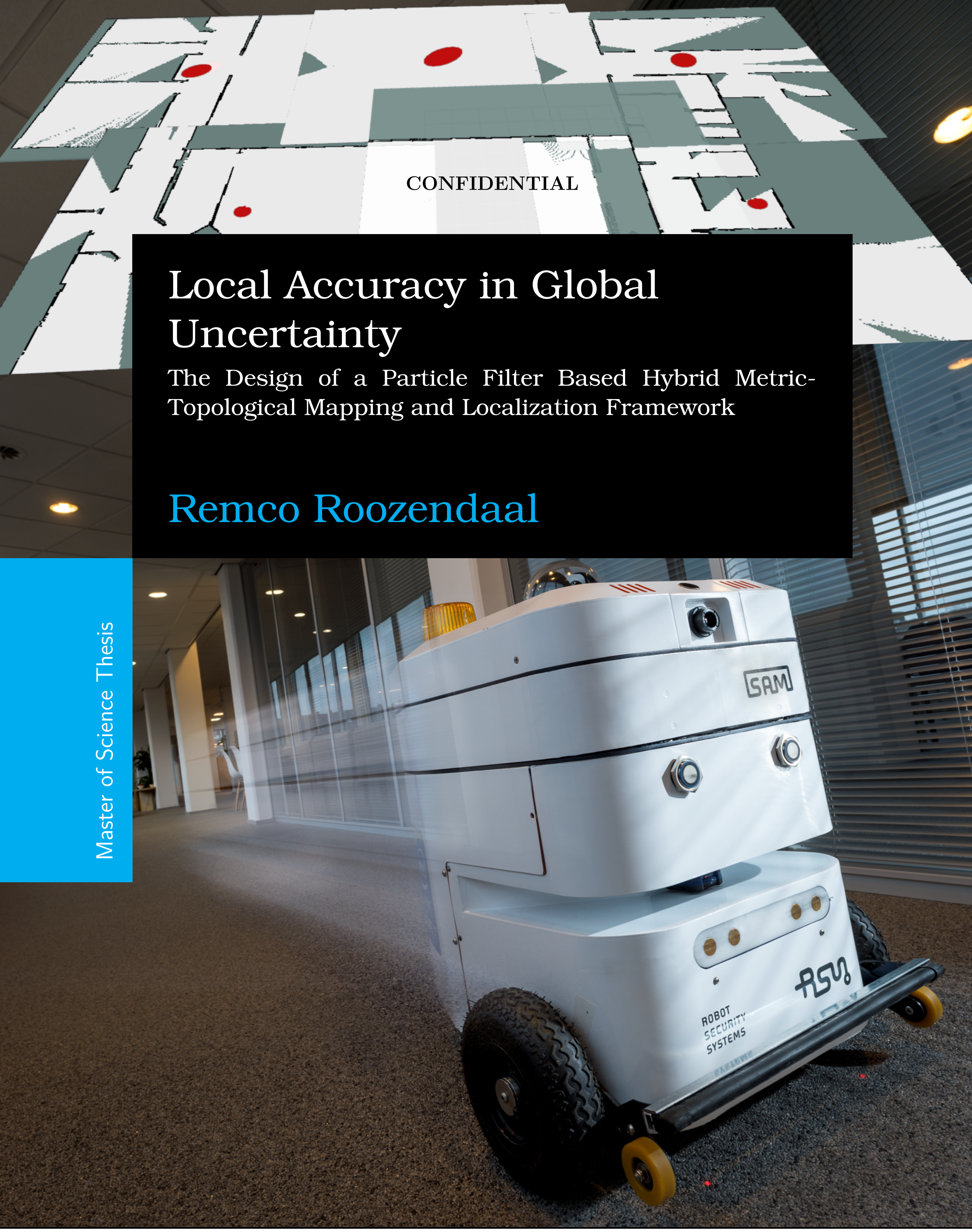
CONFIDENTIAL

Local Accuracy in Global Uncertainty

The Design of a Particle Filter Based Hybrid Metric-Topological Mapping and Localization Framework

Remco Roozendaal

Master of Science Thesis



Delft University of Technology

Delft Center for Systems and Control

Local Accuracy in Global Uncertainty

The Design of a Particle Filter Based Hybrid Metric-Topological Mapping and Localization Framework

Master of Science Thesis

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

Remco Roozendaal

October 13, 2016

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

This thesis is supported by Robot Security Systems and Robot Care Systems. Their cooperation is hereby gratefully acknowledged.

Delft University of Technology
Department of
Delft Center for Systems and Control (dcsc)

The undersigned hereby certify that they have read and recommend to the Faculty of Mechanical, Maritime and Materials Engineering (3mE) for acceptance a thesis entitled

Local Accuracy in Global Uncertainty

by

Remco Roozendaal

in partial fulfillment of the requirements for the degree of

Master of Science Systems and Control

Dated: October 13, 2016

Supervisor(s):

Prof. dr. R. Babuska (DCSC)

Ir C.J. Lekkerkerker (RCS)

Reader(s):

Dr. J Alonso Mora (DCSC)

Prof. dr. ir. P. Jonker (BMechE)

Abstract

Mobile robots need to interact with their environment to perform their tasks. To be successful they often need to know what their surrounding looks like, and where they are located in that surrounding. The act of simultaneously estimating both the state of the robot and the state of the environment is called Simultaneous Localization And Mapping (SLAM). The SLAM problem has been intensively researched since it is one of the key prerequisites for correct functioning of mobile robots.

