pose.
Most of the experiments were performed with 100 particles. More particles, representing the robot pose, would increase the accuracy of the estimated trajectory by the cost of computational time. *Particle Filters* are easy to parallelise and a SLAM implementation for the graphics processing unit (GPU) is obvious.
The performance of the feature based SLAM implementations can be enhanced by the use of a more sophisticated data association strategy. The implemented method is a simple nearest neighbour approach. Neira et al. [39] proposes a joint compatibility measure to consider the correlations between map features.
In the experiments, only one line of the 3D information, observed by the TOF camera, was used. A higher resolution would be necessary to extract geometric features, which could be used for SLAM with a feature based approach. For a 3D grid SLAM approach, a more sophisticated memory strategy would be necessary.

Appendix A

Acronyms and Symbols

List of Acronyms

CT	Computer Tomography
EM	Expectation Maximisation
FOV	Field of View
GPS	Global Positioning System
GT	Ground Truth
IMU	Inertial Measurement Unit
KF	Kalman Filter
LASER	Light Amplification by Stimulated Emission of Radiation
LRF	Laser Range Finder
MR	Magnetic Resonance
PF	Particle Filter
SLAM	Simultaneous Localisation and Mapping
SONAR	Sound Navigation and Ranging

Appendix B

How to Write a Player Driver

The Player Project [23] is a popular robot software framework, which handles the communication between the robot hardware and the software clients. It was used to record the data during the experiments and is a client server architecture with a TCP socket based communication. The framework makes interaction between different software modules, like a path planer and an obstacle detector, easy and flexible. The clients can be implemented in any programming language and the only drawback is the TCP socket based communication bandwidth. Therefore, big video streams may be transmitted via different communication methods, like firewire (IEEE 1394).
The hardware is addressed with a driver and many drivers are already implemented for common robot hardware. For example, the *PeopleBot* motion control hardware and the Sick LMS laser range finder are ready to use directly after the framework installation. For more information about the *The Player Project*, see [23].
To proceed the experiments, only a driver for the time of flight camera from PMDTec [24] was developed.
This chapter is a simple guideline, how to implement a hardware driver. The driver methods are not considered in detail, because there exist already good descriptions, see the *HowTo* from Petersen et al. [43]. This chapter provides source code templates, ready to use after a few modifications.

In the most simple case, a driver consists of three files: The main source code file, a Makefile for the compilation and finally to run the driver, a configuration file, which provides the runtime parameters. A similar example can be found in the player installation folder, called *exampledriver*, but the provided template is easier to adapt.

In the following, three template listings for a complete driver implementation are provided. This driver provides an audio interface.

