Loop closing in SLAM using ant colony optimization

B.A. Kupers
Loop closing in SLAM using ant colony optimization

MASTER OF SCIENCE THESIS

For the degree of Master of Science in Systems and Control at Delft University of Technology

B.A. Kupers

May 20, 2013

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology
Abstract

For service robots to be employed in normal human environments, autonomous navigation is an important requirement. To enable autonomous navigation, a robot needs to be able to build a map of any unknown environment. The problem of letting a robot build a map and simultaneously localizing itself within the same map is known in robotics as the Simultaneous Localization and Mapping (SLAM) problem.

An important problem in SLAM is known as the loop closing problem. This problem occurs when the trajectory of the robot contains a large loop. After traversing this loop, the accumulated pose error causes the algorithm to fail to recognize the robot has returned to its original position.

In this thesis, a novel algorithm is introduced to improve the loop closing behaviour of a widely used SLAM algorithm, called FastSLAM. FastSLAM uses a particle filter to estimate the pose of the robot. The resampling step in the particle filter algorithm causes particle depletion, which inhibits correct loop closures. The proposed algorithm, called Ant Colony Optimization (ACO-)SLAM, uses ACO to improve the resampling step of the FastSLAM algorithm. ACO-SLAM does this by incorporating a measure of the map consistency and a measure for correct loop closures into the resampling step.

Simulations are used to compare the performance of both algorithms with varying noise settings. Robby, the personal robot developed by the Delft Biorobotics lab, is used for experiments to test the performance of the algorithms in a realistic human environment.
# Table of Contents

## Preface

1. Introduction
   1-1 Formulation of the SLAM problem .............................................. 2
   1-2 Loop closing problem ............................................................... 2
   1-3 Main goal of this thesis ............................................................ 4
   1-4 Case study .................................................................................. 4
   1-5 Thesis outline ............................................................................ 6

2. Algorithms
   2-1 Introduction .............................................................................. 7
   2-2 Definitions
      2-2-1 Reference frames ................................................................. 7
      2-2-2 Robot and sensor pose ......................................................... 8
      2-2-3 Sensors and observed data ................................................... 9
      2-2-4 Landmarks and the map ....................................................... 9
      2-2-5 Control input ..................................................................... 10
   2-3 Overview of the SLAM algorithm .................................................. 10
   2-4 Mathematical models
      2-4-1 2D kinematic robot model .................................................. 11
      2-4-2 3D kinematic model ............................................................... 12
      2-4-3 Observation model ............................................................... 12
   2-5 Visual odometry .......................................................................... 13
   2-6 Standard algorithm: FastSLAM .................................................... 14
      2-6-1 General concept behind FastSLAM ...................................... 14
      2-6-2 FastSLAM algorithm, step by step ...................................... 15
      2-6-3 Loop closing in FastSLAM .................................................... 17

Master of Science Thesis

B.A. Kupers
Table of Contents

2-7 The ACO-SLAM algorithm ................................................. 18
   2-7-1 General concept of ACO-SLAM .................................. 18
   2-7-2 ACO-SLAM algorithm, step by step ................................ 19
2-8 Summary ............................................................................ 23

3 Simulated experiments with FastSLAM and ACO-SLAM ............ 25
   3-1 Introduction .................................................................... 25
   3-2 Simulation environments .................................................. 25
   3-3 Parameters and settings ..................................................... 27
   3-4 Performance measures ....................................................... 28
   3-5 Simulations with FastSLAM ................................................ 29
       3-5-1 Simple environment results with FastSLAM ..................... 29
       3-5-2 @Home lab environment results with FastSLAM .............. 37
   3-6 Simulations with ACO-SLAM ............................................. 41
       3-6-1 Convergence of the ACO ........................................... 41
       3-6-2 Simple environment results with ACO-SLAM ................. 43
       3-6-3 @Home lab environment results with ACO-SLAM .......... 47
   3-7 Summary ............................................................................ 49

4 Experiments with FastSLAM and ACO-SLAM ......................... 51
   4-1 Introduction .................................................................... 51
   4-2 Experiment setup .............................................................. 51
   4-3 Software for the experiments .............................................. 52
       4-3-1 Implementation of FastSLAM .................................... 53
       4-3-2 Implementation of ACO-SLAM .................................. 54
   4-4 Experiments with FastSLAM .............................................. 54
       4-4-1 Experiment results with FastSLAM ............................... 55
   4-5 Experiments with ACO-SLAM ............................................ 60
       4-5-1 Convergence of the ACO algorithm ............................. 60
       4-5-2 Experiment results with ACO-SLAM ............................ 60
   4-6 Summary ............................................................................ 63

5 Discussion and conclusions ..................................................... 65
   5-1 Introduction .................................................................... 65
   5-2 Discussion on the simulation results ................................... 65
   5-3 Discussion on the experiment results ................................. 67
   5-4 Main conclusions and future research ................................. 68
   5-5 Closing remarks .............................................................. 69

A Appendices ............................................................................ 71
   A-1 Extra figures .................................................................... 71

B.A. Kupers Master of Science Thesis
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bibliography</td>
<td>75</td>
</tr>
<tr>
<td><strong>Glossary</strong></td>
<td>79</td>
</tr>
<tr>
<td>A-2 List of symbols</td>
<td>79</td>
</tr>
<tr>
<td>A-3 List of abbreviations</td>
<td>81</td>
</tr>
</tbody>
</table>
List of Figures

1-1 Top: Map obtained by odometry, Middle: Map obtained by SLAM, Bottom: Map obtained after correct loop closure [1] .......................................................... 3
1-2 Picture of the Robotics Lab .......................................................... 5
1-3 Robby the service robot .......................................................... 5

2-1 The tree reference frames used in SLAM ........................................ 8
2-2 Block diagram of common SLAM algorithm ................................... 11
2-3 Transformation of point m to the sensor frame ................................ 13
2-4 Left: low spread in trajectories. Right: high spread in trajectories .... 18
2-5 Top: 5 particle trajectories, Middle: possible nodes, Bottom: One possible ant path 20

3-1 Simple circular environment ...................................................... 26
3-2 Model of the @Home lab .......................................................... 26
3-3 3D map of the simple environment with 50 particles ....................... 29
3-4 Map error for 50 particles .......................................................... 29
3-5 Pose error for 50 particles .......................................................... 30
3-6 Example of a boxplot ............................................................... 31
3-7 Boxplots of the map errors for varying number of particles .......... 32
3-8 Boxplots of the pose errors for varying number of particles .......... 32
3-9 Boxplots of the map errors for varying odometry noise and 25 particles .... 33
3-10 Boxplots of the pose errors for varying odometry noise and 25 particles .... 34
3-11 Boxplots of the map errors for varying observation noise ............... 35
3-12 Boxplots of the pose errors for varying observation noise ............... 35
3-13 Boxplots of the map errors for increasing observation noise .......... 36
3-14 Boxplots of the pose errors for increasing observation noise .......... 36
3-15 Map error for $n_p = 50$ ............................................................ 37
3-16 Pose error for \( n_p = 50 \) ................................................. 37
3-17 Boxplots of the map errors for varying number of particles ................................................. 38
3-18 Boxplots of the pose errors for varying number of particles ................................................. 39
3-19 Boxplots of the map errors for varying odometry noise and 25 particles ................................................. 39
3-20 Boxplots of the pose errors for varying odometry noise and 25 particles ................................................. 40
3-21 Boxplots of the map errors for varying observation noise ................................................. 40
3-22 Boxplots of the pose errors for varying observation noise ................................................. 41
3-23 Added particle weight for 20 tests ................................................. 42
3-24 Development of the pheromone levels on two nodes ................................................. 43
3-25 Development of the average map error with ACO-SLAM and 25 particles ................................................. 44
3-26 Pose error for ACO-SLAM ................................................. 44
3-27 Boxplots of the map errors for varying odometry noise ................................................. 45
3-28 Boxplots of the pose errors for varying odometry noise ................................................. 45
3-29 Boxplots of the map errors for varying observation noise ................................................. 46
3-30 Boxplots of the pose errors for varying observation noise ................................................. 46
3-31 Boxplots of the map errors for varying odometry noise ................................................. 47
3-32 Boxplots of the map errors for varying odometry noise ................................................. 47
3-33 Boxplots of the map errors for varying observation noise ................................................. 48
3-34 Boxplots of the pose errors for varying observation noise ................................................. 49

4-1 Top view of the @Home lab with the ground truth of the robot trajectory ................................................. 52
4-2 Estimated trajectory with 25 particles ................................................. 55
4-3 Development of the position error for 25 particles ................................................. 56
4-4 Estimated landmark positions for 25 particles ................................................. 56
4-5 Estimated trajectory with 50 particles ................................................. 57
4-6 Development of the position error for 50 particles ................................................. 57
4-7 Estimated landmark positions for 50 particles ................................................. 58
4-8 Boxplot of the robot position errors for 25 and 50 particles ................................................. 58
4-9 Boxplot of the landmark position errors for 25 and 50 particles ................................................. 59
4-10 Normalized pheromone per particle ................................................. 60
4-11 Estimated trajectory with ACO-SLAM ................................................. 61
4-12 Development of the position error for ACO-SLAM ................................................. 61
4-13 Estimated landmark positions for ACO-SLAM ................................................. 62
4-14 Boxplot of the robot position errors for ACO-SLAM and FastSLAM ................................................. 62
4-15 Boxplot of the landmark position errors for ACO-SLAM and FastSLAM ................................................. 63

A-1 Boxplots of the map errors for varying odometry noise and one particle in the simple environment ................................................. 71
A-2 Boxplots of the pose errors for varying odometry noise and one particle in the simple environment ................................................. 72
A-3 Boxplots of the map errors for varying odometry noise and one particle in the @Home environment ................................................. 72
A-4 Boxplots of the pose errors for varying odometry noise and one particle in the @Home environment ................................................. 73
List of Tables

3-1 Settings for all the FastSLAM simulations ........................................ 28
3-2 Settings for all the ACO-SLAM simulations ....................................... 43
Preface

The master thesis that is lying in front of you is the result of a journey into the world of robotics. My enthusiasm for robotics was first sparked in late 2011 when I joined the RoboCup@Home team of the university of Delft. Since then, the RoboCup@Home competition has brought me halfway around the world and back. In the almost one and a half year long period I was active in the team, I learned an incredible amount about robotics and today, my interest in the field has never been higher.

I would like to thank my supervisors, Robert Babuška and Rob Prevel for always staying critical, giving me useful advice and providing me with the necessary structure. Together with robot Robby, the "three Robs" formed the perfect team for me.

Also, I would like to thank my parents and sister for helping me through difficult times. I promise I’ll come to Groningen more often than I did in the last few months. Lastly, I owe a great deal to Florien, who has always been there for me. She has never stopped believing in me, even at times when I doubted myself. Thank you very much for everything you have done for me. I can’t wait for the future.
Chapter 1

Introduction

Since the first half of the 20th century, humanoid robots have spoken to the imagination of millions of people, especially since the books by Isaac Asimov or the play Rossum’s Universal Robots by Karel Capek. Although the first industrial robot, Unimate, was put into service in 1961 by General Motors, the desired or even feared future of humanoid robots having a place in everyday life is still far from realized. Through advances in control and artificial intelligence, the field of robotics has made big steps toward creating robots that can be used in everyday life. Last two decades the interest in the field has greatly increased. One of the goals of current research is to take the robots from the laboratories into the real world. In other words, to create fully autonomous robots that can be used in any (home) environment and aid people in simple tasks.

One initiative to promote research in these domestic robots is the RoboCup@Home, which aims to develop service robots that can be used in a realistic home environment. The RoboCup@Home is the largest international annual competition for autonomous service robots. Some of the areas this competition focuses on are human-robot interaction and cooperation, navigating through and mapping of dynamic environments, computer vision, object recognition and manipulation, adaptive behavior and behavior integration.

The @Home Team of Delft Robotics has developed service robot Robby. Robby can be controlled by speech, it can recognize faces and complete simple household tasks. For Robby to be employed in normal human environments, autonomous navigation is an important requirement. To enable autonomous navigation, the robot needs to be able to build a map of any unknown environment. The problem of getting a robot to build a map of the environment while moving through the same environment is known in robotics as the Simultaneous Localization and Mapping (SLAM) problem.

In the following section the SLAM problem will be formulated and some existing algorithms will briefly be discussed. Section 1-2 introduces the loop closing problem. This is a specific problem that affects most SLAM algorithms. Section 1-3 presents the main objective of this thesis. A new algorithm is introduced to reduce the loop closing problem. The new algorithm is tested in a realistic case study, which is described in Section 1-4. This chapter ends with an overview of the coming chapters of this thesis.
1-1 **Formulation of the SLAM problem**

Simultaneous Localization and Mapping, or SLAM, is the problem of letting a robot build a map of an unknown environment while at the same time localizing itself within that map. The goal of any SLAM algorithm is to produce a map and an estimation of the robot pose. The map is a list of the positions of easily detectable points in the environment. These points can be distinct points on walls, edges of furniture or other items in the environment. The robot pose consists of the position of the robot in 3D and the orientation of the robot, given by the yaw, pitch and roll angles.

In recent years much research has been done on the topic of SLAM and many different algorithms have been developed for numerous applications. Every SLAM application needs a sensor to measure the position of points in the environment. Commonly used sensors include laser scanners [2], monocular camera’s [3], stereo vision systems [4] and Microsoft’s Kinect[5]. Most algorithms use odometry to calculate the initial estimate of the robot pose. This pose estimate is refined by using measurements of the environment, made by the other sensor.

The SLAM algorithms can be divided into probabilistic approaches and scan-matching methods. The probabilistic approaches estimate a probability distribution that describes the joint posterior density of both the robot pose and the map. This probability distribution can be estimated by an Extended Kalman Filter (EKF) [6], an Unscented Kalman Filter (UKF) [7] and by Rao-Blackwellized Particle Filters (RBPF)[8], [9]. The scan-matching methods build a map by merging consecutive scans of the environment. Optimization algorithms are used to optimally align two consecutive scans. The pose increment of the robot is found by calculating the rotation and the translation that were applied to the two scans in order to align them. Scan-matching SLAM algorithms all have the Iterative Closest Point (ICP) algorithm [10] at their core.

The main advantage of ICP over the other (probabilistic) SLAM algorithms is the relatively high computational speed and the ease of implementation. It does however, not provide any information on the likelihood of the estimated pose or the map. The disadvantage of using the Extended Kalman Filter for SLAM is that linearized versions of the motion and observation models are used, which can cause estimation errors. Neither the UKF nor the RBPF have this problem. Compared to the UKF, the RBPF is better at handling non-Gaussian noise.

Initial papers on SLAM all considered the 2D case. The extension of these algorithms to full 3D is mathematically straightforward. However, the added complexity makes SLAM algorithms in 3D more computationally expensive. In recent years more algorithms capable of 3D SLAM were introduced, for example in [11] and [12].

