Pose estimation with Sonar

Jan Willem Marck
s1162381

February 27, 2006

Supervised by:
dr. Bart de Boer
drs. Gert Kootstra

Artificial Intelligence
Rijks Universiteit Groningen

Abstract

Probabilistic Robotics is a relatively young approach to robotics. It emphasizes uncertainty in robot perception and action. Using probability theory it is able to represent this uncertainty explicitly. Probability density distributions are used instead of a single "best guess". This way it can model ambiguity and belief in a mathematically sound way.

The research for my thesis focuses on pose estimation. Pose estimation is the estimation of location and orientation of the robot. Odometry gives a very good estimation of the current pose, but still gives only an estimation. Over time the cumulative odometry error can grow quite large. This error can be corrected using sensor information. But using only one scan is not always adequate. Matching multiple sensor readings can provide extra information. This is called scan matching. If these two errors are combined with odometric information using a Kalman filter, scan matching should provide an improved pose estimation.

Usually scan matching is done with laser range finders. For my implementation I adapted the most widely used scan match algorithm so it can be used with sonar range finders. I tested four models for determining the translational and rotational update in the Kalman Filter. I will give the results to these tests and a general overview of Probabilistic Robotics.

Preface

Robotics has always had my interests. Finishing my artificial intelligence (AI) studies at the Rijks Universiteit Groningen (RUG) in autonomic systems was a logical choice. After the course Robotics at the RUG I became very interested in the field of Probabilistic Robotics. I preferred to do my MSc abroad. So I decided to email Dieter Fox, an important writer on probabilistic robotics literature, whether I could do my master thesis at his university. During a long and unsuccessful flirt with his Robotics and State Estimation lab at the University of Washington in Seattle I had all the time to assimilate literature of this interesting robotics field. The longevity of this episode was such that, when the US options dwindled away, I wanted to start as soon as possible with my master thesis. More lengthy negotiations with other foreign universities would have taken to much time. I decided to devise a proposal for a probabilistic robotics master thesis at the RUG. This resulted in the end in this thesis.

Acknowledgements

A word of thanks goes to my supervisors Bart de Boer en Gert Kootstra for supervising me. Also I would like to thank Corina, Kees, Michiel and Dieuwke for doing the so needed test readings. Without their help this thesis wouldnt have been possible. Thanks you's go as well to Hedde and Wimer for the use of their computers when mine gave up. For the rest I would like thank all my friends and fellow ultimate players in providing the fun and distractions needed in every day live.

Contents

Chapter 1

Introduction

Mapping has been researched avidly the last two decades. Reliable maps are needed in various mobile robot applications, for example guided museum tours, mine exploration or vacuum cleaners. Probabilistic Robotics is the current fashionable paradigm in this research area. Using probability densities it models the world and the robot. In doing so Probabilistic Robotics creates robust and reliable options for different applications.

A central theme within Probabilistic Robotics is the problem of self localization. The biggest problem is a cumulative error in odometry information. Various approaches exist to deal with this error. One specifically is scan matching. By comparing range finder scans from two different times, it corrects odometry information. This technique is very successful with laser range finders. It has not been implemented with laser range finders as yet.

Sonar range finders are unreliable (40% failure rate), are noisy and provide less data, that is less reliable, when compared with a laser rangefinder. Succesfull scanmatching with sonar rangefiders would provide a method for mobile robots with sonar range finders to create more reliable maps. If the sonar variant it would also provide method to do scan matching with sparse data.

1.1 Research Objectives

The main goal of my master thesis is doing a reconnoitering study of *simultaneous localization and mapping* (SLAM) in the paradigm of probabilistic robotics. One branch of SLAM research deals with *scan matching*, in which two or more scans are combined to improve the estimation of the location of the robot. In my thesis an algorithm normally used with laser range-finders is adapted to one usable with sonar range-finder. The research question of this thesis is:

> *Is it possible to adapt a laser scan match algorithm to a sonar scan match algorithm?*

The laser scan match algorithm that is adapted is described in [22] by Lu and Milo's. Experiments with the Pioneer 2DX and experiments in simulation are done to test the effectiveness of the new algorithm. Parameters are needed for some robot models. Experiments are done to estimate these as well.

Chapter 2

Theoretical Background

Research on simultaneous localization and mapping (SLAM) consists, as the name already implies, of the study of two entwined problems: *localization* and *mapping*. Localization is hard without a map and mapping is impossible without proper knowledge of where you are. Various simplifications of this problem allow us to look at different subproblems of SLAM. For example, current state of the art localization techniques are very capable if the map and the starting position are known [33]. If the map is known but the starting position not, it is called the *kidnap problem*. In this case there are also various solutions that work well [33, 30, 7].

There are two different approaches on how to model the position of the robot. In the Markov approach the map is sliced in smaller portions to create a topological network. For each element of the Markov network the probability is calculated that the robot is there at that time [20]. Though this is a robust approach, it takes more computational power but is less susceptible to *catastrophic failure*, an unrecoverable error or deviation, when compared to the second option [10]. Which is that robot position is modeled with a probability density function (p.d.f.). I will use the latter, because there is more literature covering it. I will ignore the former because investigating this as well lies out of the scope of my master thesis.

An other often applied simplification is the use of landmarks. Landmarks are distinctive features in the environment of the robot. For example electromagnetic beacons embedded in the ground or the colored goals in robocup. Landmarks were used in the beginning of SLAM research. They work easily and good as long as the robot knows which landmark it is sensing. Landmarks are used to get an better location estimation instead of only using *odometric* information. Sometimes landmarks are indistinguishable or unavailable. Non-landmark techniques are developed for these scenarios [12]. Combinations of both are used as well [9]. In this study landmarks will not be used. Landmarks require consistent recognition, which would have taken too much time to implement and test properly.

In the case of simultaneous localization and mapping there are two basic approaches. One is called *on-line SLAM*. On-line SLAM processes one chunk of information at a time, be it odometric or sensor information, and in doing so it updates its knowledge of the world after each information chunk. The second solution is called *full SLAM*. This approach considers all sensor information

available in one batch and gives the most probable map afterward. The great difference is that on-line SLAM is usually implemented iteratively and operates in real time. The full SLAM calculations take much longer and are performed after all the data is available. On the other hand full SLAM is more accurate. However in most cases the increased accuracy is not worth the extra computational demands. I have chosen for on-line SLAM, because it is more plausible in a cognitive way. Also it is a more elegant solution than the brute force solution of full SLAM, considering terms like memory usage and computational demands.

To summarize the SLAM approach for this thesis:

- The position of the robot is modeled by a probability density function (p.d.f.).
- Landmarks are not used.
- on-line SLAM is implemented instead of full SLAM.

In the next sections probabilistic robotics theory is discussed. The most important aspect of this theory is the Bayes filter. In the following section three applications of the Bayes filter and how they are applied are discussed. The third section discusses different pose estimation algorithms and how they interact with the Bayes filter. The fourth section explains in detail a specific scan match algorithm. This technique is adapted from laser range finder usage to sonar range finder usage.

2.1 Probabilistic Robotics

Probabilistic Robotics is a relatively young approach to robotics. It emphasizes uncertainty in robot perception and action. Using probability theory it is able to represent this uncertainty explicitly. It utilizes probability density distributions instead of a single "best guess". This way it can model ambiguity and belief in a mathematically sound way. To quote the writers of the book Probabilistic Robotics [33]:

> A robot that carries a notion of its own uncertainty and that acts accordingly is superior to one that does not.

In the next subsections I will explain how probabilistic robotics models the world with state and how it models the position of the robot in the world over time. The interaction with the world and this interaction influences the knowledge about the position of the robot is discussed thereafter. The Markov assumption and its consequences are discussed next. This leads to Bayes filter, which is the foundation on which most probabilistic robotic research rests.

2.1.1 State and Belief

If robots are to interact with the world, they need to have knowledge about the world. Exceptions are the Braitenberg vehicles [1], flocks [25] or similar reflex based robots. For example, an autonomous vacuum cleaner robot needs to collect dust in a room. To be able to clean the room properly it should have a well designed body, so it can collect dust in corners. It must have decent sensors to detect dust and the room with furniture, and a well designed computer program to control the robot [34]: the robot must find the dust in the room and avoid obstacles. To be able to do this, the program needs a good model or description of the world and of the robot itself.

Most robots use models of the world. These descriptions can consist of either the robot's location and orientation in the world or a map of the world or anything else you want your robot to consider. This description is called *state*. State at time t is denoted as x_t. State can be dynamic, such as the location of the robot or state can be static, for example a map. In my thesis state x_t is defined as the robot pose. The robots I work with are harbored in a 2 dimensional environment. As such the robot pose consists of x and y coordinates and orientation.

A sequence of states is used to describe the robot in time. As the state sequence gets longer and longer, the difference between the modeled state, called belief (bel), and the actual state in the real world increases. Probabilistic Robotics tries to model the uncertainty of the robot with belief. Belief is defined as the probability density function of the chance that the world is in state x_t. Using statistical calculus and clever algorithms it tries to minimize the uncertainty in belief and the difference between it and the actual state of the robot.

$$bel(x_t) = p(x_t) \tag{2.1}$$

2.1.2 Environment interaction

A robot has two different types of interaction with the world. It can perceive or manipulate the world. In other words the robot can gain information about the state of the world or it can change this state.

- Sensor measurements: z stands for all the information by which the robot perceives the world. For example laser/sonar range finders, a camera or a touch sensor. The sensor measurements on time t will be denoted as z_t through out this thesis. Sensor data from t_1 till t_2 is denoted as $z_{t1:t2}$
- Control actions, u stands for the actions by which the robot changes the state of the world. For example by movement or actuators. Odometers are considered control data, because they measure the turning of the wheels and as such control the wheel gyration. State change on time t will be denoted as u_t throughout this thesis. Control data from t_1 till t_2 is denoted as $u_{t1:t2}$

There is one important difference between sensor measurements and control actions. Sensor data increases the robots knowledge about the environment, thus decreasing the total uncertainty in the next state. Control actions, on the

other hand, are always performed with a bit of noise. For example a robot wheel may slightly slip. Control actions always add some uncertainty to the next state. A more detailed explanation will follow later on.

In probabilistic robotics belief is represented by conditional probability distributions [33]. It assigns a probability to every possible state. To determine the belief of state x_t, you need to take into account everything that happened before t. In other words $bel(x_t)$ is dependent on all past sensor and action data. This is shown in following conditional:

$$bel(x_t) = p(x_t|z_{1:t}, u_{1:t}) \tag{2.2}$$

Later on it will be necessary to calculate $bel(x_t)$ without using the newest sensor information. This will be denoted as:

$$\overline{bel}(x_t) = p(x_t|z_{1:t-1}, u_{1:t}) \tag{2.3}$$

2.1.3 The Markov Assumption

If state x_t is the best estimator for the future it is called a complete state [33]. State x_t being the best estimator means that states x_0 to x_{t-1} are not necessary for predicting state x_{t+1} and that the only necessary information for predicting state x_{t+1} is state x_t. State x does not have to be discrete and deterministic it can have continuous and stochastic variables. Of course it is impossible to model the perfect complete state for a robot but if made well enough a model can be assumed complete.

Time processes that comply to completeness as defined above are commonly known as Markov chains [33]. Markov chains have the Markov property:

$$p(q_t|q_{t-1}, q_{t-2}, ..., q_0) = p(q_t|q_{t-1}) \tag{2.4}$$

This a first order Markov assumption. A second order Markov assumption would mean that in the conditional on the right side of equation 2.4 q_{t-2} would be used as well as a prior knowledge. This Markov property is an important property for probabilistic robotics. As long as the modeled state is complete enough this allows recursive solutions for localization, mapping and comparable estimation problems. The Markov assumption is very important for the Bayes filter and keeps the mathematics tractable.

2.1.4 The Bayes Filter

A well known method to calculate belief in robotics is the Bayes filter. The use of the Bayes filter and variants is widespread among the SLAM research corpus (some examples:[27, 32, 30]) and it plays a central role in probabilistic robotics and its literature. The recursive function requires $bel(x_{t-1}), u_t$ and z_t to calculate $bel(x_t)$ is shown in equation 2.5.

$$bel(x_t) = \eta p(z_t|x_t) \int p(x_t|x_{t-1}, u_t) bel(x_{t-1}) dx_{t-1} \tag{2.5}$$

To provide extra insight in the importance of the Markov assumption and the inner workings of the Bayes filter I will give a short prove of the correctness of the Bayes filter by induction. The proof comes from [33]. It will show that

the Bayes filter correctly calculates $Bel(x_t)$ from $Bel(x_{t-1})$. The Bayes rule [17] is first applied to equation 2.2:

$$\begin{aligned} bel(x_t) &= p(x_t|z_{1:t}, u_{1:t}) \\ &= \frac{p(z_t|x_t, z_{1:t-1}, u_{1:t})p(x_t|z_{1:t-1}, u_{1:t})}{p(z_t|z_{1:t-1}, u_{1:t})} \\ &= \eta p(z_t|x_t, z_{1:t-1}, u_{1:t})p(x_t|z_{1:t-1}, u_{1:t}) \end{aligned} \tag{2.6}$$

The term $p(z_t|z_{1:t-1}, u_{1:t})$ is the same for every possible configuration of state x_t and thus is considered a constant. Since the Bayes filter is a probability density function, η is used to make the total probability one. η normalizes the Bayes filter. State x_t is known. The application of the Markov assumption renders all knowledge before this state irrelevant. $p(z_t|x_t, z_{1:t-1}, u_{1:t}) = p(z_t|x_t)$ This leads to following equation:

$$\begin{aligned} bel(x_t) &= \eta p(z_t|x_t)p(x_t|z_{1:t-1}, u_{1:t}) \\ bel(x_t) &= \eta p(z_t|x_t)\overline{bel}(x_t) \end{aligned} \tag{2.7}$$

The theorem of total probability is applied to $\overline{bel}(x_t)$.

$$\begin{aligned} \overline{bel}(x_t) &= p(x_t|z_{1:t-1}, u_{1:t}) \\ &= \int p(x_t|x_{t-1}, z_{1:t-1}, u_{1:t})p(x_{t-1}|z_{1:t-1}, u_{1:t})dx_{t-1} \end{aligned} \tag{2.8}$$

The Markov assumption is applicable again. If state x_{t-1} is known knowledge before $t-1$ is irrelevant. $p(x_t|x_{t-1}, z_{1:t-1}, u_{1:t}) = p(x_t|x_{t-1}, u_t)$. It is obvious that u_t can be omitted from $p(x_{t-1}|z_{1:t-1}, u_{1:t})$, since it does not convey information about state x_{t-1}. Equation 2.9 and equation 2.7 together form equation 2.5.

$$\begin{aligned} \overline{bel}(x_t) &= \int p(x_t|x_{t-1}, u_t)p(x_{t-1}|z_{1:t-1}, u_{1:t-1})dx_{t-1} \\ &= \int p(x_t|x_{t-1}, u_t)bel(x_{t-1})dx_{t-1} \end{aligned} \tag{2.9}$$

The Bayes filter has two steps. First it incorporates u_t in the distribution of $bel(x_{t-1})$. This step is the state update step or is called prediction. To update $bel(x_{t-1})$ it uses the *state transition probability*: $p(x_t|x_{t-1}, u_t)$. The second step is the correction step. $\overline{bel}(x_t)$ is corrected with measurements z_t. Which is done by the *measurement probability*. Finally $\int bel(x_t)dx_t$ has to be 1 for it to be a probability density function (p.d.f.). It is normalized by η to make it a p.d.f..

