An experimental comparison of SLAM methods

David de Bos

December 20, 2011

Imagine you are in the jungle of Mexico. The goal of your journey is to find the long searched for tomb of Quetzalcoatl. After weeks of chopping your way with a machette through the dense forest you suddenly see the entrance of a cave partly covered by a golden gate. That's it, you have found it! But now comes the most tricky part: these Aztecian caves are known for their complicated structure. Only with utterly precise mapping techniques you will be able to enter the cave and be sure to also come out again. So once you set foot into the cave you carefully measure the length and direction of every step you take and write it down. After each step you draw the walls around you on your paper with black lines and mark your position relative to the walls with a red cross. In this way you will always be able to know where you are nd, once you come out again, you will have a map of the cave with the trajectory you have walked.

This process of localizing oneself and mapping the environment at the same time is known in the field of robotics as Simultaneous Localization and Mapping (SLAM). We wish the reader a warm welcome!

Contents

Chapter 1

Introduction

This introductionary chapter will put things into perspective. Since this thesis is part of the master Artificial Intelligence at the University of Amsterdam, we will start with a broad overall view of Artificial Intelligence and slowly narrow down to the specific topic of this thesis, Simultaneous Localization and Mapping. Next we will present the research objective followed by an outline of the thesis.

1.1 Artificial Intelligence

Artificial Intelligence (AI) is an immensely broad research field with very vague borders. It can refer both to the fundamental problem of understanding and modelling human intelligence in general and to a more applied view of creating machines that exhibit intelligent behaviour for specific problems. The vagueness of it's borders arises from the field's second word, intelligence, for this is a concept for which there is no clear definition. Yet we can identify closely related concepts: creativity, memory, learning, insight, problem solving, planning, reasoning, thinking, language, understanding, brains, sensing, acting, consciousness and so forth. Does intelligence depend on all these concepts? Put differently, are all these concepts necessary for intelligence? Is intelligence possible without for example language or consciousness? These are philosophical questions that can be investigated but might turn out to be unanswerable. But it illustates nicely both the broadness and vaguesness of AI.

Human intelligence is mainly studied in fields like psychology, biology and more recently neurology. These fields, and then I mean the subfields of them that concern human intelligence, have in common that they observe a human being just as a physicist observes the inanimate world. For example, they observe human behaviour under certain circumstances, they try to come up with theories about the origin and development of humans, they study human anatomy, both physical and functional, ranging from the whole body, to brain areas, to single brain cells. All these studies contribute to our understanding of intelligence, just by observing it.

Now the articifial part of AI is not so much concerned with *observation of humans*, but with the *creation of machines*: inanimate entities created by hu-

mans to perform a certain task. Naturally, the task of an AI-machine is to behave intelligently. But as said above, what intelligence is, we cannot really say. So the ultimate goal of AI is to create machines for which we cannot really say what their task is...Of course this doesn't make sense. Therefore most AI researchers build machines that can perform a very specific task. Some examples of such tasks are playing chess, speech recognition, car driving and searching the web. These tasks can be divided into two groups: structured tasks and unstructured tasks. Structured tasks are well defined and the input for the machine is always controlled. Such tasks have yet only been found in artifically created environments, such as in games (like chess), in mathematical problems (like the traveling salesman problem) and in factories (like a car factory, where a robot arm performs the same task over and over again). These structured tasks need 'intelligent' machines because of the combinatorial explosion of possible actions (like chess and the traveling salesman problem) or because of the precise repeatability of the actions (like in the factory). It can be questioned if these structured tasks really need intelligence or just a 'stupid' machine that is very good at repeated calculations. But again, since we do not yet know what intelligence is, it might once turn out that intelligence is indeed nothing more than mere calculations.

Unstructured tasks are regarded as 'real-life' tasks: the machine needs to operate in the real world, which is unstructured (or structured, but yet so complex that for now we treat it as unstructured). Examples of such tasks are interpreting natural languague, whether spoken or written, visual recognition of objects, like faces, and finding patterns in complex datasets, such as weather- and stock exchange data. For all these tasks the machine needs to 'understand' what it senses. Although natural language consists of rules and exceptions, there exists many ambiguities in every language which can only be resolved by some sort of 'understanding'. The same holds for recognizing objects: any object might appear very different under different conditions, such as light, the distance and angle to the viewer, and so forth. An artifically intelligent machines needs to cope with these infite number of possibilities and associated uncertainties. We will turn special attention to uncertainty in chapter 2.2, since it plays an important role in this thesis.

Many subfields of AI concentrate on these unstructured tasks and the topic of this thesis falls under one of them: robotics. Robotics can be seen as th subfield of AI that encompasses everything that is needed for an artifical brain. Naturally we do not mean the robots that solve structured problems, such as in factories, but we mean robots that act in the real world. Such machines need a materialistic body to be able to exist in this real world, they need sensors to sense the world and they need actors to act in the world. Furthermore they require a 'brain', which is used for processing the senses, reasoning, and planning actions. The topic of this thesis is concerned with the 'brain' part of the robot: it processes the sensor information and reasons about it. The task this 'brain' needs to solve is SLAM.

1.2 Simultaneous Localization and Mapping

The SLAM scenario is that a robot moves through an environment and during it's journey it takes measurements from that environment. Now the goal is to build a map of the environment and to calculate the trajectory of the robot's journey relative to that map. The complexity of the problem lies in the fact that the robot is facing two problems which are dependent on each other: localization and mapping. Suppose that the robot was provided a map of the environment then the problem would only be to localize itself relative to the given map; this is called localization. Or the other way around, suppose the robot trajectory is known (e.g. based on GPS-data) then the problem would only be to make a map of the environment relative to the given trajectory; this is called mapping. These two problems are in themselves already challenging enough, but now in SLAM these two problems need to be solved at the same time. In chapter 2 we will discuss the SLAM problem in more detail and in a more formal way.

1.3 Research objective

In this thesis we present an experimental comparison of three well-known methods to solve the SLAM problem. These methods are EKF-SLAM, FastSLAM 1.0 and it's successor FastSLAM 2.0. Many more SLAM algorithms exist but the EKF-SLAM and FastSLAM algorithms can be viewed as the general classes of which most other SLAM algorithms are an instance. Therefore we restrict the comparison to these three fundamental algorithms. Given that we strive for a fair comparison of these fundamental and general SLAM algorithms, we have defined the following set of requirements for the experimental setting:

- The algorithms should be tested on the same dataset(s).
- The dataset(s) should provide a realistic SLAM problem.
- The dataset(s) should provide a complex enough SLAM problem such that differences between algorithms can be detected.
- The dataset(s) should provide a general enough SLAM problem such that really the fundamental workings of the algorithms are tested.
- A groundtruth should be available to be able to quantitavely compare the algorithms.

These requirements have led us to select two datasets. One is a synthetic dataset in which all parameters can be controlled. The other is a dataset published by Stephan Gutmann based on a Robocup scenario. The advantage of the Gutmann dataset is that it gives a very manageable set of landmarks (only six), it provides a groundtruth, and that it still offers a challenging SLAM problem with realistic noise. Gutmann has used this dataset to perform an experimental comparison of localization method, but to the best of our knowledge this dataset has not been used yet to perform a comparison of SLAM methods.

The research objective of this thesis can now be formulated as to

- present the Gutmann dataset as a benchmark for comparing SLAM methods, and to

- present the first published experimental comparison of SLAM methods on the Gutmann dataset.

1.4 Outline

We start by defining the SLAM problem in chapter 2. The three different SLAM algorithms are then treated in chapter 3, followed by chapter 4 which presents the two datasets used for the experimental comparison. In chapter 5 the actual experiments and results are discussed, and finally we conclude the thesis with a conlusion in chapter 6.

Chapter 2

The SLAM problem

In the introduction we have presented the SLAM problem in an informal way. This chapter will define the SLAM problem in a more mathematical way. We start by introducing all basic SLAM elements and their representation and notation, such as state representation, control actions and measurement data. In the second section we will look at the SLAM problem as a probabilistic sequential state estimation problem in which the Bayes Filter is introduced as the most general method of solving that problem. Next the two models needed for all SLAM methods are discussed, the motion and measurement model.

2.1 Representation and notation

> *The SLAM scenario is that a robot moves through an environment and during it's journey it takes measurements from that environment. Now the goal is to build a map of the environment and to calculate the trajectory of the robot's journey relative to that map.*

To solve this problem mathematically we need to identify all elements of the problem, represent them in a formal way and give them a notation so we can later refer to them.

Time Since the robot moves throug space-time some representation for time is needed. Following convention we will treat time as consecutive discrete time steps represented by a natural number t. Each time step has the same magnitude and can represent any time unit (seconds, hours, years, etc) depending on the scenario. In this thesis time will always begin at $t = 1$.

Space A representation for space is needed as well. Since we will focus only on scenarios in which the robot is moving on a plane (a soccerfield) we use a common 2D-Euclidean space. The coordinate frame of this 2D-space can be chosen arbitrarily. We will use the pose of the robot at $t = 1$ to define the frame: the origin is at the position of the robot, the x-axis runs in the direction the robot is looking in, and the y-axis run perpendicular (counterclockwise) to the x-axis. We call this coordinate frame the world frame.

Robot pose and trajectory At each time step the robot pose is estimated. A pose consists of the robot position and it's orientation and is represented by the vector $x_t = (x\ y\ \theta)^T$. x and y are given in world coordinates and θ is the angle relative to the x-axis of the world. All angles in this thesis are in the range $(-\pi, \pi)$. Please note that a pose x_t is always associated with a time step; this distinguishes x_t from x which is the x-coordinate of the pose under discussion. A trajectory up to time t is then built up from consecutive robot poses and denoted $x_{1:t}$.

