Monte Carlo Localization

P. Hiemstra & A. Nederveen

August 24, 2007

Abstract

In this paper we investigate robot localization with the Augmented Monte Carlo Localization (aMCL) algorithm. The goal of the algorithm is to enable a robot to localize itself in an known world. The map of the environment where the robot has to localize itself must be given to the robot beforehand. In our case the map is constructed by another robot. This map is a so called occupancy grid. An occupancy map contains a value for every location which indicates the probability that this location is occupied by a object such as a wall. We investigate the influence on the performance of the aMCL algorithm when using occupancy maps as input for the aMCL algorithm. We compare the performance of the aMCL algorithm with different levels of uncertainty and two ways of dealing with this uncertainty. We found that the performance of the aMCL algorithm is best when we convert the occupancy map to a binary map by applying a threshold. In that case each location above a certain threshold is considered occupied. This simple method provided the best performance.

1 Introduction

Localization is one of the most fundamental problems in Artificial Intelligence and especially in mobile robotics (Cox & Wilfong, 1990). Because in an unknown world a robot does not know where it is, it can be difficult to determine what to do next in the robotic movements. The localization problem can be split into two sub problems. The first problem is known as the *global localization problem* or wake-up robot problem (Fox, Burgard, & Thrun, 1999). In this problem the initial pose of the robot is unknown (Thrun, Fox, Burgard, & Dellaert, 2000) and it has to determine his own position in the world from scratch. Therefore the robot must be able to deal with multiple hypotheses about it's initial pose and in this way it has to handle with big errors in the hypotheses. The second problem is the *robot kidnapping problem*. Here the robot is already localized but he is suddenly transferred to another position without being told (Thrun et al., 2000). The problem here is that the robot believes to be somewhere else in the world then he is for real. In this way the robot has to deal with great catastrophic localization failures (Thrun et al., 2000).

The Augmented Monte Carlo Localization (aMCL) can solve the global localization problem and the robot kidnapping problem in a highly robust and efficient way (Thrun et al., 2000; Fox, Thrun, Burgard, & Dellaert, 2001). One of the problems of aMCL is that it works well with maps without uncertainty. In this case an exact map of the environment is given to the robot. This is a problem if you want to use the algorithm in the real world. When creating maps real time with sensors of the robot there will be a lot of noise and the map will be full of uncertainty. One method of generating maps by the robot is to construct so called occupancy maps (Moravec & Elfes, 1985; Thrun, 2003). These maps are not perfect representations of the environment which simply show if a certain location is occupied or not (e.g. by a wall), but give a probability for a certain location to be occupied. Once constructed the occupancy map can be used for robot navigation as is discussed in the articles by Elfes (1989) and Buhmann et al. (1995). These articles focus on the general performance of the robots in real life situations. We will instead focus on the influence on the localization of the amount of uncertainty of the occupancy maps.

Our main question is to investigate the performance of the aMCL algorithm under multiple levels of uncertainty and to compare different ways to deal with this uncertainty. Is it better to convert the map to a binary map, which tells us a certain location is free or occupied, first or is it better to work use the uncertainty present in the map. The precise implementation of the latter is discussed in section 6.

In this paper we will first explain the aMCL algorithm and the construction of the occupancy maps. Subsequently, we will discuss the experiments and will present the result of the experiment. Finally, we will discuss the findings from the experiments.

2 Monte Carlo Localization

In probabilistic robotics a key concept is belief. A robot never knows his own pose because it is not measureable directly. Instead, the robot must infer its pose from map and sensor data. So the robot's true state is distinguished from its internal belief or state knowledge. Monte Carlo Localization (MCL) is a localization algorithm which represents the belief $bel(x_t)$ by particles where x_t is the state of the robot at time t. By incorporating probabilistic motion and perceptual-sensor models into the particle filter algorithm (see subsection 2.3) the MCL algorithm is obtained.

2.1 Pose robot

The kinematic state of the robot in the planar environment is described by three variables and is called the robot's *pose*. The pose consists the two-dimensional planar coordinates x and y and it's heading θ in that environment. So the vector for the pose is described as follows:

$$x_t = \begin{pmatrix} x_t \\ y_t \\ \theta_t \end{pmatrix}. \tag{1}$$

The pose is graphically shown in figure 1. If $\theta = 0$ then the heading of the robot points into the direction of the x-axis.

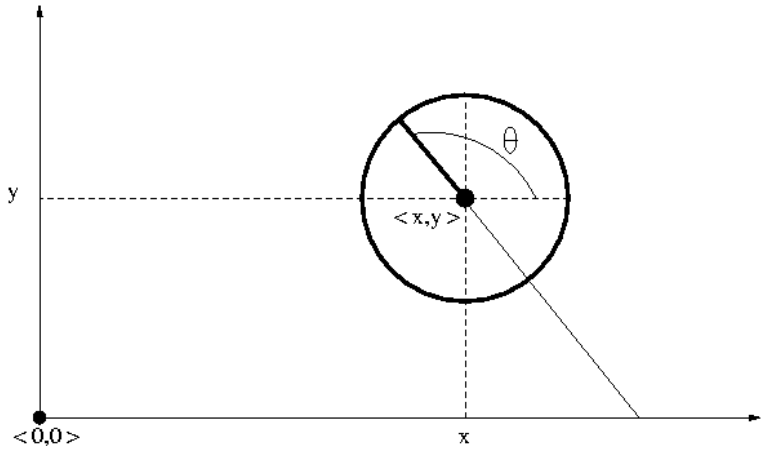


Figure 1: The robot's pose.

2.2 The Bayes Filter

The most general algorithm for calculating beliefs is the Bayes filter algorithm.

In probabilistic robotics belief is represented by conditional probability distributions. It assigns for every possible state a probability. To determine the belief of state $bel(x_t)$ you need to take into account all sensor and action data that happened before t. This is denoted in the following equation:

$$bel(x_t) = p(x_t|z_{1:t}, u_{1:t}) \tag{2}$$

Here $z_{1:t}$ is all the sensor data from timestep 1 until time step t. The control data , actions by which the robot changes the world, is denoted as $u_{1:t}$. The Bayes Filter is a recursive function that requires

$bel(x_{t-1})$, u_t and z_t to calculate the belief $bel(x_t)$. First the Bayes Rule (Hogg & Tanis, 2001) is applied to equation 2

$$\begin{aligned} bel(x_t) &= p(x_t|z_{1:t}, u_{1:t}) \\ &= \frac{p(z_t|x_t, z_{1:t-1}, u_{1:t})p(x_t|z_{1:t-1}, u_{1:t})}{p(z_t|z_{1:t-1}, u_{1:t})} \\ &= \eta p(z_t|x_t, z_{1:t-1}, u_{1:t})p(x_t|z_{1:t-1}, u_{1:t}) \end{aligned} \tag{3}$$