1-2 **Loop closing problem**

This thesis will focus on a particular problem that plagues most SLAM algorithms. This problem is called the loop closing problem. It is the problem of detecting the return to a previously visited location. The loop closing problem occurs when the trajectory of the robot contains a large loop. After traversing this loop, the accumulated pose error causes the algorithm to fail to recognize the robot has returned to its original position. When this
happens, the algorithm will continue adding new parts to the map, while the same parts are in fact already in the map, but at a slightly different locations.

The Loop closing problem is illustrated in Figure 1-1, taken from [1].

In this example, a robot moves through a rectangular hallway and produces a map in 2D. The black lines represent walls. The red line represents the estimated robot trajectory. The top picture shows the map and the robot trajectory based only on the odometry information. It can be seen that the map based on odometry is quite far from reality; the map should be rectangular and the end of the trajectory should overlap with the first part of the trajectory. The middle picture is made using a SLAM algorithm. This map is a closer representation of the true environment, however the end of the trajectory doesn’t overlap with the first part of the trajectory. This can be seen from the two parallel red lines on the right, which should overlap. Also the resulting map shows that the some parts of the map are mapped twice at different locations. The bottom picture shows the map after a loop closing algorithm is applied. A single consistent map now results.

Several algorithms have been developed to tackle the loop closing problem. One approach is to actively try to recognize previously visited locations. Possible loop closures can be detected by finding correspondences between the current observed image and all the previously observed images, as was first implemented in [13]. In [14], the same principle is applied, but the search for correspondences is much more efficient. Here, each observed image of the environment is represented by a collection of attributes, or visual words. The algorithm uses these attributes to efficiently search for correspondences between images. The disadvantage of these methods is that the algorithm that checks for correspondences is quite computationally expensive.
Hierarchical SLAM \cite{15} takes a different approach to the loop closing problem. In this algorithm the entire map is cut up into smaller submaps. After the submaps have been built, an optimization step is used to align the submaps to form a single consistent map.

To solve the loop closing problem, without the need of a posterior optimization step, techniques from artificial intelligence can be used to integrate the detecting and closing of loops into the overall algorithm. In \cite{16} a genetic algorithm is applied to find the optimal map and in \cite{17} swarm optimization is used. However, these algorithms can not be used in a probabilistic framework.

\section*{1-3 Main goal of this thesis}

This thesis proposes a new algorithm called Ant Colony Optimization (ACO-SLAM). This algorithm uses ACO \cite{18} to improve the loop closure performance of FastSLAM \cite{8}, a widely used algorithm based on a RBPF.

The choice for FastSLAM is based on its good real-time performance on other applications, robustness against non-linear observation models and dynamic models, and its ability to cope with non-Gaussian noise. Another advantage is the fact that several hypotheses of the robot trajectory are maintained simultaneously, making it more likely a global optimal solution is found. A problem in FastSLAM that inhibits the loop closing behavior is caused by the resampling step in the particle filter algorithm. In the new algorithm, ACO will be used to optimize this resampling step.

The choice for the use of ACO to optimize the FastSLAM algorithm is made for two reasons: Firstly, it is quite a new approach in SLAM, so a lot of research opportunities still exist. Secondly, the ACO-algorithm is aimed at finding optimal paths between several points, which makes this a logical choice to optimize the estimated trajectory of the robot. One approach exists that uses ACO improve the performance of FastSLAM \cite{19}, however this approach does not optimize the resampling step of the FastSLAM algorithm.

The main question that will be investigated in this thesis is:

\emph{Does optimizing the resampling step of the FastSLAM algorithm by using ant colony optimization lead to increased loop closing performance?}

\section*{1-4 Case study}

The proposed algorithm will be implemented in simulations and in experiments. The results of the simulated and real experiments will be compared to the existing FastSLAM algorithm.

The environment in which the algorithms are tested has a large influence on the performance of the algorithm. (Both in terms of calculation time and estimation errors of the robot pose and the map.) For the final algorithm to be usable for the RoboCup@Home challenge, the environment needs to represent a normal home environment. For this purpose, the RoboCup@Home lab, in the TU Delft Biorobotics Lab is very suitable. It is furnished deliberately to represent a normal human environment. By including adjacent hallways, the robots’ trajectory can be a cyclic one. This means the robot can return to its starting position.
without passing the same point twice. This is a good way to test the loop closing abilities of the algorithm.

The lab is shown in the figure below:

![Picture of the Robotics Lab](image1.jpg)

**Figure 1-2:** Picture of the Robotics Lab

The algorithms will be implemented on Robby the service robot. Robby is based on the Delft Personal Robot 2 (DPR2), developed by the Delft Biorobotics laboratory. This platform has four wheels, two of which are driven by DC-motors. The motors are controlled independently, this enable the robot to turn around its yaw axis. A picture of the robot is given below:

![Robby the service robot](image2.jpg)

**Figure 1-3:** Robby the service robot
1-5 Thesis outline

The next chapter provides background information on the FastSLAM algorithm and the proposed ACO-SLAM algorithm. The necessary concepts are defined, mathematical models are presented and the algorithms are explained. Chapter 3 presents the simulated experiments. The simulated environments and the settings that are used are discussed. Also the results of the simulations will be presented. Chapter 4 is about the experiments done with the robot. The implementation of the algorithms is discussed and the results are presented. The final chapter discusses the results found in Chapters 3 and 4. Conclusions are drawn based on these results and suggestions for future research are made.
Chapter 2
Algorithms

2-1 Introduction

This chapter provides background on SLAM and the mathematical models that are used for SLAM. Firstly, all the necessary concepts will be defined in Section 2-2. Secondly, Section 2-3 explains how the probabilistic SLAM algorithms work. The stochastic dynamic models that describe the robot and sensor movement are presented in Section 2-4. Also an observation model is given to describe the observations made by the sensor. Thirdly, one commonly used SLAM algorithm: FastSLAM is introduced in Section 2-6. This algorithm will be used in the simulations and the experiments as a baseline test. Finally, Section 2-7 introduces a novel algorithm, ACO-SLAM, that is designed to improve the loop closure performance of the current SLAM algorithms.

2-2 Definitions

This section introduces the necessary definitions that will be used throughout the report. Definitions are given of the reference frames, the robot and sensor poses, the observations, the map and the control input.

2-2-1 Reference frames

In this thesis, three reference frames are used for SLAM: the world frame, the robot frame and the sensor frame. The world frame is defined as a stationary Cartesian frame in 3D Euclidean space. It is attached to a fixed point in space. The robot frame is a Cartesian frame rigidly attached to the base of the robot. The position and orientation of the robot frame are measured with respect to the world frame. The sensor frame is a Cartesian frame rigidly attached to the sensor of the robot. The transformation between the sensor frame and the robot frame is assumed to be known. The position and orientation of the sensor frame are measured with respect to the world frame.
The reference frames are drawn in Figure 2-1. In this figure, \( \begin{bmatrix} x & y & z \end{bmatrix}^T \) denotes the position of the robot frame with respect to the world frame. The vector \( \begin{bmatrix} x_s, y_s, z_s \end{bmatrix}^T \) denotes the position of the sensor frame with respect to the world frame.

### 2-2-2 Robot and sensor pose

The pose of the robot is given by the position and orientation of the robot frame. The position is denoted by the \( x, y, z \) coordinates, relative to the world frame. The orientation is given by the pitch, roll and yaw angles of the robot, denoted by \( \phi, \theta, \psi \). The world frame and the robot frame are shown in Figure 2-1. The pose at time \( k \) is defined as the vector:

\[
\mathbf{x}_k = \begin{bmatrix} x_k \\ y_k \\ z_k \\ \phi_k \\ \theta_k \\ \psi_k \end{bmatrix} \tag{2-1}
\]

Note that the pose vector \( \mathbf{x}_k \) is denoted in boldface to differentiate it from the \( x \)-position of the robot, denoted by \( x_k \).

The pose of the sensor is also measured with respect to the world frame. Its pose is given by:

\[
\mathbf{x}_{s,k} = \begin{bmatrix} x_{s,k} \\ y_{s,k} \\ z_{s,k} \\ \phi_{s,k} \\ \theta_{s,k} \\ \psi_{s,k} \end{bmatrix} \tag{2-2}
\]
In this thesis, the hat symbol is used to denote the estimate of a variable. For example, the estimate robot pose is denoted by $\hat{x}_k$.

### 2-2-3 Sensors and observed data

The sensor used in this project is a Microsoft Kinect. The Kinect produces a 2D RGB image and a depth image. This depth image is created by measuring the distance between the sensor and the physical point in space for each pixel in the 2D RGB image. The accuracy of the depth measurement increases quadratically with the distance of the physical point to the sensor [20]. A Kanade-Lucas-Tomase (KLT) tracker [21] is used to extract and track features from the image. Features are interesting points in an image which can be recognized and tracked robustly. Features are sharp changes in contrast or color. Examples of features are the corners of a table or a black spot on a white wall.

The position of each feature with respect to the sensor frame is calculated using the Point Cloud Library (PCL) [22]. The PCL is an open source project for image and point cloud processing. It provides software that is able to process the data from the Kinect and convert its measurements to a set of 3D points. This software takes the RGB image and the depth information as inputs and produces the Cartesian coordinates of the features in the sensor frame.

The position of each feature with respect to the sensor frame is denoted by:

$$
z_{n,k} = \begin{bmatrix} x_{z,n,k} \\ y_{z,n,k} \\ z_{z,n,k} \end{bmatrix}
$$

(2-3)

The observation at time $k$ is defined as the vector containing all features stacked together:

$$
z_k = \begin{bmatrix} z_{1,k} \\ z_{2,k} \\ \vdots \\ z_{n_k,k} \end{bmatrix}
$$

(2-4)

### 2-2-4 Landmarks and the map

A landmark is a distinct point in space, given by a $x, y, z$-position with respect to the world frame. The landmarks are obtained by transforming the features from the sensor frame to the world frame. Each landmark is numbered by $n = 1 \ldots n_m$. Where $n_m$ is the total number of landmarks observed from the first until the current time step. A landmark with number $n$ at time $k$ is denoted by:

$$
m_{n,k} = \begin{bmatrix} x_{m,n,k} \\ y_{m,n,k} \\ z_{m,n,k} \end{bmatrix}
$$

(2-5)

The map is the set of all landmarks. During SLAM, newly observed landmarks will continuously be added to the map, so the map can grow unboundedly large. For the algorithm’s sake, the map is represented by a single vector, in which all landmarks are stacked.
\[ m_k = \begin{bmatrix} m_{1,k} \\ m_{2,k} \\ \vdots \\ m_{n_m,k} \end{bmatrix} \] (2-6)

Since the landmark positions cannot be measured directly, all landmarks are in fact estimated landmarks. For readability, the hat symbol is omitted.

2-2-5 Control input

The control input a user can send to the robot consists of a translational velocity and a rotational velocity. The translational velocity is defined to be in the \( x \)-direction of the robot frame and is denoted by \( u_k \). The rotational velocity is defined as a rotation around the \( z \)-axis of the robot frame and is denoted by \( \omega_k \). The input vector is given by:

\[ u_k = \begin{bmatrix} u_k \\ \omega_k \end{bmatrix} \] (2-7)

PID controllers in the robot convert this input to the signals sent to the motor drivers. The dynamics of the control loops are neglected and instead the velocities are assumed to be reached instantaneously. This assumption can be made because the velocity of the robot is relatively low compared to the dynamics of the motors and controllers. The input vector that is used for the SLAM algorithm is therefore given by (2-7).

2-3 Overview of the SLAM algorithm

Most probabilistic SLAM methods follow the same structure. At every discrete time step \( k \), the inputs to the SLAM algorithm are: the control input to the robot (\( u_k \)) and the observation made by the sensors (\( z_k \)). The desired outputs of the SLAM algorithm are: the estimated robot pose, \( \hat{x}_k \) in 6D and the positions of landmarks in 3D, represented by \( m_k \).

The general structure every SLAM algorithm follows, is represented by the block diagram in Figure 2-2. Firstly, the current pose estimate is calculated according to a dynamic model of the robot. This dynamic model uses the previous pose estimate and the last control input to compute the new estimated pose. Then, the current measurement is estimated by an observation model. This observation model uses the current pose estimate and the previous landmark positions to make a prediction of the current measurement.

To obtain the error signal, the estimated measurement is subtracted from the real measurement. This error signal, together with the pose estimate and the previous landmark positions is used in the SLAM update step. The update step is used to improve the pose estimate and to update the landmark positions. This is usually done by some filter, e.g. a Rao-Blackwellized particle filter (RBPF) [23], an Extended Kalman Filter (EKF) [6], or other.

After this step, new landmarks can be added to the map. A new landmark is first added to a temporary map and only if it is observed for a minimum of \( n_{min} = 5 \) times it is added to
the overall map. Finally, the current pose estimate and the current landmark positions are fed back into the algorithm.

Since the SLAM algorithm will most likely be used in an unknown environment, the initial conditions are all set to zero. The map starts as an empty set and the initial robot pose is set to \( \hat{x}_0 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T \). This initial pose is arbitrary, since any other initial pose would lead to the same map, only translated and rotated with respect to the world frame.

### 2-4 Mathematical models

#### 2-4-1 2D kinematic robot model

As stated before, the pose of the robot is given by its position, given by \( x, y, z \) coordinates, and its orientation, given by the yaw, pitch and roll angles: \( \phi, \theta \) and \( \psi \). Since the robot movement is constrained to the \( x, y \)-plane, a simple 2D model can be used to estimate the current pose based on the previous pose and current control inputs. This 2D model only includes the \( x, y \)-position of the robot and a rotation around the yaw axis.

The estimate of the robot pose is calculated according to the kinematic model given by: (2-8).

Here, the robot pose \( \mathbf{x}_k \) is represented by the vector \( \begin{bmatrix} x_k & y_k & \phi_k \end{bmatrix}^T \). The input \( \mathbf{u}_k \) consists of a linear velocity, \( u_k \) and an angular velocity \( \omega_k \). The difference in position and yaw are calculated by multiplying the input with the length of a time step, \( \Delta t \). \( w_k \) represents the process noise, this can be zero mean white noise or any other stochastic variable.

![Block diagram of common SLAM algorithm](image-url)
\[
\begin{bmatrix}
  x_{k+1} \\
y_{k+1} \\
\phi_{k+1}
\end{bmatrix} =
\begin{bmatrix}
x_k \\
y_k \\
\phi_k
\end{bmatrix} +
\begin{bmatrix}
u_k \Delta t \cos(\psi_k + \omega_k \Delta t) \\
u_k \Delta t \sin(\phi_k + \omega_k \Delta t)
\end{bmatrix} + w_k
\]

(2-8)

### 2-4-2 3D kinematic model

Since the sensor is not constrained to the \(x, y\)-plane, its motion needs to be described in three dimensions. The sensor is modeled as a single rigid body. Its pose is denoted by \(x_{s,k}\) and consists of the position and the orientation of the sensor with respect to the world frame.