Algorithm 2.1.1: BAYES FILTER$(bel(x_{t-1}), u_t, z_t)$

for $\forall x_t$

do $\begin{cases} \overline{bel}(x_t) = \int p(x_t|u_{xt-1})bel(x_{t-1})dx_{t-1} \\ bel(x_t) = p(z_t|x_t)\overline{bel}(x_t) \end{cases}$

Normalize $bel(x_t)$

return $(bel(x_t))$

Algorithm 2.1.1: The Bayes filter

2.2 Bayes Filter Implementations

There is an abundance of possible implementations for the *state transition probability* and the *measurement probability* an thus there are plenty of implementations of the Bayes filter. The simplest implementations are for finite state machines in which simple numerical solutions can be found [33]. The next step up are probability estimators which use Gaussian distributions. They try to estimate the mean and covariance of the estimatee. The most important example is the Kalman filter [36]. It has an entire franchise of upgrades and variations: the extended Kalman filter [33], the unscented Kalman filter [35] or the information filter [33]. There is a rich body of knowledge and literature about this.

There are situations where the unimodal Gaussian distributions cannot be used for a Bayes filter implementation. For example when there are 2 distinct hypotheses a unimodal distribution cannot model this ambiguity. In such cases numerical approaches are used with good results. In the probabilistic robotics literature they are called particle filters. Particle filters are used for localization [30] or for estimation of a driven path [32].

I will give a short overview of the Kalman filter, the particle filters and occupancy grids, which is a Bayes implementation often used for mapping.

2.2.1 The Kalman filter

The oldest en best understood Bayes filter implementation is the Kalman filter. Kalman published his paper in 1960 [19]. Since then it has undergone extensive research and it has spawned a couple of variations. It is widely used in the area of autonomous systems. The Kalman filter calculates belief in such way that it is solely applicable to continuous states. It is incompatible with discrete or hybrid states. It is part of the *Gaussian Filters* family [33]. Which means that the belief is modeled by a multivariate normal distribution.

In the Kalman filter belief is modeled as the first two moments of a probability density function: The mean μ_t and covariance Σ_t. The Kalman filter makes two assumptions about the state model in addition to the Markov assumption of the Bayes filter. These assumptions make sure that bel_t is normal multivariate distribution [33].

- The first assumption is that the state transition probability, $p(x_t|x_{t-1}, u_t)$, and the measurement probability, $p(z_t|x_t)$, are linear functions with Gaussian noise. They are expressed by the following equations:

$$x_t = A_t x_{t-1} + B_t u_t + \varepsilon_t \tag{2.10}$$

$$z_t = C_t x_t + \delta_t \tag{2.11}$$

  x_t, u_t and z_t are state vectors, with lengths, respectively, n,m and p. Matrix A_t has the size $n \times n$, matrix B_t has the size $n \times m$ and matrix C has the size $p \times n$. ε_t and δ_t are vectors that model the Gaussian noise in the equation. They have lengths of , respectively, n and p. Each element of these noise vectors has a mean of zero. The covariance Matrices that describe the noise vectors are respectively R and Q.

- The second assumption is that bel_{t-1} is a multivariate normal distribution.

Algorithm 2.2.1: KALMAN FILTER$(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$

$\overline{\mu}_t = A_t\mu_{t-1} + B_t u_t$
$\overline{\Sigma}_t = A_t\Sigma_{t-1}A_t^T + Q$

$K_t = \frac{\overline{\Sigma}_t H^t}{H\overline{\Sigma}_t H^t + R}$
$\mu_t = \overline{\mu}_t + K_t(z_t - H\overline{\mu}_t)$
$\Sigma_t = (I - K_t H)\overline{\Sigma}_t$
return (μ_t, Σ_t)

Algorithm 2.2.1: The Kalman filter

The Kalman filter is described in algorithm 2.2.1. The algorithm models belief with mean μ and covariance Σ. It calculates μ_t and Σ_t from μ_{t-1} and Σ_{t-1} using u_t and z_t. The first two lines constitute the prediction step as they incorporate u_t in μ_{t-1} and Σ_{t-1} to calculate $\overline{bel}(x_t)$. The next line describes the Kalman gain, K_t. This stands for the amount of trust in the measurement vector z_t in relation to the prediction of $\overline{bel}(x_t)$. If the K_t goes towards I,measurement z_t is trusted greatly. The consequences of this for the next two lines, the correction step, are that μ_t is corrected more towards z_t and that the covariance, Σ_t, will become smaller. If the Kalman gain is very small z_t is not trusted in comparison to $\overline{bel}(x_t)$. This means that z_t is only slightly incorporated and that the covariance decrease is tinier.

The Kalman filter is a very efficient and effective estimation technique. However it has limited applicability. It can only model linear systems and it has a unimodal normally distributed probability density. This linearity makes the Kalman filter a bit crude for most applications. As answer to the problem of linearity, the extended Kalman filter uses a first order Taylor expansion to give a better approximation of non linear systems [35]. This gives improved results but it does not deal with the problem of the fixed shape of the probability density. For example, in the kidnap problem, as explained earlier, the algorithm must be capable of having multiple hypotheses of where it could be, to finally choose the best. With a Kalman filter or a Kalman-like filter this is impossible, because of the unimodality of Gaussian distributions.

The extended Kalman filter functions as the basis for more advanced SLAM techniques [16, 12].

2.2.2 Particle Filters

Particle filters are a popular type of Bayes filter implementation. Most of the time particle filters are used as localization techniques [5, 30], or as path estimation techniques [32]. Particle filters try to model belief with a numerical approximation of the state of the world. For this they use an arbitrary amount of hypotheses of the state of the world. The distribution of the hypotheses in the state space describes the belief. This is where particle filters differ greatly from the Gaussian filters. The shape of the distributions of the Gaussians is determined by the parameters of the Gaussian. Particle filters belong to the family of the *nonparametric filters*. This means that the $bel(x_t)$ is not deter-

mined by parameters, such as mean or covariance of the distribution [33]. The mathematical derivation can be found in [5].

$bel(x_t)$ is modeled with χ_t which is a set M hypotheses in it.

$$\chi_t = \{x_{t,1}, x_{t,2}, \ldots, x_{t,M}\} \tag{2.12}$$

The particle filter algorithm is described in algorithm 2.2.2. It has two important parts. The first part processes χ_{t-1} with u_t and z_t to the proposal distribution $\overline{\chi}_t$ and W_t. First a new hypothesis$\overline{x}_{t,m}$ is sampled from $p(x_t|u_t, x_{t-1,m})$, the state transition probability. Then $w_{t,m}$ is calculated, which is the probability that hypothesis $\overline{x}_{t,m}$ could have produced sensor measurements z_t. The measurements probability, $p(z_t|x_t)$, is used for this. Distribution W_t is a probability density constructed from the weighting factors $w_{t,M}$ for all state $x_{t,M}$ elements. W_t is a probability density and as such the integral over W_t should be one. To make sure of this the weights are normalized by dividing each weight by the sum of all the weights.

The second part of the algorithm is called the *importance sampling step*. Hypotheses are sampled from $\overline{\chi}_t$ using probability density W_t to fill the set χ_t with hypotheses from $\overline{\chi}_t$. Before the importance sampling step the hypotheses are distributed according to $\overline{bel}_t$, after the re-sampling step the distribution of hypotheses is $bel(x_t)$.

Over time belief will converge to the hypotheses that produces the most likely output. Essentially this works like evolution. The best hypotheses survive.

Algorithm 2.2.2: PARTICLE FILTER(χ_{t-1}, u_t, z_t)

$\overline{\chi}_t = \chi_t = \emptyset$
for $m = 1$ **to** M
  do { Sample $\overline{x}_{t,m}$ from $p(x_t|u_t, x_{t-1,m})$
  $w_{t,m} = p(z_t|\overline{x}_{t,m})$
  Insert $\overline{x}_{t,m}$ in $\overline{\chi_t}$
Normalize W_t
for $m = 1$ **to** M
  do { Sample $x_{t,m}$ from W_t)
  Insert $x_{t,m}$ in χ_t
return (χ_t)

Algorithm 2.2.2: The particle filter

The choice of how many hypotheses to use in the filter is not without influence. There are two aspects to consider when choosing. Having plenty of particles to consider goes together with greater computational complexity. However if the chosen amount of particles is too small, the *particle depletion* problem will show up. The particle depletion problem implies that groups of hypotheses can be exhausted of particles, because there aren't enough particles. When the filter is converging, this can cause catastrophic failure. There are approaches how to deal with dynamically[4, 7]. Aspects like available computational power and filter convergence influence the amount of particles used.

2.2.3 Occupancy Grids

The robot Minerva [29] gave guided museum tours in the Smithsonian National Museum of American History for two weeks. It navigated from exhibit to exhibit avoiding people and objects. If some one was blocking her way Minerva would ask the person to move out of her way or honk her horn to get the blocker to move on. So Minerva maneuvered around in the museum. To do this she needed a valid internal map to be able to navigate and to be able to plan a path around objects to the next exhibit. Minerva used an occupancy grid.

Occupancy grids are a Bayes Filter implementation used for estimating which space around the robot is empty and which is occupied. They have been the standard mapping paradigm from 1985 when Moravec published the first article about it [23]. The map used by the algorithm is a fine grained grid. Using sonar- or laser range-finder sensor input, the algorithm tries to determine the chance for grid cell $m_{x,y}$ whether it is occupied or empty. $m_{x,y}$ is the map at coördinates x and y. If $m_{x,y}$ is 1, the algorithm is completely certain that it is occupied. If it is 0, the cell is unoccupied.

There are multiple ways to calculate this probability. In [26] three occupancy grid algorithms are compared and discussed: A probabilistic approach, a Dempster-Shafer theory approach and an approach based on fuzzy sets. The probabilistic and the Dempster-Shafer approach were both equally superior to the one based on fuzzy sets. Dempster-Shafer theory is based on finding evidence in favor and finding evidence not in favor of your estimation [24]. Therefor it uses 2 functions to handle the sensor input and calculate $p(m_{x,y})$. The probabilistic approach uses only the *measurement probability* to do this. The probabilistic algorithm is used for two reasons. The first is Ockham's Razor. Although it is not superior, the Dempster-Shafer theory is more complex than the probabilistic approach, The other reason is that the probabilistic way falls more in line with the rest of this thesis. The most straightforward implementation of the probabilistic algorithm is described by Thrun in [28].

Most variations make two important assumptions. First they consider the estimation for each grid cell an independent process. This means that in the Bayesian update algorithm only the information from state x_{t-1} from the current grid cell is used. The algorithm ignores information in other cells. This assumption allows computationally elegant and real time solutions but it is inferior to the approaches that do take these dependencies into account [28]. Unfortunately these approaches are not real time.

The second assumption is that the environment of the robot is static. Walls and doors do not change position. This simplification is of great importance to occupancy grids, see below. Unfortunately it is only valid in simple and artificial surroundings in which nothing moves except the robot itself. Adaptations to accommodate coinhabitants have been developed and work well [15, 16].

Occupancy grids work like this: sensor input, $z_{1:t}$, is used to calculate $m_{x,y}$. z can be a sonar, or laser scan combined with pose information to give the scan a location and orientation in space. Occupancy grids are calculated with a special variation of the Bayes filter. It tries to estimate a static target. Walls, occupied and empty space do not move around. The variation of the Bayes filter is called the static Bayes filter. Again I will give a small proof of correctness of the algorithm. The algorithm tries to calculate the following probability: $p(m_{x,y}|z_{1:t})$. The Bayes' rule is applied:

$$p(m_{x,y}|z_{1:t}) = \frac{p(z_t|z_{1:t-1}, m_{x,y})p(m_{x,y}|z_{1:t-1})}{p(z_t|z_{1:t-1})} \tag{2.13}$$

The static world assumption implies that past sensor readings are conditionally independent, given knowledge of map m [28]. Occupancy grid maps make a stronger claim. They claim a conditional independence for each grid cell:

$$p(z_t|z_{1:t-1}, m_{x,y}) = p(z_t|m_{x,y}) \tag{2.14}$$

Although this is technically incorrect it allows for a very convenient simplification.

$$p(m_{x,y}|z_{1:t}) = \frac{p(z_t|m_{x,y})p(m_{x,y}|z_{1:t-1})}{p(z_t|z_{1:t-1})} \tag{2.15}$$

The Bayes rule is applied this time to $p(z_t|m_{x,y})$. This gives us:

$$p(m_{x,y}|z_{1:t}) = \frac{p(m_{x,y}|z_t)p(z_t)p(m_{x,y}|z_{1:t-1})}{p(m_{x,y})p(z_t|z_{1:t-1})} \tag{2.16}$$

This is the probability that grid cell $m_{x,y}$ is occupied. The same derivation can be done for the probability that this grid cell is free: $p(\overline{m}_{x,y}$. Divide the probabilty of occupation by the probabilty of being unoccupied and a lot of difficult to calculate terms disappear.

$$\frac{p(m_{x,y}|z_{1:t})}{p(\overline{m}_{x,y}|z_{1:t})} = \frac{p(m_{x,y}|z_t)p(\overline{m}_{x,y})p(m_{x,y}|z_{1:t-1})}{p(\overline{m}_{x,y}|z_t)p(m_{x,y})p(\overline{m}_{x,y}|z_{1:t-1})} \tag{2.17}$$

$p(m_{x,y})$ obviously equals $1 - p(\overline{m}_{x,y})$. And $p(m_{x,y}|Q)$ equals $1 - p(\overline{m}_{x,y}|Q)$, for any conditioning variables Q. This turns the division in following:

$$\frac{p(m_{x,y}|z_{1:t})}{1 - p(m_{x,y}|z_{1:t})} = \frac{p(m_{x,y}|z_t)(1 - p(m_{x,y}))p(m_{x,y}|z_{1:t-1})}{(1 - p(m_{x,y}|z_t))p(m_{x,y})(1 - p(m_{x,y}|z_{1:t-1}))} \tag{2.18}$$

If you take the natural logarithm of this chance it turns into:

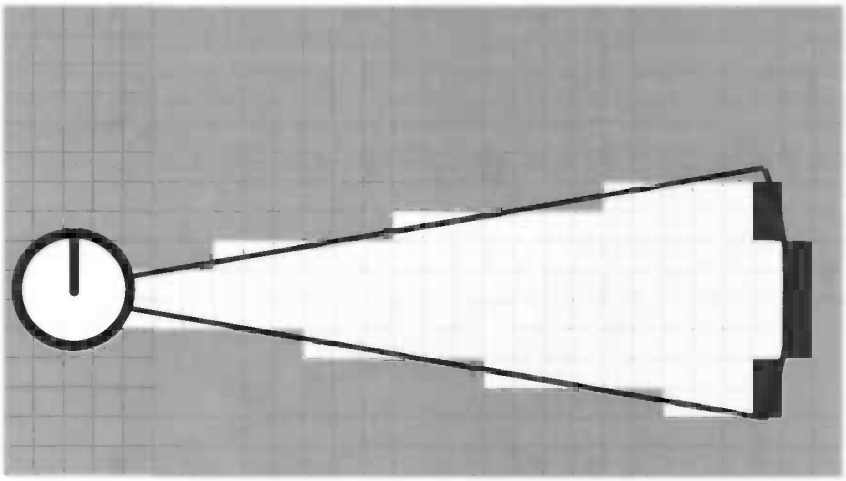
$$l^t_{x,y} = ln\frac{p(m_{x,y}|z_t)}{1 - p(m_{x,y}|z_t)} - ln\frac{p(m_{x,y})}{1 - p(m_{x,y})} + l^{t-1}_{x,y} \tag{2.19}$$

Equation 2.19 is a nice recursive function. It depends on three terms.$l^{t-1}_{x,y}$, which is the estimation from $t-1$. The term $p(m_{x,y})$ determines the initialization of the grid map and is constant. It initializes the occupancy grid with [28]:

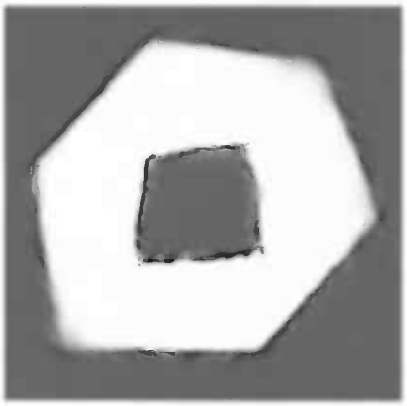
$$l^0_{x,y} = ln\frac{p(m_{x,y})}{1 - p(m_{x,y})} \tag{2.20}$$

The final and third term which defines equation 2.19 is $p(m_{x,y}|z_t)$. This term stands for the chance that according to measurement z_t grid cell $m_{x,y}$ is occupied or empty. This probability relates the sensor measurements back to its causes and is called the *inverse measurement model.* The shape of of this function is determined by the sensors and where they are mounted on the robot. They look like figure 2.1(a). My implementation is discussed in section 3.4 and is derived from the probabilistic sonar model in section 3.3.