Driver Source Code

```
/*
 *  Player - One Hell of a Robot Server
 *  Copyright (C) 2003
 *     Brian Gerkey
 *
 *
 *  This program is free software; you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation; either version 2 of the License, or
 *  (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License
 *  along with this program; if not, write to the Free Software
 *  Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 *
 */

/*
 * A simple example of how to write a driver that will be built as a
 * shared object.
 */

#linux config
driver
(
```

```
  name "TemplateDriver"
  plugin "libTemplateDriver"
  provides ["audio:0"]
)

// ONLY if you need something that was #define'd as a result of configure
// (e.g., HAVE_CFMAKERAW), then #include <config.h>, like so:
/*
#if HAVE_CONFIG_H
  #include <config.h>
#endif
*/

#include <unistd.h>
#include <string.h>
#include <stdio.h>
#include <math.h>

#include <libplayercore/playercore.h>
#include <libplayerxdr/playerxdr.h>

////////////////////////////////////////////////////////////////////////////
// The class for the driver
class DriverTemplate : public Driver
{
  public:

    // Constructor; need that
    DriverTemplate(ConfigFile* cf, int section);

    // Must implement the following methods.
    virtual int Setup();
    virtual int Shutdown();
```

```
    // This method will be invoked on each incoming message
    virtual int ProcessMessage(QueuePointer &resp_queue,
                               player_msghdr * hdr,
                               void * data);

  private:

    // Main function for device thread.
    virtual void Main();

    player_devaddr_t source_audio_adr0;

    float *data0;

};

// A factory creation function, declared outside of the class so that it
// can be invoked without any object context (alternatively, you can
// declare it static in the class).  In this function, we create and return
// (as a generic Driver*) a pointer to a new instance of this driver.
Driver*
DriverTemplate_Init(ConfigFile* cf, int section)
{
  // Create and return a new instance of this driver
  return((Driver*)(new DriverTemplate(cf, section)));
}

// A driver registration function, again declared outside of the class so
// that it can be invoked without object context.  In this function, we add
// the driver into the given driver table, indicating which interface the
// driver can support and how to create a driver instance.
void DriverTemplate_Register(DriverTable* table)
{
  table->AddDriver("DriverTemplate", DriverTemplate_Init);
```

```
}

////////////////////////////////////////////////////////////////////////////
// Constructor.  Retrieve options from the configuration file and do any
// pre-Setup() setup.
DriverTemplate::DriverTemplate(ConfigFile* cf, int section)
    : Driver(cf, section, false, PLAYER_MSGQUEUE_DEFAULT_MAXLEN,
             PLAYER_AUDIO_CODE)
{
  //request memory
  memset (&this->source_audio_adr0, 0, sizeof (player_devaddr_t));
  data0 = NULL;

  //no interface to add, because this driver provides only one interface

  return;
}

////////////////////////////////////////////////////////////////////////////
// Set up the device.  Return 0 if things go well, and -1 otherwise.
int DriverTemplate::Setup()
{
  puts("Example driver initialising");

  // Here you do whatever is necessary to setup the device, like open and
  // configure a serial port.

  //Allocate memory
  data0 = new float[100];

  // Start the device thread; spawns a new thread and executes
  // DriverTemplate::Main(), which contains the main loop for the driver.
  StartThread();
```

```
  return(0);
}


////////////////////////////////////////////////////////////////////////////
// Shutdown the device
int DriverTemplate::Shutdown()
{
  puts("Shutting example driver down");

  // Stop and join the driver thread
  StopThread();

  // Here you would shut the device down by, for example, closing a
  // serial port.

 //free memory
  if(data0 != NULL)
    delete[] data0;

  puts("Example driver has been shutdown");

  return(0);
}

int DriverTemplate::ProcessMessage(QueuePointer & resp_queue,
                                   player_msghdr * hdr,
                                   void * data)
{
  // Process messages here.  Send a response if necessary, using Publish().
  // If you handle the message successfully, return 0.  Otherwise,
  // return -1, and a NACK will be sent for you, if a response is required.
```

```
  assert(hdr);

  if(Message::MatchMessage(hdr, PLAYER_MSGTYPE_DATA,
     PLAYER_AUDIO_DATA_WAV_REC, source_audio_adr0))
  {
    player_audio_wav_t *waveData = reinterpret_cast<player_audio_wav_t *> (data);
    memcpy(data0, waveData->data, waveData->data_count);
  }


  return(0);
}



////////////////////////////////////////////////////////////////////////////
// Main function for device thread
void DriverTemplate::Main()
{
  pthread_setcanceltype(PTHREAD_CANCEL_DEFERRED,NULL);

  while (1)
  {
    ProcessMessages();
    pthread_testcancel();

    pthread_testcancel();
  }
  pthread_exit(NULL);
}

////////////////////////////////////////////////////////////////////////////
// Extra stuff for building a shared object.
```

```
/* need the extern to avoid C++ name-mangling  */
extern "C" {
  int player_driver_init(DriverTable* table)
  {
    puts("Template driver initializing");
    DriverTemplate_Register(table);
    puts("Template driver done");
    return(0);
  }
}
```

Makefile

```
LIBPATH = lib
INCPATH = inc

CFLAGS = -Dlinux32 -Wall -fpic -g3 -I$(INCPATH) `pkg-config --cflags playercore`
LDFLAGS += -Wl,-rpath,$(LIBPATH) -L$(LIBPATH) -lpthread -ldl
CC = g++

OBJ = MCSoundV1.o


TARGET = libDriverTemplate.so

all: $(TARGET)

%.o: %.cc
$(CC) $(CFLAGS) -c $? -o $@

$(TARGET): $(OBJ)
$(CC) $? -shared -o $@ $(LDFLAGS)

clean:
rm -f *.o *.so.0
```

Configuration File

```
driver
(
  name "DriverTemplate}"
  plugin "libDriverTemplate}"
  provides ["audio:0"}]
)
```

Conclusion

For a specific implementation, only a few modifications are necessary. First replace the class name *DriverTemplate* and than modify the publishing method *ProcessMessage*. Finally, the configuration file needs to be altered according to the provided interfaces, defined in the *ProcessMessage*.