As an input for this kinematic model the change in sensor pose between \(k\) and \(k - 1\) is used. This can be calculated in two ways. The first way calculates the sensor motion by using odometry of the robot base and applying the known transformation between the robot frame and the sensor frame. The second method uses visual odometry to calculate the movement of the sensor, directly as will be described in section 2-5. This change in sensor pose is denoted by \(\Delta x_{s,k}\):

\[
\Delta x_{s,k} = \begin{bmatrix}
\Delta x_{s,k} \\
\Delta y_{s,k} \\
\Delta z_{s,k} \\
\Delta \phi_{s,k} \\
\Delta \theta_{s,k} \\
\Delta \psi_{s,k}
\end{bmatrix}
\]

The current sensor pose is calculated from the previous pose and \(\Delta x_{s,k}\) by:

\[
\begin{bmatrix}
x_{k+1} \\
y_{k+1} \\
z_{k+1} \\
\phi_{k+1} \\
\theta_{k+1} \\
\psi_{k+1}
\end{bmatrix} =
\begin{bmatrix}
x_k \\
y_k \\
z_k \\
\phi_k \\
\theta_k \\
\psi_k
\end{bmatrix} +
\begin{bmatrix}
c\phi \cdot c\theta & c\phi \cdot s\theta \cdot s\psi - s\phi \cdot c\psi & c\phi \cdot s\theta \cdot c\psi + s\phi \cdot s\psi & 0 & 0 & 0 \\
s\phi \cdot c\theta & s\phi \cdot s\theta \cdot s\psi + c\phi \cdot c\psi & s\phi \cdot s\theta \cdot c\psi - c\phi \cdot s\psi & 0 & 0 & 0 \\
-s\theta & c\theta \cdot s\psi & c\theta \cdot c\psi & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
\Delta x \\
\Delta y \\
\Delta z \\
\Delta \phi \\
\Delta \theta \\
\Delta \psi
\end{bmatrix} + w_k
\]

(2-9)

with:

\[
\begin{align*}
c\phi &= \cos(\frac{\Delta \phi}{2}) \\
c\theta &= \cos(\frac{\Delta \theta}{2}) \\
c\psi &= \cos(\frac{\Delta \psi}{2}) \\
s\phi &= \sin(\frac{\Delta \phi}{2}) \\
s\theta &= \sin(\frac{\Delta \theta}{2}) \\
s\psi &= \sin(\frac{\Delta \psi}{2})
\end{align*}
\]

(2-10)

### 2-4-3 Observation model

The observation model projects every landmark from the 3D world frame to the sensor frame. The input to the observation model is a landmark position \(m_{n,k}\) and the pose of the sensor.
Figure 2-3: Transformation of point m to the sensor frame

\[ \begin{bmatrix} \hat{x}_m \\ \hat{y}_m \\ \hat{z}_m \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \hat{\psi} & \sin \hat{\psi} \\ 0 & -\sin \hat{\psi} & \cos \hat{\psi} \end{bmatrix} \begin{bmatrix} \cos \hat{\theta} & 0 & -\sin \hat{\theta} \\ 0 & 1 & 0 \\ \sin \hat{\theta} & 0 & \cos \hat{\theta} \end{bmatrix} \begin{bmatrix} \hat{x}_z \\ \hat{y}_z \\ \hat{z}_z \end{bmatrix}. \] (2-11)

2-5 Visual odometry

The pose change of the sensor can be estimated by the kinematic models in the previous section. However, errors in the transformation between the robot frame and the sensor frame can make this method unreliable. A different way of calculating pose change is visual odometry. This method uses the the difference between two consecutive measurements to estimate the sensor pose change.

As stated before, each measurement consists of a list of features. These features are transformed to points in 3D, creating a point cloud. The measurement \( \mathbf{z}_k \) is a stacked vector of these 3D points. Two consecutive feature vectors, \( \mathbf{z}_{k-1} \) and \( \mathbf{z}_k \) are considered. The features
from \( \mathbf{z}_{k-1} \) are tracked and matched against the features in \( \mathbf{z}_k \). The data association matrix, \( N \) keeps track of the corresponding points between \( \mathbf{z}_{k-1} \) and \( \mathbf{z}_k \). A value of 1 is assigned to entry \( i,j \) of \( N \) if the \( i \)th point in \( \mathbf{z}_{k-1} \) corresponds to the \( j \)th point in \( \mathbf{z}_k \). Otherwise, a value of zero is assigned. The two point clouds are overlayed and the second point cloud, \( \mathbf{z}_k \), is rotated and translated such that the corresponding features of both point clouds are as close together as possible. The rotation applied to point cloud \( \mathbf{z}_k \) is denoted by \( R \). The translation is denoted by \( t \). The pose change of the sensor is easily deduced from this rotation and translation.

The pose change is calculated by the Iterative Closest Point (ICP) algorithm, as used by [24], and [25], among others. The ICP algorithm finds the roto-translation that minimizes the cost function given by: (2-12). This cost function represents the distance between the points that are matched between measurement \( \mathbf{z}_{k-1} \) and \( \mathbf{z}_k \).

\[
E(R,t) = \sum_{i=1}^{n_i} \sum_{j=1}^{n_j} N_{i,j} \| \mathbf{z}_{k-1,i} - (Rz_{k,j} + t) \|^2
\]  

(2-12)

The cost function given by (2-12) must be minimized to find the best estimate of the roto-translation. This can be done indirectly by optimization methods, such as Levenberg-Marquardt, gradient descent or the Gauss-Newton method. It is however much faster to calculate the transformation in closed form. This is done according to the method in [26].

2-6 Standard algorithm: FastSLAM

This section explains the FastSLAM algorithm. This algorithm is widely used in practice and will serve in this thesis as a baseline test. Firstly, the general concept behind FastSLAM will be explained. After this, the separate steps of the algorithm will be described in detail. Note that in this section, the hat symbols are omitted for the estimated robot pose \( \hat{x}_k \) and the estimated landmark positions \( \hat{m}_k \). This is done to increase the readability and because all the poses and landmark positions are estimates. There are two versions of the FastSLAM algorithm, FastSLAM [8] and FastSLAM 2.0 [27]. The main difference between the two is the proposal distribution, as will be explained later. In this thesis FastSLAM 2.0 will be used, because of the increased performance. For readability, FastSLAM 2.0 will be refered to as FastSLAM.

2-6-1 General concept behind FastSLAM

The FastSLAM algorithm is an application of a Rao-Blackwellized Particle Filter to the SLAM problem. This particle filter is used to estimate the combined posterior of the current robot pose \( \mathbf{x}_k \) and the positions of landmarks in the map \( \mathbf{m}_k \). This estimation is based on the current control input \( \mathbf{u}_k \) and measurement \( \mathbf{z}_k \). The combined posterior is given by:

\[
p(\mathbf{x}_k, \mathbf{m}_k|\mathbf{z}_k, \mathbf{u}_k)
\]  

(2-13)

The main principle behind RBPF-SLAM is to use Bayes rule to split the joint posterior into two separate probability distributions. The first depends on the trajectory of the robot \( \mathbf{x}_{0:k} \),
the second depends on the map. This application of Bayes rules is given by:

$$P(x_{0:k}, m_k | z_{0:k}, u_{0:k}) = P(m_k | x_{0:k}, z_{0:k}) P(x_{0:k} | z_{0:k}, u_{0:k})$$  \hspace{2cm} (2-14)$$

The posterior of the trajectory is estimated recursively by a particle filter. In FastSLAM the landmark positions can be estimated independently, this is made possible by taking the probability distribution of the entire robot trajectory as opposed to just of the latest pose. Therefore, the posterior of the map can be updated using $n_m$ independent EKFs. Since only the robot trajectory needs to be sampled for the particle filter, the complexity of the problem is greatly reduced.

**Particles** Each particle is denoted by $X^i_k$, each particle is associated with a number $i = 1 \ldots n_p$, where $n_p$ is the total number of particles. Every particle contains an estimated robot trajectory, denoted by $x_{0:k}$, and its covariance matrix: $P_{x,k}$. Also contained in each particle is a map, consisting of the position estimates of each landmark $m_n$ and the covariance matrix corresponding to that landmark: $P_{n,k}$. In total, each particle is given by:

$$X^i_k = \{ x^i_{0:k}, m^i_{1,k}, P^i_{1,k} \ldots m^i_{N_m,k}, P^i_{N_m,k} \} \hspace{2cm} (2-15)$$

### 2-6-2 FastSLAM algorithm, step by step

The FastSLAM algorithm consists of the following steps, which will be explained in more detail below.

- Prediction step
- Sample from proposal distribution
- Update landmark estimate
- Update particle weights
- Resampling step

**Prediction step**

The estimated robot pose and sensor pose are calculated, based on the the kinematic models of the robot, given by (2-8) and (2-9). For ease of notation, the motion model (2-8) will be represented by $f(x^i_{k-1}, u_k)$. The estimate of the current measurement is calculated according to the observation model. This observation model is denoted by $h(x^i_k, m^i_{n,k})$. The short notation of the prediction step is given by:

$$x^i_k = f(x^i_{k-1}, u_k)$$

$$z^i_k = h(x^i_k, m^i_{n,k}) \hspace{2cm} (2-16)$$
Sample from proposal distribution

The FastSLAM algorithms samples only the current pose \( \mathbf{x}_k \) from the proposal distribution, as opposed to the entire trajectory \( \mathbf{x}_{0:k} \) which is stored in each particle. FastSLAM 2.0 differs from regular FastSLAM in the fact that also the current measurement, \( \mathbf{z}_k \), is used for pose estimation, as opposed to just using the kinematic robot model.

For this algorithm, the proposal distribution is given by:

\[
x_i^k \sim p(x_k|x_{0:k-1}^i, u_0 : k, \mathbf{z}_{0:k}, n_{0:k}) = \eta^i \int p(x_k|m_{n,k}, x_k, n_k)p(m_{n,k}|x_{0:k-1}^i, \mathbf{z}_{0:k-1}, n_{0:k-1})dm_{n,k}
\]

(2-17)

Where \( \eta \) is a normalizing constant, which is the inverse of the probability of the measurement:

\[
\eta^i = p(\mathbf{z}_k|x_{0:k-1}^i, u_0 : k, \mathbf{z}_{0:k-1}, n_{0:k})^{-1}
\]

\( n_k \) is a function which represents the correspondence between the observed features and the landmarks in the map. The proposal distribution (2-17) consists of a product of the next state distribution, calculated according to (2-8), and the probability of the current measurement. Calculating this measurement probability is done by an integration over each landmark in the map.

In order to obtain a closed form solution, the observation model \( h(x_k, m_{n,k}) \) is linearized around \( \bar{x}_k = x_{k-1} \) and \( \bar{m}_k = m_{k-1} \). The linearized model is given by:

\[
h(x_k, m_{n,k}) \approx \bar{z}_k^i + H_m(m_{n,k} - \bar{m}_{n,k}) + H_x(x_k - \bar{x}_k^i)
\]

(2-18)

With:

\[
\begin{align*}
\bar{z}_k & = h(\bar{x}_k^i, \bar{m}_{n,k}) \\
\bar{x}_k^i & = f(\bar{x}_k^{i-1}, u_k) \\
H_m & = \nabla_{m_{n,k}} h(x_k, m_{n,k}) \\
H_x & = \nabla_{x_k} h(x_k, m_{n,k})
\end{align*}
\]

Using this approximation of the observation model, (2-17) can be approximated by a Gaussian with the following mean and covariance:

\[
\begin{align*}
x_i^k &= P_{xx,k}H_x(S_k^i)^{-1}(z_k - \bar{z}_k^i) + x_{k-1}^i \\
P_{xx,k}^i &= (H_x^T(S_k^i)^{-1}H_x + P_{xx,k}^{-1})^{-1}
\end{align*}
\]

(2-19)

With:

\[
S_k^i = R_k + H_mP_{n,k-1}H_m^T
\]

Updating the landmark estimate

The landmark estimate is updated according to standard EKF measurement update. The update is done according to:

\[
\begin{align*}
W_k^i &= P_{n,k-1}H_m^T(S_k^i)^{-1} \\
m_{n,k}^i &= m_{n,k-1}^i + W_k^i(z_k - \bar{z}_k^i) \\
P_{n,k}^i &= (I - W_k^iH_m)P_{n,k-1}^i
\end{align*}
\]

(2-20)

Note that in this case, the standard EKF update step is used, which uses a linearized version of the observation model. When the observation model is highly non-linear, the performance of the EKF will deteriorate. One way of circumventing this problem is by using an unscented Kalman filter [7].
Update particle weights

A weight is assigned to each particle. This weight is proportional to the probability of the current observation, according to the information in the particle. These weights are given by (2-21). Using the linearized version of the observation model, the expression can be approximated by a Gaussian with mean, $\hat{z}_k$, and covariance given by (2-22).

$$w_k^i \propto p(z_k^i|x_{0:k-1}^i, u_{0:k}^i, x_0, z_0, n_0)$$

$$P_k = H_k P_{xx,k} H_k^T + H_m P_{n,k-1} H_m + R_k$$

Resampling step

The resampling step is necessary to avoid the problem of particle degeneration. This problem occurs when one of the particle weights is close to one and all the others are close to zero.

Firstly, the number of effective particles is calculated according to:

$$N_{eff} = \frac{1}{\sum_{i=1}^{n_p} (w_k^i)^2}$$

If the number of effective particles is below a certain threshold, i.e. if $N_{eff} < N_{thr}$, the particles are resampled. The resampling consists of two steps:

1. Draw $n_p$ particle from the particle set, with probabilities proportional to the particle weight.
2. set each particle weight to $w_k^i = \frac{1}{n_p}$

2-6-3 Loop closing in FastSLAM

The standard FastSLAM algorithm proves to be suboptimal when it comes to loop closures. For a large part, the problem lies with the resampling step. During this resampling step, particles with a low weight are discarded and replaced by particles with a higher weight. With this step, some information is thrown out and the spread in possible trajectories becomes smaller. Figure 2-4 illustrates the effect of the size of the spread on the loop closing behavior. Both plots show possible estimated trajectories from 10 different particles. The right figure shows the ideal case: the spread of the trajectories is so large some of the particles close the loop correctly. The left figure shows the negative effect resampling can have: because the spread in trajectories is smaller, none of the particles contain a trajectory with a closed loop.

The current resampling step does not explicitly take the loop closure into account. The proposed ACO-SLAM algorithm aims to do just that.

The resampling step in the FastSLAM algorithm also causes another problem. In FastSLAM the landmarks are estimated independently from each other, this means information on the
cross-correlations between the landmarks is discarded. In FastSLAM this can be done, because it is assumed the trajectory is fully represented by the particle filter. However, because of particle depletion, the particle filter produces an over-confident pose estimate [28]. This means the assumption does not hold and therefore, the map can become inconsistent. The proposed ACO-SLAM therefore also aims to take the map consistency into account.

2-7 The ACO-SLAM algorithm

This section introduces a novel algorithm, ACO-SLAM, which is meant to improve the loop closure behavior of FastSLAM. Firstly, the general idea behind ACO-SLAM is introduced. Finally the ACO-SLAM is explained step by step.

2-7-1 General concept of ACO-SLAM

The main idea behind the new algorithm is to include the optimality of the robot trajectory and the consistency of the map into the resampling process. Therefore less particles will be discarded unnecessarily and a solution can be reached that is closer to a global optimum.