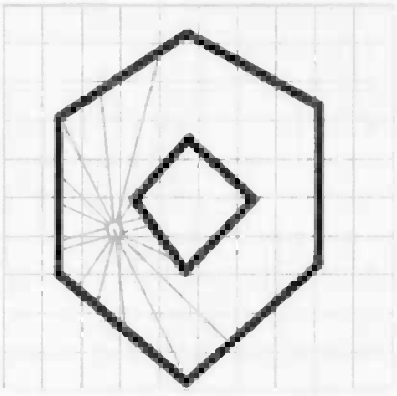
An example of my implementation of occupancy grids is given in figure 2.1(b). The reliability of the grid depends on the size of the grid and the preciseness of the range finder used. Another important cause of errors in occupancy grids is the cumulative error in the odometry information of the robot. The next section will elaborate on this.



(a) Thrun's simplistic model. The white squares are empty with a fixed probability and the black squares are likewise empty. This figure is adapted slightly from [28]



(b) An example occupancy grid map. White is most probably empty while black is occupied. Grey areas are uncertain.



(c) The simulated map in which the robot drove.

Figure 2.1: An occupancy grid example.

2.3 Pose Estimation

The main part of the research and the implementation for this thesis is about pose estimation. Pose estimation is the estimation of the difference between two states x_t and x_{t-n}. Odometers give a very good estimation of the true pose, but still give only an estimation. Over time the cumulative odometry error can grow quit large, see figure 2.2. For SLAM this is a serious problem. *Scan matching*, matching multiple sensor readings, can provide extra information such as a rotational and translational error. If these are combined with odometric information using an (extended) Kalman filter, scan matching should provide an improved pose estimation with lower variance, compared with only odometric information usage.

There are two approaches to scan matching: local and global pose estimation.

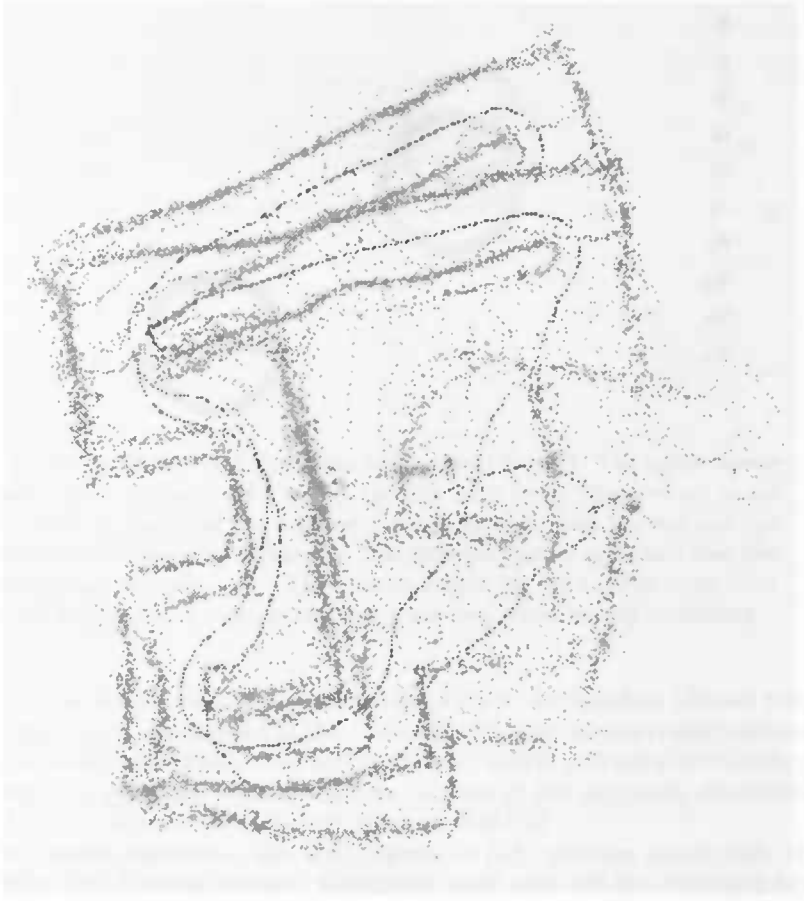


Figure 2.2: The black dots are the path driven by the robot. The gray dots are sonar readings from the surrounding walls. The cumulative error in the path grows larger as the robot drives more.

2.3.1 Local Pose Estimation

Local pose estimation tries to improve the estimation between two sequential poses. These so called scan matching techniques reveal a rotational and translational deficit of the odometry estimation, as shown in figure 2.3. By integrating sensor measurements from both states into a better pose estimation, scan matching tries to make successive poses more *locally consistent.*

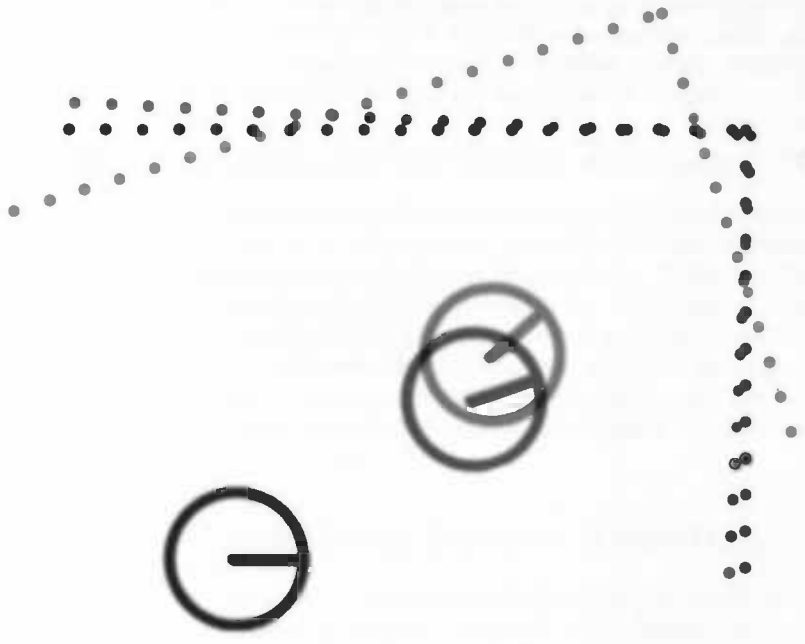


Figure 2.3: At time T-1 the robot make a scan (black). The robot moves over a short distance and makes another scan (red). Somewhere along the path of the robot the odometry made an error and the red and the black dots do not align properly. The red scan can be realigned over the black scan: the blue scan. The blue scan now fits the scan at time $T-1$ much better and states x_T and x_{T-1} are now more locally consistent.

The most widely used scan match algorithm is the Iterative Closest Point (ICP) algorithm, as described in [12]. Actually this is an adapted combination of two scan match algorithms. The first works very well in polygonal surroundings and works by matching the current scan to a set of line segments, determined from either an a-priori map or from previous scans [2].

The second algorithm, the IDC algorithm [22], matches scans with two heuristics after filtering outliers: the closest point rule and the matching rule. The closest point rule attempts to maximize point to point correspondence, while the matching rule tries to minimize a distance function, which is defined as a one dimensional search problem for orientation, and a least squares solution for the remaining two dimensions. This search/least squares algorithm is discussed in detail in section 2.4.

2.3.2 Global Pose Estimation

Global pose estimation works by inspecting the driven path of the robot for loops. If global pose estimation detects a closing loop in the robot's path it tries by using correlation filters [21] or a least squared distance function [31] to match its current surroundings with how they looked in the past. It distills a translational and rotational error from the matching process, which is used to correct the driven path. In this way the scan matching technique tries to make the path estimation topological correct or as it is sometimes called *globally consistent*. This problem of making the path estimation globally consistent is called the *loop closure problem*. This will not be discussed further.

2.3.3 Scan Matching and the Markov Assumption

To apply scan matching is accepting that the Markov assumptions in the Bayes filter do not hold entirely [11]. More sensor and pose information instead of only z_t and x_t is used to correct the Bayes filter output. While the Markov assumption acts as a sturdy and sound foundation for all the Bayes filter implementations, scan matching shows that there is not a simple answer to the questions SLAM poses. Each estimation technique, heuristic and algorithm, has different assumptions, limitations, strengths and weaknesses. The best results are produced knowing these properties and taking them into account in a concoction of different techniques.

2.4 The Search/Least Squares Algorithm

The search/least squares part of the scan match algorithm, as described by Lu and Milios in [22] is adapted in chapter 5 to sonar range finders. This section will explain the original algorithm. The interested reader should read the article by Lu and Milios for more depth and details.

The algorithm tries to align two scans to find the rotation ω and translation T that optimizes the alignment. The scan with new information is called S_{new}. This scan is aligned with reference scan S_{ref}. The algorithm has the following building blocks:

- The scan filter. Both scans are filtered for measurements that are unusable for matching the scans.
- Combining the scans. Both scans have scanned the same objects. Points that have the same correspondence to the real world are paired.
- The pair filter. Some pairs are not usable and are filtered away.
- Search. The rotation and translation for the best alignment are found by a search procedure.

2.4.1 The Scan Filter

For each point on S_{new} a tangent line is calculated using inference statistics. N neighbor points are used for this. These tangents are used for the aligning the scans.

Some of these tangents are not used. The tangent lines near occlusions or corners usually have little correspondence with the actual surroundings and make their scan point unsuitable for matching. Tangent lines have to pass two checks before they can be used in the search for ω and T. First of all the value of E_{fit} should be below a chosen threshold. E_{fit} is defined in equation 2.21. This is described in [22]. E_{fit} describes how well the used points fit to the line and a high value indicates that the scanned surface has too much curve or it indicates a range-finder error. Figure 2.4 is an example of this check.

$$E_{fit} = S_x^2 + S_y^2 - \sqrt{4S_{xy}^2 + (S_x^2 - S_y^2)^2} \tag{2.21}$$

In which S_{x^2} is the variance of the X-coordinates ,S_{y^2} the variance of the Y-coordinates of the current scan points and its neighbors. S_{xy} the covariance of both type of coordinates.

The second hindrance to be passed by the tangent lines is that the angle of the the tangent must not point straight away from the location of the robot. These tangents are very likely erroneous as surfaces aligned with the direction of the range-finders rarely produce decent data (section 3.3.3). More often they are produced by occlusions as well. A well chosen threshold for this angle is used as a check.

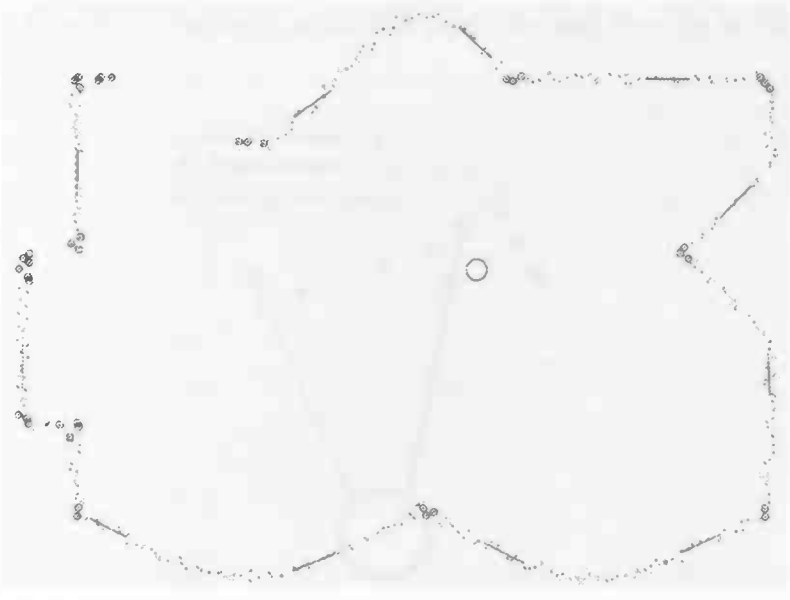


Figure 2.4: The scan filter. The picture is adapted from [22]. The circled scan points are filtered out. The lines are examples of the tangents lines fitted to the scan measurements.

2.4.2 Combining The Scans

Both scans are put together in the same coordinate system, with the robot at time of S_{new} in the origin. Invisible points on both scans are discarded. The

points on the scans are ordered by their polar angle. From the perspective of the robot in the other scan, some scan points may be ordered reverse. These points indicate a surface pointing away from the robot, a surface invisible for the robot. These points are discarded, since they cannot be used to align the scans.

The point P_1 is a point on S_{new}. P_2 is the point on S_{ref} that corresponds to the same physical point. The difference in the scans suggest an ω and translation T to correct the information from the odometry. The relation between the points is defined in:

$$P_2 = R_\omega P_1 + T \tag{2.22}$$

In which $R_\omega = \begin{pmatrix} \cos(\omega) & -\sin(\omega) \\ \sin(\omega) & \cos(\omega) \end{pmatrix}$ is the matrix that rotates the scan around the origin. The tangents at both points have their normal directions defined by the vectors $\vec{n_1}$ and $\vec{n_2}$.

To calculate the optimal translation and rotation (the next section) a method is needed to determine which point on S_{ref} corresponds with P_1. This point is P_*, the estimation of P_2, has to be chosen on S_{ref} close to P_2. P_* and P_1 form the so called *correspondence pair*. In [22] Lu and Mylo's decided to use the intersection of S_{ref} with the extrapolation of $R_\omega P_1$. The intersection has to be between two successive points that passed the scan filter. In figure2.5 is an example.