Popular mapping algorithms use Kalman or particle filters to describe the robot state and the environment with probability distributions. When the mapped environment becomes larger these algorithms need more computational resources and have trouble recognizing known locations. This is mostly because the stored map becomes too large to process and the uncertainties have become too big to properly be described or used in the SLAM framework.

This work proposes a Hybrid Metric-Topological (HMT) mapping method to solve problems regarding place recognition and computational resources commonly experienced when mapping large environments. The framework will create local maps using a particle filter and connect the local maps in a graph constrained with edges. The local environment is mapped accurately, while the relative uncertainties are stored in the edges. In this work the devised framework is called "Forced Resampling hybrid Metric-Topological Mapping" or FoRMeT Mapping. It is characterized by being an approximate solution to the SLAM problem that makes justifiable assumptions to be a lightweight HMT framework. The biggest innovation is the forced resampling strategy which allows the particle filter ambiguity to be stored in the last two visited local maps, while keeping all other maps in a shared topology.

An implementation of the framework is devised in C++ with ROS (Robot Operating System) and tested on a real world environment containing a large loop, and a simulated large scale environment. The same experiments are also performed with the commonly acknowledged mapping algorithm "GMapping" for comparison. FoRMeT Mapping outperforms GMapping on all performance criteria. FoRMeT uses about 3 times less computational resources for each task and the local areas look as good as GMapping's local areas, with exceptions around the loop closure areas where FoRMeT looks better with less discrepancies. It can be concluded that FoRMeT can close large loops, map large environments and outperform GMapping while remaining a lightweight SLAM framework.

Table of Contents

List of Figures

List of Tables

Preface

Since the classical times cartographers and alike have tried to abstract the world around them into maps to be used for navigation, strategic warfare, science and more. As we know now with satellite imaging, these maps are not always accurate on a global scale. Even the rectangular map we nowadays often use is not accurate since the earth is round. Nevertheless people know how to get from place to place using this map, because the local areas on the map are accurate enough. With the rivers, mountains, forests and coastlines as landmarks in the map people can navigate from local area to local area without even knowing their global map is skewed and deformed.

This document is a part of my Master of Science graduation thesis. The subject came into existence after a talk with the people at Robot Security Systems who had trouble mapping large environments because of increasing computational complexity and inaccuracies. I set out to develop a framework in which the robot does not create a single global map, but a network of local maps. The result of which is visible in this work.

I want to express my thanks to the people at Robot Security Systems for letting me pursue this project at their offices and for making my thesis time enjoyable. I want to thank Pieter Jonker for connecting me with the company giving me the opportunity to graduate there. I would like to thank Robert Babuska for helping me focus my thesis subject and for making the arrangements for my graduation. And also I want to thank my daily supervisor Koen Lekkerkerker for the times he lend me his knowledge about both SLAM and programming in C++.

The Hague, Remco Roozendaal
October 13, 2016

I would like to dedicate this work to my parents Bernard Roozendaal and Ineke Roozendaal Koelemeijer who raised me free and unconditionally, facilitating my curiosity and creativity.

Chapter 1

Introduction

Our lives are surrounded with machines using advanced technology. We are entering an age where nobody is surprised anymore about the improvements made between mobile phone releases, where everybody expects their lives to be made easier soon by all sorts of robotic applications and where some already have the possibility to take their hands of the steering wheel at the highway. It is undeniable that computers, electronics and mechanics combined can excel in specific tasks in a way humans would never be able to. The entrepreneurial opportunities lay in fields where robots can do human work with reduced costs, efforts or increased safety. Many robotic applications are being developed to fill the human need for automation, and research is being done in all parts of this multidisciplinary field. Soon also mobile robots will show up in our daily lives, if they are not there already. A characteristic of mobile robots is that they need to interact and move around in the environment to perform their task. To be successful they often need to have an understanding of what their surrounding looks like and where they are located in the environment. The research in this field is referred to with the words *mapping* and *localization*.

1-1 Mapping and localization

A large body of literature is available, and still being supplemented, about mapping and localization. It is regarded as one of the key prerequisites for the functioning of a mobile robot, but it is a difficult problem to solve and it involves many disciplines. Mapping and localization are tightly interwoven, because the output of either one is necessary to solve the other. For the robot to know where the wall is in the which he observes w.r.t. itself he needs to know where he is in the map. And when he reobserves this wall later, the difference in relative location of the wall indicates a difference in the robots pose. Because of uncertainties in sensor data the location and map can not always be estimated accurately and the mapping and localization is commonly solved as an estimation problem with uncertainties in the states. Because of this inaccuracy robots often produce a skewed and deformed map when mapping larger areas. Also the production of a large global map often requires a lot of computation power. Therefore this work will focus on creating a network of accurate local maps, and keeping the uncertainty in the relative location of the maps by use of a graph with nodes and edges.

1-2 Thesis statement

Future robots should be able to perform their tasks regardless of the size or layout of their environment. This thesis will produce a framework that will efficiently represent the environment as a network of small local maps. The small maps will be attached to topological nodes which are defined relative to each other with edges carrying uncertainty. It is hypothesized this localization and mapping framework can handle larger environments than other modern localization and mapping methods.