An Ant Colony Optimization (ACO) step is used to find the optimal trajectory. The measure of optimality consists of two weighed factors. The first factor is the map consistency, which is a measure of how well the current measurement fits the overall map. The second factor is the inverse of the length of the path between the last robot position and the first. This measure exploits the fact that when a loop is closed correctly, the path between the first and last position will become a lot shorter.

It is assumed the optimal trajectory is somewhere in between all the trajectory estimates of the particles. Each ant therefore creates its own path by combining the different estimated trajectories from the particles. A path is created by letting the ant move from one node to another. The nodes are the collection of all the estimated positions of the robot from \( k = 1 \) to \( k = k_{\text{end}} \) for all \( n_p \) particles. Each node corresponds to a certain position estimate at a...
certain time instant, therefore, the estimated landmarks that were last observed from that position are said to belong to that node. Every ant can therefore build a map by combining the landmarks that correspond to all the nodes in its path.

The map consistency is calculated by letting an ant create a path from the beginning of the robot trajectory to the end and then evaluating the resulting map. For the second part of the optimality measure, the ant searches for the shortest path from the end of the robot trajectory back to the beginning.

Figure 2-5 represents how an ant creates a new path from the different particle trajectories. In this example, \( n_p = 5 \) particles are used for \( k_{\text{end}} = 10 \) time steps. In the top plot, the five different trajectories from the five particles are shown. The nodes are created by taking every estimated robot position. The nodes are shown in the middle plot. The ant can now create its path by moving from one node to another. The possibly of selecting a certain node depends on the amount of pheromones on that node. The bottom plot shows a possible path of a single ant.

The optimality measure is used as a fitness function. Every ant creates its own path and every path is evaluated according to this fitness function. The pheromones are distributed over the nodes according to the fitness of the path.

### 2-7-2 ACO-SLAM algorithm, step by step

The ACO-SLAM algorithm consists of the following steps, which will be explained in more detail below.

- Initialization
- Moving ants, part 1
- Moving ants, part 2
- Building maps from ant trajectories
- Evaluating fitness
- Pheromone distribution
- Repeat until convergence is reached

#### Initialization

Firstly, a list of all nodes is created from the estimated robot trajectories. This is done by taking every estimated robot position from each particle at each time step. Each time step \( k \) from the robot trajectory therefore results in \( n_p \) nodes. A label is assigned to each node according to the time step \( k \) that is associated with it. Each node is initialized with the same amount of pheromone.

It is necessary to create a map based on the path of each ant, because the consistency of the resulting map is part of the fitness function. Each node is associated with one or more
Figure 2-5: Top: 5 particle trajectories, Middle: possible nodes, Bottom: One possible ant path
estimated landmark positions, therefore a look-up table is made that links each node to each corresponding landmark estimate. A map is created from an ant’s path by looking up the landmark positions that correspond to the traversed nodes.

As initial position, the ants are distributed randomly over the nodes that correspond to \( k = 1 \), i.e. to the first estimate position of each particle trajectory.

**Moving ants, part 1**

The ants create their own path by moving from node to node. The first part of their path contains as many steps as the particle trajectory contains time steps. For \( n_p \) particles, each time step \( k \) corresponds to \( n_p \) possible nodes. Every ant starts from the \( n_p \) nodes corresponding to time step \( k = 1 \) and moves to one of the nodes with a label \( k = 2 \). Then the ant moves to one of the \( n_p \) nodes corresponding to \( k = 3 \) and continues until it has reached one of the nodes corresponding to \( k = k_{\text{end}} \).

At each step, the ant can choose from \( n_p \) nodes to move to. The probability of selecting a certain node is proportional to the amount of pheromones on the node and the inverse of the distance to the node:

\[
P = \frac{\tau \eta}{\sum \tau \eta}
\]

In (2-24), \( \tau \) is the amount of pheromone on a node and \( \eta \) is the inverse of the distance from the current node to the next. The ants keep moving until they reach one of the \( n_p \) nodes corresponding to the last time step, i.e. when they reach one of the last positions of the particle trajectories.

**Moving ants, part 2**

After the first part of the ant path, the ants have traveled from the nodes corresponding to \( k = 1 \) to the nodes corresponding to \( k = k_{\text{end}} \). Next, the ants have to find the shortest path back to the nodes corresponding to \( k = 1 \). In this way, it can be evaluated if an ant’s path contains a correctly closed loop. If the path does contain a closed loop, the way back to \( k = 1 \) will be much shorter than the path from \( k = 1 \) to \( k = k_{\text{end}} \). In the first part, the ants were only allowed to move in one direction; only nodes were selected that corresponded to increasing time steps. In the second part, the ants are allowed to move in every direction.

To determine which node an ants moves to next, a number \( n_{\text{close}} \) of the nodes closest to the current position are selected as possible directions. The probability of selecting a certain node is again calculated with (2-24). So the ants move in random directions until they arrive at one of the nodes corresponding to \( k = 1 \). The length of this part of the path is a good measure of whether a loop is closed.

**Building maps from ant trajectories**

As each node corresponds to its own estimated landmark positions, many different maps can be built by combining the landmarks from the nodes. Each ant has made its own path by combining different nodes, this means each ant’s path corresponds to a different map.
As described earlier, a look-up table is made that links certain nodes to specific landmark position estimates. A map is created by looking up all the landmark positions that correspond to the nodes an ant has selected for its path.

**Evaluating fitness**

As explained earlier, the ACO tries to find the trajectory that optimizes the map consistency as well as the loop closure. The fitness function therefore consists of two weighted parts, $F_{\text{map}}$ and $F_{\text{path}}$. The first is a measure of the map consistency, which is a measure of how well the features in the latest observation fit in the overall map constructed up to that point. Therefore, the measured features are transformed from the sensor frame to the world frame. This is done based on the current pose estimate and the inverse of the observation model. This results in a local landmark map, denoted by $\mathbf{m}_f$. The map consistency is calculated by taking the Euclidean distance between the local landmark map and the map created by the ant, $\mathbf{m}_{\text{ant}}$. Since the fitness function should be higher when the map is more consistent, the inverse of the Euclidean distance is taken as the first part of the fitness function:

$$F_{\text{map}} = \frac{1}{\|\mathbf{m}_{\text{ant}} - \mathbf{m}_f\|} \quad (2-25)$$

The second part of the fitness function is a measure of the length of the ant path from $k = k_{\text{end}}$ and back to $k = 1$. This exploits the fact that traveling back through a closed loop results in a much shorter path. The path length is taken to be the number of steps in the second part of the ant path, denoted by $n_{\text{path}2}$. The fitness function should be higher when the path is shorter, so the inverse of the path length is taken as the second part of the fitness function:

$$F_{\text{path}} = \frac{1}{n_{\text{path}2}} \quad (2-26)$$

Since the loop is only closed if the second part of the ant path is shorter than the first part, a fitness of $F_{\text{path}} = 0$ is assigned to paths which are too long, i.e. if $n_{\text{path}2} \geq k$.

**Pheromone distribution**

Before the new pheromone is distributed, an evaporation rate is applied to the pheromone on each node, (2-27).

$$\tau_i = \rho \tau_{i-1} \quad (2-27)$$

Then, for each ant trajectory every node that ant passes will receive an amount of pheromone corresponding to the fitness ($F_{\text{ant}}$) of that ant. So for every ant and each node in that ant’s trajectory, (2-28) is used to update the pheromone levels.

$$\tau_{i+1} = \tau_i + \Delta \tau \ast \frac{F_{\text{ant}}}{\sum F_{\text{ant}}} \quad (2-28)$$
Repeat

All the previous steps are repeated until the ants converge to a single solution or when a maximum number of iterations is reached. The ants are initialized at the beginning of each iteration, the pheromones obviously are not.

Finally the nodes are grouped back together according to the particles from which they originated. The weight of a certain particle is updated according to the normalized sum of pheromones on all the nodes associated to that particle.

2-8 Summary

This chapter introduced the necessary concepts for SLAM. The reference frames were defined, as were the inputs and outputs to the algorithm. The mathematical models that are used for SLAM were also presented. FastSLAM is a solution to the SLAM problem, based on a Rao-Blackwellized particle filter. Bayes rule is used to split the joint posterior into two separate probability distributions, one depending on the trajectory of the robot, the other on the map. The resampling step in the FastSLAM algorithm causes particle depletion, which inhibits correct loop closure. Also, FastSLAM discards information on the relations between landmarks, which becomes a problem when particle depletion occurs. ACO-SLAM is a new algorithm that improves the resampling step of FastSLAM. The main idea behind this new algorithm is to include the optimality of the robot trajectory and the consistency of the map into the resampling process. Therefore less particles will be discarded unnecessarily and a solution can be reached that is closer to a global optimum.
Chapter 3

Simulated experiments with FastSLAM and ACO-SLAM

3-1 Introduction

This chapter discusses the simulated experiments done with the SLAM algorithms introduced in the previous chapter. Firstly, two different environments are presented in which both algorithms are tested. The simulations are run with different noise settings and a varying number of particles. All the different settings will be discussed in the Section 3-3. The performance of the algorithms is evaluated according to two performance measures. These performance measures are presented in the Section 3-4. The results of the simulations will be presented in the last two sections, Section 3-5 will cover the results obtained with the standard FastSLAM algorithm. Section 3-6 will cover the results obtained with the novel ACO-SLAM algorithm. Finally the main observations will be summarized.

3-2 Simulation environments

Both algorithms will be tested in two simulated environments. During the simulations no visual processing or feature extraction is done, instead, the features are assumed to be robustly extracted by the KLT tracking algorithm. This assumption is made to simplify the simulations and because feature extraction is not a part of the research itself. Zero mean Gaussian noise is added to the observations to simulate errors made in the feature extraction.

Both environments are represented by a set of points in the world frame. The first test environment is a simple circular environment, shown in Figure 3-1. The environment consists of two sets of landmarks, one in horizontal plane $z = 0$ and one in plane $z = 1$. The landmarks are represented by the asterisks. For clarity, the landmarks are connected by a thin line, these lines are not used in the algorithm. The robot can only measure the position of the dots with respect to the sensor frame. This environment is used as a first test bed to tune the algorithms and quickly check the influence of different noise settings.
Simulated experiments with FastSLAM and ACO-SLAM

The trajectory of the robot is represented by the thick black line. Since controlling the robot is not a part of the SLAM problem, the control inputs are generated beforehand, such that the simulated robot perfectly follows the trajectory. Since the odometry used to estimate the pose of the robot is not perfect, zero mean Gaussian noise is added to the odometry.

The second environment is a simplified model of the @Home Lab and is shown in Figure 3-2. The simulations done in this environment will serve as a basis for the real life experiments. Each landmark corresponds to a corner of the room or furniture. In Figure 3-2 the landmarks are represented by asterisks. For clarity, the edges between landmarks are shown here as well to visualize the shapes of the room and furniture. Again, the simulated robot can only measure position of the landmarks with respect to the sensor frame.

The trajectory of the robot is represented by the thick black line. Again, the simulated robot follows this trajectory perfectly, and noise is added to the odometry.
3-3 Parameters and settings

To test the performance of the algorithms, both algorithms will be tested with different noise settings and a varying number of particles. With these tests, three things are investigated. The first is the influence of the number of particles on the performance of the SLAM algorithm. The second thing that is investigated is the influence of odometry noise on the performance. The third is the influence of the observation noise on the performance.

It was proven in [27] that for a linear SLAM problem with Gaussian noise FastSLAM converges with only a single particle if all the landmarks are observed infinitely often. To test this, the first simulations will be performed with a single particle. These results will be compared to the results of the simulations with 10, 25, 50 and 100 particles. Clearly, running the simulation with a single particle will not be useful when testing the ACO-SLAM, since this algorithm improves the resampling step in the FastSLAM algorithm, which is not needed for a single particle. However, since the complexity of the calculations is linear in the number of particles, the simulations with a single particle will be a lot faster. This makes fine-tuning easier.

The new robot pose is estimated by using the odometry. In the simulations, the odometry is calculated by using the control input $u_k$ and the kinematic robot model. Noise is added to the odometry to represent the errors in the odometry. The odometry noise is chosen to be zero mean Gaussian noise. The standard deviation of the position change is in the interval $\sigma_{\text{pos}} = [5 \cdot 10^{-3}, 5 \cdot 10^{-4}]m$, the standard deviation of the orientation change is varied in the interval $\sigma_{\text{ornt}} = [5 \cdot 10^{-3}, 5 \cdot 10^{-4}]^\circ$. The values for the odometry noise are chosen for two reasons. Firstly, these values are in the same range as the standard values used in the Matlab toolbox by Sola [29], which is used as a framework for the implementation of FastSLAM. Secondly, these values turn out to yield similar results to the results presented in [27].

Observation noise is added to the simulated sensor data to simulate disturbances in the sensor signal and to simulate the errors made by the feature extraction. The standard deviation of the observation noise is chosen to be in the interval $\sigma_{\text{obs}} = [0.5, 5]$ pixels for the RGB image. The standard deviation of the corresponding depth information is taken the same as the standard deviation of the position error, i.e. it is in the interval $\sigma_{\text{depth}} = [5 \cdot 10^{-3}, 5 \cdot 10^{-4}]m$. These values are chosen in the same way the values for odometry noise are chosen. In [20] it was found that for planes at a distance of 2.0m the depth measurements by the kinect have a standard deviation of about 0.5cm. This leads to the conclusion that the values for the noise on the depth information are reasonable.
Table 3-1 shows the settings for all the simulations performed in both environments.