Since the Bayes Filter is a probability density function, η is used to make the total probability one. η normalizes the Bayes Filter. In probabilistic robotics an important property is the *Markov Assumption*. Hypothetically you can say that if we knew state x_t and want to predict the measurement z_t, no past measurements or control would provide additional information (Fox, Thrun, & Burgard, 2005). This is called the Markov assumption, mathematically denoted as follows:

$$p(z_t|x_t, z_{1:t-1}, u_{1:t}) = p(z_t|x_t). \tag{4}$$

If the Markov Assumption is applied to equation 3 it leads to the following equation:

$$bel(x_t) = \eta p(z_t|x_t)p(x_t|z_{1:t-1}, u_{1:t}). \tag{5}$$

If we say

$$\overline{bel}(x_t) = p(x_t|z_{1:t-1}, u_{1:t}) \tag{6}$$

then we can write equation 5 as

$$bel(x_t) = \eta p(z_t|x_t)\overline{bel}(x_t) \tag{7}$$

Equation 5 calculates the belief distribution $bel(x_t)$ from measurement and control data. A Bayes filter is recursive and has two essential steps. First it processes the control u_t by calculating the belief $\overline{bel}(x_{t-1})$. This is called the *prediction* of the filter. $\overline{bel}(x_t)$ predicts the the state at time t based on the previous state before incorporating the *measurement* at time t. The second step is the *measurement update*. It multiplies the belief $\overline{bel}(x_t)$ by the probability that the measurement z_t may have been observed. It does so for each hypothetical posterior state x_t by this recursive algorithm. The complete mathematical derivation of the Bayes Filter is given in (Fox et al., 2005).

2.3 Particle filters

Particle filters are a sample based variant of the Bayes filter. The key idea of the particle filter is to represent the belief $bel(x_t)$ by a set of samples drawn from $bel(x_{t-1})$. The samples of a distribution are called particles and are denoted as:

$$X_t = x_t^{[1]}, x_t^{[2]}, ..., x_t^{[M]}. \tag{8}$$

In equation 8 each particle $x_t^{[m]}$ with $(1 \leq m \leq M)$ is a hypothesis to what the true world state may be at time t. X_t is the set of all M particles. Because a particle filter is to approximate the belief $bel(x_t)$ by the set of particles X_t, the equation is as follows:

$$x_t^{[m]} \sim p(x_t|z_{1:t}, u_{1:t}). \tag{9}$$

Just as the Bayes algorithm in equation 7 the particle filter algorithm constructs the belief $bel(x_t)$ recursively from the belief $bel(x_{t-1})$ one timestep earlier. So since the belief is represented by the particle set as in equation 9 the particle filter construct a new set of particles X_t from the set X_{t-1}.

The particle filter has two important parts. The first part generates a hypothetical state $x_t^{[m]}$ on time t based on the particle $x_{t-1}^{[m]}$ and control u_t. This step involves sampling from the next state distribution $p(x_t|u_t, x_{t-1})$. Then the *importance factor* $w_t^{[m]}$ is calculated based on measurement z_t and the calculated particle $x_t^{[m]}$. The *importance factor* is interpreted as the weight of each particle. The resulting set $\overline{\chi}_t$ is a temporary set with weighted particles. The second part is called the *importance re-sampling*, and

redraws a new set X_t with replacement from the temporary set $\overline{\chi}_t$ and contains only the particles with a high probability. So over time the set is converge to the hypotheses that produces the most likely output.

The particle approximation is the core part of the MCL algorithm, which itself came by with some new improvements. The advantages of the algorithm is that it copes well with noisy data, it concentrates only in interesting space regions. For further mathematical derivation of the particle filter see (Fox et al., 2005).

2.4 Motion Model

Like in the particle filter there is the prediction step which is used for the motion model. The motion model $p(x|u_t, x_{t-1})$ gives the probability of finding the robot at pose x after preforming motion command u_t if the initial pose of the robot was x_{t-1}. For a visualization of the motion model , see figure 2.

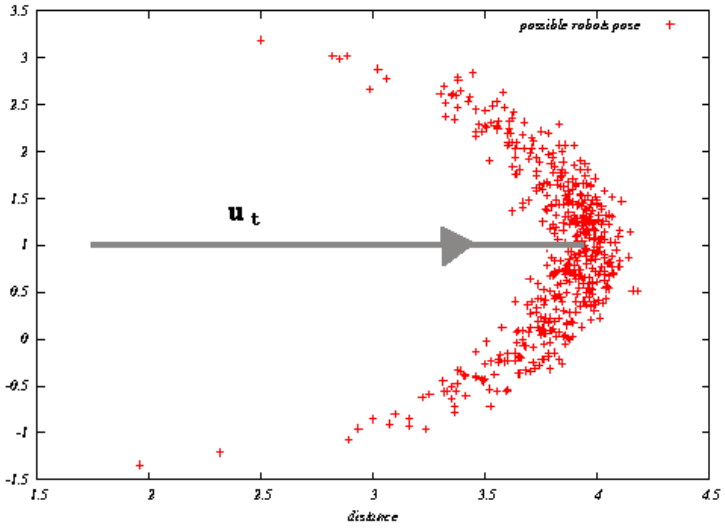


Figure 2: The motion model. Posterior distributions of the pose of the robot upon executing the motion command illustrated by the solid line.

There are two specific probabilistic motion models $p(x_t|u_t, x_{t-1})$, both for mobile robots operating in the plane. The first model assumes that the motion data u_t specifies the velocity commands given to the robot's motors. The second model assumes that one is provided with odometry information. In this project we are working with the first model, the velocity model. The choice for this model is because we are working with a simulation program Stage [1] and in this simulation model the accuracy is up to date. Unlike robotics in the real world the odometry model is only available post-the-fact and is always a small step behind.

The velocity model assumes that we control a robot through two velocities, namely rotational en translational velocity. In this way the motion command u_t is

$$u_t = (v_t, w_t) \tag{10}$$

Where v_t is the translational velocity in time and w_t is the rotational velocity in time. The positive rotational velocities induce a left turn (counterclockwise). A positive translational velocity v_t correspond to forward motion. The algorithm used for calculating the motion model velocity is given in algorithm 1. The input is the motion command u_t and the pose x_t. The algorithm returns the pose $x_t = (x, y, \theta)^T$. First the algorithm is mixing some noise in the command controls v and w by disturbing them with the noise parameters , which are given through robot-specific motion error parameters. The parameters $\alpha_{1,2}$ influence the translational velocity, $\alpha_{3,4}$ influence the rotational velocity and $\alpha_{5,6}$ are giving noise over

[1] Stage simulates a population of mobile robots, sensors and object in a two-dimensional bitmapped environment. Player is a network server for robot control. Running on your robot, Player provides a clean and simple interface to the robot's sensors and actuators over the IP network. `http:\\playerstage.sourceforge.net`

the final rotation result of the robot in line 7 of the algorithm. The higher those noise parameters are, the less accuracy there is in the calculation of the motionpart. In other words, if the parameters have a high value, then the distribution of the particles are more spread over the map. With those new calculated parameters x', y' and θ' in line 5 through 7, the algorithm is generating the sample's new pose x_t. In algorithm 1 the function **sample** is denoted

$$sample(b) = b \cdot rand(-1, 1) \cdot rand(-1, 1) \tag{11}$$

This is a triangular distribution with variance b and mean 0 (see figure 3). If the variance is small the function will return a value close to zero. In other words, if the input b is small there is relatively little noise and the other way around. At line 2 of the algorithm, the input b of sample is a combination of the noise parameters $\theta_{a,2}$, the translational speed $|v|$ and the rotational speed $|w|$. If the speed is small, then there will be not much noise over the calculation of the pose. So the higher the speed, the more noise is in the calculation of the pose x_t. For our experiments we set $\alpha_1 \ldots \alpha_6$ to 0.5.