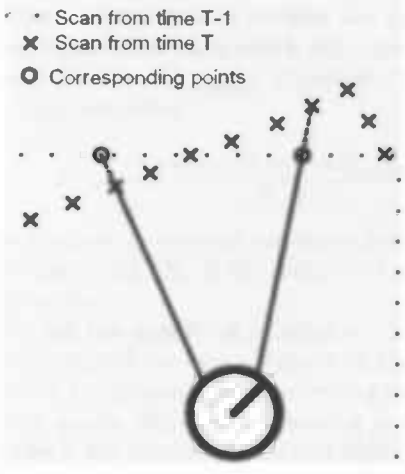


Figure 2.5: Correspondence pairs. The crosses are scan points on S_{new} that are matched to scan points on S_{ref}. The circles are their estimated corresponding points on S_{ref}.

2.4.3 The Pair filter

The pair has to pass two threshold tests before it can be used in the next step. Both points can not be too distant and the difference in directions of both

tangents can't be too large either. These are the two thresholds:

$$(R_\omega \vec{n_1}) \cdot \vec{n_*} \geq \cos(\alpha_{threshold}) \quad (2.23)$$

$$|R_\omega P_1 - P_*| \leq D_{threshold} \quad (2.24)$$

These two thresholds are important for how well the scan matcher works as a pose estimator. In the next section these threshold will return and in chapter 5 the optimal settings for these will be discussed.

2.4.4 The Search For The Optimal ω

In [22]Lu and Milios use a distance function to compare different scan alignments. The alignment with lowest distance is considered the best. The squared error of all correspondence pairs is used for this, as transcribed in the following equation from Lu and Milios:

$$D(\omega, T) = \sum_{i=1}^{N_p} ((R_\omega \vec{n_{1,i}} + \vec{n_{*,i}}) \cdot (R_\omega P_{1,i} + T - P_{*,i}))^2 \quad (2.25)$$

For each given ω the P_* and n_* are estimated. The T that minimizes the distance function is found by setting $\frac{dD}{dTx}$ and $\frac{dD}{dTy}$ to zero. This makes the distance function solely dependent on ω.

Unfortunately if there are plenty of discarded outliers, the distance D can be overestimated. In [22] they use a truncated quadratic robust estimator for this to make sure that arbitrarily bad outliers are not arbitrarily bad for the estimation. The correspondence pairs which fall outside the thresholds in pair filter will get a default penalty $D^2_{threshold}$. Equation 2.26 implements the robust distance D. The robust estimator:

$$D_{robust} = \frac{D(\omega, T) + N_o D^2_{threshold}}{N_o + N_p} \quad (2.26)$$

In which N_o is the number of omitted correspondence pairs and $D_{threshold}$ is the penalty of omitting a pair. N_p is the number of correspondence pairs that passes the threshold tests.

D_{robust} indicates the the quality of rotation ω. In figure 2.6 D_{robust} is set out against ω. This is typical for how ω relates to D_{robust}. To find the ω that minimizes the distance, techniques like hill-climbing are not suitable, because of the ruggedness of the graph. Simulated annealing is discarded as well because stochastic method like it are deemed by Lu and Milios to be too demanding on computational needs.They use the golden section algorithm to approximate the optimal ω instead.

In the golden section algorithm [18] a window around ω is searched for the optimal ω. Two points in the search window are checked for the distance. The first lies at 61,8%, the golden section, of the search window. The second at 38.2%. The point with the worst distance replace the border on its side of the search window. After enough iterations the window around the optimal ω becomes smaller and smaller, until a decent approximation of the optimal ω is found. On wikipedia there is plenty of information on the golden section.

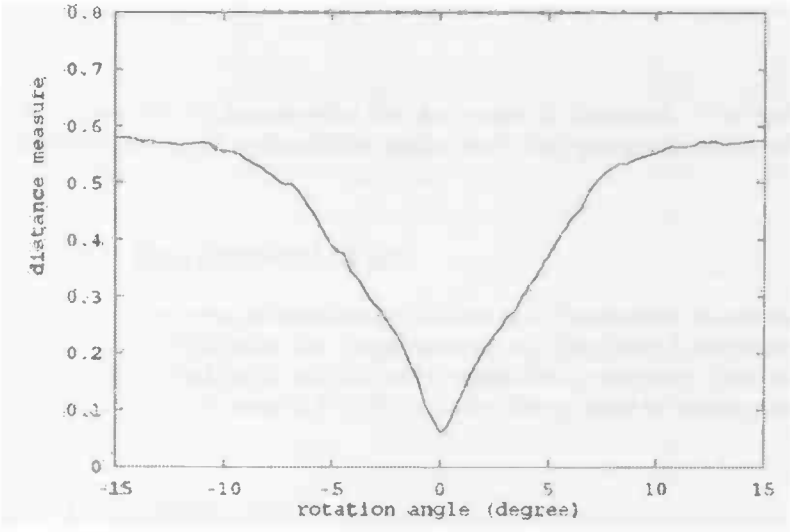


Figure 2.6: The distance function in the search window of ω. The picture is adapted from [22].

Chapter 3

Implementation and Formalization

In this chapter the implementation for this thesis is discussed. The formalizations of the necessary probabilistic models and their parameter follow afterwards.

3.1 The Implementation

Two programs have been implemented. The first is a Player agent that communicates through TCP/IP with the Player simulation. The other is the program that does all the probabilistic calculus and presents this to the user. Everything is implemented in C++ using QT. QT is a C++ library used to create graphic application.

3.1.1 The Player Agent

The Player Agent is a multi threaded program. The first thread is an object avoiding Braitenberg vehicle [1] (vehicle 2a "fear"). This thread also puts all the odometry information ,location and rotation, and the senor measurements in a Singleton class [8], named "Control". The object avoiding module is in this thread because it has to be able to steer the robot through the map without collisions. Lengthy calculations prevent the other thread from acting fast enough to do this reliably.

The "Control" class adds noise to the information from the simulation and makes the noised information available to the other thread. To generate the noisy it uses the probabilistic model that are discussed later on in this chapter. The second thread starts the Viewer program and provides it with the information it needs.

3.1.2 The Viewer

The viewer program is used to plot the maps, occupancy grids, the driven path and the particle location estimation. The output window was made using the QT 3-3 libraries. Halfway during the thesis it appeared that the wrong class was

used to be able to create movies. A Qimage object was used while a QPixmap object would have provided easier movie production.

Two variations of the viewer program exist. One gets its input from the simulation the other gets its information from the datasets. The datasets are explained in chapter 4.

These are the different functions by which the viewer produces its output.

- Path: draws the driven path of the robot.
- Location: provides a numerical estimation of the robots location.
- Naive: plots the sonar output. This makes maps as figure 2.2.
- Scanmatch: this function implements the sequence matcher as described in the methods chapter.
- Occgrid: this draws occupancy grids.
- Seqfilteroccgrid: this draws occupancy grids with the points that pass the sequence filter. This filter is described is described as well in the methods chapter.

All the probabilistic models need values for their parameters. When the viewer program is started it reads a file named "default.dat". This file contains all the parameter settings for the program itself and for the the probabilistic models.

3.2 Probabilistic Kinematic Models

To model the uncertainty of a moving robot a probabilistic model of its kinematics is needed. The *state transition probability* as described in section 2.1.4 is used for this: $p(x_t|x_{t-1}, u_t)$. Using nothing but the latest known pose and the last motion command issued, it projects a probability density function over state x_t.

In [33] two ways to model such a kinematic model are discussed: a velocity model and a odometry model. The velocity model uses rotational and translational speed as the action command u_t to calculate the state transition probability. The odometry model uses odometry information as u_t instead.

The writers of [33] consider the odometry model superior to the velocity model. While both suffer from drift and slippage. Drift is a constant bias in the odometry, while slippage is the accidental slipping of the wheels. The velocity model suffers from a mismatch between actuators and crude mathematics as well. However the velocity model predicts the state one frame further, since odometry is only available after the robot has moved. The only model used in relevant literature is the odometry model [13, 10]. The velocity model is mentioned only in [33]. The odometry model has more literature coverage and the approval of the writers of [33] therefore I will use it.

3.2.1 The odometry Model

The difference between pose x_t and pose x_{t-1} consists of translation and a change of bearing. This transition is modeled in three steps. Step1: A rotation

to the direction of the translation: δ_{rot1}. Step2: The translation: δ_{trans}. Step3: A rotation to the bearing of x_t: δ_{rot2} These three deltas make the vector u_t. Figure 3.1 illustrates this.

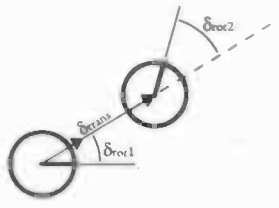


Figure 3.1: Step1:δ_{rot1} Step2:δ_{trans} Step3:δ_{rot2}

$$u_t = \left(\delta_{rot1}, \quad \delta_{trans}, \quad \delta_{rot2} \right) \tag{3.1}$$

The odometry model assumes that each element of change u_t is corrupted by independent noise.

$$\begin{pmatrix} \hat{\delta}_{rot1} \\ \hat{\delta}_{trans} \\ \hat{\delta}_{rot2} \end{pmatrix} = \begin{pmatrix} \delta_{rot1} \\ \delta_{trans} \\ \delta_{rot2} \end{pmatrix} + \begin{pmatrix} \varepsilon_{rot1} \\ \varepsilon_{trans} \\ \varepsilon_{rot2} \end{pmatrix} \tag{3.2}$$

$\varepsilon_{rot1}, \varepsilon_{trans}and\varepsilon_{rot2}$ are normally distributed deviations with mean zero and variances, respectively:

- $(\alpha_1 \delta_{rot1} + \alpha_2 \delta_{trans})$
- $(\alpha_3 \delta_{trans} + \alpha_4 (\delta_{rot1} + \delta_{rot2}))$
- $(\alpha_1 \delta_{rot2} + \alpha_2 \delta_{trans})$

The α parameters are the influences of translation and rotation on themselves and on each other. They are robot specific and need to be estimated for a valid robot model. The next section will elaborate on this. With this model, if control u_t and pose x_{t-1} are known, it is possible to calculate the probability of pose x_t. A probability density function for pose x_t can be calculated also. Sampling from this distribution is used to generate the noise for the simulation. The distribution is also used for sampling poses for Monte Carlo localization.

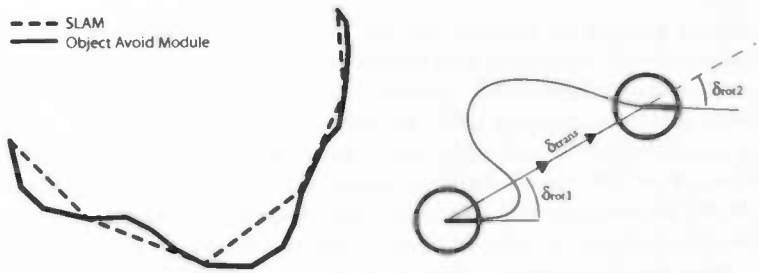
The odometry models time as discrete steps of undetermined length. The model uses only u_t and ignores the duration between the steps. Iteration of these steps can be used to model a moving robot (fig. 3.2.1). Unfortunately the kinematic model does not take into account at what speed the robot moves and therefore loses all information that could be gained this way. This is not really a problem as long as the u_t's are sampled at such a rate that the u_t succession provides a good approximation of the original robot movement.

(a) A probability density of the robots position.

(b) A numerical approximation of the robots position.

Figure 3.2: A probability density and numerical estimate.

The reason that the rotation should be split up in two parts instead of only one rotation is that the SLAM algorithm might update insufficiëntly fast, when compared to a module that steers and controls the robot, for example an *Object Avoidance* (OA) module. The OA module makes sure that the robot does not bump into object and walls. It need to react fast and reflexively. The update time is an order of magnitude smaller than the update time of the SLAM part. If a robot control module would steer the robot in an S shaped curve between a SLAM update (figure 3.2.1), control u_t would have a too small bearing adjustment. The two rotations in the model are to make sure that the variance induced by bearing change is not underestimated. For an example look at figure 3.2.1.



(a) Update time of Object Avoiding vs SLAM. The u_t vectors for SLAM are longer and cover more time than those of OA.

(b) Riding a S-curve.

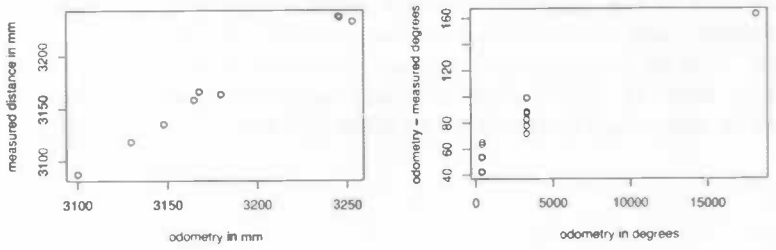
Figure 3.3: SLAM update times

3.2.2 Parameter estimation

The α parameters and how to estimate them lack sufficient literature coverage. In the course "robotica" [14] a comparable odometry motion model [14]

was used. This model used slightly different parameters. These parameters were estimated by experiments. Some of the experiments are re-usable for the odometry model from [33].

To estimate α_3, the variance in translation caused by translation, the robot rode three meters in a straight line. The odometry readings and the exact measured distance were compared ten times. A regression analysis from odometry information to measured distance was made from this. We used the regression line to estimate the variance of the point estimation of the measured distance as described in [17]. Scaled properly this variance is α_3. The standard deviation for driving 1 meter is 35mm.



(a) Driving roughly 3 meters.

(b) Driving 5, 10 and 50 circles.

Figure 3.4:

The same procedure is used to get the bearing on bearing variance. The pioneer spun around for 5 round, 10 rounds and 50 rounds. The angle as given by odometry is compared with the measured one. The scaled standard deviation for this is α_1, Which is 0.03π for 1 π turning. The experiments shows that turning when standing still first gives a big initial bias. Another experiment was done in which the pioneer rode 50cm straight ahead, turned 90° and repeated this a number of times. This experiment showed no startup error for the measured angle. The startup error is ignored. When the robot is standing still and starts driving it always shocks a bit. This is probably caused by loose axels.

No experiments which would cleanly estimate the other two α's were done. Looking at the test results from the other experiments a rough estimation is made that seems to work quite well:α_2, the rotation variance caused by translation, is certainly not more than 1 degree per meter. There was not a lot of angle variance in the experiment for α_3 and most of that variance was probably caused by slight θ variance in the starting conditions. 0.01π per driven meter is probably an overestimation of α_2.

There was no notable translation in the rotation experiment. The newest version of the stage simulation uses a cruder form of this noise model. Stage's documentation shows that α_4 is omitted. Therefore α_4 is removed from the implementation. This a reasonable assumption because of the reasons above.

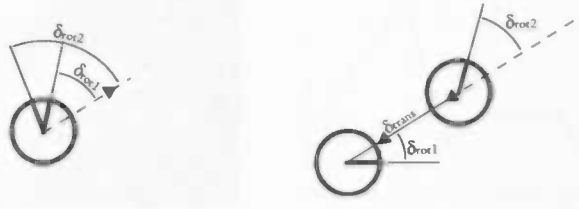
To summarize, these are the α values used throughout the thesis:

- $\alpha_1 = 0.03\pi$ per π rotation.
- $\alpha_2 = 0.01\pi$ per driven meter.
- $\alpha_3 = 35mm$ per driven meter.