Because of the special character of player drivers, one needs to distinguish between drivers with a single interface and drivers with multiple interfaces. The source code listing, provided before considers the first case, where the player server is responsible for the interface registration and the constructor calls a specialised base constructor.
If multiple interfaces are implemented, a registration of each interface is necessary in the constructor section of the source code file. The following constructor code listing describes the modifications for a driver, which provides a laser and a camera interface. Note that the base constructor call in this example is simply *Driver(cf, section)*.

```
PMDV3::PMDV3(ConfigFile* cf, int section)
    : Driver(cf, section)
{
  //request emory
  memset (&this->i_cam_addr,  0, sizeof (player_devaddr_t));
  memset (&this->laser_addr,  0, sizeof (player_devaddr_t));

 provideLaser = FALSE; provideICam = FALSE;

// Outgoing laser interface
```

```
  if (cf->ReadDeviceAddr(&(this->laser_addr), section, "provides",
      PLAYER_LASER_CODE, -1, NULL) == 0)
  {
    if (this->AddInterface (this->laser_addr) != 0)
    {
      this->SetError (-1);
      return;
    }
    provideLaser = TRUE;
  }

  // Outgoing intensity::camera:0 interface
  if (cf->ReadDeviceAddr (&(this->i_cam_addr), section, "provides",
      PLAYER_CAMERA_CODE, -1, "intensity") == 0)
  {
    if (this->AddInterface (this->i_cam_addr) != 0)
    {
      this->SetError (-1);
      return;
    }
    provideICam = TRUE;
  }

  return;
}
```

The development of player drivers is very intuitive and easy to understand. With the driver *HowTo* from Petersen et al. [43] and the provided source listings, the reader is able to develop new hardware drivers at minimum costs.

Appendix C

How to use Realtime Player Data within Matlab

This chapter describes a library, permitting a communication between *Matlab*, a common scientific programming tool, and the player server. The developer enjoys the advantages of *Matlab*, like complex plotting procedures or easy signal processing, and has access the many player client modules in realtime.
Thanks to Kevin Barry [5], who developed the *Matlab* client library, called *PlayerMex*.

The latest version of *PlayerMex* can be downloaded from [5]. The reader has the choice between pre compiled linux and windows versions, which is often sufficient. If an interface is not implemented, the source code version of *PlayerMex* is the proper choice. A new interface implementation would affect the file *playermex_generated.c*, but this modification is not explained in this work.

In the following, a simple matlab code example explains the basic communication process. Note that the *PlayerMex* library folder is added to the *Matlab* path variable.

Matlab Player Communication

The following example demonstrates a communication between *Matlab* and a player client, implementing a *position2d* interface (*PeopleBot* driver for instance).

```
%% init player connection
```

```
try, player('destroy', pos2d); end;
try, player('client_disconnect', client); end;
try, player('destroy', client); end;

% Now we can create a client to connect to the player server (192.168.0.1)
client = player('client_create', '192.168.0.1', 6665);
if (player('client_connect', client))
    error 'Could not connect to player';
end
%subscribe
pos2d = player('position2d_create', client, 0);
player('subscribe', pos2d);
%enable motors
status = player('position2d_enable', pos2d, 1);
%loop
while quit_flag == 0

        %read data from player server
        player('client_read_raw', client);

        if (player('isfresh', pos2d))
         %display robot pose
         [pos2d.px pos2d.py pos2d.pa]
        end
end

% Clean up
try
    player('unsubscribe', pos2d);
    player('destroy', pos2d);

    player('client_disconnect', client);
    player('destroy', client);
catch
```

Appendix D

ICG Environment Characteristics

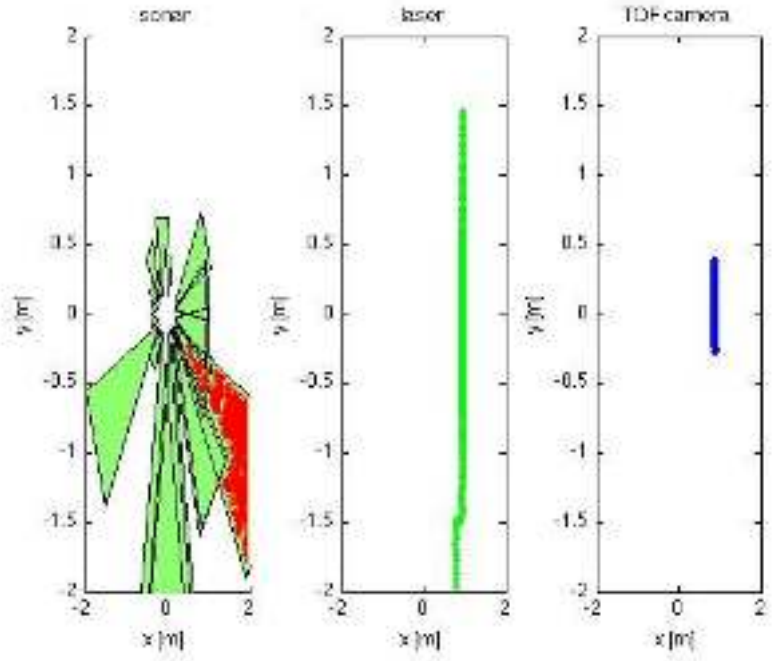


Figure D.1: ICG static indoor dataset, Concrete Walls: Front view of a concrete wall. Sonar, laser and TOF camera scans.