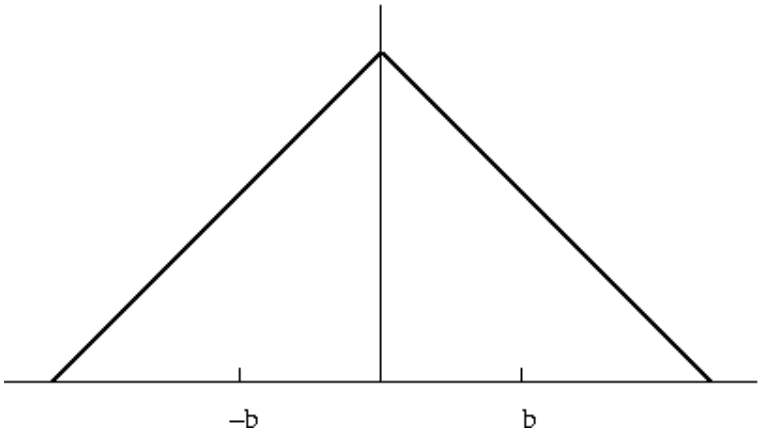


Figure 3: Triangular distribution with mean zero and variance b.

Algorithm 1 Motion model.

1: **procedure** SAMPLE MOTION MODEL(u_t, x_{t-1})
2: $\hat{v} = v + \text{sample}(\alpha_1|v| + \alpha_2|w|)$
3: $\hat{w} = w + \text{sample}(\alpha_3|v| + \alpha_4|w|)$
4: $\hat{\gamma} = \text{sample}(\alpha_5|v| + \alpha_6|w|)$
5: $x' = x - \frac{\hat{v}}{\hat{w}} \sin\theta + \frac{\hat{v}}{\hat{w}} \sin(\theta + \hat{w} \triangle t)$
6: $y' = y + \frac{\hat{v}}{\hat{w}} \cos\theta - \frac{\hat{v}}{\hat{w}} \cos(\theta + \hat{w} \triangle t)$
7: $\theta' = \theta + \hat{w} \triangle t + \hat{\gamma} \triangle t$
8: **return** $x_t = (x', y', \theta')^T$
9: **end procedure**

2.5 Sensor Model

The sensor model returns the probability of a real measurement given the pose of a particle and a map. In other words, what is the probability of getting the current sensor reading when the robot is at a hypothesized position. The sensor model deals with the inherent uncertainty associated with sensor readings. The sensor model is modelled as conditional probability distribution $p(z_t|x_t, m)$, where z_t is the current measurement, x_t is the assumed pose and m represents the map. This distribution is modelled as a weighted mixture of several probability distributions. The individual distributions are plotted in

figure 4. An example of a resulting mixture is plotted in figure 5. This mixture is given by:

$$p(z_t^k|x_t, m) = \begin{pmatrix} z_{hit} \\ z_{short} \\ z_{max} \\ z_{rand} \end{pmatrix} \cdot \begin{pmatrix} p_{hit}(z_t^k|x_t, m) \\ p_{short}(z_t^k|x_t, m) \\ p_{max}(z_t^k|x_t, m) \\ p_{rand}(z_t^k|x_t, m) \end{pmatrix} \tag{12}$$

The first is a distribution we will call p_{hit} (figure 4a) . This Gaussian distribution, $\sigma = 0.5$, centered around z_t^{k*} models a measurement with a correct range but with a local measurement noise..Where z_t^{k*} is the distance which would be measured by a sensor if the robot has the pose represented by the particle. This distance is measured with ray tracing in the available map. A line, representing the sonar beam, is drawn in the map until the simulated sonar beam is reflected of an obstacle in the map. This produces the distance the sonar beam would measure at this location. The second distribution, p_{short} (figure 4b), models the presence of unexpected objects. Examples of such objects are tables, chairs and people. These objects not present in the map are modelled by an exponential distribution. This distribution has one parameter λ which has to be adjusted to suit the environment. The third distribution, p_{max} (figure 4c), models failures: the chance that objects are missed altogether. This is done with a point mass distribution centered at the maximum measurement of the sonar. The fourth and final distribution, p_{rand} (figure 4d), models random measurements with a uniform distribution over the entire measurement range. Because our robot will not meet objects not present in the map, we will not use the second distribution in our simulation. For our experiment we set the weight parameters z_{hit}, z_{rand}, and, z_{max} to 0.9, 0.05, and 0.05 respectively.

2.6 Monte Carlo Localization Algorithm

Monte Carlo Localization (MCL) is an algorithm which persists on the particle filter algorithm. It substitutes the motion model and the measurement model into the particle filter algorithm. It represents the $bel(x_t)$ as a set of M particles $\chi_t = x_t^1, x_t^2, ..., x_t^M$. The basic MCL algorithm is shown in algorithm 2. In line 4 the motion model is applied using the particles from the present belief x_{t-1}. In line 5 the measurement model is applied to calculate the *importance weight* of the particles what will be used for generating the next distribution. In line 6 the particle and his weight are combined in the new distribution.

The initialization state of the set of particles represents the knowledge about the system at the beginning. If there is no information about the possible pose x_t of the robot all particles are uniformly distributed on the map. All n particles then have the same weight value, namely $1/n$. The next step is to repeat the next sequence n times for every particle.

First, sample a particle from the distribution $bel(x_{t-1})$, a weighted re-sampling of particles. The higher the weight of the particle, the larger the probability for selection of the particle when the next sampling is occurring. Second, the motion model and (see section explained 2.4) and the measurement model (see section 2.5) are applied. Finally, the particles which represent the better hypothesis, i.e. the highest probabilities are selected.

3 Sampling

Re-sampling of the particles can be a source of error. Given, a robot who has no access to its environment. The robot will start with a uniform distribution of samples across its sample space. If these particles are re-sampled there is a good chance that one of the particles will not be chosen and another particle will be chosen more than one time. Because no new particles are introduced, repetition of this sampling process will eventually result in all particles having the same pose. Meaning that a robot with no information about its environment has located itself.

The loss of diversity in the particle population manifests itself as approximation error, also known as the *variance* of the estimator. There are two options available to decrease the variance of the estimator. The first is reduction of the sampling frequency. If for example the robot is not moving, no re-sampling of the particles should be done. To determine at which frequency to sample in each situation is a matter of experience. A low frequency will potentially waste particles to low probability position. A high frequency will possibly lead to a loss in diversity.