Control actions The movement of the robot through the environment is caused by control actions. Many different kind of control actions are possible ranging from very general to very specific commands. For example 'turn left' is quite a general control action whereas 'turn the left-wheels with 10 rotations per minute and the right-wheels with 50 rotations per minute for the next 4 seconds' is quite specific. Which control actions are used depends on the scenario and in some cases, like with hand-held cameras, control actions are not available at all. Formally a control action is given by a vector u_t, it's dimension being dependent on the choice of control actions, and is representing the action that moves the robot from pose x_{t-1} to pose x_t. The set of all consecutive control actions up to time t is denoted $u_{1:t}$.

Environment and sensing We assume the environment consists of a space containing objects, e.g. houses, cars, walls, trees, humans, etc. The robot is capable of measuring these objects by sensors. How the environment, the map and the measurements are formally represented is depended of the sensor(s) being used. In this thesis a VGA-camera will be used as a sensor. For cameras the most common representation of the environment is that of a set of feature based landmarks. At each time step the camera captures an image of the environment. Using image processing techniques this image can be analyzed and features detected. Commonly used features are corners (SIFT, Harris), lines (line detector) or even complete objects, like faces, pedestrians or cars. The detected features are then described as landmarks: a landmark is just a coordinate in the world frame representing the location of the feature accompanied with a signature describing the landmark. This signature is needed for distinguishing between different landmarks and clearly depends on the feature-detector used: for corners a SIFT-descriptior might be used, whereas for faces a totally different signature is needed. Since sensing, image processing, feature extraction and description is a research field in it's own we will just happily use the results from that research. In our case the robot is capable of detecting and quite reliably (but not 100%!) distinguishing between the six poles in the image captured by the camera.

An important assumption made by most SLAM methods so far, including the ones discussed in this thesis, is that the environment is static. This means that the environment doesn't change over time: the position and appearance of landmarks remains constant. To account for dynamic environments landmark-motion and landmark-appearance models that keep track of the changes of the landmarks over time are needed.

Landmarks and map So the environment consists of (static) landmarks: locations accompanied with a signature. The location of a landmark i is represented by the vector $m_i = (m_{i,x}\ m_{i,y})^T$ and it's signature by s_i, which in our case is just an integer in the range $(1, 6)$. A map m is then nothing more than the combined vector of all N landmarks: $m = (m_1\ s_1\ m_2\ s_2\ \ldots\ m_N\ s_N)^T$. The map uses the world coordinate frame to store the landmark locations. Please note that since the environment is assumed to be static the map does not depend on time t.

Measurements At each time step the robot senses the environment. Using image processing techniques briefly mentioned above the robot transforms the sensed images into measurements of landmarks. What kind of measurements are being done is again dependent on the scenario. In our case we measure the distance from the robot to the landmark, called range, and the angle under which it is observed, called bearing. We will discuss this in more detail in chapter 4.4. A measurement of landmark i at time step t is then represented by the vector $z_t^i = (r_t^i\ \phi_t^i\ s_t^i)^T$, where r is the range and ϕ is the bearing. All measurements done at time t are denoted z_t and represent the measurements done by the robot at position x_t. Note that the coordinate frame in which the measurements are done is relative to the robot, see again chapter 4.4 for more details. All consecutive measurements taken by the robot up to time t are denoted $z_{1:t}$.

State SLAM is about estimating the robot location and map simultaneously. So we combine the robot's pose and the map at time t in a single state vector denoted by $y_t = (x_t\ m)^T$. This state vector y_t is the variable which we are going to estimate in the following section and chapters by making use of the control actions $u_{1:t}$ and measurements $z_{1:t}$. A related variable is $y_{1:t} = (x_{1:t}\ m)^T$ which consists of the whole robot trajectory up to time t and the map at time t. Estimating this variable is known as global SLAM, but we will focus only on estimating y_t. This is known as the online SLAM problem.

2.2 Representing uncertainty

This section introduces the concept uncertainty and how to deal with that in a robotic context. Why do we need a representation for uncertainty, what role does it play in SLAM and what are the difficulties? The section starts with an informal discussion of uncertainty after which probability distributions are introduced as a way to represent uncertainty. Finally we will discuss two representations for probability distributions, Gaussians and particle sets.

2.2.1 Uncertainty in the real world

The reasoning part in real world robotic problems is mainly concerned with reasoning about uncertainty, because in the real world nothing is exact. By this we mean that measurements done in the real world are always prone to smaller or larger errors. We humans can deal with these uncertainties very well. To function in the real world we do not need to know with nanometer precision where we are, nor do we need to calculate the exact force with which we control

our limbs. Maybe our brains perform these exact calculations, we must admit that we do not know that yet, but intuitively our body seems perfectly capable of moving around without such calculations.

For robots the case is totally different. Robots are built up from scratch by mechanical and electrical parts. Their 'intelligence' is reduced to exact calculations with only zeros and ones. This makes it very hard for robots to reason about uncertainty: either the robot is in this position (up to point-precision!) or it is not. To allow a robot to cope with measurement errors inherent to real world functioning, some kind of uncertainty representation and reasoning is needed.

Consider the case that a robot's camera is looking at the Eiffeltower and now it wants to estimate the city in which it is located. An exact estimate would yield 'Paris'. But an exact estimate might also be wrong. Maybe the robot's image processing techniques are not so good and it matches the view of the Eiffeltower with that of an ordinary transmission tower; the exact estimate might then yield 'Eindhoven' or any other insignificant town with a transmission tower. Or, a more likely case, the robot is actually standing in Las Vegas where some hotel has built a replica of the Eiffeltower. So how do we represent these uncertainties?

2.2.2 Probability theory

We adopt a probabilistic view and use probability density functions to represent the uncertainties in the world. Such functions describe a belief over knowledge by assigning a number to the different possibilities. In the Eiffeltower-case, the robot might assign the highest number to 'Paris', a slightly lower number to 'Las Vegas' and a very small number to all cities and towns that have transmission towers. These numbers are called probabilities.

More formally, a probability density function is a function p defined over the possible values x (like 'Paris', 'Las Vegas', etc.) that a random variable X (like 'Which city I am in?') can have: $p(X = x)$ or just $p(x)$. It assigns a probability to each value x, where a probability is a non-negative real number, with $p(X = x) = 0$ meaning 'random variable X can absolutely not have the value of x'. The integral of a PDF must always equal 1, representing the knowledge that a random variable must necessarily take up a value lying in the domain of the variable, in the case above that is the domain of cities.

For the SLAM problem we will use these probability density functions for all aspects in which uncertainty might arise. Uncertainty does not only play a role in the estimation of the robot trajectory and the map, but also in the control and sensing parts of the robot. Viewing the SLAM problem as such, we are viewing it as a probabilistic estimation problem; at each time step not a single state y_t is considered anymore, but a belief over states $bel(y_t)$. And since this believe is dependent on the consecutive control actions and measurements we will represent it by the following conditional probability distribution:

$$bel(y_t) = p(y_t|z_{1:t}, u_{1:t}) \tag{2.1}$$

Such probability distributions need a representation to be able to work with them. We will now turn our attention to the two representations used in this thesis, Gaussian and particle sets.

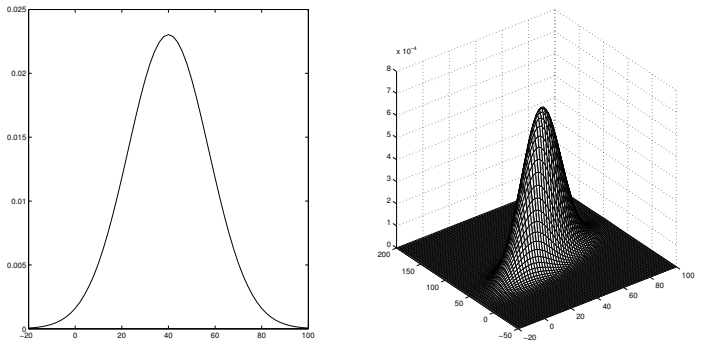

Figure 2.1: A one- and two-dimensional Gaussian distribution

2.2.3 Gaussians

One of the most common representations for a probability distribution is the one-dimensional Gaussian distribution, also known as the normal distribution. In the multi-dimensional case this distribution is officially called the multivariate normal distribution, but in practice most people just refer to it as a Gaussian. So do we.

The Gaussian distribution of a k-dimensional vector x is parameterized by two variables, the mean vector μ and the covariance matrix Σ, and is given by:

$$p(x) = 2\pi^{-\frac{1}{2}k}|\Sigma|^{-\frac{1}{2}}e^{[-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)]} \tag{2.2}$$

If a multidimensional variable x is distributed according to a Gaussian distribution this is written as:

$$p(x) \sim \mathcal{N}(\mu, \Sigma) \tag{2.3}$$

Some properties of the Gaussian distribution are:

- The mean vector represents the most likely value for x, sometimes denoted the expected value.
- The covariance matrix describes the amount of uncertainty and the correlation between the different dimensions of x.
- The distribution has a single peak, centered at it's mean, and symmetrically falls down according to the covariance matrix.
- It is a continuous distribution and all values have a probability greater than 0.

In figure 2.1 two example distributions are shown.