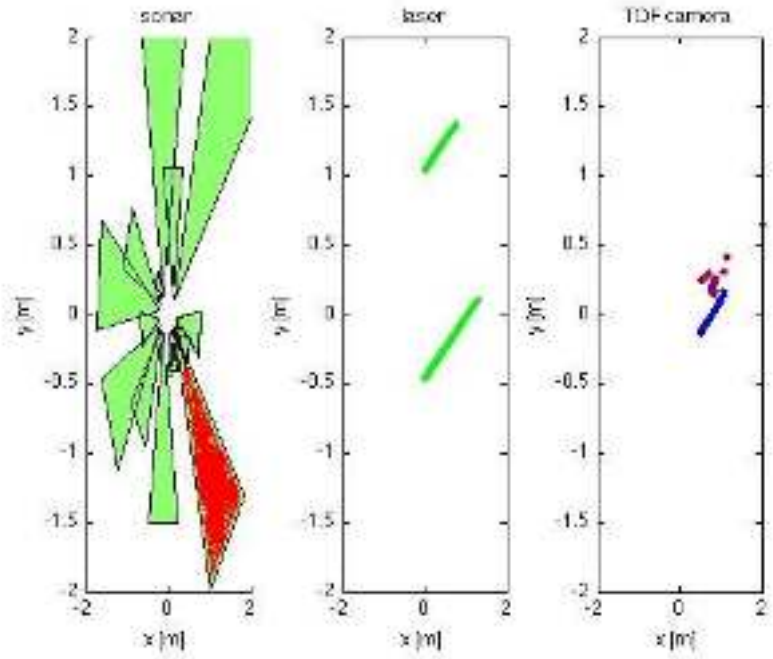
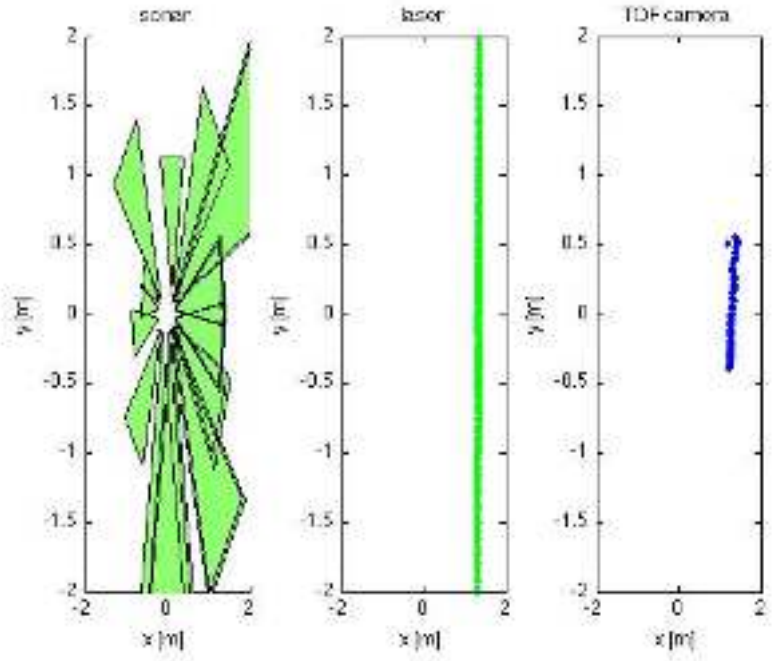
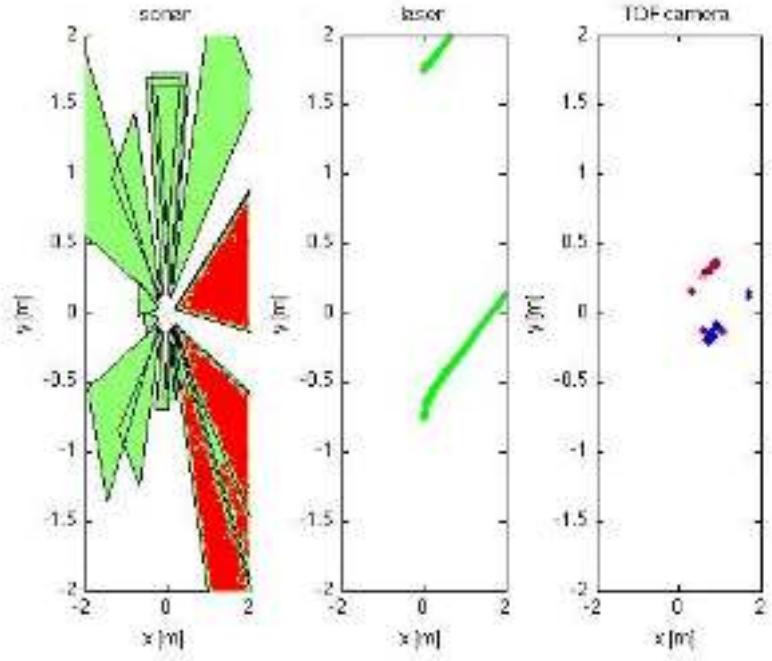