Algorithm 2 Monte Carlo localization.

procedure MCL(χ_{t-1}, u_t, z_t, m)
  $\bar{\chi}_t = \chi_t = \emptyset$
  for $m = 1$ to m **do**
    $x_t^{[m]}$ = **sample motion model**(u_t,x_{t-1})
    $w_t^{[m]}$ = **measurement model**(z_t,$x_t^{[m]}$,m)
    $\bar{\chi}_t = \bar{\chi}_i + \langle x_t^{[m]}, w_t^{[m]} \rangle$
  end for
  for $m = 1$ to m **do**
    draw i with probability $\propto w_t^{[i]}$
    add $x_t^{[i]}$ to χ_t
  end for
  return χ_t
end procedure

The second strategy is to use a *low variance sampler*. A sequential stochastic process is used to sample the particles. The algorithm is best illustrated with 3. The weights of the particles are presumed to have a sum of 1. An initial random number r between 0 and M^{-1} is chosen. By repeatedly adding M^{-1} to r one can select the particles. A unique particle is selected with the following formula:

$$i = \underset{j}{\operatorname{argmin}} \sum_{m=1}^{j} w_t^{[m]} \geq u$$

There are three advantages of the low-variance sampler. First, the sample space is sampled in a more systematic way as opposed to independent selection. Second, if all samples have identical weights the collection of particles will be the same after re-sampling. Third, the algorithm has a lower complexity $O(M)$.

Algorithm 3 Low variance sampler.

procedure LOW VARIANCE SAMPLER(a, b)
  $\bar{\mathcal{X}}_t = \emptyset$
  $r = rand(0; M^{-1})$
  $c = w_t^{[1]}$
  $i = 1$
  for $m = 1$ to M **do**
    $u = r + (m-1) \cdot M^{-1}$
    while $u > c$ **do**
      $i = i + 1$
      $c = c + w_t^{[i]}$
    end while
    add $x_t^{[i]}$ to $\bar{\mathcal{X}}$
  end for
  return $\bar{\mathcal{X}}_t$
end procedure

4 Augmented MCL

The normal implementation of the MCL algorithm suffers from the problem that it will not recover from so called kidnapping of the robot or a global localization failure. Once the particles have converged to one position it is not possible for the algorithm to contemplate a alternative position. Simply because other positions are not considered anymore. Any particle which suggested an alternative location is removed by the sampling procedure. As a consequence, when *teleporting* the robot to another position, the algorithm

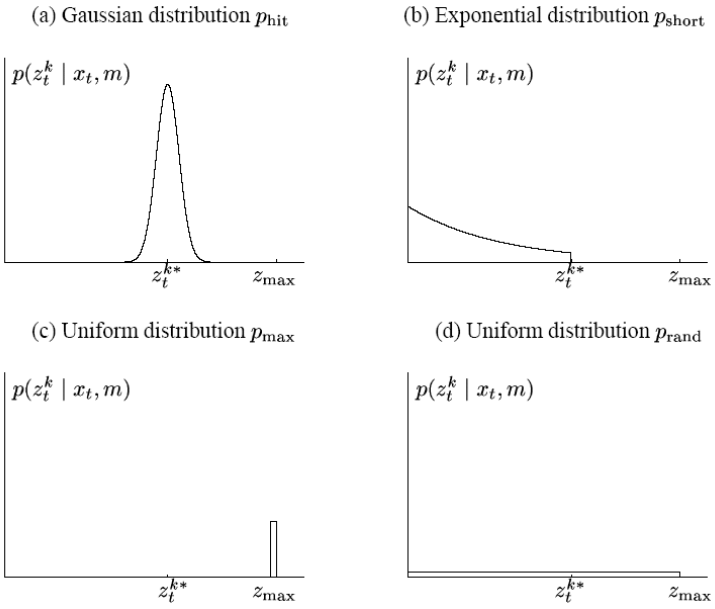


Figure 4: (a) Gaussian distribution. (b) Exponential distribution (c,d) Uniform distribution. The four component distributions of the sensor model.

will not be able to find the new location of the robot. This is also known as the *kidnapping problem*. The same holds if the algorithm converges on a wrong location. Once converged other positions will not be considered.

A solution to this problem is to add random particles, which represent a random position. This of course raises the question how many random particles to add. Too many particles will undermine the ability for localization and too little will have no meaningful effect on the performance of the algorithm. As explained in Fox et al. (2005) a better way to add particles would be to look at the current quality of the localization. One possibility is to look at the average weight of the particles but this can be misleading in case of high sensor noise. Instead we can look at the change of a short term average compared to a long term average the weights.

Two variables are introduced: w_{slow} which is an exponential filter of the weight over a relatively long time and w_{fast} which is a exponential filter over a shorter time. The variables α_{slow} and α_{fast} are used to determine decay rate of both filters. A high decay rate will make this value sensitive to abrupt changes. A small decay rate will smooth out the averaging. This scheme only works if $0 \leq \alpha_{slow} \ll \alpha_{fast}$. A sudden drop in the value of w_{fast} compared to the value of w_{slow} indicates a decrease in the quality of the localization.

The ratio of w_{fast} and w_{slow} can be used to determine the chance of adding random particles. A particle is chosen from uniform distribution of the pose space. Using the measurement model a initial chance is assigned to that particle. Algorithm 4 is an implementation of the random particle injection into the MCL algorithm.

For our experiments we have chosen 0.2 for α_{fast} and 0.05 for α_{slow}. With these values the robot is capable to recover from a kidnap.

5 SLAM and Occupancy maps

As discussed the aMCL algorithm is used to enable a robot to localize itself in a given map. This map has to be known up front and is given to the robot. Therefore, the robot knows exactly how its environment looks, only its position is unknown. However, it is more interesting if a robot is able to learn its own environment and use that map to localize itself. This would enable to function in a previously unknown environment. The problem to both build the map and localize itself in that map is called simultaneous

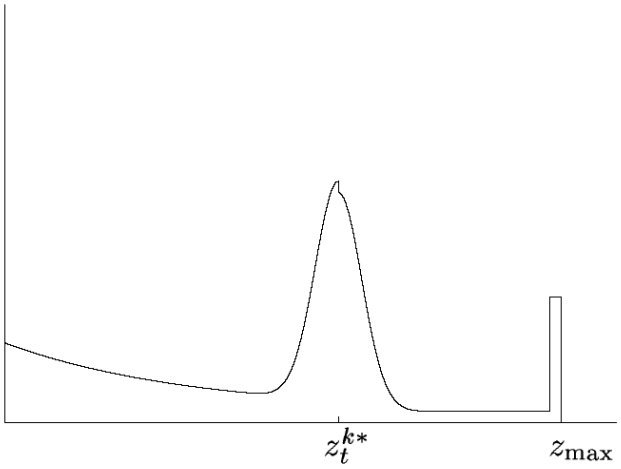


Figure 5: Mixture distribution used by sensor model. The distribution is a summation of the distributions shown in figure 4.