2.2.4 Particle sets

A different kind of representation is that of a particle set, which is parameterized by it's number of particles M. The set $\mathcal{X}$ then represents the uncertainty of the variable x by listing M possible values for x, distributed according to $p(x)$:

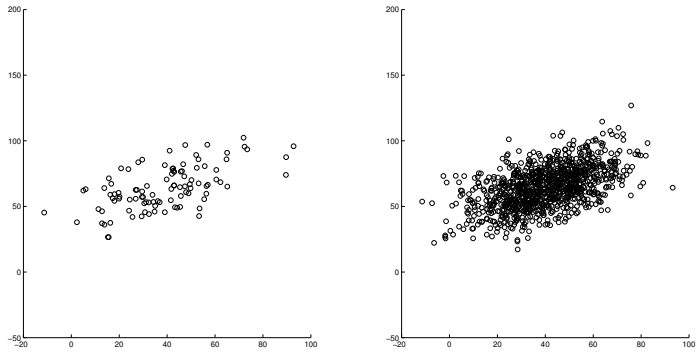

Figure 2.2: A 100 and 1,000 particle distribution

$$\mathcal{X} := x^{[1]}, x^{[2]}, \ldots, x^{[M]} \tag{2.4}$$

Some properties of particle sets are:

- A particle set can represent any distribution, it doesn't make assumptions about the shape of the function.
- A particle set can have multiple peaks.
- Since only a finite number of particles is possible to represent the distribution, the distribution can only be approximated.
- It is a discrete distribution: all listed values for x have equal probability, all other values have a probability of 0.

In figure 2.2 the same two-dimensional distribution as in figure 2.1 is shown but than with a particle set representation.

Sampling

For particle sets it is required to draw samples from the distribution $p(x)$. This can be done

2.3 SLAM as an estimation problem

In this section we look at the online SLAM problem as a *probabilistic sequential state estimation problem* and present the Bayes Filter as the most general method of solving that problem.

2.3.1 Sequential estimation

Not only can the SLAM problem be seen as a probabilistic estimation problem, but also as a *sequential estimation problem*. This arises from the fact that robot poses, with their corresponding control actions and measurements, are dependent on each other. The robot is not placed at random in different positions

at the soccerfield, but it is constantly moving from one position to another position nearby. So knowing the past poses of the robot gives some information about the future poses. Any problem in which information from the past might influence the estimate of the future, can be viewed as a sequential estimation problem.

Take weather prediction for example and let the state you wish to estimate is whether or not it is going to rain each day. Starting one day you take daily measurements, like temperature, wind direction and force, air pressure, and humidity. Based on these measurements you estimate each following day if it is going to rain or not. Although in this case you can easily find out at the end of the day whether or not your estimate was correct, you are not allowed to use this information in order to keep the analogy with the SLAM problem correct. During SLAM you also don't know whether or not your estimate of the robot's pose and map is correct! So our weather sequential estimation problem consists of the state 'Is it going to rain today?' and you can make use of your previous estimates (possbily incorrect) and the daily measurements. Now the most important issue in sequential estimation problems is not how to use the information from the past (because that differs for each scenario), but how far you go back in history. Although theoretically it might be, and I think probably is, the case that something that has occured many years ago still has an influence on the future, there are good reasons to set a limit to this. Intuitively the temperature at a certain day fifty years ago has probably little or no influence on the weather today. A counter-argument might be that not the specific temperature at that day is important, but the daily fluctations over the years nevertheless might be. So where to draw the line?

Clearly it takes a lot more computation to use the whole history of measurements. But you also don't want to disregard important information from the past. So a trade-off is made between computation and information and this is done by using the Markov assumption. The Markov assumption states that the state is complete, which then renders the next state independent of past measurements. A state is complete if it contains all the information that exactly enables this independence. More formally, the Markov assumption states the following:

$$p(y_t|y_{t-1}, z_{1:t}) = p(y_t|y_{t-1}, z_t) \tag{2.5}$$

In practice it is difficult to meet this criterion, but it allows for much less computation.

2.3.2 Bayes Filter

Starting off from equation 2.1, which is the basic function that all probabilistic SLAM methods try to estimate, we will now derive the Bayes Filter by making use of basic rules from probability theory and the Markov assumption. The Bayes Filter is the most general solution for the SLAM problem and underlies the three SLAM methods compared in this thesis.

[0]The derivation follows the one provided in PROBABILISTIC ROBOTICS but will be repeated here for completeness.

We start by applying Bayes Rule

$$\begin{aligned} p(y_t|z_{1:t}, u_{1:t}) &= p(y_t|z_{1:t-1}, z_t, u_{1:t}) && (2.6)\\ &= \frac{p(z_t|y_t, z_{1:t-1}, u_{1:t})p(y_t|z_{1:t-1}, u_{1:t})}{p(z_{1:t}|z_{1:t-1}, u_{1:t})} && (2.7)\\ &\sim p(z_t|y_t, z_{1:t-1}, u_{1:t})p(y_t|z_{1:t-1}, u_{1:t}) && (2.8) \end{aligned}$$

Next we make use of the Markov assumption

$$p(z_t|y_t, z_{1:t-1}, u_{1:t}) = p(z_t|y_t) \tag{2.9}$$

Then we use the Theorem of Total Probability

$$p(y_t|z_{1:t-1}, u_{1:t}) = \int p(y_t|y_{t-1}, z_{1:t-1}, u_{1:t})p(y_{t-1}|z_{1:t-1}, u_{1:t})dy_{t-1} \tag{2.10}$$

Followed again by the Markov assumption

$$p(y_t|y_{t-1}, z_{1:t-1}, u_{1:t}) = p(y_t|y_{t-1}, u_t) \tag{2.11}$$

Finally we assume that the control actions at time t do not depend on the state at time $t-1$

$$p(y_{t-1}|z_{1:t-1}, u_{1:t}) = p(y_{t-1}|z_{1:t-1}, u_{1:t-1}) \tag{2.12}$$

Plugging all equations back into equation 2.6 we finally get

$$p(y_t|z_{1:t}, u_{1:t}) \sim p(z_t|y_t) \int p(y_t|y_{t-1}, u_t)p(y_{t-1}|z_{1:t-1}, u_{1:t-1})dy_{t-1} \tag{2.13}$$

or equivalently using the belief notation from equation 2.1

$$bel(y_t) \sim p(z_t|y_t) \int p(y_t|y_{t-1}, u_t)bel(y_{t-1})dy_{t-1} \tag{2.14}$$

This equation is known as the Bayes Filter and will be the basis for all SLAM methods discussed in this thesis.

[0] Note that u_t happens before y_t

2.4 Bayes Filter in practice

In equation 2.13 we can identify the following three probability distributions: $p(y_t|z_{1:t}, u_{1:t})$, $p(z_t|y_t)$ and $p(y_t|y_{t-1}, u_t)$. These distributions are the ingredients of all probabilistic SLAM solutions based on the Bayes Filter. How to calculate them depends on their representation and may differ for each SLAM method.

How does it work:

1. You start with an initial state estimate: y_0.
2. You use the state transition probability/state transition function/motion model. This function/model is governed by a function denoted g. This step is called the prediction step and results in a prediction of the new state, denoted by $\bar{y}$.
3. The prediction is updated according to it's likelihood based on the measurements using a measurement probability/measurement model/measurement likelihood, which is governed by a function h. This is called the update step.
4. These two steps results in a new estimate and you go to step 2 again

What decisions do we need to make:

- **representation of the three distributions**. as will become clear in the next chapter, these representations assume and/or pose certain dependencies between the variables. in this aspect the three methods differ
- **Motion model** we need to choose an underlying function g that governs the motion model. all three methods share the same function g. This g describes how the state is affected by control actions:

$$y' = g(u, y) \tag{2.15}$$

  The probabilistic interpretation and calculation of g, which we are ultimately interested in, differs per method.

- **Measurement model** we need to choose the function h which governs the measurement model. all three methods share the same function h. the probabilistic interpretation differs per method. h describes the relation between measurements and the state:

$$z = h(y) \tag{2.16}$$

[0]The fourth distribution, $p(y_{t-1}|z_{1:t-1}, u_{1:t-1})$, represents the same distribution as $p(y_t|z_{1:t}, u_{1:t})$ but then one time step earlier.

Chapter 3

EKF-SLAM and FastSLAM

The previous chapters presented the SLAM problem as a probabilistic sequential state estimation problem. It identified all elements and introduced notation. The Bayes Filter was derived as the most general solution for the SLAM problem, which is repeated here:

$$p(y_t|z_{1:t}, u_{1:t}) \sim p(z_t|y_t) \int p(y_t|y_{t-1}, u_t) p(y_{t-1}|z_{1:t-1}, u_{1:t-1}) dy_{t-1} \tag{3.1}$$

The Bayes Filter is an exact but only theoretical solution. Implementing it in a computationally feasible way is only possible by making assumptions about the underlying structure of the probability distributions. And in this respect the three methods of this thesis, EKF-SLAM, FastSLAM 1.0 and FastSLAM 2.0, differ both in the representation of the probability distributions as in the dependencies of the random variables. What they do have in common are the functions g and h underlying the motion and measurement models respectively. We will now discuss each of the methods.