Figure D.2: ICG static indoor dataset, Concrete Walls: Side view of a concrete wall. Sonar, laser and TOF camera scans. Red labelled structures denote wrong measurements. Some TOF measurements are wrong because of sun light reflections.

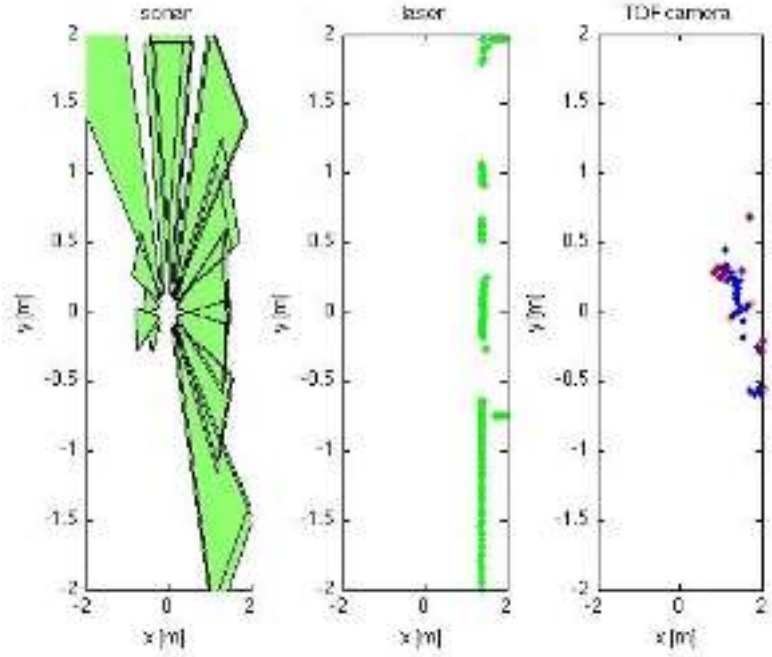


(a)

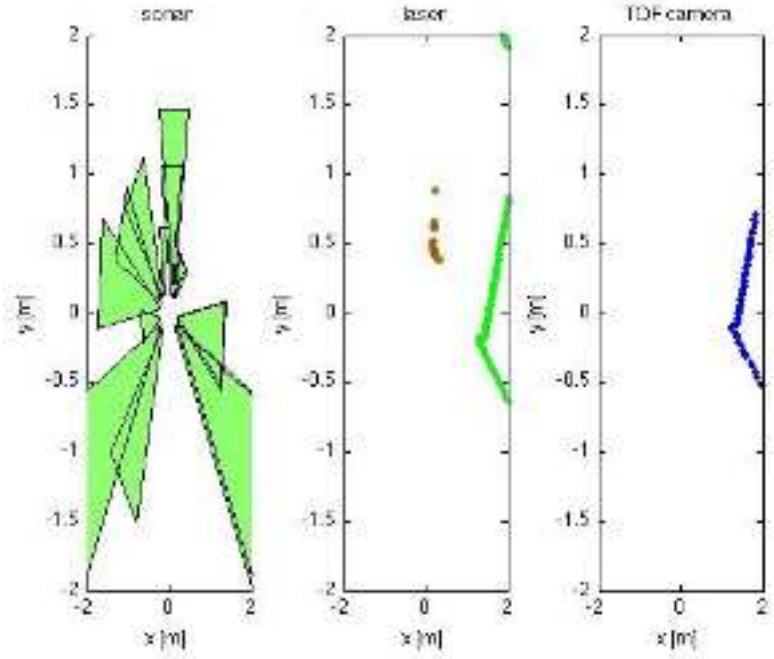


(b)

Figure D.3: ICG static indoor dataset, Metal Doors: (a and b) Front and side view of metal doors. Each image with sonar, laser and TOF camera scans. Metal doors lead to reflections, which causes error prone distance measurements of the sonar transducers and the TOF camera.