1-3 Design specifications

The design specifications are given below.

- The robot should be able to map larger environments than other modern mapping methods while using similar resources.
- The map should be globally consistent by approximation for human interpretation.
- The lower the computational load the better, there is room for offline processing.
- The framework should not limit itself to a specific environment but be general.
- A grid based map representation is desired for easy human interpretation.

1-4 Thesis outline

This work will explain the proposed framework called Forced Resampling hybrid Metric Topological (FoRMeT) Mapping. Also an implementation of the framework in C++/ROS (Robot Operating System) is discussed with which the experiments are done to check the performance. For the readers that are not mapping and localization experts it is recommended to read the background chapter 2, and supplement anything unknown with information in the appendices. For the more experienced readers it is recommended to read into the HMT mapping section (2-5) and continue to the explanation of the framework and further.

Chapter 2: Background The basics of probabilistic robotics are introduced here. It contains definitions, and background information necessary for the subsequent chapters.

Chapter 3: Framework The proposed framework is treated here in its theoretical form.

Chapter 4: Implemented Framework The implementation of the proposed framework used to test the functionality is discussed here.

Chapter 5: Experiments The implemented framework is tested on different datasets.

Chapter 6: Conclusion Conclusions are made about the frameworks performance.

Chapter 7: Discussion and Recommendation What can be said about the framework, what are the limitations, and where can improvements be made?

Appendices Appendices about the Bayes filter (A), Kalman filter SLAM (B), particle filters (C), Monte Carlo localization (D), shortest path algorithms (E), homogeneous coordinates (F) and iterative closest point (ICP) (G) are attached at the end to supplement the reader with extra background information.

Chapter 2

Background Information

This chapter will treat the information necessary to understand hybrid metric-topological maps and their applications in robotics. First metric SLAM will be treated, then topological SLAM and the hybrid combination of both on which this work focuses. For a more detailed description of Kalman filter SLAM and Particle filter SLAM, the reader is referred to appendices B and C.

2-1 SLAM

Mapping is the action of finding the state of the environment. For a mobile robot the state of the environment includes the places it can and cannot travel and the objects it can use for localization called *landmarks*. By using distance sensors the robot estimates the relative distances of objects to its own coordinate frame. To create a map with these measurements it needs to know its own location in the map. The estimate of its location based on its motion sensors is called *odometry* . Since any sensor is prone to noise and the odometry typically integrates motion to get distance, it gives a quite erroneous estimation. Landmarks can be observed with the robots observation sensors and they are used to aid localization and determine where the landmark is to create a map. We have now arrived at the origin of a large body of research in mapping and localization, because the landmarks are used for localization, and the localization is needed to estimate the locations of landmarks in the map. This chicken and egg problem is referred to as Simultaneous Localization and Mapping (SLAM).

2-1-1 Interaction variables

The robot state is denoted as x, the state at time t is x_t and the state evolves from t_1 to t_2 as $x^{t_1:t_2} = \{x_{t_1}, x_{t_1+1}, .., x_{t_2-1}, x_{t_2}\}$. Notice the superscript notation to indicate the sequence up to a time, and the subscript notation to indicate the value at a specific time. The two other generalized variables are observation measurements and control input. The control input contains the input given to change the robot state and therefore the observed world. The control input at time t is denoted as u_t and evolves from t_1 to t_2 as $u^{t_1:t_2} = \{u_{t_1}, u_{t_1+1}, .., u_{t_2-1}, u_{t_2}\}$. Because the translation from control input to robot motion often involves complex models, odometry is frequently taken as the source of the control input variable. The observation measurement at time t is denoted as z_t and evolves from t_1 to t_2 as $z^{t_1:t_2} = \{z_{t_1}, z_{t_1+1}, .., z_{t_2-1}, z_{t_2}\}$.

2-1-2 Map appearance

The map will consist of the locations of the landmarks as estimated by the robot. These landmarks can be represented with their mean and uncertainty ellipse derived from their covariance. Such a landmark map is visible in Figure 2-1.

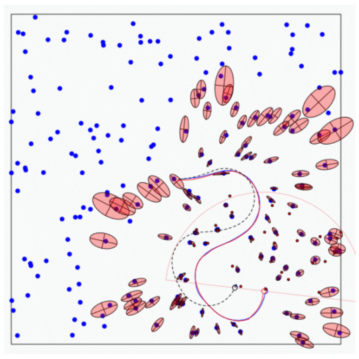


Figure 2-1: A map containing the robot odometry (dashed), the corrected path (red line), the real (blue), and the estimated (red) locations of the landmarks. From [1]

Another popular map type is the point map, where observed points are projected into the map as visible in Figure 2-2. This contains more information than a landmark map, but this also means it uses more memory.

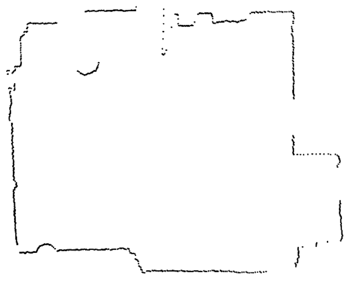


Figure 2-2: A pointmap containing projected range measurements points.