3.2.3 Model adaptations

After the model was implemented there were two problems that created explosive amounts of noise in test runs. In these test runs the model was used to add noise to the odometry information from the simulation. Roughly ten thousand hypotheses estimated the robots position. Every frame sampled for every hypothesis a relative pose update from the noise model and updated the hypothesis with this update. The distribution of the hypotheses is the same kind of approximation used in Monte Carlo Localization. For an example look at figure 3.2.1. In two scenarios the particle acted with too much variance.

When the robot drove backwards the first problem reared its head. The two rotational delta's would both be almost π, figure 3.5(b). In other words the bearing needed a 180 degree flip, when the robot drives backwards, to fix this.



(a) A very small translation under an awkward angle can cause to much variance.

(b) If the robot moves backwards, the bearing needs to flip 180o.

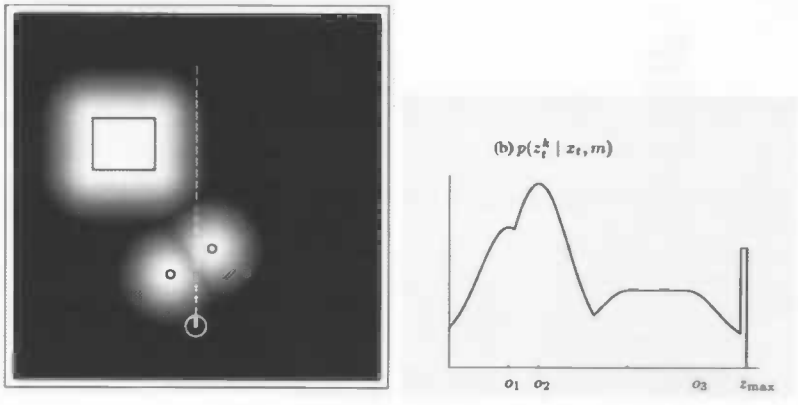
Figure 3.5: Problematic situations.

When using odometry data from the pioneer robot, from the standard research setup in section 4.1, instead of the noised simulation data a slight translation noise created too much variance in the particles. Normally when the robot is moving this type of noise is too small to have any significant consequences. There is the possibility though that, when the robot is standing still or only rotating, there is translation noise in a random direction, figure 3.5(a). The possibility rises that the bearing difference between two states is 1° and that the translation noise happens in a 44° direction. δ_{rot1} rotates to the direction of the translational noise. δ_{rot2} rotates back to the new bearing of the robot. In this scenario there is suddenly a humongous and unrealistic amount of variance introduced. The following adaptation is made: the rotation is not split up in 2 parts for state change u_t. If the translation is smaller than 4 standard deviations of noise caused by rotating ($4\sqrt{\alpha_4}$). This fixed this problem. The justification

for this perhaps ad-hoc seeming fix is that when the robot is in the situation of causing the problem, it can never ride an S-curve and as such it does not need to accommodate the extra variance precautions for it.

3.3 Probabilistic Sonar Beam Models

A probabilistic model of the sonar beam of our pioneer 2 robot is needed to do calculations with the *measurement probability*, $p(z_t|x_t)$, and to be able to generate realistic noise for the simulation. In [33] two models are presented. The first option is an approximative physical model. By combining four different types of measurement errors it forms a computationally very cheap measurement probability. How will be explained later on. The other option is better suited for cluttered environments and works with likelihood fields. Figure 3.6 is an example of a measurement probability derived from such a field.



(a) There are three objects interfering with the measurement of the range finder. The whiteness indicates the likelihood of producing a measurement.

(b) The resulting density function from the likelihood approach.

Figure 3.6: Adapted from [33]

When compared with the approximative physical model the likelihood fields are more realistic, though it is complexer and more computational demanding approach. It takes more time to implement and test properly as well. Only empty environments are used in this thesis, therefor the extra realism of the likelihood fields would be a minimal gain and not worth its price. The approximative physical model is used instead.

3.3.1 The noise model

To calculate $p(z_t|x_t)$, the model uses z_{sim}, the exact distance provided by the range finder in the simulation.

The model breaks down the measurement probability in four different conditionals as is shown in equation 3.3 and figure 3.7. There is a chance that the sonar suffers from unexplainable noise. This results in a uniform distribution over the reach of the sonar range finders (p_{random}). There is also a chance that the sonar beam hits an object and bounces away(p_{fail}). The range finder senses no object and gives maximum range. The third possibility is that an object close to the range finder beam interferes so that the range finder produces a shorter distance than z_{sim} (p_{short}). This probability is exponentially distributed [33]. If none of these three situations occur, the sonar range finder produces results normally distributed around z_{sim} (p_{hit}). This specification results in the following equation:

$$\begin{aligned} p(z|x_t) &= Z_{hit}p_{hit}(z|x_t) + Z_{short}p_{short}(z|x_t) + \\ & \quad Z_{random}p_{random}(z|x_t) + Z_{fail}p_{fail}(z|x_t) \end{aligned} \tag{3.3}$$

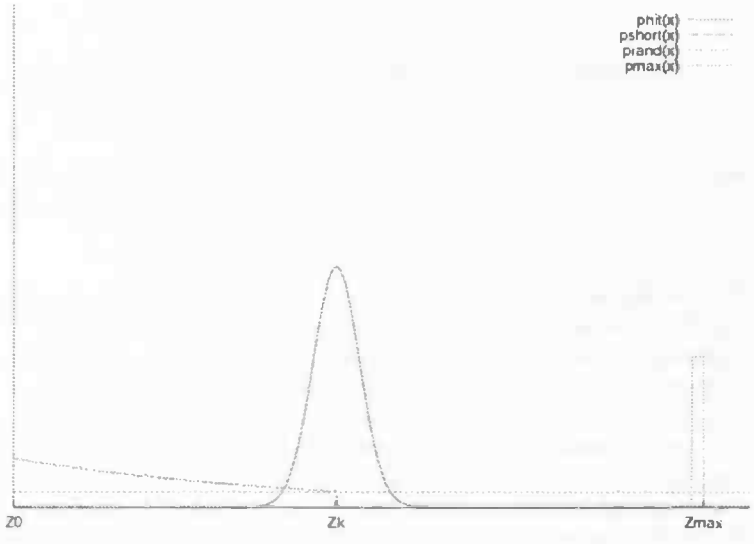


Figure 3.7: Constructing the measurement probability from Zk.

In other words, if there is no inexplicable noise (*random* hits) and if there is no failure (*fail* hits) and if there are no other objects interfering with the range finder (*short* hits) than z is normally distributed around z_{sim} (*hit* hits).

3.3.2 Parameter estimation

This model has four parameters to estimate. To estimate these parameters empirical results are needed. The course "robotica", [14] indicated that the sonar range finder is far from perfect. For example if the beam hit an object at an angle greater than a certain angle, *the critical angle*, the sound wave would bounce away and the range finder produced erroneous results. Also there is a slightly positive bias if the measured distance is shorter than a meter and a

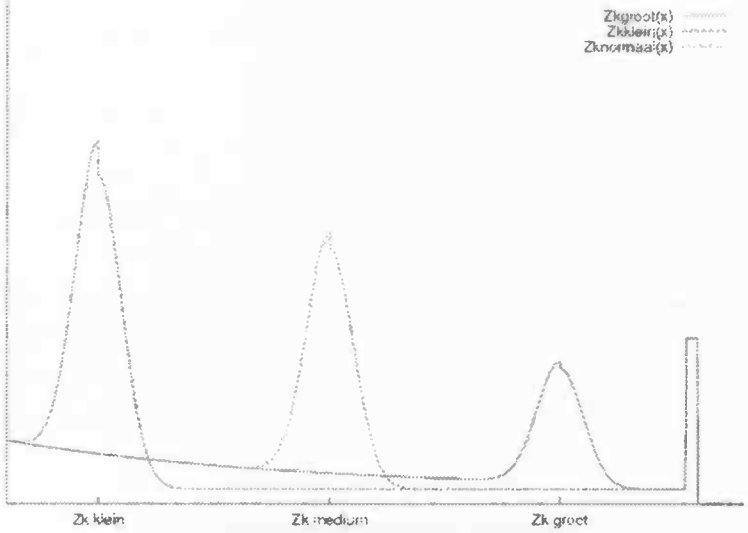


Figure 3.8: Different Zk values generate different pdf's.

slightly negative bias if z was greater than 1 meter. Figures 3.9(b) and 3.9(a) show the results of these experiments.

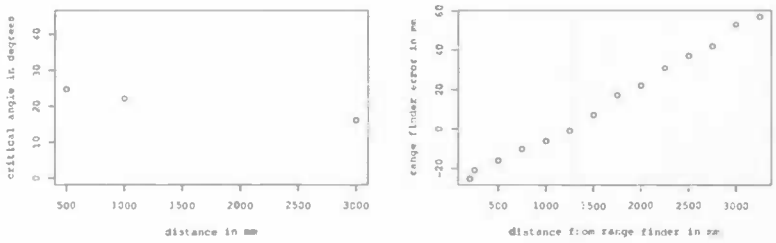
More experimentation was needed. The pioneer robot was placed with θ 25° and 35° on variable distances from a wall (figure 3.10). The center of the pioneer is placed 50, 100, 150 and 200 cm away from the wall. The configuration is such that the angle under which the sonar beam hits the wall is varied from 5° to 85° with variable distances. Each range finder produced a dataset for each θ and distance from the wall with 2500 readings. In these datasets there are almost no *short* and *random* hits (figure 3.11(a)). There are a lot of failures. When the angle is larger then the critical angle there is an 100% failure rate.

The experiment was too noisy to say anything conclusive about the bias, so concerning the bias I will use the findings pictured in figure 3.9(b). The bias is omitted in the simulation. When SLAM is done on the real pioneer the sensor output of the range finders will be adjusted instead. In figures 3.11(a) and3.11(b) each dataset minus it's mean is plotted . Failures are omitted. There are very few outliers. Notice the tight Gaussian bell around the mean.

3.3.3 Model adaptations

The experimental setup (chapter 4) used for the final experiment has no objects to generate *short* and *random* noise just like the last experiment. This does not justify *short* and *random* hits in the model. The noise model is downscaled to only the Gaussian curve and a chance of failure. This makes it less suitable for dense office environments or other noisy environments.

The critical angle is a problem and gives me two choices: The first option is to calculate in simulation the angle with which the range finder hit its object. In



(a) The critical angle becomes smaller if the detected object is farther away and less objects are detected on long range. Unfortunately the data from the course is quite sparse and no information is available between one and three meter.

(b) There is a slight bias in the distance as detected by the range finder.

Figure 3.9: Data from [14].

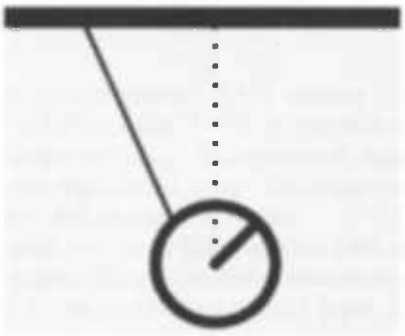
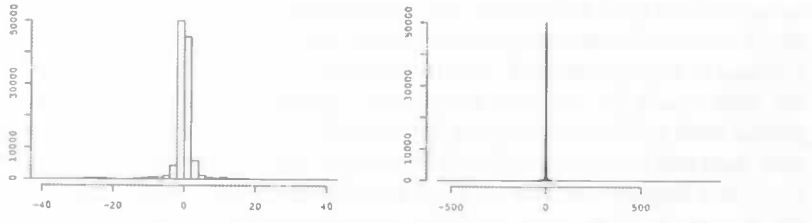


Figure 3.10: Experiment setup. The robot is placed at different distances from the wall. Data is produced by all sonar range finders. They hit the wall at different angles and different distances.

the Player/Stage simulation this is very difficult, because the map is a bitmap. Choice two is to model this type of noise independent of angle. This seemed easier and more time effective at the time. But it leads to a bit awkward solution.

Range finder failure is almost 100% if the struck angle is above the critical angle and almost 0% if below. With the sonar rangefinders of the pioneer 2DX this results in sequences of failures and succesful hits. The angle struck by the rangefinder is unknown to the simulation. This disables the possiblity to generate this noise realistically. This is not really a problem if the structure of the angle noise is not used in the SLAM algorithm. The SLAM algorithm just needs to be robust enough to handle this type of noise, sequences of failures and hits. To prepare my SLAM algorithm for the real world, noise that looks like this *angle-failure noise* would do. I modeled this noise with a counter. To this counter at random 1 is added or subtracted. The counter has a certain maximum. If the counter is above a predefined maximum or below zero the



(a) (b)

Figure 3.11: For each dataset the failures are omitted and subtracted the mean of that dataset. The following data of all the datasets are combined in these figures. The deviation around the mean is plotted against its plentitude. Two different scales are shown. On the left 4 cm scale to show the Gaussian bell. On the right 50 cm range showing no *short* and *random* hits. Failures are omitted.

counter switches to the other extreme. If the counter is above a certain level the range finder produces failure noise. If the counter below this level it produces output normally distributed around z. This results in sequences of failure noise and sequences of normally distributed noise. The sonar range finder creates the same kind of sequences of failure noise in reality.

The model as suggested by Thrun [33] considers each measurement an independent process. This angle-failure noise makes successive noised measurements not independent, which is considering the critical angle, a valid assumption.

3.4 Inverse measurement model for occupancy grids

In section 2.2.3, occupancy grids are discussed. Occupancy grids need an inverse measurement model: $p(m_{x,y}|z_t)$. The inversion means that the model relates the sensor readings back to its origins: the surroundings. It models where the object, that caused the readings, could be and the likelihood that it is there. In the literature some models are discussed. Moravec [23] does not use a static Bayes estimator and the rest of his occupancy grid mathematics are not easily adaptable to the static Bayes estimator. In [28] Thrun describes, what he calls, a "simplistic" model. This gives either an occupied or an empty output (fig. 2.1(a)) instead of a probability, as demanded by the static Bayes filter. Ribo and Pinz describe in their probabilistic approach in [26] a complex sensor model that is not explained in any detail. Unsatisfied with the enigmatic and strange models I encountered, I developed an inverse measurement model myself.

The model works somewhat the same as the probabilistic approach used in [26]. It uses two functions. The first, *the information function*, determines from measurement z the probability whether location (x, y) is occupied or not, as in figure 3.12(a) and equation 3.4. This function is solely dependent on the

distance to the range finder. The second function, *the spread function*, spreads the information from the first function over the width and length of the sonar beam. If the sonar range finder returns a distance it is unknown where in the width of the sonar beam the results were produced. This uncertainty is modeled with a normally distributed function over the isodistance, all places with the same distance to the range finder. To prevent long range readings from adding too much information, the spread function is also divided by the distance from the range finder (figure 3.12(b), equation 3.5). This way the integral over every infinitesimal slice along a isodistance has the same size. A final note: if the range finder produces a failure,that is when it returns the maximal distance, it is ignored. Otherwise failed detections introduce to much erroneous grid cell changes.