3.1 EKF-SLAM

Extended Kalman Filter SLAM (EKF-SLAM) was one of the first SLAM implementations and is based on the Kalman Filter. REFERENCE The Kalman Filter is a direct implementation of the Bayes Filter for Gaussian linear processes. Such processes are characterized by the following three assumptions:

1. The initial belief of the state is represented by a normal distribution

$$p(y_0) \sim \mathcal{N}(\mu_0, \Sigma_0) \tag{3.2}$$

2. The motion model g is linear with added Gaussian white noise $\epsilon_t \sim \mathcal{N}(0, R_t)$

$$g(u_t, y_{t-1}) + \epsilon_t = A_t y_{t-1} + B_t u_t + \epsilon_t \tag{3.3}$$

3. The measurement model h is linear with added Gaussian white noise $\delta_t \sim \mathcal{N}(0, Q_t)$

$$h(y_t) + \delta_t = C_t y_t + \delta_t \tag{3.4}$$

These three assumptions guarantee that the belief of each following state remains a Gaussian, see chapter 3.2.4 in [4] for the proof of this. And since a linear transform of a Gaussian returns a Gaussian again, each PDF for Kalman Filter SLAM can be represented by a Gaussian, all parameterized by a mean vector and covariance matrix:

$$\begin{aligned} p(y_t|z_{1:t}, u_{1:t}) &\sim \mathcal{N}(\mu_t, \Sigma_t) && (3.5)\\ p(y_t|y_{t-1}, u_t) &\sim \mathcal{N}(g(u_t, y_{t-1}), R_t) && (3.6)\\ p(z_t|y_t) &\sim \mathcal{N}(h(y_t), Q_t) && (3.7) \end{aligned}$$

Given these representations and the linear motion and measurement models the prediction step looks then as

$$\begin{aligned} \bar{\mu}_t &= A_t\mu_{t-1} + B_t u_t && (3.8)\\ \bar{\Sigma}_t &= A_t\Sigma_{t-1}A_t^T + R_t && (3.9) \end{aligned}$$

and the update step as

$$\begin{aligned} K_t &= \bar{\Sigma}_t C_t^T (C_t\bar{\Sigma}_t C_t^T + Q_t)^{-1} && (3.10)\\ \mu_t &= \bar{\mu}_t + K_t(z_t - C_t\bar{\mu}_t) && (3.11)\\ \Sigma_t &= (I - K_t C_t)\bar{\Sigma}_t && (3.12) \end{aligned}$$

For the derivations of these formulas we refer the reader to [4].

In case the motion and measurement models are not linear the Extended Kalman Filter is used, which linearizes these models using a first order Taylor expansion around the state most likely at the time of linearization. For the prediction step this is around the estimated mean of the state at the previous time step, μ_{t-1} and for the update step this is around the predicted mean of the state at the current time step, $\bar{\mu}_t$. These linearizations are used as an approximation of the nonlinear models and are given by

$$\begin{aligned} g(u_t, y_{t-1}) &\approx g(u_t, \mu_{t-1}) + G_t(y_{t-1} - \mu_{t-1}) && (3.13)\\ h(y_t) &\approx h(\bar{\mu}_t) + H_t(y_t - \bar{\mu}_t) && (3.14) \end{aligned}$$

Given the same representations for the PDFs as in the Kalman Filter case, but now with nonlinear motion and measurement models, the prediction step looks like

$$\begin{aligned} \bar{\mu}_t &= g(u_t, \mu_{t-1}) && (3.15)\\ \bar{\Sigma}_t &= G_t\Sigma_{t-1}G_t^T + R_t && (3.16) \end{aligned}$$

and the update step as

$$\begin{aligned} K_t &= \bar{\Sigma}_t H_t^T (H_t\bar{\Sigma}_t H_t^T + Q_t)^{-1} && (3.17)\\ \mu_t &= \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t)) && (3.18)\\ \Sigma_t &= (I - K_t H_t)\bar{\Sigma}_t && (3.19) \end{aligned}$$

Again we refer to [4] for their derivations.

3.2 FastSLAM 1.0

As said above the (Extended) Kalman Filter is an implementation of the Bayes Filter for linear(ized) Gaussian processes. Now we will discuss the FastSLAM algorithm, which also implements the Bayes Filter but it makes some different assumptions about the underlying structure of the SLAM problem.

3.2.1 Particle filter

The belief representation of FastSLAM is that of a particle set as discussed in the previous chapter. To repeat, the belief of a state is then represented by a set $\mathcal{Y}_t$ of M samples:

$$\mathcal{Y}_t := y_t^{[1]}, y_t^{[2]}, \ldots, y_t^{[M]} \tag{3.20}$$

Calculating this particle-set at each time step will be done following the predict and update steps from the Bayes Filter. Algorithms that uses a particle representation and implement the Bayes Filter are known as particle filters.

The predict step for particle filters samples from the motion model:

$$\bar{\mathcal{Y}}_t = \{\text{sample } y_t^{[k]} \sim p(y_t|u_t, y_{t-1}^{[k]}) : y_{t-1}^{[k]} \in \mathcal{Y}_{t-1}\} \tag{3.21}$$

How to sample from the motion model depends on the underlying function g and it's probabilistic interpretation. We will discuss these issues in chapter 4.3, but for now we just point out that some sampling scheme is needed for the prediction step.

$\bar{\mathcal{Y}}_t$ is called the proposal distribution and approximates $p(y_t|u_t, y_{t-1}^{[k]})$. After that the update step from the Bayes Filter incorporates the measurements using the measurement model by calculating a weight:

$$w_t^{[k]} = p(z_t|\bar{y}_t^{[k]}) \tag{3.22}$$

So for each predicted state $\bar{y}_t^{[k]}$ from the proposal distribution the likelihood is calculated using a probabilistic interpretation of the function h (see chapter 4.4). These likelihoods are then stored and interpreted as weights $w_t^{[k]}$ for each particle. The proposal distribution $\bar{\mathcal{Y}}_t$ together with the associated weights then constitutes the posterior distribution $p(y_t|z_{1:t}, u_{1:t})$.

Notice that the particles from the proposal distribution are distributed according to $p(y_t|u_t, y_{t-1}^{[k]})$ and not distributed according to the posterior. Still this can be achieved by redistributing the particles from the proposal using the weights. What you basically do is drawing M samples (with replacement) from the proposal distribution $\bar{\mathcal{Y}}_t$ in proportion to their weights. This process is called importance resampling.

So given the predicted set and the corresponding weights, the update step for particle filters looks like:

$$\mathcal{Y}_t \quad = \quad \text{resample}(\bar{\mathcal{Y}}_t, w_t^{[1:M]}) \tag{3.23}$$

With an initial belief $\mathcal{Y}_0$ represented by a set consisting of M copies of the initial state y_0, equations 4.21 to 4.23 specifiy the general particle filter.

3.2.2 SLAM Factorization

FastSLAM makes two independence assumptions:

- The robot trajectory and the map are independent.
- The individual landmark positions are conditionally independent of each other given the robot's trajectory.

These two assumptions can be expressed in the following factorization of the full SLAM posterior, where the state vector has been split into the robot trajectory and the map components:

$$p(y_{1:t}|z_{1:t}, u_{1:t}) = p(x_{1:t}|z_{1:t}, u_{1:t}) \prod_{n=1}^{N} p(m_n|x_{1:t}, z_{1:t}) \tag{3.24}$$

Finally FastSLAM makes the same assumption as EKF-SLAM about the measurement model:

- The measurement model h is linear with added Gaussian white noise $\delta_t \sim \mathcal{N}(0, Q_t)$. In case it is non-linear, FastSLAM uses the same linearization as EKF-SLAM.

Please note that FastSLAM, in contrast to EKF-SLAM, doesn't make any assumptions about the motion model or the form of the posterior distribution.

3.2.3 FastSLAM representation

FastSLAM needs representations for the posterior, $p(y_{1:t}|z_{1:t}, u_{1:t})$, the robot trajectory, $p(x_{1:t}|z_{1:t}, u_{1:t})$ and for each of the landmarks, $p(m_n|x_{1:t}, z_{1:t})$. As discussed in the previous section, the posterior and the robot trajectory are represented by particles. But given the factorization of the SLAM problem and the assumption about the measurement model, each landmark can be represented by a single Gaussian. The landmark estimates are all conditionally independent on each other given the trajectory, so for each particle (which represents a trajectory) we need a set of N Gaussians to represent the landmarks; this results in a total of MN Gaussians for the landmarks. Each of these Gaussian is parameterized by it's own mean and covariance and is updated using an EKF. Filters that use particles to represent some variables and Gaussians to represent the other variables are known as Rao-Blackwellised particle filters. REFERENCE.

Given the general particle filter, the SLAM factorization and the representations used by FastSLAM, we now have enough information to present the FastSLAM implementation of the predict- and update-steps of the Bayes Filter. The derivations of the equations below can all be found in [4].

3.2.4 Predict step

In practice only new robot poses are sampled, because the prediction step leaves the landmark estimates unchanged. So when writing out the state into the pose

and map components, the prediction step for one particle looks like:

$$\bar{x}_t^{[k]} \sim p(x_t|x_{t-1}^{[k]}, u_t) \tag{3.25}$$
$$\bar{\mu}_i^{[k]} = \mu_i^{[k]} \tag{3.26}$$
$$\bar{\Sigma}_i^{[k]} = \Sigma_i^{[k]} \tag{3.27}$$

The predict step for the complete set is just performing the above calculations for each particle in turn.

3.2.5 Update step

The update step involves both the landmark positions and the predicted robot pose. We will discuss them seperately, starting with the landmark updates.

$$K_i^{[k]} = \bar{\Sigma}_i^{[k]} H^T (H\bar{\Sigma}_i^{[k]} H^T + Q_t)^{-1} \tag{3.28}$$
$$\mu_i^{[k]} = \bar{\mu}_i^{[k]} + K_t(z_t^i - h(\bar{\mu}_i^{[k]})) \tag{3.29}$$
$$\Sigma_i^{[k]} = (I - K_t H)\bar{\Sigma}_i^{[k]} \tag{3.30}$$

Notice that these equations are almost identical to the update equations in EKF-SLAM as given in chapter 3.1. The main difference is that in EKF-SLAM these equations worked on the full state vector and in FastSLAM they work on a single landmark i of a specific particle k.