As a more compressed format of the point map, the occupancy grid map is often used. In a grid map each grid cell represents a square plane in space, and that cell can be either occupied, free or unknown. Hence a ternary representation of space is obtained with a resolution depending on cell size.

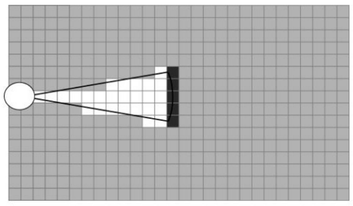


Figure 2-3: An occupancy grid map. White cells are free, black cells are occupied and gray cells are unknown. From [2].

2-1-3 Traditional SLAM

Traditional SLAM is based on the Kalman filter. It estimates the location of the robot often referred to as robot *pose*, and the locations of the landmarks. More about the KF can be found in appendix A. The KF is a linear solution, but the SLAM problem contains non-linear motion and observation models, hence the pose distribution is not normally distributed. Therefore the Extended Kalman Filter (EKF) can be used which linearizes the models is used for traditional SLAM. The EKF linearizes the models at operating point so it can perform the linear estimation, the result will be Gaussian estimates of th robot and landmark poses. A more detailed description of the EKF can be found in appendix B.

The KF based solutions are nowadays not the standard anymore. The linearizations cause the estimates to be suboptimal and the complexity grows quadratically with the amount of observed landmarks ($\mathcal{O}(n^2)$). Also it assumes uniquely identifiable landmarks, therefore any wrong data association will cause an erroneous map.

2-2 Particle Filter SLAM

This work will make use of a modern SLAM method, Particle Filter (PF) SLAM. PF SLAM is based on the Particle Filter theory where a distribution is approximated by multiple hypotheses called particles. An introduction to particle filters is included in appendix C. Particle filters work optimal when they are estimating only a couple of variables, therefore they are not suited to do the estimation for the whole KF state vector including landmark locations. But they are excellent to estimate the pose of the robot, just the x, y, θ variables. By separating the estimation of robot pose and landmarks, the landmark locations can be estimated by n Kalman filters with 2×2 covariance matrices for every particle. Separating the SLAM problem into a set of particles for the pose estimation and a Kalman filter for each landmark per particle is called Rao Blackwellized Particle Filter (RBPF) SLAM.

2-2-1 Rao Blackwellized particle filter

Particle filter SLAM tries to avoid the chicken and egg problem where localization and mapping are continuously interwoven. It takes advantage of the fact that creating a map is easy when the traveled path is known. PF SLAM decouples the SLAM problem into two separate estimation problems [6]. One estimates the path of the robot, based on the measurements and the controls, and the other estimates the landmark locations based on all the previous. The factorization of the SLAM problem in two estimation problems where the first part is estimated by a PF and the second part by a EKF is known as *Rao Blackwellization* (eq. 2-1).

$$P(s^t, l|z^t, u^t, n^t) = P(s^t|z^t, u^t, n^t) \prod_k P(l_k|s^t, z^t, u^t, n^t) \tag{2-1}$$

Here the k landmarks are estimated based on a known path, simply all robot poses up to time t. The new variable n^t carries the associations between the landmarks which are needed for correct estimation of both the map and the path. In particle filter SLAM the path s^t is estimated with a particle filter. In KF SLAM the Markov assumption (A-2-3) is made, which assumes that the previous pose is estimated perfectly and thus is the only necessary parameter from the previous time steps. PF SLAM does not have to make this assumption for previous poses because it estimates the whole path, all subsequent poses from t_0 up to t. It can however make the assumption that the landmark poses are conditionally independent when all poses are known, allowing the factorization in equation 2-1.

2-2-2 Resampling

The resampling step is usually an computationally expensive step. In the classic particle filter the resampling step is performed each iteration, however it is often found unnecessary in RBPF SLAM because the particle set might describe the pose posterior well. Typically a healthy particle set has many particles around a high state probability. This way there are allways a couple of particles very accurate. However, when the particles evolve over time they spread out, and not enough particles may be left around the true pose. RBPF SLAM chooses the time to resampling based on a performance criterion, when the variance in particle scores becomes too high, it resamples. This performance criterion is called the *number of effective particles*, η_{eff}. This number is calculated as in Eq. (2-2), with w_k begin the weight of a particle.

$$\eta_{eff} = \frac{1}{\sum_{k=1}^{n}(w_k)^2} \tag{2-2}$$

By only resampling at these strategic moments, RBPF SLAM reduces computational load while maintaining a healthy particle set. The resampling happens the same as in the classic particle filter, the chance for a particle to be sampled is proportionate to its weight, hence the resampled set is expected to have more particles with higher weights.

2-2-3 Loop closure

The loop closure for a Kalman filter has been described in B-3. For a particle filter loop closure works a bit differently because an observation cannot change the pose, it can only weigh a particle. The right particle must thus already be present at the moment of a loop closure and the observation is used to point out which one is right.

2-2-4 Rao Blackwellized particle filter

The estimation of the robot path by particle filter has been covered now. To solve the full SLAM problem the landmark locations still need to be estimated. Due to the high dimensionality of this problem, particle filters are not suitable for this, and in practical algorithms a Kalman filter is typically used to estimate the landmark positions. A particle filter SLAM algorithm will therefore have k particles containing the robot pose x_t^k, and an Extended Kalman Filter with mean $\mu_{j,t}^k$ and covariance $\Sigma_{j,t}^k$ for each landmark l_j.