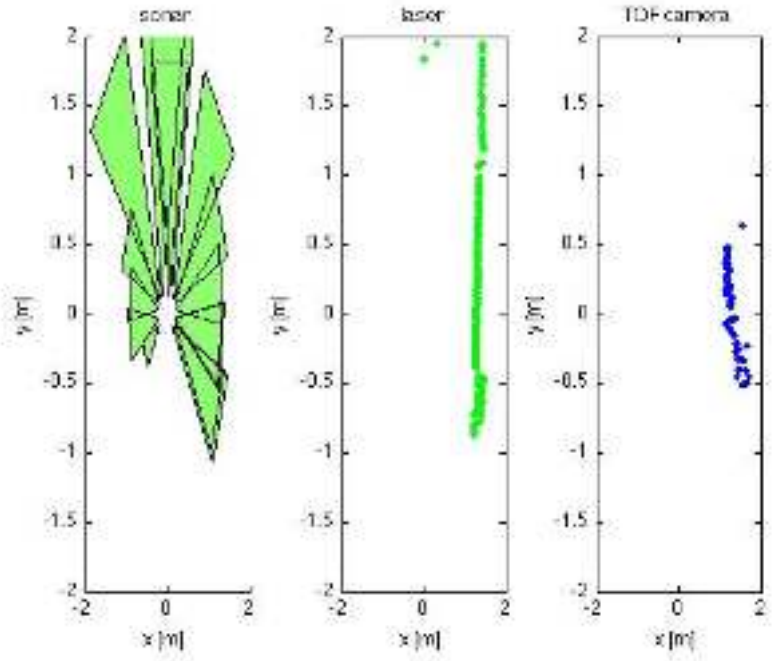


(a)

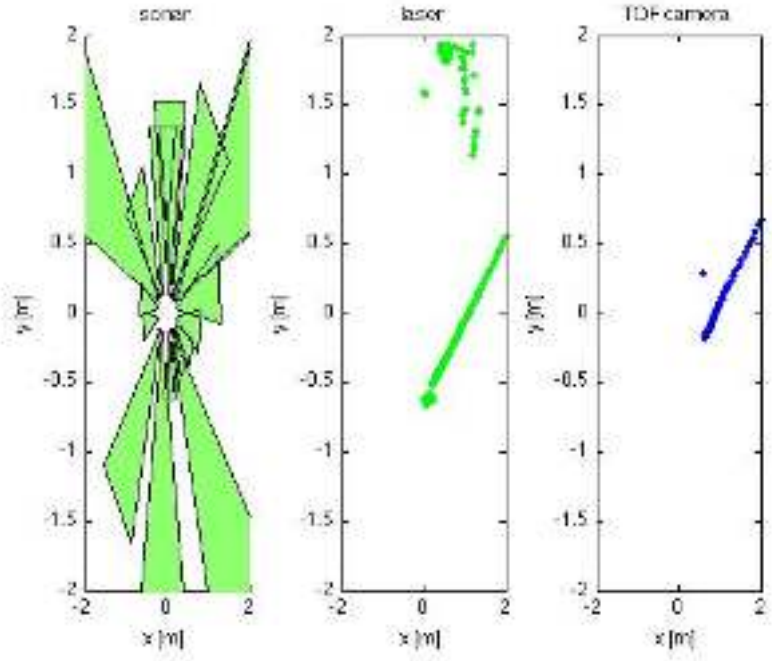


(b)

Figure D.4: ICG static indoor dataset, Window Panes: (a and b) Front and side view of a window pane. Each image with sonar, laser and TOF camera scans. Laser readings and the TOF measurements are corrupted.



(a)



(b)

Figure D.5: ICG static indoor dataset, Metal Fences: (a and b) Front and side view of a metal fence. Each image with sonar, laser and TOF camera scans. Present imprecisions can be lead back to the sensor noise.

Bibliography

[1] ActivMedia (2009). Mobile robots inc. `http://www.activmedia.com`. Visited on december 6th 2009.

[2] Altermatt, M., Martinelli, A., Tomatis, N., and Siegwart, R. (2004). SLAM with Corner Features Based on a Relative Map. In *None*.