As explained above, the update step of the state involves calculating importance weights and resampling from the proposal distribution according to these weights. The weight for each particle is calculated as follows:

$$L_i^{[k]} = H\bar{\Sigma}_i^{[k]} H^T + Q_t \tag{3.31}$$
$$w_t^{[k]} = \prod_{i=1}^{N} \mathcal{N}(z_t^i; h(\bar{\mu}_i^{[k]}), L_i^{[k]}) \tag{3.32}$$

For the derivations we refer the reader to [4].

3.3 FastSLAM 2.0

FastSLAM 2.0 is an improved variant of FastSLAM 1.0. The only conceptual difference is that FastSLAM 2.0 takes the measurements into account when calculating the proposal distribution. To do so, firstly each pose from the previous particle set is transformed into a Gaussian parameterized by a mean vector and covariance matrix using the motion model:

$$\mu_{x,t}^{[k]} = g(x_{t-1}^{[k]}, u_t) \tag{3.33}$$
$$\Sigma_{x,t}^{[k]} = R_t \tag{3.34}$$
$$\tag{3.35}$$

Then each measurement in turn is used to update this Gaussian:

$$Q = (H_m \Sigma_i^{[k]} H_m^T + Q_t) \tag{3.36}$$
$$\Sigma_{x,t}^{[k]} = (H_x^T Q^{-1} H_x + (\Sigma_{x,t}^{[k]})^{-1})^{-1} \tag{3.37}$$
$$\mu_{x,t}^{[k]} = \mu_{x,t}^{[k]} + \Sigma_{x,t}^{[k]} H_x^T Q^{-1} (z_t^i - h(\mu_i^{[k]})) \tag{3.38}$$

After all the measurements have been processed, each particle for the proposal distribution is then sampled from this Gaussian:

$$\bar{x}_t^{[k]} \quad \sim \quad \mathcal{N}(\mu_{x,t}^{[k]}, \Sigma_{x,t}^{[k]}) \tag{3.39}$$

After calculating this improved distribution FastSLAM 2.0 continues just like FastSLAM 1.0. Only the calculation of the covariance $L_i^{[k]}$ in the importance weight equation is slightly changed as a mathematical consequence of using the improved distribution.

$$L_i^{[k]} \quad = \quad H_{x,j} R_t H_{x,i}^T + H_{m,i} \Sigma_{i,t-1}^{[k]} H_{m,i}^T + Q_t \tag{3.40}$$

Chapter 4

The Gutmann dataset

The SLAM problem and three methods solving the problem have been discussed in the previous chapters. This chapter will introduce the Gutmann dataset as a testbed for performing the comparison between these methods. In [3] Gutmann and Fox presented a comparison of localization methods. Localization is a subproblem of SLAM in which the map is given to the robot and it's only task is to estimate his consecutive locations. For their experiment they made use of a scenario based on the RoboCup competition. The chapter starts with a description of the RoboCup competition. Then the specific experimental setup of the Gutmann dataset is treated. In the last two sections details will be given about the relevant motion and measurement models.

4.1 RoboCup

The RoboCup competition started in 1997. It's ultimate goal is to have a team of humanoid robots winning a soccer game while playing against the world champion soccerteam by the midst 21st century. This poses challenges in almost all aspects of robotics. Physically the robots need to be able to move around a soccerfield and control the ball, both at considerable speed and with taking into account enough safety precautions for other players. On the reasoning part difficulties lie in coorporation, tactics, and even basic knowledge as to where the robots are located on the soccerfield. Although this latter problem seems futile for humans, we have pointed out in chapter **??** that this really is a challenge for robots. Given the broad scope of problems and large difficulties to overcome, RoboCup is divided into different leagues each focusing on a different aspects. For example, the simulation league has a strong focus on the coorporation and tactics part, and the humanoid league is more focused on the physical aspects of the robot.

The Gutmann dataset originates from the Sony Four-Legged League which started in 1999. In this leagure all teams play with the same robot. so as to focus more on the reasoning part and keeping the physical factors equal for all the teams. Till 2008 the robot platform used was the Sony AIBO, after that it was replaced by the Nao humanoid robot from Aldebaran Robotics. Since then the league is called the Standard Platform League.

Figure 4.1: The Sony AIBO ERS-210 robot.

4.2 Experimental setup

In this section details about the experimental setup of the Gutmann dataset will be given.

The robot Gutmann and Fox used a Sony AIBO robot (ERS 210) for their experiments. This system is a four legged dog-like robot built in 2001, see figure 4.1. It is equiped with many sensors, like pressure sensors (at it's feet, back, chin and head), a CMOS digital camera sensor (placed in it's head), an acceleration sensor, and a infrared distance sensor. In total it has 20 degrees of freedom of which only the legs (12 degrees of freedom) and head movement (3 degrees of freedom) play a role here.

Environment The size of the soccerfield is 3-by-2 meters. Six poles are placed around the field: four at the corners and two in the middle of the long sides of the field. Each pole is colored with two colors, one color for the upper part and one color for the lower part. This allows the robot to reliably detect and distinguish between the different poles.

The task For their experiments they let the robot follow a figure-of-eight trajectory specified by 5 marked points on the soccer field for one hour. The robot was moved around by a human controller with a joystick. The head of the robot was constantly swinging it's head from side to side. This way the total field of view (over a number of time steps) was in the range -120 degrees to 120 degrees. During the trajectory the CMOS camera sensor captured color labeled images, which were processed using the CMVision software library [1]. This resulted in timestamped range and bearing measurements of the observed landmarks, where the direction of the head relative to the body was already taken account for. This will be further discussed in section 4.4. Furthermore each time step odometry measurements were used to obtain an estimate of the robot's pose. In section 4.3 we will explain how we used these pose estimates for the control data.

Each time the robot arrived on a marked point the pressure sensor on it's head was touched by a human operator. These timesteps are stored as well and will be used as the groundtruth. Please note that no information about the true location of the robot is known in between these marked points.

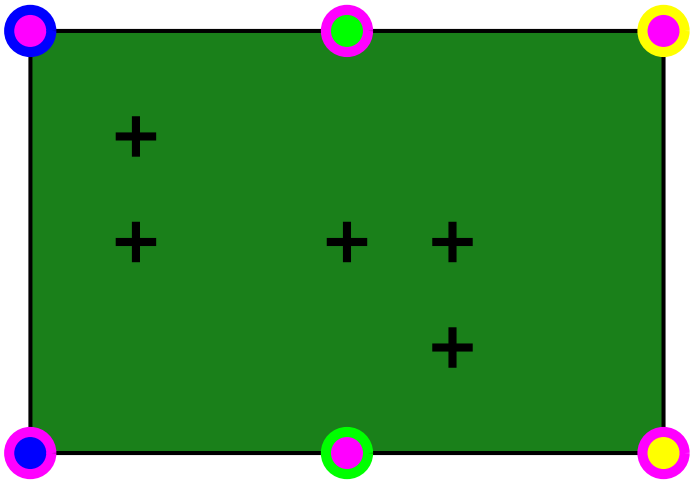

Figure 4.2: An overview of the soccerfield with the six landmarks and the five markerd points.

Figure 4.2 gives an overview of the scenario.

4.3 Motion model

For the prediction step a motion model is needed which describes how a robot state is affected by control actions:

$$p(y_t|y_{t-1}, u_t) \tag{4.1}$$

There are also cases in which no control actions are available ([2]). Still a motion model is then needed that describes how the state proceeds in the absence of control data. Although the motion model formally operates on the combined state vector y_{t-1}, in practice it is only used to alter the robot's pose x_{t-1} leaving the map estimate m unchanged. Therefore we only show below how the model works on the robot's pose.

4.3.1 Move Forward Motion Model

We have created a model that doesn't use control actions. The robot moves a constant step d forward in the direction it is looking and keeps the orientation constant, both added with gaussian white noise. This model can be formulated as follows:

$$\begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} d\cos\theta \\ -d\sin\theta \\ 0 \end{pmatrix} + \mathcal{N}(0, R) \tag{4.2}$$

For the EKF-SLAM algorithm we also need a linearization of this model, which is calculated by a first order Taylor expansion around the most likely state at the

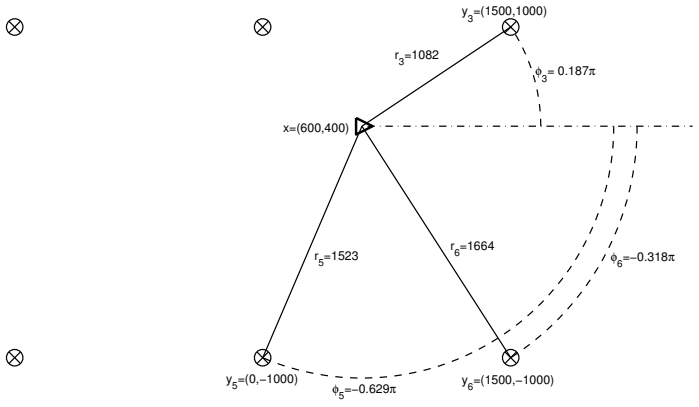

Figure 4.3: Feature based range and bearing model

time of linearization. This is given by the Jacobian of the model with respect to the robot's pose:

$$G = \begin{pmatrix} 1 & 0 & -d\sin\theta \\ 0 & 1 & -d\cos\theta \\ 0 & 0 & 1 \end{pmatrix} \tag{4.3}$$

4.4 Measurement Model

The measurement model describes the robot's perception of landmarks. Given the robot's pose the measurement model maps a landmark position to a measurement. For the Gutmann dataset we use a feature based range and bearing model.