2-3 FastSLAM

Montemerlo et al. have written the FastSLAM [6] and FastSLAM 2.0 [7] algorithms which are RBPF based SLAM algorithms. FastSLAM 2.0 is an improved version published five years after its predecessor FastSLAM.

It maintains M particles and N landmarks by a set of $MN + 1$ low dimensional filters, which require a constant update cost regardless of the path length. Because for each particle the map is calculated to estimate landmark positions, only 2x2 covariance matrices have to be calculated, in stead of an NxN matrix which grows quadratically with every landmark. The $MN + 1$ update cost is when known data association is assumed. When data association is efficiently implemented the computational complexity of FastSLAM will become $\mathcal{O}(M \log N)$).

During the resampling step the history, poses and landmark estimates, of another particle need to be copied, which can be a costly operation for large maps and trajectories. FastSLAM avoids most copying by pointing to the location of the landmarks and poses and using an ancestry tree structure for looking them up.

FastSLAM became popular for it's benefits over EKF SLAM. It could handle multiple pose hypothesis', is was more likely to recover from wrong associations because of its ability to pursue multiple data associations in different particles, it could handle significantly more landmarks because it did not add them to the big covariance matrix, but examines them per landmark per particle.

FastSLAM 2.0 overcomes some deficiencies of its predecessor. The motion update to sample the particles into their new pose now also incorporates the latest measurement besides the odometry information. So a measurement update has already been incorporated into the motion sampling. This makes for more particles being sampled in the posterior distribution and thus for a higher accuracy of the particle filter.

2-3-1 RBPF based occupancy grid mapping

Currently the most used open source mapping algorithm is *GMapping*, which uses a form of RBPF SLAM to perform occupancy grid mapping [8]. The GMapping software library is open source available on ROS[1]. The path traveled by the particles is recorded, and the observations are projected from these poses. The occupancy grid mapping algorithm identifies which grid cells are occupied and free for each observation, and keeps the average stored in each cell.

[1]http://wiki.ros.org/gmapping

When the average is over a certain occupation threshold this cell can be regarded as occupied. The landmark estimation is thus a mere average of projected observations, but it works well in practice. No data association algorithms are necessary to identify landmarks, just a projection of the observations into the map to update grid cell averages. GMapping is based on FastSLAM 2.0 and also uses an ancestry tree with poses to store particles dependencies on each other. The map visualized at runtime, which is the map of the best scoring particle, is constructed real time with this tree structure and the observations. This way not every particle explicitly carries the whole global grid map.

Even though this is a non linear solution, still small odometry errors, or bad resampling could deform the resulting map over longer distances. Also the particles may be locally accurate, but for large maps the particle able to close a large loop (see section B-3) may have been lost in sampling. This is why this work will focus on using a RBPF SLAM for creating local maps, and keeping these local maps in a *topological* network.

2-4 Topological SLAM

A very different approach to mapping is *topological mapping.* Topological mapping is based on the topology of the environment instead of the metric properties.

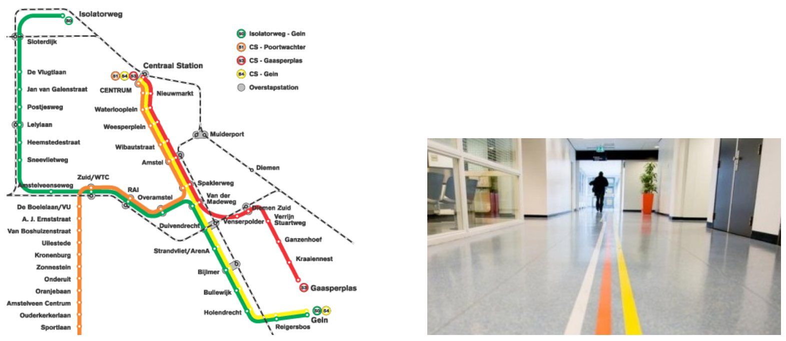
2-4-1 Topological maps

Topological maps are defined by *nodes* connected by *edges.* The nodes represent places in the environment and the edges define relations between the nodes. The usual relation captured in an edge is *navigability*, which means that you can navigate between two nodes if they are connected by an edge. A good example of a topological map is a subway map. A subway map is not made to represent the exact metric locations of stations and tracks, but rather which stations can be reached with which tracks in a way that is clear for the user (Figure 2-4 (a)).

With such a philosophy in mind topological maps can also be used by robots, the nodes and edges should then contain information the robot can use to navigate. In a hospital for example this could mean one node is the main entrance and another node is the intensive care department, a colored line system is occasionally used to indicate a route to a department, which a robot could perceive as an edge. Using these nodes and edges the navigation system could say: "take the red line until you arrive at the intensive care department", without the need of any metric information.

2-4-2 Semantic maps

A human would usually not know his exact metric location with respect to some reference point, but can reason through the objects in his environment where he is. A human would not know he is 4 meters in x, 6 meters in y and 3 meters in z direction as opposed to the main reference frame of a building. He just knows he entered the main door, walked up the stairs, took the second door on the right in the hallway and that he is now standing in a room that looks like an office, because there are many chairs and desks. This can only be understood if the objects in the human surrounding, like entrance, stairs, hallways, desks,



(a) metro map of Amsterdam [9]

(b) guiding lines in a hostpital [10]

Figure 2-4: Topological examples.

chairs have a *semantic* meaning to him. Semantics give meaning to raw data, and a robot that uses semantics can use this for localization and mapping.