δ is the distance from the grid cell to the range finder. α is the deviation from the center axis of the sonar beam in radials. These equations describe the information and the spread function.

$$I(\delta) = \begin{cases} P(T > \delta) & \text{if } \delta > z_t \\ -.5 + 2P(T < \delta) & \text{otherwise} \end{cases} \tag{3.4}$$

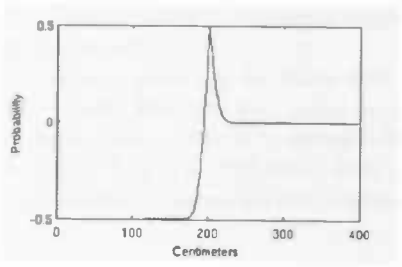
In which T is normally distributed with mean z_t and σ^2_{hit}, from the probabilistic sonar model.

$$\lambda(\delta, \alpha) = \frac{\theta P(A > \alpha)}{\delta} \tag{3.5}$$

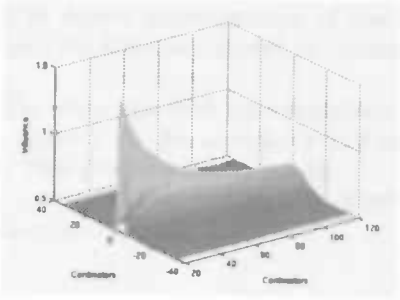
In which θ is a scaling variable. A is normally distributed with standard deviation of a quarter of the width of the sonar beam.

$$p(m_{\alpha,\delta}|z_t) = .5 + I\lambda \tag{3.6}$$

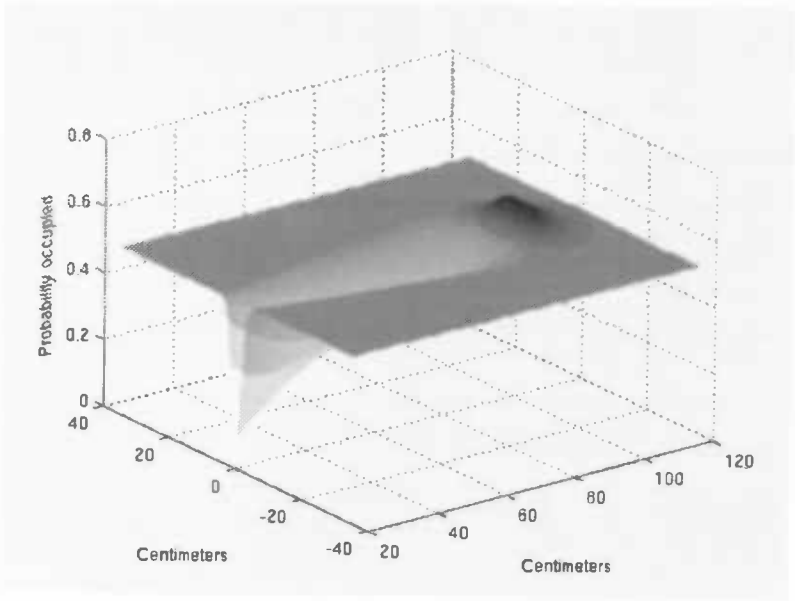
$p(m_{x,y}|z_t)$ is transformed into a location relative to the range finder with polar coordinates: $p(m_{\alpha,\delta}|z_t)$. This is calculated as in equation 3.6.



(a) The information function.



(b) The spread function.



(c) The inverse measurement model. Black is probably occupied and white probably empty. In this example the range finder produces a measurement of one meter. The rangefinder produces reliable measurements from a distance of 20 centimeters. Therefore the model strarts from there.

Figure 3.12: The creation of the inverse measurement model.

3.5 Examples

To show what the implementation actually produces, some sample outputs are made. The experimental setup, that is explained in the next chapter, is used. Figures 3.13 and 3.14 are two sonar plots. Both consist of roughly 2000 frames. The first is made using a dataset produced by a robot in the robotlab. The latter is made with the simulation. The output from the simulation is noised with probabilistic models. This results in decent approximation of the real world. There is noise in the sonar scans and the kinematic model has produced a cumulative error.

Occupancy grids can be made with the robot and with the simulation. An example made with the real robot is in figure 3.15. An occupancy grid made after the sequence filter is in figure 3.16. This produces better results.

In figure 3.14 is a blue area. This is a numerical estimation of its position. This estimation is based on the kinematic model as well.

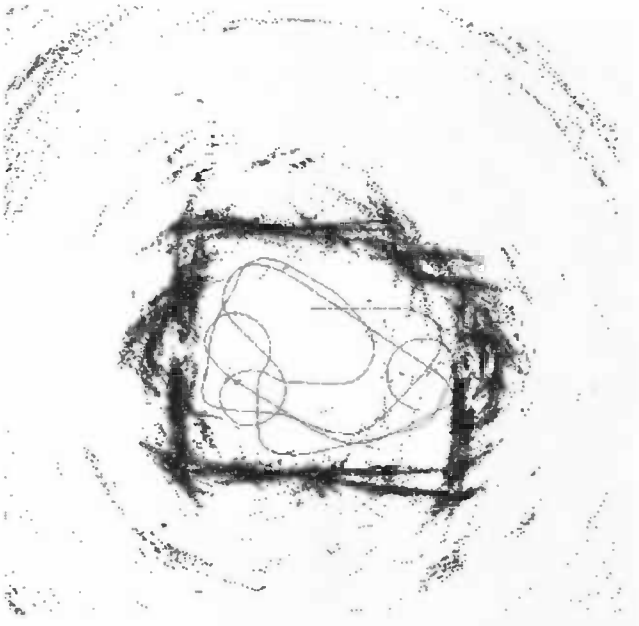


Figure 3.13: A basic sonar map with the robot. The map is relatively similar to the map of the robot lab (figure 4.2). The black line is path, that is driven by the robot.

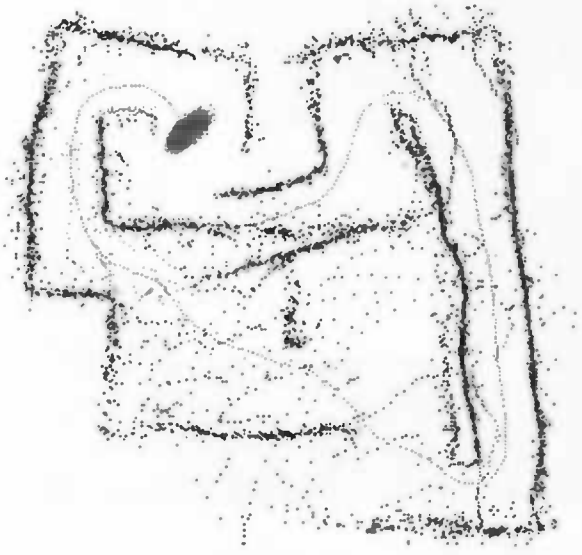


Figure 3.14: A basic sonar map with the simulation combined with a numerical estimation of the robots position.

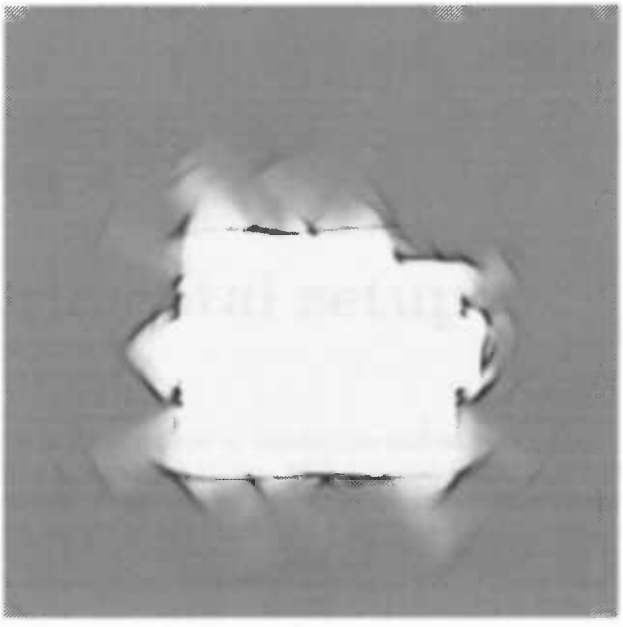


Figure 3.15: An occupancy grid map made with the robot.

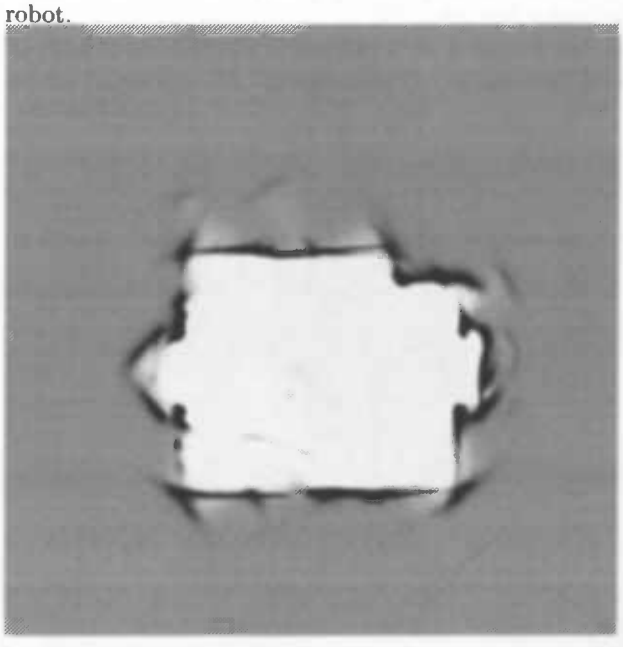


Figure 3.16: The points that passed the sqeuence filter are used to produce a occupancy grid. Data from the robot lab is used for this.

Chapter 4

Experimental setup

SLAM research is usually done in simulations and on real robots. The simulations are easier and cheaper to use, while the goal will always be to perform as good as possible on real robots. My experiments are done in both environments. The laboratory setup is explained first, the simulation setup will follow thereafter.

4.1 The Robot Laboratory

The robot laboratory at our faculty has an old fashioned robocup field with two goals. The field at our faculty is roughly 4 by 5 meters and is walled by 50 cm high boards, figure 4.1. All the experiments on robots in this thesis are performed on the field.

Figure 4.1: The robot lab at our faculty.

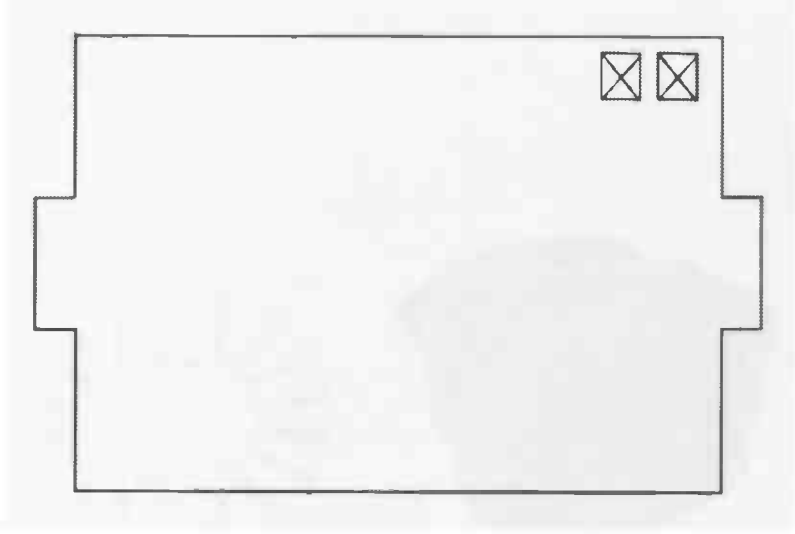


Figure 4.2: The map of the robot lab. It is five meters long and 4 meters wide. Two boxes were input in the corner to make it less symetrical.

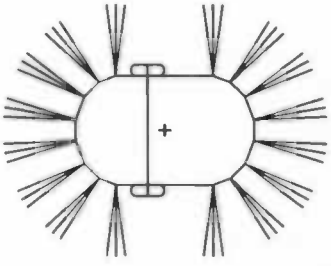
4.1.1 Pioneer 2 DX Robot

Different robots are used for SLAM research. Localization research is done on AIBOs to improve their robocup game [9]. There is a research project involving the Groundhog robot. This big four wheeled robot explores and maps abandoned and perhaps dangerous mines [16]. The robot that is most commonly used for SLAM research is the Pioneer robot of Activ media (figure 4.3(b)). Being easily programmable, remotely controllable and the added cheap range finders made the robot widely used in SLAM research.

The robots available at our laboratory are Sony's walking robot AIBO and the Pioneer 2 DX. The Pioneer robot is used in favor of the AIBO because it has accessible and reliable odometry. The AIBO lacks these. The Pioneer has sixteen sonar range finders as well. It propels itself with two wheels and uses one wheel at the back for support. The locations and directions of where the sonar range finders are mounted on the Pioneer Robot are listed in table 4.3(c) and shown in figure 4.3(a).

4.1.2 The Sonar Range Finder

Sensors are needed for mapping. Cameras are used sometimes [3, 6], but sonar or laser range finders are used more often. Processing camera data is computationally very costly and it is more difficult to determine distance to objects, when compared with range finders for obvious reasons. Most SLAM research is done with laser range finders, because of the precise plentitude of distance data it provides. The combination of camera and range finder has been researched



(a) A simplified pioneer model, with wheels and rangefinders.

(b) A picture of the Pioneer in action.

N	X	Y	α
1	0.115	0.130	90
2	0.155	0.115	50
3	0.190	0.080	30
4	0.210	0.025	10
5	0.210	-0.025	-10
6	0.190	-0.080	-30
7	0.155	-0.115	-50
8	0.115	-0.130	-90
9	-0.115	-0.130	-90
10	-0.155	-0.115	-130
11	-0.190	-0.080	-150
12	-0.210	-0.025	-170
13	-0.210	0.025	170
14	-0.190	0.080	150
15	-0.155	0.115	130
16	-0.115	0.130	90

(c) Mounting positions and directions of the sonar range finders on Pioneer 2DX.

Figure 4.3: The pioneer 2DX robot.

as well, providing techniques to make a detailed three dimensional map of the surroundings to which texture maps from the camera are added [16].

Generally there are two types of range finder sensors: Sonar and laser range finders. Laser scanners consists of one laser. Every frame the laser is rotated over an arc, usually 180 degrees, and gives for example every half a degree the distance to an object. They are very precise and can give a detailed and precise snapshot of their surroundings. The other archetype range finder is the sonar scanner. The sonar range finder emits a sound wave. After the sound waves bounces back from an object, it returns back to sensor. Using the delay between the departure and arrival of the sound wave the range finder determines the distance of the detected object. There are a couple differences between sonar and laser range finder. The first is the width of the beam with which the range finder works. The laser range finder works with a very tight beam, which gives the scan, given by the range finder, a good resolution. The sonar range finder on the other hand uses a beam which is approximately 30 degrees wide. The resolution of the scans provided by the sonar range finder are rather crude compared with the other variant.

Only sonar range finders were available at our faculty when I started with my thesis, so I was restricted in my range finder choice. At the moment a laser range finder is available. The sonar range finder has maximum range of three meters and a minimum of roughly 20 centimeters. The beam has an angular width of fifteen degrees.