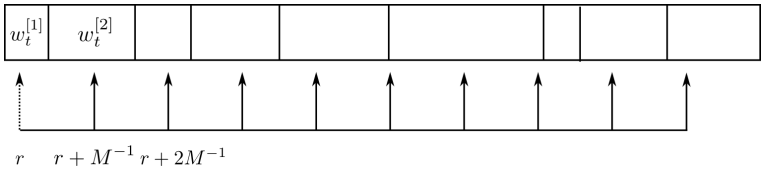


Figure 6: Graphical depiction of the low variance sampler.Where r is the initial random position and $r + M^{-i}$ the position of the selected weights.

localization and mapping (SLAM). In this case the map of the unknown environment has to be learned with the sensors present on the robot.

A well known representation of an environment is the so called occupancy map (Moravec & Elfes, 1985). An occupancy map gives a value for every location in a map which represents the probability that that location is occupied. For example, in an indoor environment the location can be occupied by a wall or a door. These maps can be used as map for robot navigation and localization (Elfes, 1989; Buhmann et al., 1995). The maps are constructed using data from laser range finders or as in our case data from sonar measurements.

The occupancy map divides an unknown space into a grid. While the robot moves through the unknown space, the robot estimates the probability that a cell in the grid is occupied based on the sensor readings. This is expressed as:

$$p(m_{x,y}|z_1, \ldots, z_T). \tag{13}$$

$m_{x,y}$ is the grid cell for which the occupancy probability is determined and $z_1, \ldots, z_T$ are the different sensor readings at all time steps. The algorithm to construct a occupancy map is presented in algorithm 5 and is based on a article by Thrun (2003). Where $p(x, y)$ is the *prior* for occupancy. This value depends on the how much obstacles are expected to be present in the environment. This value is usually between 0.2 and 0.4. $p(m_{x,y}|z)$ is the inverse measurement model. This model assigns a probability of a sonar measurement originating from a certain cell in the grid. In other words, the probability that grid cell $m_{x,y}$ causes sonar measurement z. An example is given in figure 7. The shape of the inverse sensor model depends on the sensor reading, z_t, similar to the sensor model, described in section 2.5, whose shape depends on z_k. For a precise definition of this function see Marck, 2006.

Instead of normal occupancy chances from 0 to 1, the probabilities of the occupancy of every grid cell are expressed in logodds. The logodd value for a grid cell, $-\infty < l_{x,y} < \infty$, also indicates how likely that a certain spot in the map is unreachable for the robot. The logodd value can be converted to a normal

Algorithm 4 Augmented Mcl.

procedure AUGMENTED MCL(χ_{t-1}, u_t, z_t, m)
  static w_{slow}, w_{fast}
  $\bar{\chi}_t = \chi_t = \emptyset$
  for $m = 1$ to M **do**
    $x_t^{[m]}$ = **sample motion model**(u_t,x_{t-1})
    $w_t^{[m]}$ = **measurement model**(z_t,$x_t^{[m]}$,m)
    $\bar{\chi}_t = \bar{\chi}_i + \langle x_t^{[m]}, w_t^{[m]} \rangle$
    $w_{avg} = w_{avg} + \frac{1}{M} w_t^{[m]}$
  end for
  $w_{slow} = w_{slow} + \alpha_{slow}(w_{avg} - w_{slow})$
  $w_{fast} = w_{fast} + \alpha_{fast}(w_{avg} - w_{fast})$
  for $m = 1$ to M **do**
    with probability **max**(0.0, 1 - w_{fast}/w_{slow}) **do**
      add random pose to χ_t
    else
      draw $i \in \{1, ..., N\}$ with probability $\propto w_t^{[i]}$
      add $x_t^{[i]}$ to χ_t
    end with
  end for
  return χ_t
end procedure

Algorithm 5 OccGrids: algorithm to construct occupancy maps.

procedure OCCGRIDS(Z)
  /* Initialization */
  for all grid cells $\langle x, y \rangle$ **do**
    $l_{x,y} = \log(p(m_{x,y})) - \log[1 - [p(m_{x,y})])$
  end for
  /* Calculate log-odds for each grid cell */
  for all time steps t to T **do**
    for all grid cells $\langle x, y \rangle$ in the perceptual range of z_t **do**
      $l_{x,y} = l_{x,y} + \log(p(m_{x,y}|z_t)) - \log[1 - p(m_{x,y}|z_t)] - \log p(m_{x,y}) + \log[1 - p(m_{x,y})]$
    end for
  end for
end procedure

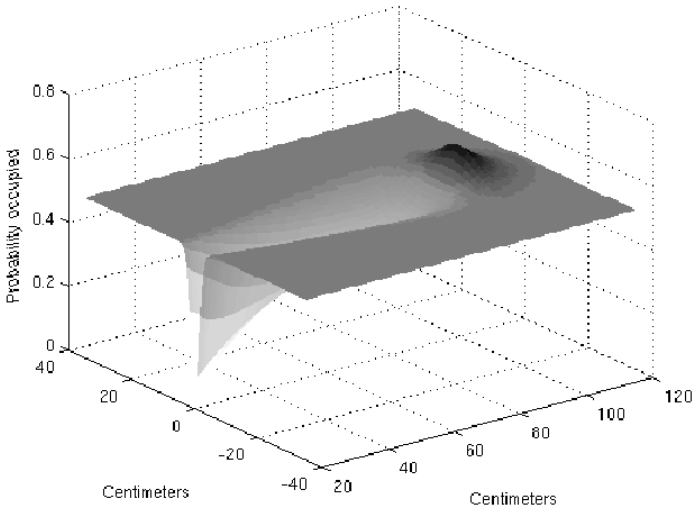


Figure 7: Inverse sensor model for a sensor reading, z_t, of 1 m. (From: Marck, 2006)

chance value ranging form 0 to 1, which we will call the *belief* value, with the following formula:

$$\textbf{belief} = 1 - \frac{1}{1 + e^{\textbf{logodd}}}.$$

Finally the perceptual range mentioned in algorithm 5 is defined a cone from the sonar sensor outwards with a width of 15 degrees. The cone starts at 20cm, because the sonar does not provide reliable measurements below that distance. The cone ends at 5m which is the maximum distance of a sonar measurement.

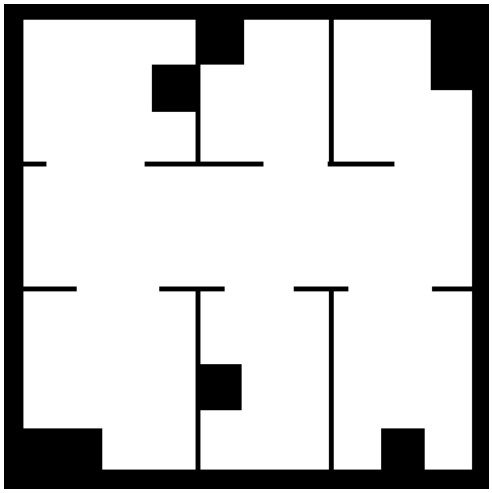
We can use the occupancy maps as input for the aMCL algorithm. An occupancy map always will always introduce some error and uncertainty in the representation of the environment. This is caused by the errors in the data obtained from the odometry and sonar sensors. Therefore, the discussed algorithm will not produce neat map which gives a binary presentation of the locations which are occupied or free.