4.4.1 Feature Based Range and Bearing Model

The feature based range and bearing model assumes that landmarks can be represented by points in world coordinates. Range is defined as the Euclidean distance between the robot and the landmark, and bearing is defined as the viewing angle under which the landmark is observed. See Figure (4.3) for a graphical explanation of this model.

Given the robot's pose $(x\ y\ \theta)^T$ and the position $(m_{i,x}\ m_{i,y})^T$ of landmark i, the range r_i and bearing ϕ_i are calculated as follows:

$$\begin{pmatrix} r_i \\ \phi_i \end{pmatrix} = \begin{pmatrix} \sqrt{\delta_{i,x}^2 + \delta_{i,y}^2} \\ \arctan\frac{\delta_{i,y}}{\delta_{i,x}} - \theta \end{pmatrix} + \mathcal{N}(0, Q) \tag{4.4}$$

with

$$\delta_{i,x} = m_{i,x} - x \tag{4.5}$$

$$\delta_{i,y} = m_{i,y} - y \tag{4.6}$$

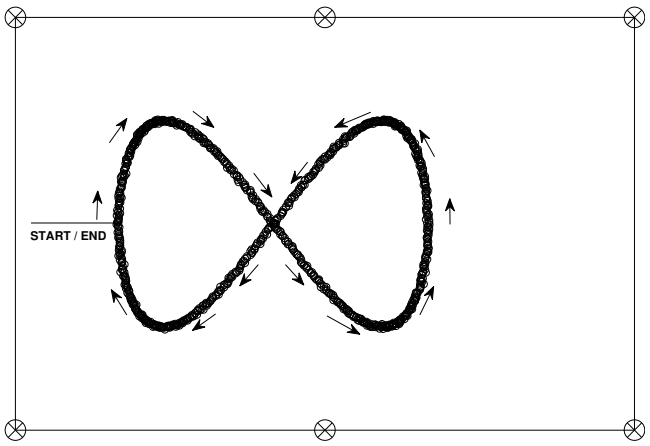

Figure 4.4: The robot trajectory for the synthetic dataset

The Jacobian of this model with respect to the landmark position and with respect to the robot position is respectively given by

$$H_{m_i} = \begin{pmatrix} \frac{dx_i}{r_i} & \frac{dy_i}{r_i} \\ \frac{-dy_i}{r_i^2} & \frac{-dx_i}{r_i^2} \end{pmatrix} \tag{4.7}$$

$$H_X = \begin{pmatrix} \frac{-dx_i}{r_i} & \frac{-dy_i}{r_i} & 0 \\ \frac{dy_i}{r_i^2} & \frac{-dx_i}{r_i^2} & 1 \end{pmatrix} \tag{4.8}$$

4.5 A simulated environment

To gain furhter insight into the workings of the different algoritms a simulated environment has been created. The advantage of a simulation is that all settings are controlable and known. The groundtruth is exactly known, noise parameters can be variated and are known and robot trajectories can be setup at will. A simulated environment is for a robotician as a sterile lab is for a chemist: you keep as much factors constant as possible except for the one you are interested in. This allows for an understanding into every detail of your model and hopefully all these details create an understanding of the whole.

We have simulated a Robocup scenario identical to the one which is used for the Gutmann dataset. This allowed us to use the same motion and measurement model. We let the simulated robot walk the trajectory over the soccerfield depicted in figure 4.4 for four rounds. Each round was divided into $1,000$ timesteps which resulted in $4,000$ robot poses to be estimated. At each timestep a random number of landmarks within the visual field of the robot was observed with added Gaussian white noise. Please note that, just like in the Gutmann dataset, sometimes no landmarks were observed and sometimes more than one landmark was observed. In the end the synthetic dataset consisted of the $1,355$ observations of the 6 landmarks with which the $4,000$ poses and the map should be estimated. Since no control data is used in our experimental comparison control actions were not simulated.

Chapter 5

Experimental comparison

In this chapter the actual experimental comparison will be presented. First some parameter experiments have been performed. By running the algorithm with different parameter settings, such as noise values, we discovered how these parameters influenced the performance and which parameter values yielded the best results. Then these optimal parameter values were used to conduct a final experiment to show the differences between the three methods on both datasets. Finally we conclude with a section describing the details of how to compare the output of the SLAM algorithms with the given groundtruth.

5.1 Parameter experiments

The goal of the parameter experiments was not to compare the different SLAM algorithms with each other, but only to see how different parameter values effected the performance. We have defined a fixed set of parameter values that were used throughout all the experiments and in each experiment only one of these parameters was made variable. This set was the following:

parameter	**symbol**	**synthetic**	**Gutmann**
motion noise	σ_x^2, σ_y^2	4.8	1.5
range noise	σ_r^2	15% of r	15% of r
bearing noise	σ_ϕ^2	0.18	0.18
# of particles	M	100	100
threshold on effective # of particles	N_{min}	75% of M	75% of M

5.1.1 Motion noise

The motion model as described in chapter 4.3 is added with white Gaussian noise to incorporate the uncertainty in the robot's motion:

$$\begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} d\cos\theta \\ -d\sin\theta \\ 0 \end{pmatrix} + \mathcal{N}(0, R) \tag{5.1}$$

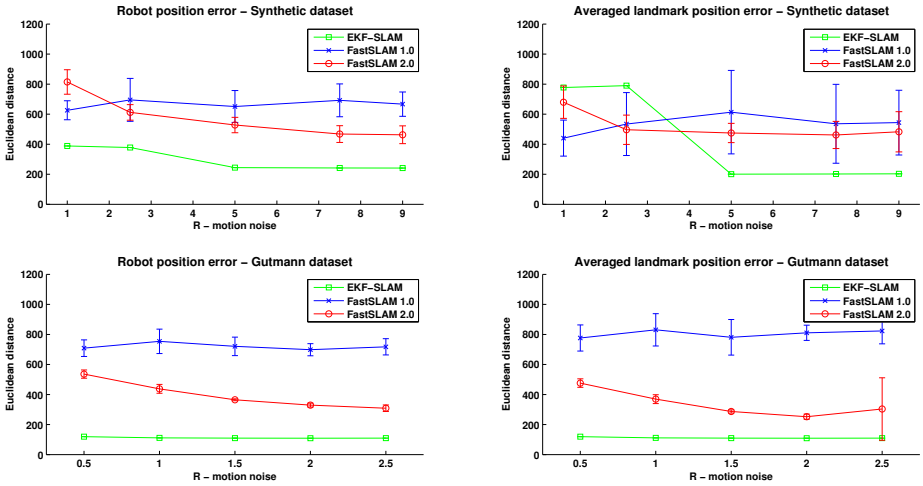

Figure 5.1: Motion noise

with

$$R = \begin{pmatrix} \sigma_x^2 & 0 & 0 \\ 0 & \sigma_y^2 & 0 \\ 0 & 0 & \sigma_\theta^2 \end{pmatrix} \tag{5.2}$$

The two parameters σ_x^2 and σ_y^2 represent the uncertainty in the robot's translational motion. To see what effect these parameters had on the performance we varied the value for both parameters from 1 till 9 for the synthetic dataset and from 0.5 to 2.5 for the Gutmann dataset. This difference in scale is because the robot takes bigger steps in the synthetic scenario than in the real Robocup scenario.

The results are given in figure 5.1. There the robot and landmark position error is plotted against the different parameter values. The overall behaviour of the algorithms can be summarized as follows. The EKF-SLAM and FastSLAM 1.0 methods perform roughly the same for all tested parameter values. This is clearly the case for the Gutmann dataset. Only for the synthetic dataset the EKF-SLAM algorithm shows an unexpected peak in the landmark position graph which we could not explain. The FastSLAM 2.0 method performs better with increasing motion noise. This might be explained from the fact that the measurement model is more accurate than the motion model. Since FastSLAM 2.0 uses both the motion and the measurement model in the prediction step, setting a higher uncertainty for the robot's motion gives more space for the measurement model to improve the prediction.