4.2 The Simulation: Player/Stage

SLAM research should always be tested on real robots, but it is very dependent on good simulation testing as well. Experiments in simulation are easily repeatable and variable, and simulations are cheaper in maintenance and purchase than real robots. There are some requirements for a simulation before it can be properly used. The simulation needs to be fast, realistic and controllable for real-time SLAM research to be useful. The saphira simulation at our faculty was lacking in the latter two aspects. A bit of research revealed the player/stage simulation.

Player is network server software for controlling and communicating with robots. It can be used with either real robots or with their digital simulacra in Stage or Gazebo. Stage is a multi robot 2D simulation using fairly simple, computationally cheap models of lots of devices rather than attempting to emulate any device with great fidelity. Figure 4.2 is an example. Stage can reliably model fast amounts of simple robots. Gazebo is used in 3D simulations for more accurate simulations. Both communicate with their clients through a TCP/IP protocol. The Player/Gazebo/Stage combination has an active open source community on playerstage.sourceforge.net. Bugs are removed regularly and new improvements and device drivers are written on a regular basis. The software came with some example worlds and robots. The geometry of the pioneer robot, handily supplied in the Player/Stage distribution, is used trough out this thesis.

The stage simulation version does not model noise. To make sure the SLAM algorithms are robust enough, it was needed to add sufficiently realistic noise to the simulation of the pioneer robot. For this probabilistic noise models are

needed: a model for kinematics of the robot and a model for the sonar range finders. These are discussed in chapter 3.2.

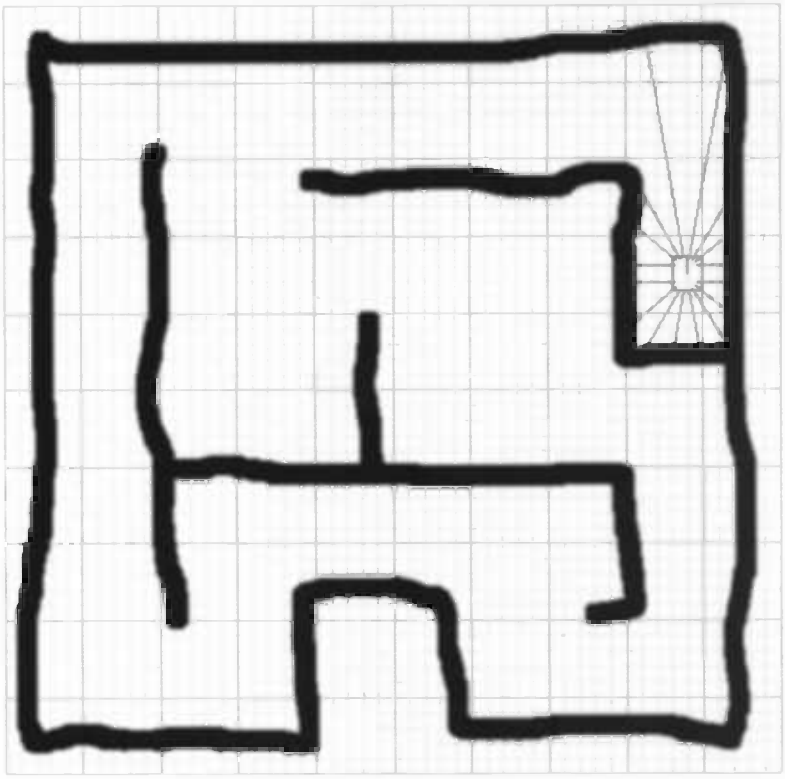


Figure 4.4: The simulation Stage. A map with a pioneer robot in it. The green beams are the sonar range finder measurements.

4.3 Datasets

Datasets are used in the experiments in favor of real-time experiments. These datasets are processed as if they were produced real-time by the robots or the simulation. This way my experiments are performed faster and they are repeatable. This is important for comparing different techniques. Using real-time information was impossible on the pioneers anyway. The computers inside the robots are too slow to perform the calculations.

The datasets consist of odometry information and sonar information, that is x- and y coördinates, orientation and the measurements of the sixteen sonar range finders, mounted on the robot. Datasets are made with the pioneers as well as the simulation.

Four pioneer 2DX robots are available in the robot lab: Sam, Frodo, Merry and Pippin. During the first experiments eight sonar range finders on the robot

Frodo were found to malfunction. This made the robot unusable for experiments. Sam needs a new mainboard. With Sam and Frodo unfit for duty, Pippin and Merry produced all the robot datasets used in this thesis.

The datasets produced in the robot lab are made with both the usable pioneers. An object avoiding Braitenberg vehicle controlled the robot in the robot lab, while the robot put the necessary data in a datafile. Two boxes were put in the corner to break the symmetry of the map to make left from right distinguishable.

The simulated datasets were made in the player/stage simulation, using the map in figure 4.2. Again the object avoiding agent piloted the robot while the needed data was stored in a data-file.

Chapter 5

Methods: Sonar Scan Matching

In section 2.4 a scan match method is explained. This method is usually used with a laser range finder. If this method is applied using sonar range finders without adapting the method, it runs into problems. The inferior precision of the sonar range finder introduces more variance. About forty percent of the sonar measurements are, usually incorrect, maximum range readings. Also sonar beams sometimes bounce twice. This results in wrong readings. Also the 16 measurements are far fewer then the amount of laser scanners produce. These add up and make it very unlikely that acceptable correspondence pairs are found, as described in section 2.4

Instead of comparing two scans from different times, the scan match method is adapted to compare sonar sensor measurements with sequences of measurements of the same range finder.

This approach is chosen because when compared with the original algorithm, this allows more acceptable tangents and more correspondence pairs. There will be more acceptable tangents when using sequences instead of scans, because the mere sixteen range finders and the geometry of how they are mounted on the Pioneer 2DX do not easily allow scan points that are right next to each other. Using a sequence from one range finder allows plenty of readings that are close to each other. This allows more correspondence pairs. In the original each scan point can produce only one correspondence pair. In the new algorithm all scan points on S_{new}can form correspondence pairs with all the sequences, even the sequence belonging to range finder that produced the reading. With the Pioneer2 DX geometry this allows in the best circumstances to form 256 correspondence pairs instead of the 16 in the original.

First the adaptations to scan match algorithm are discussed. The experiments to test these adaptations are described after wards.

5.1 The Sequence Matcher

The original algorithm consists of four steps (section 2.4): The scan filter, Combining the scans, The pair filter and Search. In the adapted algorithm the scan filter is replaced by the sequence filter. In the next step are the correspondence

pairs are formed using sequences instead of scans. The third and the fourth step remain almost the same. The sequence matcher consist of the following features:

- The Sequence filter. S_{new}, the new scan is added to sequences. Useless points are filtered.
- Finding the pairs. The new points on the sequences have scanned the same objects as previous points on the sequences. Points that have the same correspondence to the real world are paired.
- The pair filter. Some pairs are not usable and are filtered away.
- Search.The rotation and translation for the best alignment are found by a search procedure.

5.1.1 The Sequence Filter

The new scan S_{new} is added to the sequences. To calculate the tangents line that are fitted on a point in a sequence inferential statistics are used. In the calculation the position of neighboring points on the sequence are needed. For example: if five points in both sides are used, than there can only be reliable tangents from the sixth point on in the sequence. This creates a 5 five frame delay before the points can be used to find the correspondence pairs. This means that not S_{new} is used to find the pairs. The scan of which the scan points have passed the sequence filter check is used instead.

The longer the neighborhood of points, that is used for this, the more reliable the check is. However if more points are used in this check the delay gets longer and more points in the corners are discarded, because they fail the E_{fit} check. The find the length and the threshold experiments are done. This is explained in the next section. Optimal settings should filter out outliers, corners and occlusion boundaries.

$$E_{fit} = S_x^2 + S_y^2 - \sqrt{4S_{xy}^2 + (S_x^2 - S_y^2)^2} \tag{5.1}$$

E_{fit} indicates the fitness of the line. S_x^2 is the variance of the X-coordinates ,S_y^2 the variance of the Y-coordinates of the current scan points and its neighbors. S_{xy} the covariance of the coordinates.

5.1.2 Finding The Pairs

The corresponding points in $S_{filtered}$ the sequences are used for the search part. To find these pairs the scan points in $S_{filtered}$ are extended to find intersections with the sequences. Only intersections between two successive points on a sequence are accepted. Both points must have passed the sequence filter. For each scan point a sequence is searched from the newer points to the older points (figure 5.1). Multiple corresponding pairs are possible, but only the newest are used. They suffer less from cumulative error.

The check for invisible points is omitted, because if the robot drives an S-curve a range finder may sweep back and forward over the same region of its surroundings. This makes the check for invisible point check inapplicable here,

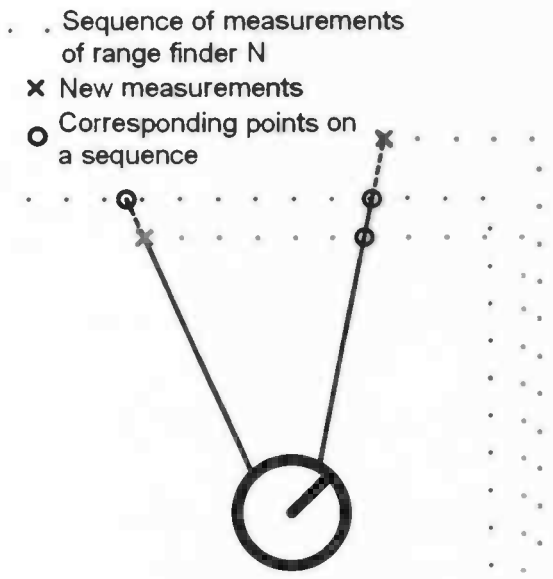


Figure 5.1: Correspondence pairs. The crosses are the scan points that are matched with the sequence. The circles are their estimated corresponding points on the sequence. The blue cross results in two corresponding pairs because it has correspondence with two sequences.

since this back and forward sweep reverses the polar angle order for a section of sequence Z_n, while the points are visible. I could not find a solution for this check as easy as the polar angle order.

Computational demands force a limit on the length of the sequences. How the length of the sequence influences the pose estimator is determined by an experiment in the next section.

5.1.3 The Pair Filter

The pair filter works in the same way. For each trial rotation ω It checks for each pair whether they are not too far apart and that the directions of their sequences are in the same direction. This results in the following two checks. The default values of Lu and Milios were fitted to their data. The value of $\alpha_{threshold}$ is .5 π. No mention was made about the translational limit. Five centimeters are used as the maximal distance.

$$(R_\omega \vec{n_1}) \cdot \vec{n_*} \geq \cos(\alpha_{threshold}) \tag{5.2}$$

$$|R_\omega P_1 - P_*| \leq D_{threshold} \tag{5.3}$$

R_ω is the rotation matrix. If a vector is multilpied with it, it rotates the vector with rotation ω. $\vec{n_1}$ is the normal of the tangent line that belongs to point P_1. $\vec{n_*}$ is the normal of the tangent line that belongs to point P_*.

5.1.4 Search

Four different distance functions to calculate $D(\omega, T)$ are tested. In all the distance functions it is possible for each rtial rotation to find T_x and T_y. This is done by finding the minimum of the distance function. Setting $\frac{dD(\omega,T)}{dT_x}$ and $\frac{dD(\omega,T)}{dT_y}$ both to zero you get two formula's with two unknown variables: T_x and T_y. This is solveble and results in the minimum distance for a given trial rotation and the translation that goes with it.

$$D(\omega, T) = \sum_{i=1}^{N_p} (\vec{n_{*,i}} \cdot (R_\omega P_{1,i} + T - P_{*,i}))^2 \tag{5.4}$$

$$(\omega, T) = \sum_{i=1}^{N_p} ((R_\omega \vec{n_{1,i}} + \vec{n_{*,i}}) \cdot (R_\omega P_{1,i} + T - P_{*,i}))^2 \tag{5.5}$$

$$(\omega, T) = \sum_{i=1}^{N_p} |R_\omega P_{1,i} + T - P_{*,i}|^2 \tag{5.6}$$

$$(\omega, T) = \sum_{i=1}^{N_p} \left(\frac{|R_\omega P_{1,i} + T - P_{*,i}|^2}{(R\omega \vec{n_{1,i}} \cdot \vec{n_{*,i}})^2} \right) \tag{5.7}$$

The first two models are suggested in [22]. Lu and Milios do not give an intuitive explanation of their formulas. The third and fourth distance functions are devised by myself. The third distance function is solely dependent on the displacement between the correspondence pairs. The fourth not only takes the distance between the correspondence pairs into account, but divides this translational difference with in-product of the normals of the tangents. This means that the more these tangents have the same direction the smaller the "distance" becomes between the points in the pair.

The algorithm which finds the ω that minimizes the robust estimator distance function differs. During initial testings it appeared that the distance function is not near as smooth as in figure 2.6. Therefore the golden section approach to find the optimal ω did not work. Instead the search window was divided on a hundred equal intervals. The interval with the lowest distance is the approximated optimal ω.

5.2 The Experiments

Three experiments are performed to optimize and test the performance of the sequence matcher. The experimental setup as discussed in chapter 4 is used for this. All experiments are done on robot ans in simulation. Two experiments were already mentioned above.

- The Pair filter. The two parameters of the sequence filter are varied to find an optimal configuration.
- The sequence matcher. In this experiment the data from the datasets (chapter 4) is used. The data are processed by the sequence matcher with the four distance functions. The sequence length is varied as well

to see what kind of influence that has. The corrections relative to the robot are plotted. The suggestions put forward by the pose estimator are incorporated in the information of the robot.

- Softer restraints. In the second experiment the parameter settings for the pair filter were copied from the [22]. These settings judged the correspondence pairs very strictly, reducing the amount of usable pairs greatly. In this experiment different settings for the pair filter are looked at. Again the corrections relative to the robot are plotted.

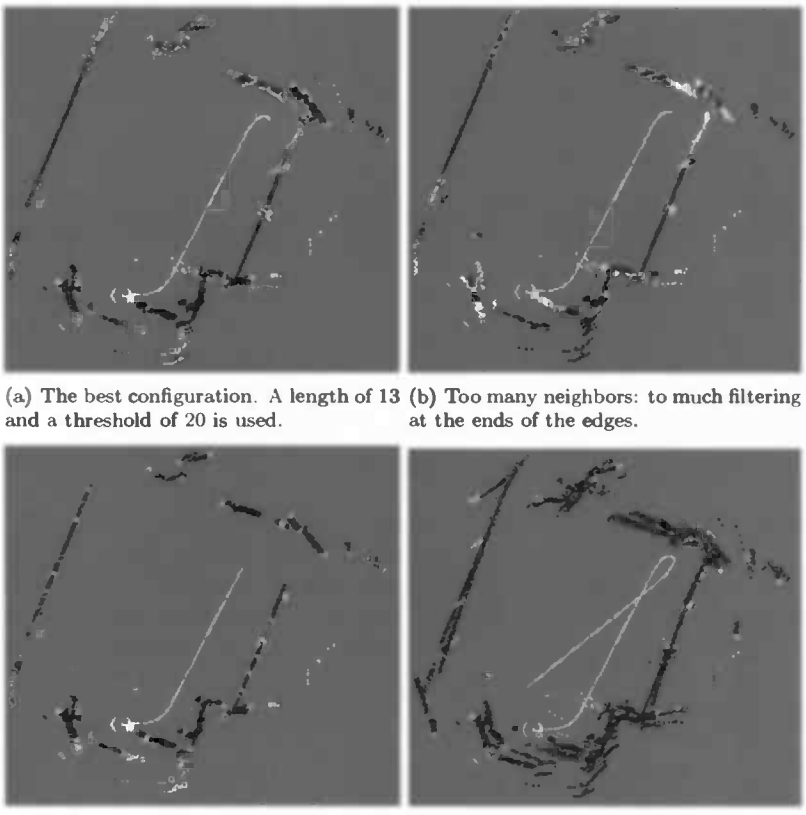
Chapter 6

Results

6.1 Experiment 1: Sequence filter

Two variables are varied in this experiment: The amount of neighbors used for calculating $Efit$ (equation 2.21). The other variable is the threshold which, if transgressed by $Efit$, makes the current scan point unacceptable. Optimal settings should filter out points near occlusion boundaries and points on corners.