6 Experiments

For the experiments we used the robot simulation software Player/Stage. This software simulates a simple two dimensional environment and various types of robots with an assortment of peripherals. We used a simulation of a pioneer dx2 and its sonar array. The environment is a map of a simple hallway with different rooms (see figure 8). The virtual size of the hallway is 16 by 16 meters and the resolution is one centimeter for both the sonar values and the values of the odometry. The map is constructed with a raster graphics program. This map can be read by the player/stage software and represents our virtual environment.

The goal for the robot is to localize itself in the simulated environment. This is achieved by using the aMCL algorithm. As explained, the aMCL algorithm has to be given a map of the environment beforehand. We supply different occupancy maps with different amounts of uncertainty. This enables us to determine the influence on the performance of the different amounts of uncertainty present in the occupancy maps.

The robot receives two types of occupancy maps in the experiments. The first uses a map directly derived from the map used by the Player/Stage simulation which we will call the *clean map*. In this case both the robot and the simulator use the same map. The map (see figure 8) was converted to a *logodds* map to make it compatible with our MCL software. This map contains no uncertainty and will give us a top-line for the performance of the aMCL algorithm.



(a) Hallway

Figure 8: Map of a hallway used as the simulated environment in the experiments.

In the second experiment we use occupancy maps generated by the Mapmaker software written by Marck. This software constructs a occupancy map of the environment depicted in figure 8 (See section 5). The Mapmaker software also uses the Player/Stage software package to provide a simulated environment from which it has to construct an occupancy map. We produce two maps (see figure 10) with different amounts of noise and therefore uncertainty. By comparing the performance of the aMCL algorithm to the results of the first experiment we can determine the effect of a increased amount of uncertainty in the occupancy maps.

Internally we convert the log odds values if the occupancy maps to normal chance values which indicate the probability for a certain location to be reachable. To perform the ray tracing as discussed in section 2.5 we need to define a stopping criteria. In this case, if the belief value of a certain grid cell in the occupancy map is smaller than 0.4 the location is considered occupied and will therefore stop the simulated sonar beam and this results in a value for the distance. For the remainder of the article we will refer to this method as *method 1*.

In the third and last experiment we introduce another way of defining the stopping criteria. Instead of simply putting a threshold at a certain value we will determine if the location is occupied on basis of chance. We will determine if a certain location is occupied by the following formula:

$$\textbf{rand} > P(l) = \begin{cases} \text{occupied}, \textbf{true} \\ \text{free}, \textbf{false} \end{cases} \tag{14}$$

Where $P(l)$ represents the probability, $0 \leq P(l) \leq 1$, that a location l is free. And **rand** is a random generator which returns a value between 0 and 1. We will refer to this method as *method 2*. This is different than simply applying a threshold, because the same location can now be both free and occupied. This is more in line with the nature of the probabilistic nature of the map. We will evaluate both methods and see if one produces a better performance on the clean map and the noisy maps made by the Mapmaker software.

In every experiment we vary a certain variable for which we determine its influence on the speed of convergence and the quality of the convergence. Convergence in this case meaning that all particles are close to one another. Convergence is determined by looking at the average distance of the particles to the particle with the greatest weight. The quality of the convergence is determined by looking at the distance of the particle with the greatest weight to the real location of the robot. The real position can be obtained from the simulation and can therefore be compared to the location hypothesized by the particles. This location is the particle with the greatest weight. If at a certain moment the average

distance to the particle with the maximum weight is smaller than 1 meter and the distance of the particle with the maximum weight to the real location of the robot is also smaller than 1 meter we assume the robot successfully localized itself.

Each experiment consists of 100 runs. A run starts with randomly placing the robot at one of nine predetermined locations with a random rotation. Furthermore, the particles, which represent possible robot poses, are given a random pose. Subsequently, the robot is allowed to move through the map guided by a simple obstacle avoid algorithm. The maximum speed of the robot is 0.1 m/s and the we sample the sonars every 0.5 seconds. The run ends after 200 seconds. After every sample taken from the sonar we update all the particles and determine the average distance to the particle with the greatest weight and the distance of the particle with the greatest weight to the real position of the robot. This enables us to determine if the robot has localized itself successfully at a particular time. We count the number of times this has happened for each time step over all runs and we divide this through the total number of runs. This gives us the *ratio of correct localization* for a given a time step. We will use the ratio of correct localization as performance measure of the aMCL algorithm.

7 Results

In this section we present the results of the experiments we conducted. In experiment I, II and, III we will test the influence of several parameters on the performance of the aMCL algorithm. The robot and the simulation both use the same map (see figure 8). In experiment IV we will test the performance with the more noisy maps generated by the Mapmaker software. Finally, in experiment V we introduce and compare another way to handle the uncertainty in the maps.

7.1 Experiment I

This experiment gives us a top-line performance of the aMCL algorithm, because we use an occupancy map without any noise. The results are shown in figure 9 which shows three ratio of localization plots for 1500, 12500, 15000 particles. When we use 1500 particles the ratio of correctly localized runs reaches a maximum of 0.86 at 200 seconds while with 12500 and 15000 we achieve a ratio correct localization close to one. Furthermore, the graph for 1500 shows a slower increase in ratio of localization than with 12500 and 15000 particles. The difference between the results between of 12500 and 15000 is negligible. This result is not very surprising as a larger number of particles represents more hypothesized locations which gives a better more defined representation of the probability distribution function of the location of the robot. This increases the chance of finding the correct location. We will use these results as basis for comparison of the performance of the aMCL with occupancy maps with more uncertainty.

7.2 Experiment II

In the second experiment we investigate the use of maps generated by a scan matching implementation by Marck (2006). We compare the performance of the aMCL algorithm on three different maps. The first is the clean map which is also used in experiment II and depicted in figure 8 and two maps generated by the Mapmaker software written by Marck (See figure 10). By adjusting the noise parameters of the program one map contains a small amount of noise and one map contained a significant amount of noise. In this experiment we used 15000 particles. If we compare the results from this experiment with the result obtained in experiment I, it is clear from figure 11 that the localization is less successful with a map with a higher noise factor. The ratio of localization with the low noise map is worse than the performance and the performance with the clean map. Moreover, the performance with the high noise map is worse than the performance with low noise map. Adding noise has an clear deteriorating effect on the performance of the aMCL algorithm.