[3] Ayache, N. and Faugeras, O. (1988). Building, registrating, and fusing noisy visual maps. *Int. J. Rob. Res.*, 7(6):45–65.

[4] Balaguer, B., Carpin, S., and Balakirsky, S. (2007). Towards quantitative comparisons of robot algorithms: Experiences with slam in simulation and real world systems. In *Workshop on "Performance Evaluation and Benchmarking for Intelligent Robots and Systems" at IEEE/RSJ IROS*.

[5] Barry, K. (2009). Playermex. `http://barryk.googlepages.com/playermex`. Visited on november 14th 2009.

[6] Burgard, W., Ruhnke, M., Stachniss, C., and Grisetti, G. (2001). The rawseeds project. `http://www.rawseeds.org/home/`. Visited on july 28th 2009.

[7] Castellanos, J. A. and Tardos, J. D. (2000). *Mobile Robot Localization and Map Building: A Multisensor Fusion Approach*. Kluwer Academic Publishers, Norwell, MA, USA.

[8] Chen, Z. (2003). Bayesian filtering: From kalman filters to particle filters, and beyond. Technical report, McMaster University.

[9] Choi, Y.-H., Lee, T.-K., and Oh, S.-Y. (2008). A line feature based slam with low grade range sensors using geometric constraints and active exploration for mobile robot. *Auton. Robots*, 24(1):13–27.

[10] Crowley, J. (1989). World modeling and position estimation for a mobile robot usig ultrasonic ranging.

[11] Cyrill Stachniss, Udo Frese, G. G. (2001). Openslam. `http://www.openslam.org/`. Visited on july 28th 2009.

[12] Defense Advanced Research Projects Agency (2009). The DARPA Urban Challenge. `http://www.darpa.mil/grandchallenge/index.asp`. Visited on july 28th 2009.

[13] Diosi, A. and Kleeman, L. (2004). Advanced sonar and laser range finder fusion for simultaneous localization and mapping. In *Proceedings of 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*.

[14] Douc, R. and Cappe, O. (2005). Comparison of resampling schemes for particle filtering. In *Image and Signal Processing and Analysis, 2005. ISPA 2005. Proceedings of the 4th International Symposium on*, pages 64–69.

[15] Durrant-Whyte, H. (1988). On uncertain geometry in robotics. 4(1).

[16] Durrant-Whyte, H. and Bailey, T. (2006). T.: Simultaneous localisation and mapping (slam): Part i the essential algorithms. robotics and automation magazine. *IEEE Robotics and Automation Magazine*, 2:2006.

[17] Eliazar, A. and Parr, R. (2003). Dp-slam: Fast, robust simultaneous localization and mapping without predetermined landmarks.

[18] Eliazar, A. and Parr, R. (2004a). Dp-slam 2.0. volume 2, pages 1314–1320 Vol.2.

[19] Eliazar, A. I. and Parr, R. (2004b). Learning probabilistic motion models for mobile robots. In *ICML '04: Proceedings of the twenty-first international conference on Machine learning*, page 32, New York, NY, USA. ACM.

[20] Everett, H. R. (1995). *Sensors for mobile robots: theory and application*. A. K. Peters, Ltd., Natick, MA, USA.

[21] FraunhoferIPA (2009). Fraunhofer institute for manufacturing engineering and automation ipa. `http://www.ipa.fraunhofer.de`.

[22] Fulgenzi, C., Ippoliti, G., and Longhi, S. (2009). Experimental validation of FastSLAM algorithm integrated with a linear features based map. *Mechatronics*.

[23] Gerkey, B., Howard, A., Koenig, N., and Vaughan, R. (2009). The player project. `http://playerstage.sourceforge.net/`. Visited on november 14th 2009.

[24] GmbH, P. (2009). Pmdtechnologies gmbh. `http://www.pmdtec.com/`. Visited on august 8th 2009.

[25] Gokturk, S. B., Yalcin, H., and Bamji, C. (2004). A time-of-flight depth sensor - system description, issues and solutions. In *CVPRW '04: Proceedings of the 2004 Conference on Computer Vision and Pattern Recognition Workshop (CVPRW'04) Volume 3*, page 35, Washington, DC, USA. IEEE Computer Society.

[26] Gonzalez, J., Ollero, A., and Reina, A. (1994). Map building for a mobile robot equipped with a 2d laser rangefinder. In *Proc. IEEE International Conference on Robotics and Automation*, pages 1904–1909.

[27] Gordon, N. J., Salmond, D. J., and Smith, A. F. M. (1993). Novel approach to nonlinear/non-gaussian bayesian state estimation. *Radar and Signal Processing, IEE Proceedings F*, 140(2):pages 107–113.