<table>
<thead>
<tr>
<th>$n_p$</th>
<th>Odometry noise (position)</th>
<th>Odometry noise (orientation)</th>
<th>Observation noise</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\sigma_{pos} = 1 \cdot 10^{-3}$</td>
<td>$\sigma_{ornt} = 1 \cdot 10^{-3}$</td>
<td>$\sigma_{obs} = 0.5$</td>
</tr>
<tr>
<td>1</td>
<td>$\sigma_{pos} = 1 \cdot 10^{-3}$</td>
<td>$\sigma_{ornt} = 1 \cdot 10^{-3}$</td>
<td>$\sigma_{obs} = 0.5$</td>
</tr>
<tr>
<td>10</td>
<td>$\sigma_{pos} = 1 \cdot 10^{-3}$</td>
<td>$\sigma_{ornt} = 1 \cdot 10^{-3}$</td>
<td>$\sigma_{obs} = 0.5$</td>
</tr>
<tr>
<td>25</td>
<td>$\sigma_{pos} = 1 \cdot 10^{-3}$</td>
<td>$\sigma_{ornt} = 1 \cdot 10^{-3}$</td>
<td>$\sigma_{obs} = 0.5$</td>
</tr>
<tr>
<td>50</td>
<td>$\sigma_{pos} = 1 \cdot 10^{-3}$</td>
<td>$\sigma_{ornt} = 1 \cdot 10^{-3}$</td>
<td>$\sigma_{obs} = 0.5$</td>
</tr>
<tr>
<td>100</td>
<td>$\sigma_{pos} = 1 \cdot 10^{-3}$</td>
<td>$\sigma_{ornt} = 1 \cdot 10^{-3}$</td>
<td>$\sigma_{obs} = 0.5$</td>
</tr>
<tr>
<td>25</td>
<td>$\sigma_{pos} = 5 \cdot 10^{-4}$</td>
<td>$\sigma_{ornt} = 5 \cdot 10^{-4}$</td>
<td>$\sigma_{obs} = 0.5$</td>
</tr>
<tr>
<td>25</td>
<td>$\sigma_{pos} = 1 \cdot 10^{-3}$</td>
<td>$\sigma_{ornt} = 1 \cdot 10^{-3}$</td>
<td>$\sigma_{obs} = 0.5$</td>
</tr>
<tr>
<td>25</td>
<td>$\sigma_{pos} = 5 \cdot 10^{-3}$</td>
<td>$\sigma_{ornt} = 5 \cdot 10^{-3}$</td>
<td>$\sigma_{obs} = 0.5$</td>
</tr>
<tr>
<td>1</td>
<td>$\sigma_{pos} = 5 \cdot 10^{-4}$</td>
<td>$\sigma_{ornt} = 5 \cdot 10^{-4}$</td>
<td>$\sigma_{obs} = 0.5$</td>
</tr>
<tr>
<td>1</td>
<td>$\sigma_{pos} = 1 \cdot 10^{-3}$</td>
<td>$\sigma_{ornt} = 1 \cdot 10^{-3}$</td>
<td>$\sigma_{obs} = 0.5$</td>
</tr>
<tr>
<td>1</td>
<td>$\sigma_{pos} = 5 \cdot 10^{-3}$</td>
<td>$\sigma_{ornt} = 5 \cdot 10^{-3}$</td>
<td>$\sigma_{obs} = 0.5$</td>
</tr>
</tbody>
</table>

Table 3-1: Settings for all the FastSLAM simulations

### 3-4 Performance measures

The performance measure for the resulting map is chosen to be the average error of the estimated landmark positions. The error at time $k$ is the average Euclidean distance between the positions of the estimated landmarks and the true landmarks. This is given by:

$$e_{k,\text{map}} = \frac{1}{n_m} \sum_{n=1}^{n_m} (\mathbf{m}_n - \hat{\mathbf{m}}_n)^T \cdot (\mathbf{m}_n - \hat{\mathbf{m}}_n)$$  \hspace{1cm} (3-1)

The true position of the $n$th landmark is denoted by $\mathbf{m}_n$. The position of the estimated landmark is denoted by $\hat{\mathbf{m}}_n$. The total number of landmarks in the map is given by $n_m$.

The performance measure for the estimated pose is divided into two parts; one describes the position error and one describes the orientation error. The error at time $k$ is the Euclidean distance between the estimated pose and the true pose of the robot. Both performance measures are given by:

$$e_{k,\text{position}} = \left( \left[ \begin{array}{c} (x_k - \hat{x}_k) \\ (y_k - \hat{y}_k) \\ (z_k - \hat{z}_k) \end{array} \right] \cdot \left[ \begin{array}{c} (x_k - \hat{x}_k) \\ (y_k - \hat{y}_k) \\ (z_k - \hat{z}_k) \end{array} \right] \right)^{\frac{1}{2}}$$

$$e_{k,\text{orientation}} = \left( \left[ \begin{array}{c} (\phi_k - \hat{\phi}_k) \\ (\theta_k - \hat{\theta}_k) \\ (\psi_k - \hat{\psi}_k) \end{array} \right] \cdot \left[ \begin{array}{c} (\phi_k - \hat{\phi}_k) \\ (\theta_k - \hat{\theta}_k) \\ (\psi_k - \hat{\psi}_k) \end{array} \right] \right)^{\frac{1}{2}}$$  \hspace{1cm} (3-2)
3-5 Simulations with FastSLAM

This section describes the simulation results obtained by the FastSLAM algorithm. First, the results for the simple environment are presented. Afterwards, the results for the @Home environment are discussed.

3-5-1 Simple environment results with FastSLAM

As an example, the map resulting from the simulation with 50 particles is shown in Figure 3-3.

![Figure 3-3: 3D map of the simple environment with 50 particles](image)

The development of the average map error and the development of the pose error are shown in the figures below:

![Figure 3-4: Map error for 50 particles](image)
Simulated experiments with FastSLAM and ACO-SLAM

![Graph showing pose error for 50 particles](image)

**Figure 3-5:** Pose error for 50 particles

It can be seen that after the first 50 iterations, the average error drops to a value between 0.06m and 0.10m. Sharp increases in the average map error are seen at iterations $k = 48$, $k = 66$, $k = 112$, $k = 123$ et cetera. At these time steps, new landmarks are added to the map. The initial estimate position of a new landmark is apparently quite far from the true position, causing a sharp increase in the average map error. As the new landmark is observed more often, its probability increases and its error decreases. As can be seen from the graph, within 5 to 10 iterations these error peaks converges back to a steady state value. The average map error slowly increases over time. This is because the average pose error also slowly increases over time. The loop is closed around $k = 450$, but the effect of the loop closure can not be seen in the error development of the map. This is because this plot shows the average mapping error of the entire map. The landmarks at the beginning of the loop will be observed for a second time, so their error decreases. However, all the other landmarks are not updated until the robot observes them again. This means that from $k = 450$ to $k = 500$ only the error of the first few landmarks decreases. This results in a very small difference in the average of the entire map.

The pose error steadily increases until $k = 450$. After which it starts to decrease. This decrease is caused by a correct loop closure. If the loop is closed correctly, the pose error will decrease, because older landmarks, with a high likelihood and a low error can be used to locate the robot.

**Influence of number of particles on the performance**

The influence of the number of particles on the performance is investigated. The number of particles is varied from 1 to 100 particles. The average is taken over 3 simulations. The odometry noise is a zero mean Gaussian with $\sigma_{pos} = 1 \cdot 10^{-3}$m for the position and $\sigma_{ornt} = 1 \cdot 10^{-3}$ $^\circ$ for the orientation. The observation error has a standard deviation of $\sigma_{obs} = 0.5$ pixels.
The distribution of the map errors from a single simulation is non-Gaussian. This is because the Euclidean distance is used as an error measure. This means many standard statistical methods, such as ANOVA, can not be used to draw conclusions on the effect of different parameters. The Wilcoxon rank sum test (also known as the Mann-Whitney U test) is a non-parametric test, that tests the null-hypothesis that two samples come from the same distribution. It is the non-parametric equivalent of a two-sample t-test. The big advantage of this test is that any distribution can be used, instead of only normal distributions. A disadvantage is that it only provides information on the similarity between two distributions. A rejection of the null hypothesis can therefore mean a significant difference in the mean, but it can also indicate a significant difference in the shape of the distribution.

Descriptive statistics is used to be able to make some statements on the performance of the algorithms under different settings. A boxplot is used to illustrate not only the average error, but also the spread of the errors. A boxplot is a way of illustrating descriptive statistical information of batches of numerical data. Five numbers are used to describe the information in the batch. The median of the data is given by a horizontal line in the middle. The upper and lower quartile are represented by another two horizontal lines above and below the median. The sample minimum and maximum are given by the outer two lines. Any outliers are represented by asterisks. An example of a boxplot is shown in Figure 3-6.

The notches are rough indicators of the significance of the difference of the medians. The width of a notch is calculated by:

$$\pm 1.5 \frac{IQR}{\sqrt{n_{\text{sample}}}}$$  \hspace{1cm} (3-3)

Here, $IQR$ stands for the interquartile range and $n_{\text{sample}}$ is the size of the sample. If the notches overlap it indicates there is no significant difference between the two medians. If the notches do not overlap, this offers an indication that there is a significant difference. It is
not possible to draw definite conclusions based on this information, but it does offer a first indication of the significance of the results.

Figure 3-7 shows the boxplots of the map error for different numbers of particles. Since the error only converges after 50 iterations, only the errors after the first 50 iterations are considered for the boxplots. Figure 3-8 shows the boxplots of the pose error for different numbers of particles.

**Figure 3-7:** Boxplots of the map errors for varying number of particles

![Boxplot of map errors](image)

**Figure 3-8:** Boxplots of the pose errors for varying number of particles

![Boxplot of pose errors](image)

The notches in the boxplots indicate that a significant difference between the medians of the
map errors is likely. The p-values, obtained by the Wilcoxon rank sum test, are all below the 0.05 significance level, which provides strong evidence the null hypothesis can be rejected. As the null hypothesis only states that two error distributions are the same, no conclusion can be drawn regarding the significance of the difference in medians. It does seem very likely there is a significant difference in the medians of the map error for different numbers of particles. In [27] it was proven that FastSLAM converges under the assumption that the SLAM problem is linear and Gaussian. The fact that a single particle leads to higher map errors, indicates non-linearities. These non-linearities are caused by the non-linearities in the 3D kinematic robot model.

The boxplots of the resulting position errors point towards a significant difference in the median error for a varying amount of particles. The dependency of the position error on the number of particles also points towards non-linearities in the SLAM problem. The boxplots of the orientation error are much more similar to each other. This makes sense, since the 3D kinematic robot model is linear in terms of orientation.

**Influence of odometry noise on the performance**

The influence of the odometry noise on the performance is also investigated. Three different noise settings are used to investigate this influence. The first situation has a standard deviation of $\sigma_{pos} = 5 \cdot 10^{-4}$m for the position noise and $\sigma_{ornt} = 5 \cdot 10^{-4}$° for the orientation noise. The second situation has noise with standard deviations of $\sigma_{pos} = 1 \cdot 10^{-3}$m and $\sigma_{ornt} = 1 \cdot 10^{-3}$°. The third situation has standard deviations of $\sigma_{pos} = 5 \cdot 10^{-3}$m and $\sigma_{ornt} = 5 \cdot 10^{-3}$°.

These simulations are performed with a 1 particle and with 25 particles. The resulting map and pose errors for 25 are shown in the boxplots below. The results of the simulations with 1 particle can be found in the appendix.

![Boxplots of the map errors for varying odometry noise and 25 particles](image)

**Figure 3-9:** Boxplots of the map errors for varying odometry noise and 25 particles
34 Simulated experiments with FastSLAM and ACO-SLAM

![Boxplots of the pose errors for varying odometry noise and 25 particles](image)

**Figure 3-10:** Boxplots of the pose errors for varying odometry noise and 25 particles

The results of the simulations with 1 particle are very similar to the results obtained with 25 particles. As the odometry noise is increased from \( \sigma_{\text{pos}} = 5 \cdot 10^{-4} \text{m} \) and \( \sigma_{\text{ornt}} = 5 \cdot 10^{-4} \text{o} \) to \( \sigma_{\text{pos}} = 5 \cdot 10^{-3} \text{m} \) and \( \sigma_{\text{ornt}} = 5 \cdot 10^{-3} \text{o} \), the median of the map error increases from 0.05m to values of 0.18m and 0.23m. The p-values are all below the 0.05 significance level which leads to the conclusion the distributions are different. The notches in the boxplots do not overlap, which is an indicator there is a significant difference between the medians. This results make sense since higher disturbances in the odometry will lead to worse estimated poses and thus to a decrease in the accuracy of the resulting map. The spread in the results is larger for a higher standard deviation of odometry noise, although there are far fewer outliers.

The median of the position error increases from 0.04m to values of 0.12m and 0.19m with an increasing odometry noise. The median of the orientation error increases from \( 2 \cdot 10^{-3} \text{o} \) to \( 10 \cdot 10^{-3} \text{o} \). The spread of the pose errors is higher for a higher standard deviation of the odometry noise. The boxplots suggest a significant difference between the medians. These results are in line with the previous observations on the influence of odometry noise on the average map error.

**Influence of observation noise on the performance**

The standard deviation of the observation noise is varied to test its influence on the performance. Three tests are performed. The first is done with an observation noise with \( \sigma_{\text{obs}} = 0.5 \) pixel. The second test is performed with a \( \sigma_{\text{obs}} = 1 \) pixel and the third with \( \sigma_{\text{obs}} = 5 \) pixels. All tests are performed with a single particle. The odometry noise during these tests is zero mean Gaussian noise with \( \sigma_{\text{pos}} = 1 \cdot 10^{-3} \text{m} \) and \( \sigma_{\text{obs}} = 1 \cdot 10^{-3} \text{o} \).
The results of the simulations for different standard deviations of observation noise are shown in the figures below:

![Boxplots of the map errors for varying observation noise](image1)

**Figure 3-11:** Boxplots of the map errors for varying observation noise

![Boxplots of the pose errors for varying observation noise](image2)

**Figure 3-12:** Boxplots of the pose errors for varying observation noise

From the boxplots in Figures 3-11 and 3-12 it can be seen that there is little difference in the map and pose errors for different settings of observation noise. The boxplots indicate that a significant difference is unlikely. This is an unexpected result since this implies that increasing the standard deviation of the observation noise from 0.5pixel to 5pixels has no influence on the average map or pose error. Also the spread of the errors is not influenced by increasing the observation noise. To investigate this, more tests are performed with an even higher observation noise. The standard deviation is increased to $\sigma_{obs} = 10$pixels and $\sigma_{obs} = 25$pixels.
The results of the experiments with the increased standard deviation of observation noise are shown below:

![Boxplots of the map errors for increasing observation noise](image1)

**Figure 3-13**: Boxplots of the map errors for increasing observation noise

![Boxplots of the pose errors for increasing observation noise](image2)

**Figure 3-14**: Boxplots of the pose errors for increasing observation noise

From these boxplots it can be seen that increasing the observation noise even further likely results in a significant increase in the map and pose errors. This result corresponds to the expectation. Apparently, the originally chosen standard deviations of observation noise are too close together to result in a significant difference.
3-5-2 @Home lab environment results with FastSLAM

Every simulation performed in the simple environment is repeated for the @Home environment. The results are shown and discussed in this section.

A plot of the evolution of the map error in time is shown in Figure 3-15. The pose errors of the same simulation are shown in Figure 3-16. This data is obtained by a simulation in the @Home environment, using 50 particles. The simulations are done with odometry noise having a standard deviation of $\sigma_{pos} = 1 \cdot 10^{-3} \text{m}$ and $\sigma_{ornt} = 1 \cdot 10^{-3} \text{o}$. The observation noise has a standard deviation of $\sigma_{obs} = 0.5 \text{ pixel}$.

![Figure 3-15: Map error for $n_p = 50$](image1)

![Figure 3-16: Pose error for $n_p = 50$](image2)
For the map error obtained in the @Home environment, three different sections can be distinguished. The first is from $k=1$ to $k=35$, the second is from $k=36$ to $k=275$. The third section is from $k=276$ to $k=470$. Each of these sections start with a sharp rise in map error. These sections correspond to the three separate rooms in the @Home environment. The landmarks in one room can not be observed from another room. Therefore each time the simulated robot enters a new room, many new landmarks are added to the map simultaneously, resulting in the sharp increase in map error.

A sharp increase in the position error can be seen at $k = 270$. This peak occurs at the same time step the average map error has a sharp increase. At that point, the landmarks that can be observed have a large error, so a large error in the estimated position results. The robot traverses the entire loop in 472 time steps. As can be seen in the plot, the pose error does not decrease after $k = 472$, indicating this simulation does not result in a loop closure.