7.3 Experiment III

As explained in section 6 we will now use a different way of handling the uncertainty present in the maps. Instead of applying a threshold on the map to determine which areas are free or occupied we will use equation 6 to determine if a location is free or occupied. To evaluate the difference between both

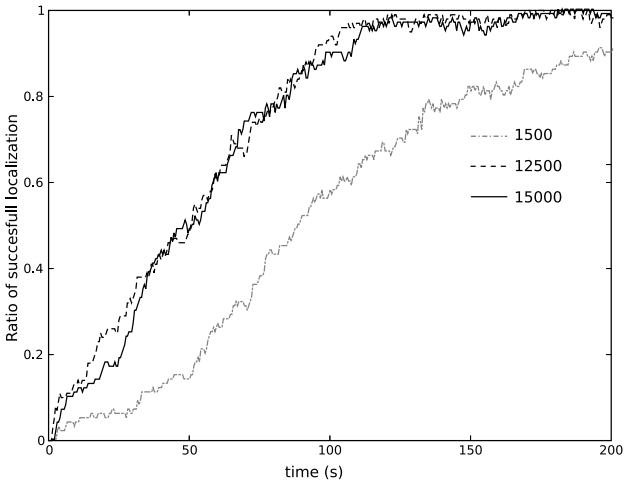
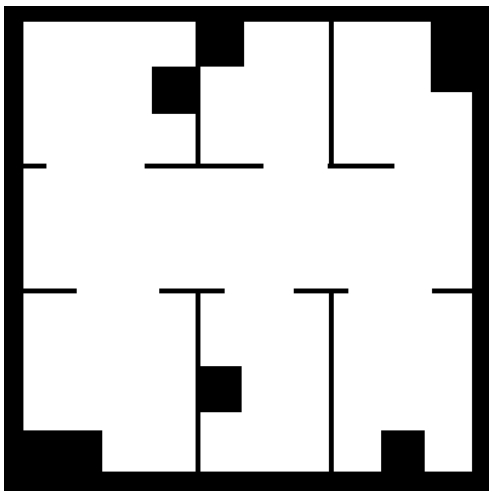
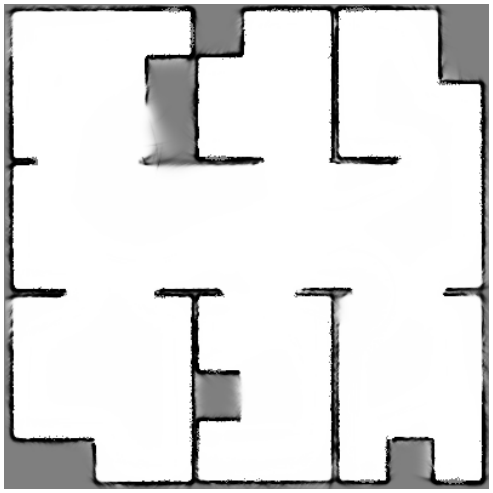


Figure 9: Convergence results for experiments with 1500, 12500, and 15000 particles over 100 runs. This figure shows the ratio of correctly localized robots at a certain time. An experiment is always divided in 100 runs. A run starts with all particles randomly distributed through pose space and the robot is positioned at a random location. Subsequently, the robot tries to determine its location for a period of 200 seconds. This enables us to compute the ratio of correct localization (see section 6). The figure shows that the aMCL algorithm performs better with 12500 and 15000 particles than with 1500 particles.



(a) Clean map



(b) Low noise

(c) High noise

Figure 10: Three occupancy maps used in the experiments. (a) Clean map directly derived from the map used by the simulation. (b,c) Two occupancy maps generated with the mapmaker software from Marck (2006) with different degrees of noise. White means certainly free and black means certainly occupied.

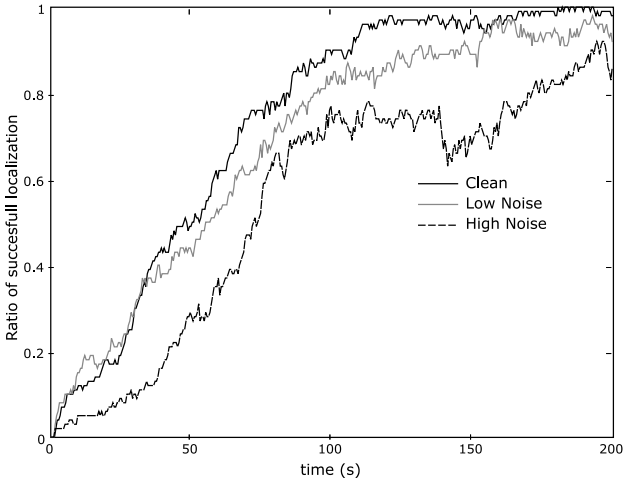


Figure 11: Correct localization ratios for 100 runs and 15000 particles with different maps of the hallway. The maps used in the experiment are the *clean* hallway shown in figure 8 and the low noise map and high noise map seen in figure 10.

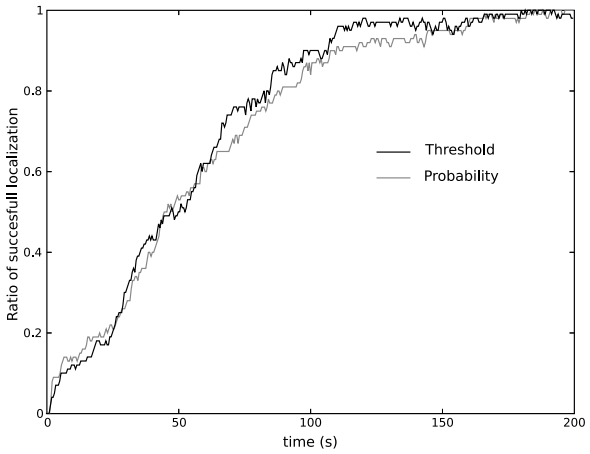
methods we conducted an experiment with the three different maps (See figure 10) with 15000 particles. The results are given in figure 12. We see that for the clean map the performance is essentially the same. This is expected because every location in the map is either a value close to zero if the location is unreachable or is close to one if that location is free. Because there are no intermediate values using method 1 to this map will result in a map identical to the original map seen in 8. When we use equation method 2 to determine if a location is reachable, the result will be very similar to the threshold map. This is caused by the fact that the probabilities in the map are either very close to zero or close to one. Therefore, there is only a small probability that a given location which is free will falsely be identified as occupied and similarly a occupied will not often be identified as free. The consequence of this is that both methods will perform very similar if there is little uncertainty present in the map.

We see different results with the maps constructed by the Mapmaking software. As seen in figure 10 these maps contain more noise and some inaccuracies as a consequence of the manner the maps were constructed. As seen in the original map (figure 10) the gray areas, representing a probability of 0.5 of being occupied, mostly correspond to the wall in the map. If we use method 1, all gray areas will be considered occupied which corresponds pretty well to the original map. On the other hand, if we use method 2 a location with a value of 0.5 than that location will be considered free in half of the cases and occupied in the other half. This will probably deteriorate the performance.