[28] Grewal, M. S. and Andrews, A. P. (1993). *Kalman filtering: theory and practice*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[29] Grisetti, G., Stachniss, C., and Burgard, W. (2007). Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE Transactions on Robotics*, 23:2007.

[30] Hähnel, D., Burgard, W., Fox, D., and Thrun, S. (2003a). An efficient fastslam algorithm for generating maps of large-scale cyclic environments fram raw laser range measurements. In *In Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 206–211.

[31] Hähnel, D., Burgard, W., Fox, D., and Thrun, S. (2003b). An efficient fastslam algorithm for generating maps of large-scale cyclic environments from raw laser range measurements. In *In Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 206–211.

[32] Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Transactions of the ASME Journal of Basic Engineering*, (82 (Series D)):35–45.

[33] Leonard, J. J. and Durrant-Whyte, H. F. (1992). *Directed Sonar Sensing for Mobile Robot Navigation*. Kluwer Academic Publishers, Norwell, MA, USA.

[34] Leonard, J. J. and Whyte, D. H. (1991). Simultaneous map building and localization for an autonomous mobile robot. In *IEEE International Conference on Intelligent Robot Systems, Osaka, Japan*.

[35] Makarenko, A. and Durrant-Whyte, H. (2004). Decentralized data fusion and control in active sensor networks. In *In Proceedings of the Seventh International Conference on Information Fusion*.

[36] Montemerlo, M. and Thrun, S. (2003). Simultaneous localization and mapping with unknown data association using fastslam.

[37] Moravec, H. (1988). Sensor fusion in certainty grids for mobile robots. *AI Mag.*, 9(2):pages 61–74.

[38] Murphy, K. P. (2000). Bayesian map learning in dynamic environments. In *In Neural Info. Proc. Systems (NIPS)*, pages 1015–1021. MIT Press.

[39] Neira, J. and Tards, J. D. (2001). Data association in stochastic mapping using the joint compatibility test.

[40] Nguyen, V., Harati, A., Martinelli, A., Siegwart, R., and Tomatis, N. (2006). Orthogonal slam: a step toward lightweight indoor autonomous navigation. pages 5007–5012.

[41] Nguyen, V., Harati, A., and Siegwart, R. (2007). A lightweight slam algorithm using orthogonal planes for indoor mobile robotics. pages 658–663.

[42] Pandey, A. K., Krishna, K. M., and Hexmoor, H. (2007). Feature chain based occupancy grid slam for robots equipped with sonar sensors. pages 283–288.

[43] Petersen, B. and Fonseca, J. (2009). Writing player/stage drivers. `http://image.diku.dk/mediawiki/images/e/e2/Driverhowtodoc_HandedIn.pdf`. Visited on november 14th 2009.

[44] Rosencrantz, M., Gordon, G., and Thrun, S. (2003). Decentralized sensor fusion with distributed particle filters. In *In Proc. of UAI*.

[45] Schröter, C. and Gross, H.-M. (2007). Efficient gridmaps for slam with rao-blackwellized particle filters.

[46] Siegward, R. (2009). Perception - sensors. `http://www.asl.ethz.ch/education/master/mobile_robotics/4a_-_Perception_-_Sensors.pdf`. Visited on december 5th 2009.

[47] Smith, R. and Cheeseman, P. (1986). On the representation and estimation of spatial uncertainty.

[48] Sukhatme, G. S. (2009). Usc robotics research lab. `http://www-robotics.usc.edu/~gaurav/`. Visited on december 6th 2009.

[49] Thrun, S., Burgard, W., and Fox, D. (2005). *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press.

[50] Vandorpe, J., Van Brussel, H., and Xu, H. (1996). Exact dynamic map building for a mobile robot using geometrical primitives produced by a 2d range finder. In *Proc. IEEE International Conference on Robotics and Automation*, volume 1, pages 901–908.

[51] Welch, G. and Bishop, G. (1995). An introduction to the kalman filter. Technical report, Chapel Hill, NC, USA.

[52] Xavier, J., Pacheco, M., Castro, D., and Ruano, A. (2005). Fast line, arc/circle and leg detection from laser scan data in a player driver. In *In 2005 IEEE International Conference on Robotics and Automation, Barcelona, Spain*, pages 3930–3935.

[53] Yang, S.-W. and Wang, C.-C. (2008). Dealing with laser scanner failure: Mirrors and windows. In *IEEE International Conference on Robotics and Automation (ICRA)*, Pasadena, California.