5.1.2 Measurement noise

The measurement model as described in chapter 4.4 is added with white Gaussian noise to incorporate the uncertainty in the robot's measurements:

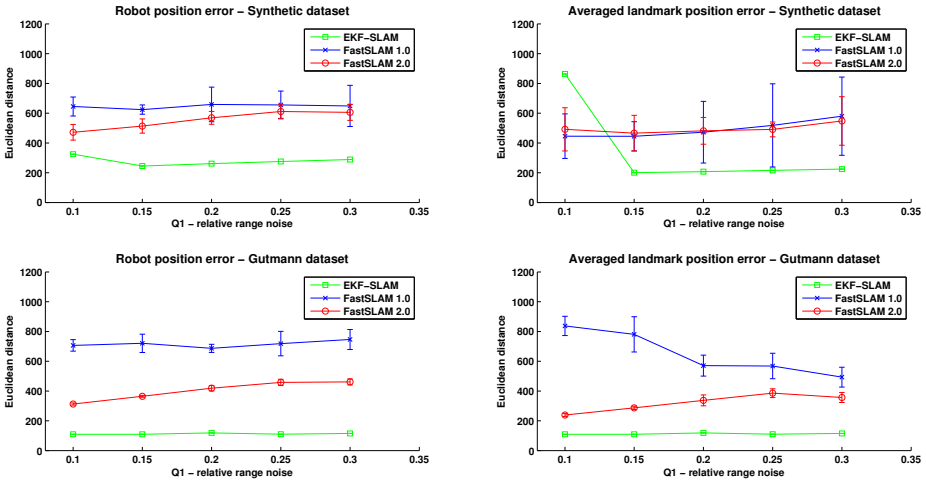

Figure 5.2: Relative range noise

$$\begin{pmatrix} r_i \\ \phi_i \end{pmatrix} = \begin{pmatrix} \sqrt{\delta_{i,x}^2 + \delta_{i,y}^2} \\ \arctan \frac{\delta_{i,y}}{\delta_{i,x}} - \theta \end{pmatrix} + \mathcal{N}(0, Q) \tag{5.3}$$

with

$$Q = \begin{pmatrix} \sigma_r^2 & 0 \\ 0 & \sigma_\phi^2 \end{pmatrix} \tag{5.4}$$

We performed two experiments, one with varying values for σ_r^2, the range noise, and the other with varying values for σ_ϕ^2, the bearing noise. For the range noise we used relative values varying from 10% to 30% of the measured range. For the bearing noise absolute values were used varying from 0.1 to 0.3 radians. The results of these two experiments are shown in figure 5.2 and figure 5.3 respectively. Again the robot and landmark position error is plotted against the different parameter values.

In the graph of the range noise we can see that for the Gutmann dataset the EKF-SLAM algorithm performs stable: it is not effected by the different values. The FastSLAM 1.0 method has better performance with increasing values of range noise. With a value of 30% the performance on the landmark positions is almost comparable with that of FastSLAM 2.0. On the contrary the FastSLAM 2.0 has a slightly decreasing performance if the values of the range noise are higher. It's graph is almost opposite than that of the graph for the motion noise in figure 5.1. Again we think that this is because of the combination of the motion and measurement model in the prediction step.

For the bearing noise we can see that the different values had little or no impact on the performance. Except for the same strange landmark error graph of EKF-SLAM for the synthetic dataset, no clear trends or peaks are visible in the graphs. This might be explained by assuming that the bearing part of the measurement model is so precise that almost no bearing noise is present in the measurements.

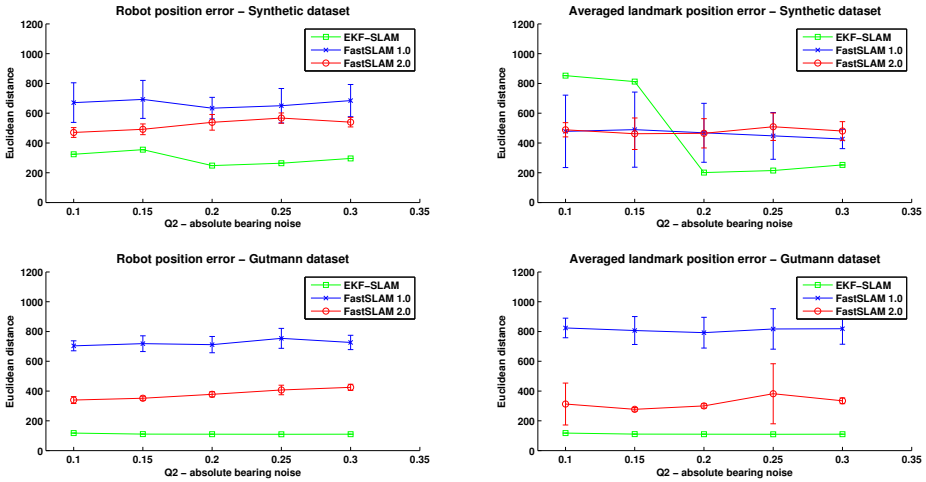

Figure 5.3: Absolute bearing noise

5.1.3 Sampling strategies

As explained in chapter **??** it has been shown that resampling at every time step might decrease performance. Therefore we have employed a stratified resampling strategy in which the effective number of particles determine if resampling is performed:

$$N_{eff} = \frac{1}{\sum_{k=1}^{M} \left(w^{[k]}\right)^2} \tag{5.5}$$

$$\text{resample if } N_{eff} < N_{min} \tag{5.6}$$

For the two FastSLAM methods we have done an experiment to see how the threshold N_{min} on the effective number of particles influences the performance. The tested values were 0%, 25%, 50%, 75%, and 100% of the total number of particles. Please note that a value of 100% means that the algorithm will always resample, whereas a value of 0% means that the algorithm will never resample. The results of this experiment are given in figure 5.4. The FastSLAM 1.0 algorithm behaves as expected, both in the synthetic dataset case as with the Gutmann dataset. The best performance is achieved for both datasets when N_{min} is set to 50% of the number of particles. If no resampling takes place the performance is lowest. For the synthetic dataset the same holds for FastSLAM 2.0, although the differences in performance are smaller than for FastSLAM 1.0. For the Gutmann dataset the performance of FastSLAM 2.0 seems even totally independent if resampling takes place or not. A possible explanation for this behaviour is that the proposal distribution already matches the target distribution so well that resampling doesn't contribute to the performance anymore. This suggestion is supported by the fact that FastSLAM 1.0, which generally generates a poorer proposal distribution, does seem to benefit from the resampling step. Furter support is given by the next experiment.

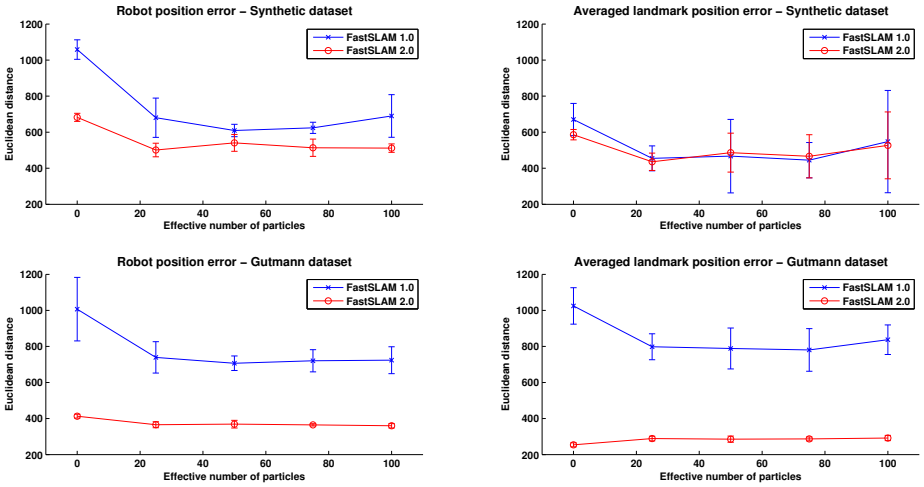

Figure 5.4: Sampling strategies.

5.1.4 Number of particles

In this last parameter experiment we investigated how the number of particles influenced the performance for the two FastSLAM methods. In chapter **??** we showed that in theory the more particles you use, the better the performance, in terms of pose and map estimates, gets. But extra particles come at the cost of extra computation time and in general there is a point in which adding extra particles doesn't improve performance anymore. Therefore we have done experiments with 1, 10, 100, and 500 particles. For the Gutmann dataset an extra experiment with 200 particles was done. Results are shown in figure 5.5.

As can be seen the performance of both FastSLAM methods increases as the number of particles gets higher. This holds both for the synthetic dataset as for the Gutmann dataset. The performance of the FastSLAM 1.0 algorithm is lowest with 1 and 10 particles; then with 100 or more particles the performance stays more or less constant. The same holds for the FastSLAM 2.0 method in case of the synthetic dataset. But for the Gutmann dataset the performance of FastSLAM 2.0 doesn't depend much on how many particles are used. Only with 1 particle there is a poorer performance for the landmark estimates, but all other number of particles perform roughly the same. This behaviour might be related to what could be seen in the case of sampling strategies. If resampling doesn't effect the performance of FastSLAM 2.0, then this could be explained by assuming that the prososal distribution is a very homogenous set of particles, where all particles gets more or less the same weight. And a homogeneous set could just as well be represented by a few particles as by hundreds of particles. We'll come back to this point in section **??**.