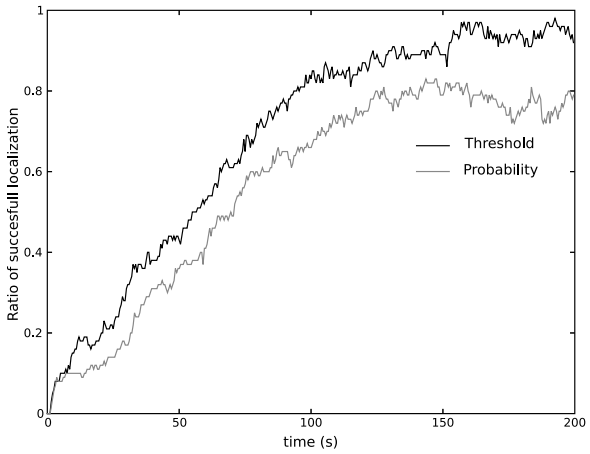
We can see this in figure 12. The performance of the aMCL algorithm is worse if we take the uncertainty into account. We see that the performance with the high noise map differs even more for both methods. Compared to the low noise map, even larger parts of the walls in the high noise map consist of gray areas. This has the consequence that parts of the wall will be falsely considered to be free space in half of the sonar measurements. We must therefore conclude that simply applying a threshold in the case gives a better performance.

8 Discussion

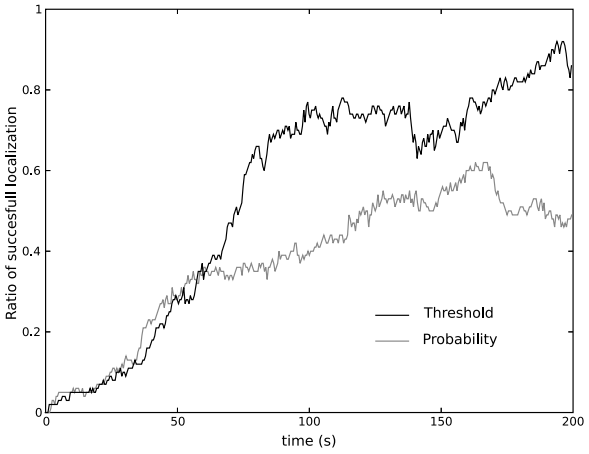
In this paper we investigated the performance of the aMCL algorithm in an artificial environment. We especially looked at the performance if the maps used by the algorithm contain uncertainty. To this end, we compared the performance of the aMCL algorithm on maps with different degrees of uncertainty. We can conclude that the performance of the aMCL algorithm is inversely proportional the amount of noise



(a) Clean hallway



(b) Low noise hallway



(c) High noise hallway

Figure 12: Comparison between two different methods: method 1 (Threshold) and method 2 (Probability). For the handling of the uncertainty present in the occupancy maps maps.

present in the occupancy map. Therefore, the occupancy maps generated by the Mapmaker software will always lead to a worse performance compared to perfect maps from a known environment. We also found that method 1 produces the best results in dealing with this uncertainty.

As explained in chapter 7 we considered a robot successfully localized if all of the particles were sufficiently close together and the particle with the biggest weight was sufficiently close to the real location of the robot. We defined sufficiently (see chapter 7) close as one meter from the robot. In a environment which is 16×16 this is a somewhat large margin and we should possibly reevaluate the data with a smaller distance. I we use a smaller distance, less situations would be considered correctly localized and this could result in different ratios of localizations. For example, the runs with the clean map could show an overall lower ratio of correct localization.

Another point of discussion is that the model we tested has a rather large set of parameters. Several parameters such as α_{slow} and α_{fast} explained in section 4 and the weight and the parameters of the mixture distributions as discussed in 2.5 were chosen quite arbitrarily. They were chosen because these values produce reasonably good results. Further investigations could be conducted to determine the influence of these parameters. Especially α_{slow} and α_{fast}, which determine how fast the robot can recover from a kidnap and how fast the robot will inject random variables which enables the robot to recover from a convergence of the particles to a wrong location, are interesting. A different set of parameters could lead to faster recovery from situations where the localization is mislead by the uncertainty. Other parameters to consider are the frequency of the sonar measurements and the speed of the robot. If the robot moves faster and we maintain a fixed interval of sampling the sonar measurements, the distance traveled and therefore the error in the odometry, would be bigger. This leads to a greater error in the estimation of the pose. Instead of fixed interval to sample the sonars we could have opted to scan every time a certain amount of change in the pose of the robot was realized. This way the robot would be less sensitive to speed changes. If we change these parameters more uncertainty would still lead to a worse performance, but different settings of the parameters could possibly lead to improved performance.

We found that method 1 gives the best performance of the aMCL algorithm. We only tested one setting of this threshold, namely 0.4, but it would be interesting to test the influence of the choice of the threshold. With an occupancy map with little noise the choice of threshold will not give a large difference in performance, but with a map with a significant amount of noise in the occupancy map the choice of threshold can produce a large effect on the performance. If we choose a lower threshold more places will be considered occupied. If choose a higher threshold more places will be considered free. In a map with much uncertainty this can lead to a large difference in interpretation of the map and therefore will influence the performance of the aMCL algorithm. It would be interesting to device a method which could dynamically determine the most optimal threshold. Thereby, improving the performance with uncertain occupancy maps.

Concluding, given enough time and a perfect map the robot will be able to localize itself with the aMCL algorithm. If a map which contains uncertainty such as the maps generated by the Mapmaker software will always lead to a worse performance. We found that the best way to deal with this uncertainty is to simply apply a threshold on the probabilities (method 1) in the map as explained in section 7. This will lead to the best performance of the aMCL algorithm.

References

Buhmann, J., Burgard, W., Cremers, A., Fox, D., Hofmann, T., Schneider, F., et al. (1995). The mobile robot Rhino. *AI Magazine*, *16*(1).

Cox, I. J., & Wilfong, G. T. (Eds.). (1990). *Autonomous robot vehicles*. New York, NY, USA: Springer-Verlag New York, Inc.

Elfes, A. (1989). Using occupancy grids for mobile robot perception and navigation. *Computer*, *22*(6), 46–57.

Fox, D., Burgard, W., & Thrun, S. (1999). Markov localization for mobile robots in dynamic environments. *Journal of Artificial Intelligence Research*, *11*, 391-427.

Fox, D., Thrun, S., & Burgard, W. (2005). *Probabilistic robotics*. MIT Press.

Fox, D., Thrun, S., Burgard, W., & Dellaert, F. (2001). *Particle filters for mobile robot localization* (A. Doucet, N. de Freitas, & N. Gordon, Eds.). New York: Springer.

Hogg, R. V., & Tanis, E. A. (2001). (6th ed.). Prentice Hall.

Marck, J. (2006). *Pose estimation with sonar range finders*. Unpublished doctoral dissertation, University of Groningen.

Moravec, H. P., & Elfes, A. (1985). High resolution maps from wide angle sonar. In *Proceedings of the 1985 ieee international conference on robotics and automation* (p. 116-121).

Thrun, S. (2003). Learning occupancy grid maps with forward sensor models. *Auton. Robots*, *15*(2), 111–127.

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