A robot could use the abstracted information from its sensors to reason about high level semantics of its environment. It could use a laser range finder to infer the size and shape of the room, its camera to identify objects in the room and it could even use other sensors like temperature sensors to infer semantics of a space.

Semantic maps try to use as little metric information as possible and are a good example of topological maps. However often it is found useful to incorporate some metric elements in the map. Such a map is then not strictly a topological map anymore, but a *hybrid metric-topological map*.

2-5 Hybrid metric-topological SLAM

Hybrid Metric-Topological (HMT) mapping combines both metric and topological paradigms by including metric information in the nodes and edges of the topological graph. Any *global mapping* system, with a single map, will represent the reality a bit deformed over larger areas. For traditional SLAM these deformations will appear sooner than for modern SLAM, but they all have a size where the map gets so inaccurate that loops cannot be closed anymore. The HMT mapping that is discussed in this work is designed to divide the global space in a set of local subspaces which are represented with one or more nodes connected with edges. The underlying SLAM algorithm, may it be traditional, or any modern type, should be accurate in the local subspace. The inaccuracy in the global space should be captured by the topological network as an *uncertainty* of where the local subspaces are opposed to each other. In this work the uncertainty is always closely related to the covariance. The covariance represents a three dimensional Gaussian, and the uncertainty is represented by an ellipse which indicates the boundary in which a point in this Gaussian will be with a defined probability. I.e. there is a probability of 0.95 that the pose of the robot is in the area of the ellipse. Only the x and y values of the covariance can be expressed in this 2 dimensional representation. The radii of this ellipse are calculated by taking the eigenvalues of the covariance matrix, and the orientation is calculated with the arctangent of the eigenvectors. The two HMT frameworks identified as most relevant to this work will be discussed in this section.

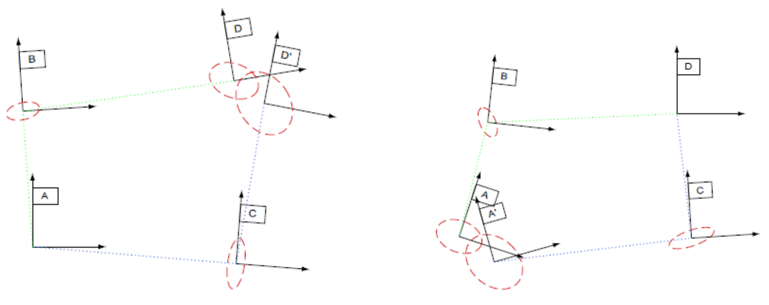
2-5-1 HMT with traditional SLAM

The first HMT framework is designed on top of EKF SLAM and called "Atlas" [3]. It maps its environment with an EKF into line segments in local maps. The lines in the maps carry a mean and covariance and the local maps also carry uncertainty about their location which is captured by edges connecting the map frames. The local maps are added to a graph which is updated after loop closure to 'stitch' them together using nonlinear optimization.

Genesis Genesis is a term used in the Atlas framework to describe the generation of a new map. When the complexity of a local map becomes too large, and a new unexplored area is entered, a new map frame is created at the origin of the current robot pose. This new map frame is connected with an edge to the previous map frame. The edge contains a mean and variance. The robot will continue exploring with his pose relative to the current map frame.

Competing hypotheses This is Atlas' version of a particle filter. The location of the robot can be explained in different map frames. Only one hypothesis can exist per map frame, there are several map frames competing and the best explanation is chosen by a performance metric based on measured features. This way the robot can be in multiple maps with respective probabilities, with the best probability being the robots most likely pose/map. The robot continuously looks for the best explanation.

Uncertainty projection The uncertainty of a node is always relative to another node and is captured in the edges with a normal distribution. If you would want to calculate the uncertainty of node D as opposed to node A, it would be a different shaped uncertainty than that of A as opposed to D (Figure 2-5).



(a) Uncertainty of D seen from A

(b) Uncertainty of A seen from D

Figure 2-5: Uncertainty projection of different routes between two nodes in [3]

Because nodes are only defined relative to each other Atlas calculates the so called *uncertainty projection* to know the uncertainty to each other node by using Dijkstra's solution from the current node (Appendix E). For the weight of the edges in the Dijkstra algorithm the determinant of the covariance is used as measure of how certain the location of the node is. It will look up the route from node to node, and it will project the uncertainties over the corresponding edges. The uncertainty projection is being calculated with the edge transforms and covariances as in Eq. (2-3). Here T_a^c is the total transform, and Σ_{nm} the covariances for each edge. J_n are the Jacobians of transformation composition, where J_1 is the Jacobian of the translational part of the transformation and J_2 the Jacobian of the rotational part.

$$\begin{aligned} T_a^c &= T_a^b + T_b^c \\ \Sigma_{ac} &= J_1 T_a^c \Sigma_{ab} J_1 (T_a^c)^T + J_2 T_a^c \Sigma_{bc} J_2 (T_a^c)^T \\ \rho &= det(\Sigma_{ac}) \end{aligned} \tag{2-3}$$

In Figure 2-5 the route A-D can be reached through node C or B both giving another mean and covariance for node D, hence it is important to use the path with the least uncertainty as defined by ρ. Besides it can be seen that the inconsistency is always kept furthest away from the current node. This uncertainty projection will be used to detect possible loop closures.