5.2 SLAM Experiments

Based on the results of the parameter experiments described above we have selected a set of parameter values for each algorithm and for each dataset with

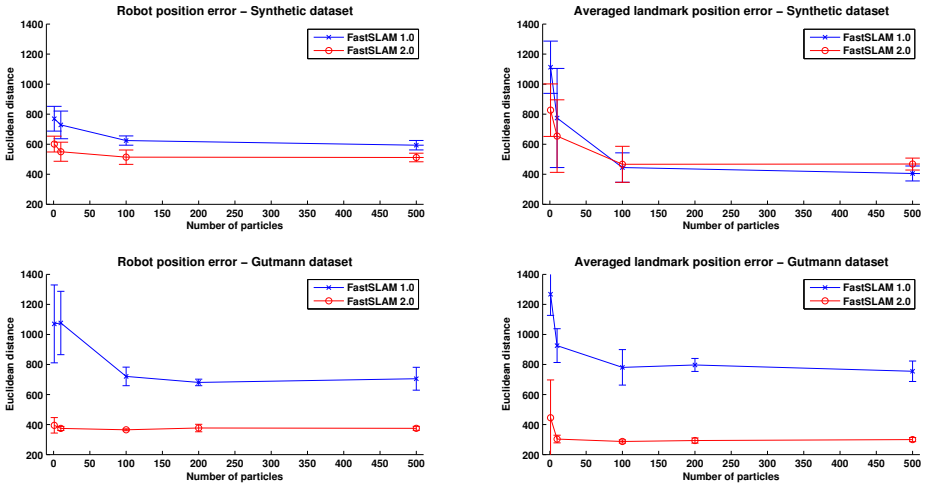

Figure 5.5: Number of particles.

which the performance was best. This set was the following:

EKF-SLAM			
parameter	**symbol**	**synthetic**	**Gutmann**
motion noise	σ_x^2, σ_y^2	7.5	2.0
range noise	σ_r^2	15% of r	20% of r
bearing noise	σ_ϕ^2	0.20	0.20
FastSLAM 1.0			
parameter	**symbol**	**synthetic**	**Gutmann**
motion noise	σ_x^2, σ_y^2	7.5	2.0
range noise	σ_r^2	10% of r	30% of r
bearing noise	σ_ϕ^2	0.20	0.20
# of particles	M	100	100
threshold on effective # of particles	N_{min}	50% of M	50% of M
FastSLAM 2.0			
parameter	**symbol**	**synthetic**	**Gutmann**
motion noise	σ_x^2, σ_y^2	7.5	2.0
range noise	σ_r^2	10% of r	10% of r
bearing noise	σ_ϕ^2	0.15	0.15
# of particles	M	100	10
threshold on effective # of particles	N_{min}	25% of M	25% of M

Using these parameters we let the three algorithms run on both datasets leading to the following results:

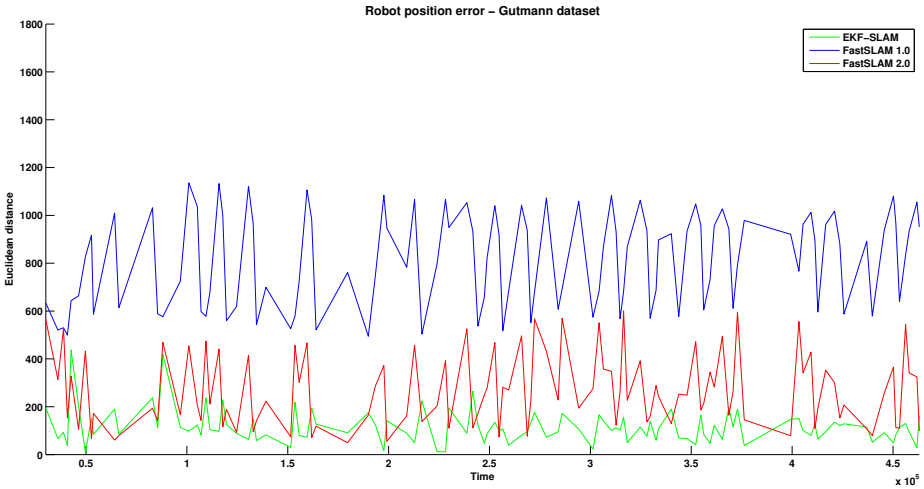

Figure 5.6: SLAM Gutmann robot.

	Position error on the Gutmann dataset			
	Robot		Landmarks	
	mean	std	mean	std
EKF-SLAM	116	n.a.	132	n.a.
FastSLAM 1.0	810	94	476	60
FastSLAM 2.0	268	6	199	16

	Position error on the synthetic dataset			
	Robot		Landmarks	
	mean	std	mean	std
EKF-SLAM	246	n.a.	204	n.a.
FastSLAM 1.0	712	130	633	331
FastSLAM 2.0	425	40	474	115

These numbers show that EKF-SLAM clearly outperforms both FastSLAM methods. Furthermore FastSLAM 2.0 has quite a low standard deviation. To get more insight into the behaviour of the algorithms we have created graphs which show the mean error over time.

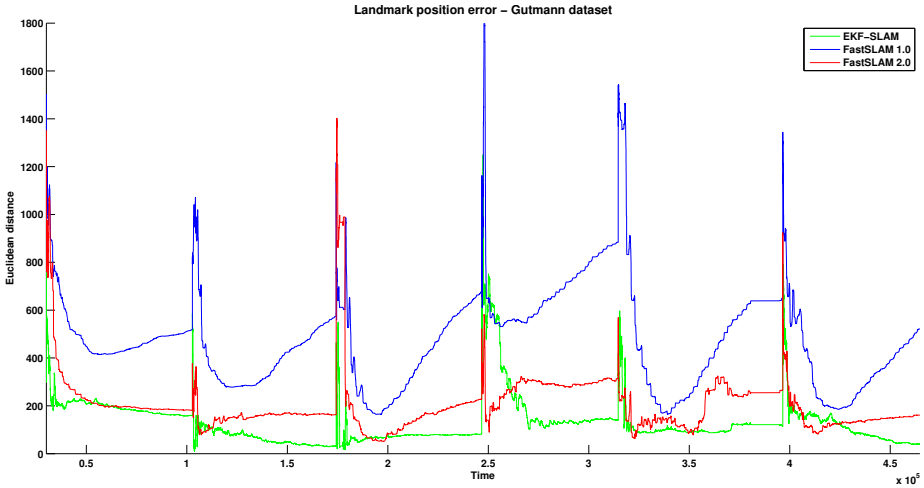

Figure 5.7: SLAM Gutmann landmarks.

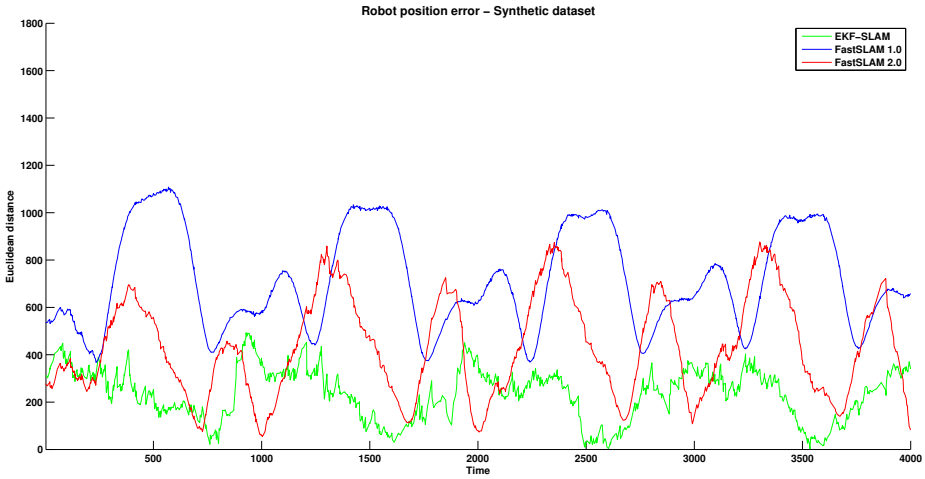

Figure 5.8: SLAM Synthetic robot

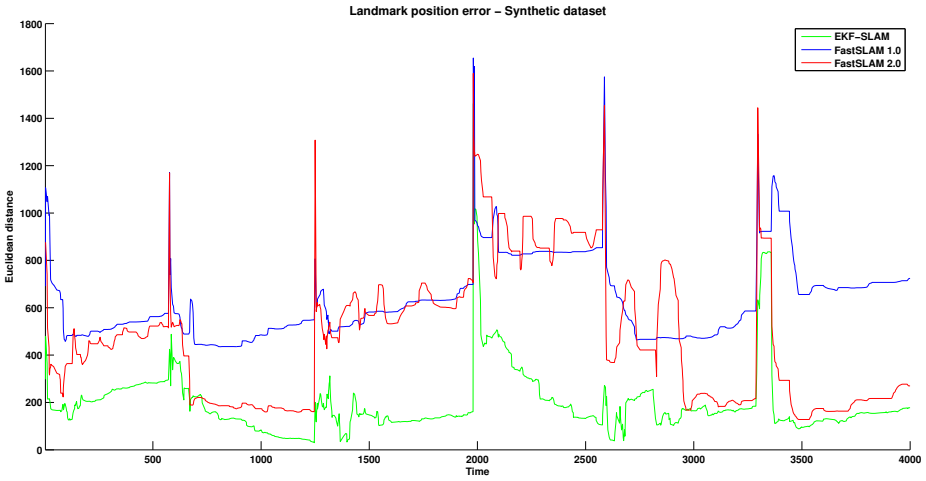

Figure 5.9: SLAM Synthetic Landmarks

Chapter 6

Conclusion

The dataset provided by Gutmann et al. [3] is an ideal benchmark to test different SLAM algorithms against each other. The map consists of only 6 landmarks, which prevents any scaling problems. The noise in both the motions and observations of the Aibo are challenging enough for state-of-the art algorithms as FastSLAM [17]. Although the dataset is easy to manage, still all aspects of the motion model, observation model, corresponding noise models, validation gate and resampling methods have to be carefully implemented before more complex SLAM problems can be solved.

Bibliography

[1] James Bruce, Tucker Balch, and Manuela Veloso. Fast and inexpensive color image segmentation for interactive robots. In *In Proceedings of IROS-2000*, pages 2061–2066, 2000.

[2] A.J. Davison. Real-time simultaneous localisation and mapping with a single camera. In *Proc. International Conference on Computer Vision, Nice*, October 2003.

[3] D. Gutmann, J.-S. Fox. An experimental comparison of localization methods continued. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2002.

[4] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. Intelligent robotics and autonomous agents. The MIT Press, September 2005.