**Influence of number of particles on the performance**

The number of particles is varied from 1 to 100 particles. The simulations are done with a odometry noise with $\sigma_{pos} = 1 \cdot 10^{-5} m$ and $\sigma_{ornt} = 1 \cdot 10^{-3} \circ$. The observation noise is set to have a standard deviation of $\sigma_{obs} = 0.5 \text{pixel}$.

Figure 3-17 represents the spread in map error for a varying number of particle. Figure 3-18 represents the spread of the pose errors for the same simulations.

![Figure 3-17: Boxplots of the map errors for varying number of particles](image)

Here, the difference between the medians of the errors is not significant. The p-values obtained when comparing the results between 25 and 50 and between 10 and 25 particles were above 0.7, indicating the samples are likely from the same distribution. This may seem like an unexpected result, considering the non-linearities in the kinematic model caused significantly different errors for the simple environment. However, the trajectory used in the @Home
environment is linear for a large part. Therefore little difference in the pose and map errors are to be expected for the @Home environment.

![Figure 3-18: Boxplots of the pose errors for varying number of particles](image)

**Influence of odometry noise on the performance**

The standard deviation of the odometry noise is in the interval $\sigma_{pos} = [5 \cdot 10^{-4}, 5 \cdot 10^{-3}]$ m and $\sigma_{ornt} = [5 \cdot 10^{-4}, 5 \cdot 10^{-3}]$ °. The results of the simulations with 25 particles are shown in the figures below. The results of the simulations with a single particle can be found in the appendix.

![Figure 3-19: Boxplots of the map errors for varying odometry noise and 25 particles](image)
Simulated experiments with FastSLAM and ACO-SLAM

The boxplots indicate a significant difference in the map error is very likely. These results are consistent with the results obtained by simulations in the simple environment. An increase in the odometry noise leads to an increase in the average map error. The resulting pose and map errors are very similar to the errors of the same simulations in the simple environment.

Influence of observation noise on the performance

The standard deviation of the observation noise is in the interval $\sigma_{obs} = [0.5, 5]$ pixels. The simulations are done with odometry noise with $\sigma_{pos} = 1 \cdot 10^{-3} \text{ m}$ and $\sigma_{ornt} = 1 \cdot 10^{-3}$°. These tests are done with a single particle. The results of the simulations are shown in the boxplots below:

Figure 3-20: Boxplots of the pose errors for varying odometry noise and 25 particles

Figure 3-21: Boxplots of the map errors for varying observation noise
As was the case for the simple environment, increasing the observation noise in the original interval does not have any influence on the median or the spread of the average mapping error. If the observation noise is increased even further, there will be a difference in the map error. Also the pose errors do not increase for an increasing standard deviation of observation noise.

### 3-6 Simulations with ACO-SLAM

This section describes the simulations done with the ACO-SLAM algorithm. Firstly, the settings for which the ACO algorithm converges are investigated. Then, simulations with ACO-SLAM are performed for both environments.

The tests with ACO-SLAM are all performed with the same number of particles. If this number is too low, the spread in possible trajectories will be too small for the ACO-SLAM to make any difference. If this number is chosen too high, the calculation time will increase greatly. Therefore a number of 25 particles is taken.

The influence of the odometry noise and observation noise on the performance of the ACO-SLAM algorithm are investigated. The same noise settings that were used for the FastSLAM simulations are used for the ACO-SLAM simulations.

#### 3-6-1 Convergence of the ACO

To test for which settings the ACO converges, the following test is performed. Firstly, a simulation is performed with FastSLAM in the simple environment for 50 time steps. The resampling step is omitted to avoid particle depletion and to ensure a large spread in possible trajectories. Normally, omitting the resampling step would lead to particles degeneration and therefore to a decreased performance. However, it is not the performance of the particle filter
that is of interest at this point, it is the convergence of the ACO. The data stored in the particles after these 50 time steps is used as the starting point for the ACO algorithm.

From this starting point, the ACO-update step is performed. The amount of pheromones on each node is stored as well as the weight that is added to each particle. The ACO-update step is performed 20 times with the same data and the same parameters. This gives an indication the ACO algorithm will converge to the same solution every time.

The bar plot in Figure 3-23 represents the normalized weights that are added to each particle. Each bar in this plot represents the added weight of one particle for a single test of the ACO-update step.

![Figure 3-23: Added particle weight for 20 tests](image)

These results are obtained with 20 ants. The evaporation rate is set to $\rho = 0.4$. The weight ratio between the map consistency measure and the loop closure measure is set to $1 : 0.005$. It can be seen that for these settings a large weight is added to particles 9 and 21 for each test. This means the ACO algorithm converges to the same solution every time.

To see after how many iterations the ACO algorithm converges, a plot is made of the two nodes with the highest amount of pheromones. The amount of pheromones is normalized according to the maximum amount of pheromones. This plot is given in Figure 3-24.
It can be seen that after 30 iterations the pheromone levels on both nodes have reached their steady state level. Some variation around this value occurs, due to the randomness in the ACO algorithm.

3-6-2 Simple environment results with ACO-SLAM

The simulations are performed with the parameters found in the previous section. All simulations will be performed with 25 particles. The standard deviation of the odometry noise is chosen to be in the interval \( \sigma_{\text{pos}} = [5 \cdot 10^{-4}, 5 \cdot 10^{-3}] \text{m} \) and \( \sigma_{\text{ornt}} = [5 \cdot 10^{-4}, 5 \cdot 10^{-3}] \text{o} \). The standard deviation of the observation noise is in the interval \( \sigma_{\text{obs}} = [0.5, 5] \text{pixels} \). Table 3-2 presents settings for all simulations performed with ACO-SLAM. The same tests will be repeated for each environment.

<table>
<thead>
<tr>
<th>No. of particles</th>
<th>Odometry noise (position) [m]</th>
<th>Odometry noise (orientation) [°]</th>
<th>Observation noise [px]</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>( \sigma_{\text{pos}} = 5 \cdot 10^{-4} )</td>
<td>( \sigma_{\text{ornt}} = 5 \cdot 10^{-4} )</td>
<td>( \sigma_{\text{obs}} = 0.5 )</td>
</tr>
<tr>
<td>25</td>
<td>( \sigma_{\text{pos}} = 1 \cdot 10^{-3} )</td>
<td>( \sigma_{\text{ornt}} = 1 \cdot 10^{-3} )</td>
<td>( \sigma_{\text{obs}} = 0.5 )</td>
</tr>
<tr>
<td>25</td>
<td>( \sigma_{\text{pos}} = 5 \cdot 10^{-3} )</td>
<td>( \sigma_{\text{ornt}} = 5 \cdot 10^{-3} )</td>
<td>( \sigma_{\text{obs}} = 0.5 )</td>
</tr>
<tr>
<td>25</td>
<td>( \sigma_{\text{pos}} = 1 \cdot 10^{-3} )</td>
<td>( \sigma_{\text{ornt}} = 1 \cdot 10^{-3} )</td>
<td>( \sigma_{\text{obs}} = 0.5 )</td>
</tr>
<tr>
<td>25</td>
<td>( \sigma_{\text{pos}} = 1 \cdot 10^{-3} )</td>
<td>( \sigma_{\text{ornt}} = 1 \cdot 10^{-3} )</td>
<td>( \sigma_{\text{obs}} = 5 )</td>
</tr>
</tbody>
</table>

Table 3-2: Settings for all the ACO-SLAM simulations
The development of the average map error and the pose errors for a simulation with ACO-SLAM are shown below:

**Figure 3-25:** Development of the average map error with ACO-SLAM and 25 particles

**Figure 3-26:** Pose error for ACO-SLAM

Compared to Figures 3-4 and 3-5, there is little difference in the performance of the ACO-SLAM algorithm. The position error is slightly lower, but no conclusions can be based on this. It can be seen that after $k = 470$ the pose error starts to decrease, which is caused by a correct loop closure.
Influence of the odometry noise on the performance

The ACO-SLAM algorithm is used in simulations with different odometry noise settings. The resulting map and pose errors are shown in the figures below:

![Boxplots of the map errors for varying odometry noise](image1)

**Figure 3-27:** Boxplots of the map errors for varying odometry noise

![Boxplots of the pose errors for varying odometry noise](image2)

**Figure 3-28:** Boxplots of the pose errors for varying odometry noise

The Boxplots indicate there is no significant difference in the performance of ACO-SLAM and FastSLAM for the lowest two standard deviations of odometry noise. The medians of the map errors for both algorithms are 0.05m and 0.08m. The medians of the pose error are slightly lower, but without a clear significance. For the odometry noise with a standard deviation of $5 \cdot 10^{-3}$m, ACO-SLAM results in a larger map and pose error. A significant difference is very likely.
Influence of the observation noise on the performance

The standard deviation of observation noise is varied from $\sigma_{obs} = 0.5\text{pixel}$ to $\sigma_{obs} = 5\text{pixels}$. The resulting map errors are shown in the figures below:

![Boxplots of the map errors for varying observation noise](image1)

*Figure 3-29: Boxplots of the map errors for varying observation noise*

![Boxplots of the pose errors for varying observation noise](image2)

*Figure 3-30: Boxplots of the pose errors for varying observation noise*

The medians of the map error are quite close for both algorithms. For an observation noise of 1 and 5pixels, the medians are in between 0.12m and 0.14m for both the ACO-SLAM and the FastSLAM. Neither the Wilcoxon rank sum test, nor the notched boxplots suggest evidence for a significant difference. For an observation noise of 0.5 pixel, the ACO-SLAM does result in lower map and pose errors. The boxplot indicate a significant difference is likely.
3-6 Simulations with ACO-SLAM

3-6-3 @Home lab environment results with ACO-SLAM

The simulations that were described in the previous section are repeated in the @Home environment.

Influence of the odometry noise on the performance

The standard deviations of the odometry noise are $\sigma_{\text{pos}} = [5 \cdot 10^{-4}, 5 \cdot 10^{-3}]$ m and $\sigma_{\text{ornt}} = [5 \cdot 10^{-4}, 5 \cdot 10^{-3}]$°. The resulting map and pose errors are shown below:

![Boxplots of the map errors for varying odometry noise](image)

**Figure 3-31:** Boxplots of the map errors for varying odometry noise

![Boxplots of the map errors for varying odometry noise](image)

**Figure 3-32:** Boxplots of the map errors for varying odometry noise
For the lowest two standard deviations of odometry noise, the medians of the map error are close to those obtained by the FastSLAM algorithm. There is little indication to assume significantly lower map errors for ACO-SLAM. The spread is smaller for the simulations done with the ACO-SLAM algorithm. For odometry noise with $\sigma_{\text{pos}} = 5 \cdot 10^{-3}$m and $\sigma_{\text{ornt}} = 5 \cdot 10^{-3}^{\circ}$, a significantly higher map error seems very likely. The median obtained with the ACO-SLAM is 0.36m, whereas the median obtained with FastSLAM is 0.27m. The pose errors obtained with ACO-SLAM are quite similar to the simulations done with FastSLAM. The medians of both the pose and the orientation errors are close to the medians obtained with FastSLAM.

**Influence of the observation noise on the performance**

The standard deviation of observation noise is varied from $\sigma_{\text{obs}} = 0.5$pixel to $\sigma_{\text{obs}} = 5$pixels. The resulting map errors are shown in the figures below:

![Boxplots of the map errors for varying observation noise](image)

**Figure 3-33:** Boxplots of the map errors for varying observation noise
It can clearly be seen that for an observation noise with a standard deviation of 5 pixels, the ACO-SLAM leads to higher map errors than the FastSLAM algorithm. The median of the map error for that observation noise is 0.36 m for ACO-SLAM and 0.12 m for FastSLAM. This is likely a significant difference. For lower standard deviations in observation noise, the performance of ACO-SLAM is comparable to that of FastSLAM. The resulting pose errors are similar to the pose errors obtained by the FastSLAM algorithm. The medians of the orientation error are slightly smaller for ACO-SLAM.

### 3-7 Summary

This chapter presented the simulations performed with both algorithms. The simulations are done in two different environments, one simple cyclic environment and an environment modeled on the @Home lab. Boxplots are used to represent the data. The notches in the boxplot and the p-values obtained by the Wilcoxon rank sum test are used as an indication of the significance of the results. However, no definite conclusions can be drawn based on these indicators.

The map and pose errors increase as the odometry noise increases. For the chosen interval of observation noise, there is no significant difference in map or pose errors. When the observation noise is increased further, the pose and map errors become larger. The nonlinearities in the kinematic robot model cause an influence of the number of particles on the map and pose errors. In the @Home environment, the robot trajectory is nearly linear, so this effect is not observed in the simulations for this environment.

For a high odometry noise, ACO-SLAM performs worse than the FastSLAM algorithm. For lower values of odometry noise, the results are comparable. For a high observation noise, ACO-SLAM leads to higher pose and map errors. For a low value of observation noise, ACO-SLAM leads to lower or comparable errors.
Chapter 4

Experiments with FastSLAM and ACO-SLAM

4-1 Introduction

This chapter presents the experiments done with service robot Robby. The experiments are performed in the @Home lab with both the FastSLAM and the ACO-SLAM algorithms. A model of the same environment was used in the simulations, so the results can be compared. The experimental setup is described in Section 4-2. The software that is used to implement the algorithms is presented in Section 4-3. The results of the FastSLAM and ACO-SLAM algorithms are presented in Sections 4-4 and 4-5 respectively.

4-2 Experiment setup

The robot motion is controlled with a joystick, which allows the user to send the desired velocity and rotational velocity to the robot base. The choice to control the robot manually is made for several reasons: Firstly, trajectory control is not a part of the SLAM problem and therefore not of this research. Secondly, it is not possible to automatically find a trajectory through the environment, since the environment is unknown at the beginning of the SLAM algorithm. Also, the implementation is straightforward and it closely resembles the way the algorithms will be used in real situations.

The robot moves through the entire lab before returning to its original position. In order to compare the actual robot path to the estimated robot path, the ground truth of the robot trajectory needs to be recorded. This is done by fixing a marker pen to the base of the robot. As the robot moves around the laboratory, the trajectory of the robot is marked on the floor. Afterwards, this trajectory is measured by hand. The figure below shows the measured robot path in the 2D map of the @Home lab.

Master of Science Thesis B.A. Kupers
To calculate the position error, the estimated 2D position of the marker tip is calculated from the estimated 6D pose. This is done by using a constant transformation matrix. The estimated 2D trajectory of the robot is then compared to the measured path, marked on the floor. The trajectory can be measured with an accuracy of ±5mm. The average position error found in the simulations is between 50mm and 150mm, so the trajectory can be measured accurately enough.

There are several advantages to this method of recording the ground truth: the implementation is straightforward, the costs are low and it can be done with the required accuracy. A disadvantage of this method is that the orientation of the robot cannot be measured directly. However, since the estimated 2D position of the marker is calculated from the full 6D pose, the accuracy of the orientation is incorporated into the overall 2D position error. Recording the ground truth with an overhead camera would have been another option, but since the robot moves through separate rooms in the lab, a lot of cameras would have to be used. Also, obtaining the pose of the robot from visual data is far from trivial. Another option could have been to put the robot on a set of rails. This would, however, be much more expensive and it would restrict the movement of the robot, so the experiments would be a less realistic representation of later use in real life.