Different settings are tried and the solution that produced the best results was selected. Criteria that are used for this are the amount of points that were wrongly discarded and the amount that are wrongly accepted. Using 13 points in the Scan filter and using a threshold of 20 results in the best filter as is shown in figure 6.1. This figure illustrates as well what happens when the variables are changed.



(a) The best configuration. A length of 13 and a threshold of 20 is used.

(b) Too many neighbors: to much filtering at the ends of the edges.

(c) High threshold: acceptable points are filtered out.

(d) Low threshold or too few neighbors: unacceptable point are accepted.

Figure 6.1: Different parameter settings for the sequence filter. The optimal settings are shown in the picture in the top left corner. The other pictures are examples of how the two variable influence the filter process.

6.2 The unstable distance function

The amount of correspondence pairs that is detected is minimal for each frame. There are generally not enough pairs to make a distance function with a curve that is similar to figure 2.6. A typical curve as it is found in the sonar pose estimator is depicted figure 6.2. An increase or decrease in pairs has more influence on this curve than the corresponding pairs can cope with. The trial rotation with the lowest distance is selected. Because the correspondence pairs appear and disappear at random the sonar pose estimator is also random.

Another aspect of having not enough pairs is that most of the time not even a pose estimator can be calculated. This is also apparent in figure 6.2. Not enough pairs are available to calculate the distance function on the left and the right side. Both aspects render the pose estimator practically useless. The results of the next two experiments will illustrate this.

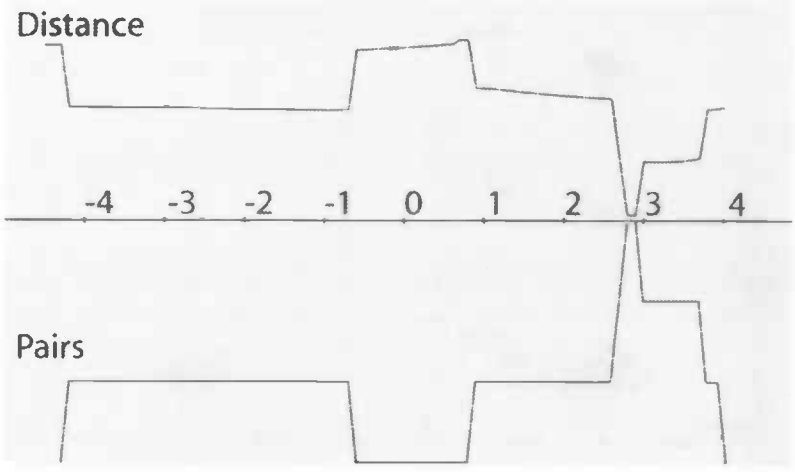


Figure 6.2: Distance and the amount of pairs set in the search window for ω. The window goes from minus 5 degrees to five degrees. An increase or decrease in correspondence pairs has more influence than the correspondence pairs can cope with. If there would be more pairs the correspondence pairs would have more influence compared with the influence of an increase/decrease in pairs.

6.3 Experiment 2: Sequence matching

The four distance function that were tested are:

- M1, equation 5.4:
  $D(\omega, T) = \sum_{i=1}^{N_p} (\vec{n_{*,i}} \cdot (R_\omega P_{1,i} + T - P_{*,i}))^2$
- M2, equation 5.5:
  $D(\omega, T) = \sum_{i=1}^{N_p} (\vec{n_{*,i}} \cdot (R_\omega P_{1,i} + T - P_{*,i}))^2$
- M3, equation 5.6:
  $D(\omega, T) = \sum_{i=1}^{N_p} |R_\omega P_{1,i} + T - P_{*,i}|^2$
- M4, equation 5.7:
  $D(\omega, T) = \sum_{i=1}^{N_p} \left(\frac{|R_\omega P_{1,i} + T - P_{*,i}|^2}{(R\omega \vec{n_{1,i}} \cdot \vec{n_{*,i}})^2} \right)$

The dataset of the pioneer 2DX Merry is used with the setup that is explained earlier. In figure 6.3 the suggestions of the pose estimator relative to the robot are plotted. A Kalman filter uses the variance of both odometry and the pose estimator to determine how much the pose estimator can correct. If the variance of the pose estimator is much larger than the variance of the pose estimator it will mostly be ignored.

The sample standard deviations of the Models M1 and M2 are larger by far. They also produce plenty of outliers when compared with M3 and M4. In all cases the measurements have smaller deviations when longer sequences are used. The variance of the pose estimator is too large when compared with that of the odometry.

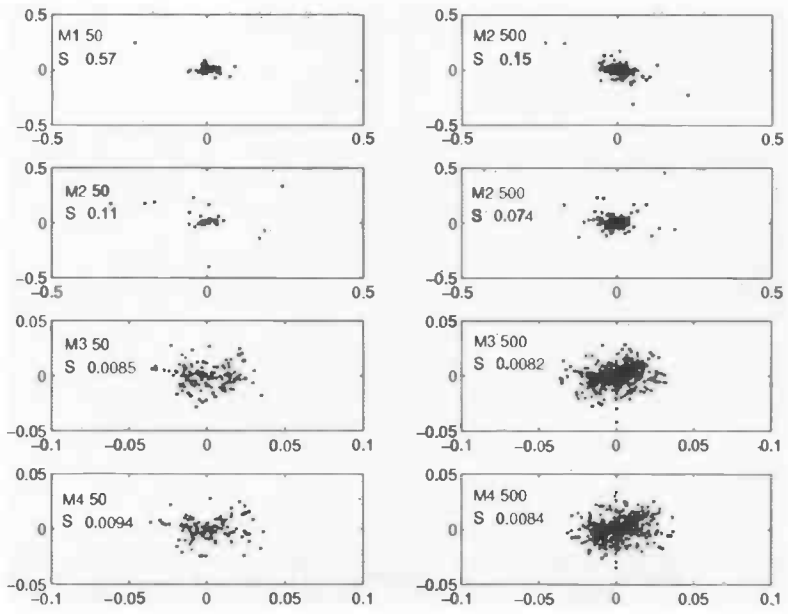


Figure 6.3: The suggestions relative to the robot are plotted. The left corner says which model is used and the length of the sequence. Underneath is the sample standard deviation. M1 and M2 have different scales to show the outliers.

6.4 Experiment 3: Softer Restraints

The sequence matcher needs far more correspondence pairs to make it at least a bit reliable. Therefore the restraints in the pair filter were softened in this experiment. The $\alpha_{threshold}$ determines how much the tangent lines in a correspondence pair can differ in alignment. If they more than $\alpha_{threshold}$ the pair is discarded. The cutoff threshold $\alpha_{threshold}$ was made 75°, 45° and 15°.

Specifically The softer restraints provided more corresponding pairs, but it did not significantly improve the estimator. Results are in figure 6.4. Although there are more corresponding pairs, the instability of the distance function is still too much.

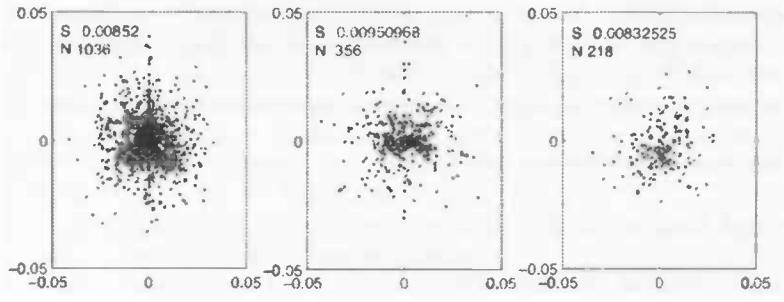


Figure 6.4: More eligible pairs are found with softer restraints in the sequence filter. This does not really improve the variance of the pose estimation.

Chapter 7

The Conclusion

As a pose estimator the sequence matcher will not work neither will it, in its' current form, suffice as a proof of concept that such a sequence matcher will work properly as a pose estimator. The main problem of the sequence matcher is the randomness of the distance function due to lack of correspondence pairs. This randomness makes the pose estimator mostly random and useless. The lack of pairs is caused by the unreliability of the sonar range finders and the lack of information. The suggested corrections should have been applied with a Kalman filter. Before the Kalman filter was implemented the randomness of the suggested corrections emerged. This randomness made the sequence matcher unusable as a estimator. No Kalman filter was implemented.

The implementations of the viewer program and the player agent both work well. I hope more work will be done with these.

Another conclusion is that the odometry of the pioneers is highly reliable as long as the robot rides without shocking or stopping. These actions create the largest amount noise in odometry.

7.1 Future work

Some suggestions for future work can be made. Perhaps more and better correspondence pairs can be made if different estimations of P_2, that is P_*, are tried. Lu and Milios [22] made the suggestions that P_2 can be estimated by intersecting the normal of the tangent line of P_1 on S_{new}. Another option is that P_* is the scan point on the sequence that is closest to the scan point P_1.

Another thing is that for the parameters of the kinematic model and the sonar beam model experiments were done in a rather ad hoc way. The parameters are in the right order of magnitude, but they are not precise. If more research is done in probabilistic robotics at our faculty these parameter estimations should be done more elaborately. A suggestion is to check whether tapestry would reduce the amount of failure of the sonar range finders. The ruggedness of tapestry on the lab walls would make the walls act less like a mirror for the sound pulses of the robot. This would lessen the forty percent failure rate. Perhaps a bachelor project would fit.

There is no future for sonar range finders except in the most basic applications. They are far from state of the art. They will probably never be used in

service robots, because they make a decent amount of annoying noise. Laser scanners are far more precise and all state of the art probabilistic robotics research is done with them. If the faculty wishes to perform state of the art research on this topic laser scanners are needed.

Bibliography

[1] Valentino Braitenberg. *Vehicles, Experiments in Synthetic Psychology.* The MIT press, 1984.

[2] I. J. Cox. Blanche: An experiment in guidance and navigation for an autonomous robot vehicle. *IEEE Trans. on Robotics and Automation*, pages 193–204, 1991.

[3] A.J. Davison, Y. González Cid, and N. Kita. Real-time 3D SLAM with wide-angle vision. In *Proc. IFAC Symposium on Intelligent Autonomous Vehicles, Lisbon*, July 2004.

[4] D.Fox. Adapting the sample size in particle filters through kld-sampling. *International Journal of Robotics Research*, 2003.

[5] Arnaud Doucet, Nando de Freitas, and Neil Gordon. *Sequential Monte Carlo Methods in Practice.* New York: Springer-Verlag, 1st edition, january 2001.

[6] Trevor Fitzgibbons and Eduardo Nebot. Bearing-only slam using colour-based feature tracking. ACRA, 2002.

[7] D. Fox. Kld-sampling: Adaptive particle filters. In *Advances in Neural Information Processing Systems 14.* MIT Press, 2001.

[8] E Gamme, R Helm, R Johnoon, and J Vlissides. *Design Patterns.* Addison Wesley, 2004.

[9] 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.

[10] J.-S. Gutmann, W. Burgard, D. Fox, and K. Konolige. An experimental comparison of localization methods. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1998.

[11] J.-S. Gutmann and K. Konolige. Incremental mapping of large cyclic environments. *International Symposium on Computational Intelligence in Robotics and Automation (CIRA'99)*, 1999.

[12] J.-S. Gutmann and C. Schlegel. Amos: Comparison of scan matching approaches for self-localization in indoor environments. *1st Euromicro Workshop on Advanced Mobile Robots (Eurobot'96)*, 1996.

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

[14] W Hazenberg, J Horst, and S. Thrun. Robotica: Monte carlo localisatie. June 2004.

[15] D. Hhnel, D. Schulz, and W. Burgardings. Map building with mobile robots in populated environments. IROS, 2002.

[16] D. Hhnel, R. Triebel, W. Burgard, and S. Thrun. Map building with mobile robots in dynamic environments.

[17] Robert V. Hogg and Elliot A. Tanis. *Probability and Statistical Inference.* Prentice Hall, 6th edition, 2001.

[18] D Kahaner, C Moler, and S Nash. *Numerical methods and software.* Prentics Hall, 1989.

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

[20] K. Konolige and K. Chou. Markov localization using correlation. Proceedings of the International Joint Conference on AI (IJCAI), 1999.

[21] Kurt Konolige. Large-scale map-making. AAAI, 2004.

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

[23] H. P. Moravec and A. Elfes. High resolution maps from wide angle sonar. *Proc. IEEE Int. Conf. Robotics and automation*, pages 116–121, 1984.

[24] Ph.Smets. What is dempster-shafer's model ? *wiley*, 1994.

[25] C. W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics*, pages 25–34, 1987.

[26] M. Ribo and A. Pinz. A comparison of three uncertainty calculi for building sonar-based occupancy grids. *Robotics and Autonomous Systems*, (35):201–209, 2001.

[27] S. Thrun. A probabilistic online mapping algorithm for teams of mobile robots. *International Journal of Robotics Research*, 20(5):335–363, 2001.

[28] S. Thrun. Learning occupancy grid maps with forward sensor models. *Autonomous Robots*, 15(2):111–127, September 2003.

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

[30] S. Thrun, D. Fox, W. Burgard, and F. Dellaert. Robust monte carlo localization for mobile robots. *Artificial Intelligence*, 128(1-2):99–141, 2000.

[31] S. Thrun, D. Hähnel, D. Ferguson, M. Montemerlo, R. Triebel, W. Burgard, C. Baker, Z. Omohundro, S. Thayer, and W. Whittaker. A system for volumetric robotic mapping of abandoned mines. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2003.

[32] S. Thrun, M. Montemerlo, D. Koller, B. Wegbreit, J. Nieto, and E. Nebot. Fastslam: An efficient solution to the simultaneous localization and mapping problem with unknown data association. *Journal of Machine Learning Research*, 2004. To appear.

[33] Sebastian Thrun, Dieter Fox, and Wolfram Burgard. *Probalistic Robotics, EARLY DRAFT*. n.p., 2004.

[34] I. Ulrich, F. Mondada, and Nicoud J.D. Autonomous vacuum cleaner. *Robotics and Autonomous Systems*, 19:233–245, 1997.

[35] E. Wan and R. van der Merwe. The unscented kalman filter for nonlinear estimation. In *Proceedings of Symposium 2000 on Adaptive Systems for Signal Processing, Communication and Control (AS-SPCC)*, Lake Louise, Alberta, Canada, October 2000. IEEE.

[36] Greg Welch and Gary Bishop. *An Introduction to the Kalman Filter*. Department of Computer Science University of North Carolina at Chapel Hill Chapel Hill, 2004.