Map Matching When the uncertainty of the current robot pose with respect to the local map frame overlaps the projected uncertainty of another map, it qualifies as a potential loop closure.

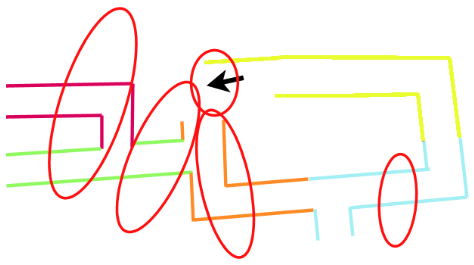


Figure 2-6: An improvised illustration of different line elements grouped by color forming local maps. The projected uncertainty for each map is visualized in red. The robots uncertainty overlaps with the uncertainty of two other maps.

To determine if the loop closure is valid, a *map matching* algorithm is deployed. Map matching is described as finding a coordinate transformation which aligns the current observations with a candidate map. To start, a signature is made of the current map, which captures important features. This signature is matched first with its own map to determine if there are any non unique features present which can be explained by multiple transformations. The non unique features are removed from the signature and now the signature can more accurately be used to match with the candidate maps.

Elements in the signature are individually matched with elements in the candidate map. So one line element could have i.e. three possible matches with respective transformations. Each of these transformations are also applied to the other elements in the signature. This is continued for each element until for each element the best matching transformation is found. These transformations are then solved in a weighted least squares optimization, using the covariances of line elements stemming from the Kalman filter estimation. A loop can then be closed and the uncertainties can be reduced.

Traversal When the robot moves from one map frame to another it uses the 'competing hypotheses' to determine which map will be the new associated or *dominant* map. There are

4 stages a map-robot association can be in: *Dominant, Mature, Juvenile* and *Retired.* A map can only move from one to another stage in the direction given in figure Figure 2-7.

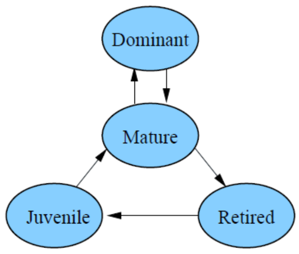


Figure 2-7: Every existing map in Atlas [3] is in one of these states and can move from state to state in the direction of the arrows.

Most of the maps are retired and not candidates to be the robots current map. Every map that has a hypothesis value is considered at juvenile stage, every non-retired map can therefore keep track of their performance. From this pool of juvenile maps the hypothesis' could mature if it has a high enough performance metric. From the pool of mature maps the map which represents the environment of the robot the best is chosen to be dominant. When the robot moves around to the next map, the previous dominant map will become mature again and retire when it is not near anymore.

Edge refinement When there are two competing map hypotheses and both can explaining the current robot pose in their own frame very accurately, the relation between the two pose estimates can be used to reduce the uncertainty between the two map frames. When the uncertainties are reduced, better estimates of loop closure and traversal candidates can be made.

2-5-2 HMT with RBPF SLAM

Blanco et al. [4] have build a RBPF SLAM based HMT framework called HMT-SLAM. They have taken inspiration from the particle filter, and have extended some of its functionality to the network of local maps. It uses not only hypotheses about the robot pose, but also about the topology.

Local maps HMT-SLAM uses local grid maps connected with edges which are rigid body transformations. As opposed to Bosse et al. [3], the map size is based on the amount of features with overlapping visibility between observations. This means that when you would go through a door for example, you would see many features not visible on the other side of the door. The features on either side off the door would likely be grouped in different maps.

By separating the maps this way it will be unlikely to observe features of two different local areas at the same time, herewith constructing the topological graph in a way that it's permissible to assume conditional independence between the nodes and do graph optimization later.

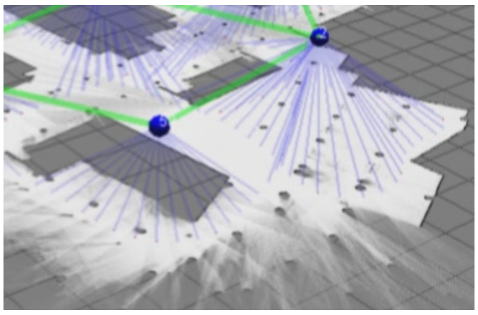
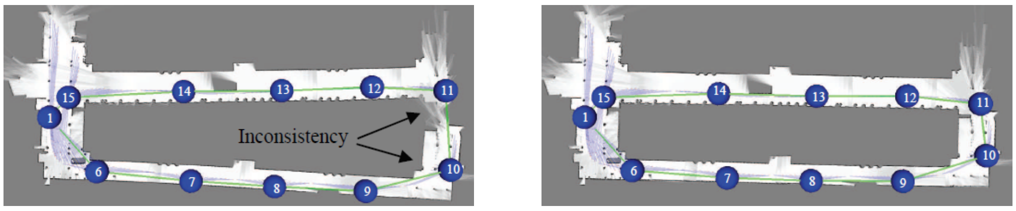


Figure 2-8: Maps created by HMT-SLAM [4], blue lines go from features to map nodes to indicate which features are grouped to one map. Nodes are shown in blue and edges in green.