The ground truth of the map is obtained by measuring the positions of the walls and large pieces of furniture. The mapping error is calculated by taking the Euclidean distance between the estimated landmark positions and the measured walls and objects. Landmarks that correspond to smaller objects are discarded.

A recording is made of the data measured by the Kinect in order to allow the algorithms to be tested and tuned offline.

4-3 Software for the experiments

The robot runs on the Robot Operating System (ROS). ROS is a software framework for robotic software development, developed by the Stanford Artificial Intelligence Laboratory
in 2007 [30] in support of the Stanford AI Robot STAIR. Nowadays, ROS is an open source platform where software developers and researchers can exchange software packages, which provide the available functionalities. The necessary packages to control the motion of the robot and to do simple 2D SLAM are already implemented. However, no package exists that implements FastSLAM in 3D. This package is created especially for this research.

### 4-3-1 Implementation of FastSLAM

The FastSLAM package uses the Mobile Robot Programming Toolbox (MRPT). This toolbox is developed by the University of Malaga and consists of a set of libraries in C++. Libraries for many different SLAM algorithms are available, among which ICP-SLAM [10], EKF-SLAM [6] and several SLAM algorithms based on particle filters. However, the MRPT does not provide a library that implements FastSLAM, so an extension is made to the existing libraries to implement FastSLAM.

The FastSLAM algorithm is implemented as described in chapter 2. The Kinect generates a 2D RGB image and the corresponding depth image. A KLT [21] tracker is used to extract and track features from this data. The odometry that is used for the prediction step in FastSLAM is calculated by visual odometry as described in 2-5. The tracked features are used as the measurement input $z_k$.

The Kinect data is recorded with a frequency of 7.5Hz. If the particles would be updated every time a new measurement is available, the computational effort would become too large. Therefore, the particle filter is only updated if the change in robot pose is above a certain threshold. Updating the particle filter less often leads to a decrease in the accuracy of the pose and landmark estimates, because less measurements are incorporated. This introduces a trade-off between computational effort and accuracy. The threshold that gives reasonable results in terms of both computational speed and accuracy is found to be 0.10m or $10^\circ$.

**Correspondences between features and landmarks**

To compute the error signal, the distance between a measured feature and the corresponding predicted feature needs to be calculated. The predicted features are found by projecting the landmarks from the world frame into the sensor frame, this is done using the observation model, described in 2-4-3. The correspondences between the measured features $z_k$ and the predicted features $\hat{z}_k$ need to be found. If a feature is tracked from one frame to the other, the correspondence is known. However, if a feature was not observed in the previous measurement, a correspondence can still exist. The algorithm searches for these correspondences by calculating the Mahalanobis distance between the measured unmatched feature and all the predicted features. A feature is said to correspond to a landmark if the Mahalanobis distance is below a certain threshold. If a feature has no correspondence it is a possible new landmark and it is added to the map. Only if a landmark is observed a minimum number of $n_{\min} = 5$ times it is used to update the pose of the robot.
4-3-2 Implementation of ACO-SLAM

The ACO-SLAM algorithm is used before the resampling step of the particle filter algorithm. It is used to prevent discarding particles that may lead to loop closure. The ACO algorithm is used to find particles that are more likely to lead to loop closure and a consistent map. Extra weight is added according to how well a particle performs in the ACO algorithm.

The ACO algorithm is implemented as described in chapter 2. For the first step of the algorithm, a network of nodes is created. For a single particle, a node is created for each robot position from time step 1 to \( k \), this means each particle results in \( k \) new nodes. The entire network of nodes consists of all the nodes taken from all the particles. To save computation time in the ACO algorithm, each node maintains a list of possible nodes that an ant can reach in the next iteration. Each node also maintains a local map, consisting of the landmarks that were last updated from its position. Each ants combines these local maps into a single map. The consistency of this map with the current measurement is a measure of the performance of the ant.

After the ACO algorithm has converged, the weight that is added to the particles is calculated. The pheromones on the set of nodes that originate from the same particle are added and normalized. This normalized total pheromone per particle is used as the added weight.

4-4 Experiments with FastSLAM

The recorded dataset is used offline with the newly implemented FastSLAM algorithm. For this dataset, a map consists of around 800 landmarks. Because the algorithm that checks for correspondences is quite computationally expensive, the algorithm is too slow to be used on this dataset in real time. Therefore, the recorded data is played back with a frequency of 3Hz. The data was recorded with 7.5Hz, so the offline experiments are performed at half the real speed.

The position error is calculated by taking the smallest distance between each estimated position and the measured trajectory. As said before, a constant transformation matrix is used to calculate the estimated 2D trajectory from the estimated 6D poses. The resulting estimated trajectory is shifted and rotated with respect to the measured trajectory. This has to be done because the algorithm takes the initial pose of the robot as the origin of the world frame, whereas the marked trajectory is measured with respect to a different origin. The transformation between the two frames is calculated afterwards, this is done by the Matlab optimization function fminsearch.

For the calculation of the mapping error, only the landmarks that correspond to a wall or the other large objects in the room are considered. The landmarks that correspond to a wall or large objects are selected afterwards by hand. Only the landmarks that certainly belong to a wall or large object are selected. This reduces the amount of landmarks to be considered for the error from around 800 to around 200. The smallest distance between the selected landmarks and the measured objects is taken as the landmark error.

B.A. Kupers Master of Science Thesis
4-4-1 Experiment results with FastSLAM

The simulations showed that if the robot trajectory is close to linear, the number of particles does not have an influence on the performance of FastSLAM. The experiments with FastSLAM are performed with 25 and 50 particles. If the trajectory is close to linear in the experiments, there will be little difference in the performance here as well.

An example of the estimated path with 25 particles is shown below. The dotted line represents the true trajectory and the solid line represents the estimated trajectory.

![Estimated trajectory with 25 particles](image)

**Figure 4-2:** Estimated trajectory with 25 particles

It can be seen that initially, the estimate is quite close to the measured data. After the second corner, the accuracy deteriorates and the estimated trajectory diverges from the true path. This is because the visual odometry becomes less accurate in the second corner. This happens because at this point in the trajectory, the orientation of the robot changes quickly and there are relatively few tracked features available for visual odometry.

The position error is given in the figure below. Here the increasing position error is clearly visible as well.
The resulting map is shown in the figure below.

Here it can also be seen that the landmarks in the bottom-right corner are quite far away from the true position.

An example of the estimated path with 50 particles is shown below.
It can be seen that in this example the estimated trajectory is closer to the ground truth. However, the position error at the end of the loop is still too large to automatically close the loop. The same result is reflected in the plot of the position error, shown below.

The resulting map is shown in the figure below.
The same experiments are carried out 3 times. The average mapping errors and position errors are shown in the boxplots below.
The experiments with 50 particles lead to lower position errors. The median of the position error with 25 particles is 0.40m with a maximum error of 2.05m. For 50 particles, the median is 0.26m and the maximum error is 0.91m. The p-values obtained by the Wilcoxon rank sum test are below the significance level of 0.05, which indicates that the two distributions are different. The notches in the boxplots also suggest there is a significantly lower error for the experiments with 50 particles. As was observed in chapter 3, the number of particles influences the pose and mapping error if the trajectory is non-linear. This is obviously the case for the experiments, so this result is to be expected. The errors are comparable to the simulation results with 25 particles and odometry noise with a standard deviation of 0.005m. The position error decreases with an increasing number of particles.

There is very little difference between the median of the map error for both cases. The experiments carried out with 50 particles do, however, result in fewer outliers and a lower maximum error. Compared to the simulations, the experiments lead to larger mapping errors. 

Figure 4-9: Boxplot of the landmark position errors for 25 and 50 particles
4-5 Experiments with ACO-SLAM

The same dataset is used offline to perform the experiments with ACO-SLAM. The ACO-update step is quite computationally expensive, so for this dataset the data has to be played back with a frequency of 0.75Hz.

The map error and the position error are calculated in the same way as the errors of the FastSLAM algorithm. The experiments are carried out with 25 particles, which is the same amount that was used in the simulations of ACO-SLAM.

4-5-1 Convergence of the ACO algorithm

Before the experiments can be carried out, the ACO algorithm has to be fine-tuned to ensure it converges. To test the convergence the following test is performed: First, the FastSLAM algorithm is used, but without the resampling step. This ensures a large spread in the particles. Then, the ACO update step is performed. After each iteration of the ACO algorithm, the normalized added weight per particle is calculated. The plot below gives the normalized added weight for each of the 25 particles.

![Normalized pheromone per particle](image)

Figure 4-10: Normalized pheromone per particle

It can be seen that the algorithm converges after 30 iterations. The ACO-algorithm converged after the same amount of iterations for the simulations, which is an indication that the simulations show a realistic performance.

4-5-2 Experiment results with ACO-SLAM

An example of the estimated path with ACO-SLAM is shown below. The dotted line represents the true trajectory and the solid line represents the estimated trajectory.
The estimated trajectory is closer to the measured trajectory than for FastSLAM. However, as can be seen, the loop is still not closed. The position error at the end of the trajectory is 0.6m, which is too large for the ACO-algorithm to close the loop.

The position error is given in the figure below. Here it can be seen that ACO-SLAM results in a lower position error.

The resulting map is shown in the figure below.
Here it can be seen that the estimated landmarks are closer to the measured data than with FastSLAM.

The same experiments are carried out 3 times. The average mapping errors and position errors are shown in the boxplots below. For comparison, also the results for FastSLAM with the same number of particles are shown.

**Figure 4-14:** Boxplot of the robot position errors for ACO-SLAM and FastSLAM
From Figure 4-14 it can be seen that the ACO-SLAM algorithm leads to lower pose errors than the FastSLAM algorithm with the same number of particles. The boxplots suggest the difference is significant. The median pose error is 0.26m for the ACO-SLAM, whereas the pose error for FastSLAM is 0.40m. The maximum error is also lower for ACO-SLAM. The resulting pose errors ACO-SLAM with 25 particles are in fact comparable to the pose errors for FastSLAM with 50 particles.

Also the mapping error is lower for ACO-SLAM than for FastSLAM. This can be seen in Figure 4-15. The median of the landmark position error 0.23m for ACO-SLAM and 0.37m for FastSLAM. Also the maximum landmark position error is lower for ACO-SLAM than it is for FastSLAM.

4-6 Summary

This chapter presented the experiments conducted with the robot in the @Home lab. The robot is controlled by hand through the @Home lab and its trajectory is marked on the floor by a marker. This path has to be measured by hand, just like the positions of walls and large objects, this can be done with the required accuracy. The disadvantage of this method is that no direct measurements of the orientation of the robot are possible.

Master of Science Thesis

B.A. Kupers
The robot runs on the Robot Operating System (ROS). A package that implements FastSLAM in 3D with a Kinect is made specially for this research. Also the ACO-SLAM algorithm has been implemented. Both algorithms are too slow to be implemented in real time, so the experiments are done offline.

The errors found in the experiments are comparable to the errors found in the simulations. The experiments with FastSLAM show that the pose and map errors obtained with 50 particles are lower than the errors obtained with 25 particles. The ACO-SLAM algorithm results in lower position and landmark error than the FastSLAM algorithm with the same number of particles.
Chapter 5

Discussion and conclusions

5-1 Introduction

This chapter discusses the results of both the simulated experiments and the real experiments. The first section discusses the simulation results presented in Chapter 3. The second section discusses the results of the experiments, presented in Chapter 4. The main observations made in the previous chapter are repeated and conclusions are drawn based on these observations. Section 5-4 presents the conclusions and makes suggestions for future research. This chapter ends off with some closing remarks.

5-2 Discussion on the simulation results

The simulations with the FastSLAM algorithm result in very similar pose and map errors for the two different environments. Because the @Home environment seems more complex, one might expect the FastSLAM algorithm to perform worse for this environment. However, there is only a small difference between the two environments in terms of size and number of landmarks. This explains why the error for the @Home environment is not significantly larger than the error for the simple environment. Furthermore, in the current implementation of the FastSLAM algorithm a maximum amount of 10 landmarks is used to update the particle filter. Therefore, the particle filter is updated with the same amount of information for both environments. This further explains why the @Home environment does not result in higher pose and map errors than the simple environment.

Varying the amount of particles used for the simulations results in a significant difference in position and map error for the simulations in the simple environment. The capability of the particle filter to handle non-linearities in the kinematic model is increased by increasing the number of particles, therefore the pose and map error decrease when the number of particles is increased from $n_p = 1$ to $n_p = 50$. Unexpectedly, increasing the number of particles from $n_p = 50$ to $n_p = 100$ leads to a higher pose and map error. This is most likely caused by the
fact that the resampling step is not performed every time step, but every 10 time steps. A similar result is found in [31], in this work the error also increases when the amount of particles is increased above a certain value. No significant difference was found in the orientation error for a varying number of particles. This is because the kinematic model is linear in terms of orientation, therefore a single particle would be sufficient to estimate the orientation. In [27] it was proven that in the linear case with Gaussian noise, FastSLAM converges with a single particle.

In the @Home environment, the amount of particles does not influence the pose and map errors. This is because the robot trajectory in the simulated @Home environment is mostly linear. As was the case for the orientation error, FastSLAM only needs a single particle to converge for the linear Gaussian case.

Increasing the odometry noise results in higher map and pose errors. This result is to be expected since a higher odometry noise leads to a higher prediction error with a higher variance. If both the prediction error and its variance are higher, and if the same measurements are used, the update step will result in a higher pose and map error. The distribution of the errors is not Gaussian, since the Euclidean distance is used as an error measure. Therefore, the usual methods of statistical inference (like ANOVA) can not be used. The Wilcoxon rank sum test is used to test the null hypothesis that the medians of the pose and map error are the same for different noise settings. A very low likelihood was found to support that hypothesis, so it seems likely the distributions are different. Also the notches in the Boxplots indicate the medians are significantly different. No definite conclusions can be drawn based on this information, but a significant difference seems very likely and logical.

It is expected that increasing the observation noise results in higher map and pose errors. However, for the first experiments this is not the case. Therefore, more tests are performed with even higher standard deviations of observation noise. These tests turned out that increasing the observation noise even further does lead to an increase in error. The initially chosen range of observation noise was apparently too small to yield significantly different errors.

ACO-SLAM leads to smaller or comparable pose and map errors for a low standard deviation of observation noise. For observation noise with a higher standard deviation, ACO-SLAM leads to a larger error than FastSLAM. Before the first loop is completed, the ACO-SLAM algorithm will not be able to find a loop closure, so only the measure of map consistency is taken into account. This measure depends on the estimated map and the current measurement, therefore it introduces a sensitivity to high observation noise. This is reflected in the higher pose and mapping error. For a lower observation noise, the fact that ACO-SLAM takes the map consistency into account leads to improved performance. FastSLAM updates all the landmarks independently from each other, so information on the cross-correlations between landmarks is not taken into account in FastSLAM. Since ACO-SLAM uses the consistency of the map with the current measurement, it does take the relations between landmarks into account. This means that for a low observation noise, ACO-SLAM is able to improve the map and pose estimate. As was the case for simulations with FastSLAM, the error distribution is non-Gaussian, so normal statistical techniques cannot be used to base conclusions on. The notches in the boxplots do indicate that a significant difference is likely.

For a high standard deviation of odometry noise, ACO-SLAM leads to significantly higher map and pose errors than FastSLAM. This is because the current pose estimate is used to
project the measured features from the sensor frame into the world frame. As seen in the simulations with FastSLAM, a high odometry noise leads to a higher error in the estimated pose, therefore the measure of map consistency is less accurate and the performance of ACO-SLAM deteriorates. For a low standard deviation of odometry noise, the performance of ACO-SLAM is slightly lower than or comparable to FastSLAM.

5-3 Discussion on the experiment results

The experiments are conducted in a realistic environment. This means the results obtained in the experiments are a good indicator of the applicability of the algorithms in a real life situation.

The errors found in the experiments are comparable to those found in the simulations. The pose errors from the experiments are close the errors obtained in the simulations. The map errors from the experiments are not more than two to three times as high as the map errors from the simulations. This indicates the simulations were performed with realistic parameters. The results also correspond to the results found in literature, which indicates the implementation of FastSLAM in 3D with the Kinect as a sensor works well.

As opposed to what was found in the simulations for the @Home environment, a higher amount of particles does lead to a lower median of the position error. The real trajectory is much less linear than the simulated trajectory and a particle filter with more particles is more capable of handling the non-linearities. The notches of the boxplots and the Wilcoxon rank sum test indicate a significant difference is likely. However, as was the case for the simulations, it is not possible to draw definite conclusions based on this information.

The ACO-SLAM algorithm leads to lower position and map errors than FastSLAM. The notches in the boxplots indicate a significant difference is likely. The improved performance is caused by the fact that ACO-SLAM optimizes a measure of the map consistency. Therefore, information on the relations between landmarks is not thrown away, as is the case with FastSLAM, but it is used in the resampling step. From the results of the simulations it became clear that if either the observation noise or the odometry noise becomes too high, the ACO-SLAM will lead to higher pose and map errors. Clearly, in the experiments this is not the case. The position error at the end of the loop is still too large for the algorithm find a loop closure. To enable ACO-SLAM to find loop closure in this environment, the position needs to be estimated more accurately.

Currently the computation time is too long for the current algorithms to be implemented in real time for an environment of this size. There are currently two parts in the algorithms that are responsible for a large part of the computational effort. The first is the algorithm that checks for correspondences between features and the landmarks in the map. The second is the creation of the network of nodes for the ACO. In the current implementation, this network is created every time the ACO-update step is performed. This is done because the network of nodes changes every time the particles are resampled. Some particles are discarded during the resampling step and replaced with other particles. This means the nodes corresponding to these particles are also discarded and replaced by other nodes.
5-4 Main conclusions and future research

The two previous sections discussed the results obtained in the simulations and the experiments. This section will present the conclusions based on this discussion. Also suggestions for future research are made.

As was discussed earlier on, because the Euclidean distance is used as the error, the distribution of the pose and map errors are non-Gaussian. Therefore descriptive statistics and the Wilcoxon rank sum test are used to analyse the data. These methods can offer strong indications of significance, but no definite conclusions can be based on this.

If the trajectory of the robot is non-linear, a higher amount of particles results in a lower map and pose error for the FastSLAM algorithm. If the trajectory of the robot is close to linear, there is no significant difference in the pose and map errors for a varying amount of particles. These conclusions are based on the simulations and the experiments. It was proven in [27] that in the linear Gaussian case FastSLAM converges with a single particle, however from the experiments it became clear that this is not a realistic assumption for this set-up.

For FastSLAM, both the pose and the map errors increase when the standard deviation of the odometry noise is increased. A higher standard deviation in the odometry noise leads to a higher prediction error with a higher variance, this in turn leads to the observed increase in pose and map errors.

For FastSLAM, the errors increase with an increasing amount of observation noise. This effect is however not observable if the interval in which the standard deviation of the noise is varied is too small.

ACO-SLAM can lead to lower pose and map errors if the standard deviations of the observation noise and the odometry noise are relatively low. This increased performance is mainly caused by the use of the measure of map consistency, which incorporates relations between the landmarks into the resampling algorithm. These relations are discarded in the normal FastSLAM algorithm. If the odometry noise becomes too high, ACO-SLAM performs worse than FastSLAM, because the pose estimate is used to calculate the map consistency. In this way, the increase in pose error, caused by the higher odometry noise, is also incorporated into the map consistency. If the observation noise becomes too high, ACO-SLAM also performs worse than FastSLAM, because the measurement is used directly to compute the consistency of the map. From the experiments it can be concluded that in a realistic environment with real noise, ACO-SLAM performs better than FastSLAM in terms of pose and map error. However, in the experiments the resulting pose error at the end of the loop is still too large for automatic loop closure.

Future experiments are needed to be able to make statements about the significance of the results. These experiments should not take the Euclidean distance as the error, but they rather should calculate the error separately for each direction. The orientation of the robot is not recorded in the current set-up, however, this may be necessary for future experiments that do not take the Euclidean distance as the error. Since the experiments take place in an indoor environment a compass cannot be used to record this angle. A way of calculating the yaw angle of a robot is presented in [32]. This method uses patterns on the ceiling to calculate the heading angle of the robot. This algorithm only achieves an accuracy of 4.5°, which is not precise enough for this application. If the accuracy of the algorithm can be improved, this
might be a good method to record the yaw angle of the robot. Since in [32] only off-the-shelf smart phones were used as sensors, a higher accuracy can likely be achieved by using sensors with a higher precision.

The pose errors from the experiments are too large to automatically close the loop. A higher accuracy can be obtained by updating the particle filter more often. For example after every 0.05m or 5°. Also, increasing the amount of particles will make finding a correct loop closure more likely.

In the current implementation both the FastSLAM algorithm and the ACO-SLAM algorithm are too slow to be used in real time in the chosen environment. The suggested improvements to increase the accuracy of the estimated robot pose and landmark positions would further increase the computational time. Some techniques can be used to reduce the computation time, making real time implementation more feasible. The hierarchical SLAM algorithm [15] divides the entire map of the environment into smaller submaps. These small submaps contain only a part of the landmarks, so fewer correspondences have to be checked. This makes the entire algorithm faster. The ACO-algorithm can then be used to optimally align the submaps to form the entire map of the environment.

By updating the network of nodes after each resampling step, rather than recreating it before every ACO-update step, a big reduction in computational effort is possible.

5-5 Closing remarks

For this thesis the FastSLAM algorithm was implemented for full 6D pose estimation using the Kinect as a sensor. This thesis introduced a new algorithm to improve the loop closing behavior of FastSLAM. This novel algorithm, called ACO-SLAM, uses ant colony optimization to improve the resampling step in the current FastSLAM algorithm. During this resampling step particles with a low weights are discarded. The ACO-update step adds a higher weight to particles that lead to the most consistent map and to particles that lead to a loop closure.

The main question of this thesis was: Does optimizing the resampling step of the FastSLAM algorithm by using ant colony optimization lead to an increased loop closing performance?

ACO-SLAM can improve the performance of FastSLAM in terms of pose and map error. However, the performance of the algorithm is more sensitive to odometry noise and observation noise. If the standard deviations of the observation noise and odometry noise do not become too large, ACO-SLAM performs significantly better.

ACO-SLAM does not succeed in automatically closing the loop for the given environment, but this likely becomes possible if the accuracy of the estimates of the pose and the landmark positions is increased.

The current implementations of both the FastSLAM and the ACO-SLAM algorithms are too slow to be used in real time in the given environment. The computational time can be reduced by implementing the suggestions made in the previous section.
Appendix A

Appendices

A-1 Extra figures

This section contains the figures that were omitted from the main text. The boxplots that show the map and pose errors for varying odometry noise and a single particle are shown here for both simulated environments.

Figure A-1: Boxplots of the map errors for varying odometry noise and one particle in the simple environment
Figure A-2: Boxplots of the pose errors for varying odometry noise and one particle in the simple environment

Figure A-3: Boxplots of the map errors for varying odometry noise and one particle in the @Home environment
Figure A-4: Boxplots of the pose errors for varying odometry noise and one particle in the @Home environment


A-2 List of symbols

- $x_k$ : $x$-position of the robot frame w.r.t. the world frame
- $y_k$ : $y$-position of the robot frame w.r.t. the world frame
- $z_k$ : $z$-position of the robot frame w.r.t. the world frame
- $\phi_k$ : Yaw angle of the robot frame w.r.t. the world frame
- $\theta_k$ : Pitch angle of the robot frame w.r.t. the world frame
- $\psi_k$ : Roll angle of the robot frame w.r.t. the world frame
- $x_{s,k}$ : $x$-position of the sensor frame w.r.t. the sensor frame
- $y_{s,k}$ : $y$-position of the sensor frame w.r.t. the sensor frame
- $z_{s,k}$ : $z$-position of the sensor frame w.r.t. the sensor frame
- $\phi_{s,k}$ : Yaw angle of the sensor frame w.r.t. the world frame
- $\theta_{s,k}$ : Pitch angle of the sensor frame w.r.t. the world frame
- $\psi_{s,k}$ : Roll angle of the sensor frame w.r.t. the world frame
- $w_k$ : Stochastic variable to represent noise
- $x_{z_n,k}$ : $x$-position of feature $z_n$ in the sensor frame
- $y_{z_n,k}$ : $y$-position of feature $z_n$ in the sensor frame
- $z_{z_n,k}$ : $z$-position of feature $z_n$ in the sensor frame
- $x_{m_n,k}$ : $x$-position of landmark $m_n$ in the world frame
- $y_{m_n,k}$ : $y$-position of landmark $m_n$ in the world frame
- $z_{m_n,k}$ : $z$-position of landmark $m_n$ in the world frame
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u_k$</td>
<td>Translational velocity of the robot</td>
</tr>
<tr>
<td>$\omega_k$</td>
<td>Rotational velocity of the robot</td>
</tr>
<tr>
<td>$x_k$</td>
<td>Robot pose at time $k$</td>
</tr>
<tr>
<td>$x_{0:k}$</td>
<td>Robot trajectory from time 0 to $k$</td>
</tr>
<tr>
<td>$\hat{x}_k$</td>
<td>Estimated robot pose</td>
</tr>
<tr>
<td>$x_{s,k}$</td>
<td>Sensor pose</td>
</tr>
<tr>
<td>$\hat{x}_{s,k}$</td>
<td>Estimated sensor pose</td>
</tr>
<tr>
<td>$z_{n,k}$</td>
<td>Position of feature $n$ in the sensor frame</td>
</tr>
<tr>
<td>$\hat{z}_{n,k}$</td>
<td>Estimated observation</td>
</tr>
<tr>
<td>$z_{0:k}$</td>
<td>Observations from 0 to $k$</td>
</tr>
<tr>
<td>$m_{n,k}$</td>
<td>Position of landmark $n$ in the world frame</td>
</tr>
<tr>
<td>$m_{k}$</td>
<td>Stacked vector of all landmarks up to time $k$</td>
</tr>
<tr>
<td>$m_f$</td>
<td>Local landmark map, based on the current observation</td>
</tr>
<tr>
<td>$m_{ant}$</td>
<td>Landmark map constructed by an ant</td>
</tr>
<tr>
<td>$u_k$</td>
<td>Input vector of desired velocities</td>
</tr>
<tr>
<td>$u_{0:k}$</td>
<td>All inputs from 0 to $k$</td>
</tr>
<tr>
<td>$\Delta t$</td>
<td>Length of one time step</td>
</tr>
<tr>
<td>$N$</td>
<td>Data association matrix</td>
</tr>
<tr>
<td>$R$</td>
<td>Rotation matrix applied to a feature vector</td>
</tr>
<tr>
<td>$t$</td>
<td>Translation applied to a feature vector</td>
</tr>
<tr>
<td>$X^i_k$</td>
<td>Particle number $i$</td>
</tr>
<tr>
<td>$i$</td>
<td>Particle counter</td>
</tr>
<tr>
<td>$n_p$</td>
<td>Number of particles</td>
</tr>
<tr>
<td>$n_{sample}$</td>
<td>Sample size</td>
</tr>
<tr>
<td>$w^i_k$</td>
<td>Particle weight</td>
</tr>
<tr>
<td>$\eta^i$</td>
<td>Normalizing constant for a particle $i$</td>
</tr>
<tr>
<td>$n_k$</td>
<td>Data association function</td>
</tr>
<tr>
<td>$f(x_{k-1}, m_{n,k})$</td>
<td>Motion model</td>
</tr>
<tr>
<td>$h(x_k, m_{n,k})$</td>
<td>Observation model</td>
</tr>
<tr>
<td>$H_x$</td>
<td>Jacobian of the motion model</td>
</tr>
<tr>
<td>$H_m$</td>
<td>Jacobian of the observation model</td>
</tr>
<tr>
<td>$n_m$</td>
<td>Number of landmarks in the map</td>
</tr>
<tr>
<td>$n_z$</td>
<td>Number of features in one observation</td>
</tr>
<tr>
<td>$\sigma_{pos}$</td>
<td>Standard deviation of odometry noise for the position</td>
</tr>
<tr>
<td>$\sigma_{ornt}$</td>
<td>Standard deviation of odometry noise for the orientation</td>
</tr>
<tr>
<td>$\sigma_{obs}$</td>
<td>Standard deviation of observation noise</td>
</tr>
<tr>
<td>$\tau_i$</td>
<td>Pheromone level on node $i$</td>
</tr>
<tr>
<td>$\rho$</td>
<td>Evaporation rate</td>
</tr>
<tr>
<td>$n_{path_2}$</td>
<td>Length of the second part of the ant path</td>
</tr>
<tr>
<td>$F_{ant}$</td>
<td>Fitness of a particular ant</td>
</tr>
<tr>
<td>$F_{path}$</td>
<td>Fitness function of the ant path</td>
</tr>
<tr>
<td>$F_{map}$</td>
<td>Fitness function of the ant map</td>
</tr>
</tbody>
</table>
### A-3 List of abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACO</td>
<td>Ant colony optimization</td>
</tr>
<tr>
<td>EKF</td>
<td>Extended Kalman filter</td>
</tr>
<tr>
<td>ICP</td>
<td>Iterative closest point</td>
</tr>
<tr>
<td>IQR</td>
<td>Inter quartile range</td>
</tr>
<tr>
<td>KLT</td>
<td>Kanade-Lucas-Tomasi feature tracker</td>
</tr>
<tr>
<td>MRPT</td>
<td>Mobile robot programming toolbox</td>
</tr>
<tr>
<td>PCL</td>
<td>Point cloud library</td>
</tr>
<tr>
<td>RBPF</td>
<td>Rao-Blackwelized particle filter</td>
</tr>
<tr>
<td>ROS</td>
<td>Robot operating software</td>
</tr>
<tr>
<td>SLAM</td>
<td>Simultaneous localization and mapping</td>
</tr>
<tr>
<td>UKF</td>
<td>Unscented Kalman filter</td>
</tr>
</tbody>
</table>