Particles There are particles maintaining a hypothesis about both the topological configuration and the current local map. Each hypothesis of the topological path implies a different topological structure of the environment and thus, a different hypothesis for the current local map. The robots hybrid metric-topological path is now described by $s^t = \langle \gamma^t, x^t \rangle$ with γ^t the topological path and x^t the metric path traveled up to time t. The hybrid path is estimated by a particle filter. This is nothing more than a particle filter for the metric path, but the particles may have different paths and therefore their topological areas look different and are organized differently, the particles keep track of these nodes and edges.

Loop closure The particles keep moving on while segmenting the features into maps. Whenever a new map is segmented e.g. an explored area is left behind as being one map, the particle checks if the features in this map do not already belong to another map and if so performs a loop closure. The algorithm is thus performing SLAM, and in the background it does segmentation and loop closure. To extend the particle filter hypotheses strategy to the topological structure a multiple hypothesis system is used for traversing between local areas. There is a hypothesis of revisiting a local area, and of entering an unexplored area, both sum up to one. The likelihood of both hypotheses is tracked until a single hypothesis becomes evident enough.
Every time the robot moves to an already mapped local map, there will be an estimate of how the nodes are constraint. The algorithm will reduce the uncertainty in the edges and make a new estimate of the relative locations. When a global map is desired the optimal global node locations are computed iteratively by linearizing the optimization problem defined by the edge constraints between the nodes (Figure 2-9). Note that HMT-SLAM also uses the Dijkstra solution and uncertainty projection (Subsection 2-5-1) to visualize the maps, which positions the inconsistency the furthest away from the robot pose.



(a) before optimization **(b)** ater optimization

Figure 2-9: Node locations with corresponding maps projected from current node (15) in HMT-SLAM [4]. Optimization is performed for global consistency.

Scalability It would appear that this method will be as suffer from similar scalability issues as RBPF SLAM since it keeps a memory of the whole path s^t (which also contains the topological path γ^t), which keeps on increasing when the robot keeps on moving. However, the particles only maintain the part on which they don't agree in their memory. Local maps can be fixed from the path on which they agree, these are stored in memory and only used when reentered. The algorithm thus only holds an immediate memory of the part in which the particles do not agree, making it a scalable algorithm with less complexity than RBPF SLAM for large environments.

Results HMT-SLAM is tested over a 2km path in an approximately 150m x 150m building at the University of Malaga campus. It is compared to RBPF SLAM based on the same article as GMapping (perhaps it was GMapping). It measures how long both algorithms take to process the data into a map, and how much memory it uses. HMT-SLAM performs better on both datasets as can be seen in table 2-1.

Table 2-1: Results from the HMT-SLAM [4] algorithm on two datasets.

Method	**Málaga dataset**	**Edmonton dataset**
Global RBPF	197Mb, 103min	84Mb, 39min
HMT-SLAM	36Mb, 26min	28Mb, 8min

Chapter 3

The FoRMeT Framework

It has been made clear that theoretically HMT-SLAM can have a better scalability than global SLAM systems. This work will take inspiration from the discussed works and devise a new framework which can map large scale environments while remaining a lightweight application. These frameworks have proved themselves, but improvements can be made in computational complexity when looking at RBPF SLAM and HMT SLAM. From the discussed HMT-SLAM systems [3], [4], a number of key HMT elements can be extracted:

1. Local mapping algorithm
2. Map size and separation
3. Map relations
4. Matching
5. Traversal
6. Optimization

The designed framework is called Forced Resampling hybrid Metric Topological (FoRMeT) Mapping and this chapter will discuss in general how the different elements of this framework are designed. An implementation to validate this framework's functionality is discussed in Chapter 4.

3-1 Local mapping algorithm

The choice is made to use RBPF SLAM as a basis for the FoRMeT framework. RBPF SLAM is a nonlinear solution to the SLAM problem which can yield very accurate results locally. Some properties of RBPF SLAM are excellent for building a global map, but these properties might get in the way when building a network of local maps. Also some solutions for practical implementation of RBPF SLAM should be evaluated since they might be unnecessary. These topics will be discussed in this section. When deciding how to optimally use RBPF SLAM one should keep the philosophy behind FoRMeT in mind: "Local accuracy in global uncertainty". The steps of a regular particle filter SLAM are the following.

1. Sample new pose from distribution.
2. Weight update.
3. Resampling.
4. Map update.

PF-SLAM has already been treated in 2-2, however to be extended to FoRMeT some of the functionality can be adjusted. The adjustments will be discussed in the following paragraphs.

Sample from motion model First let's look at the motion model sampling. Normally the particles are sampled in the prior distribution, but in RBPF SLAM it is common to include the measurement model into the sampling to increase the amount of particles sampled in the posterior distribution (2-3). This also helps with odometry errors caused by e.g. wheel slipping. In practice GMapping does this by moving all the particles a bit around their sampled pose, and check if their weight improves. This is a costly operation, but works in practice. In HMT mapping however there are often observed features that are outside the range of the map, even a map without features is possible. The method of GMapping does thus not always work in the HMT case. It is therefore decided to compare the current observation with a key frame observation from the past. The odometry will give a prior estimate, and the key frame observation matching will correct this estimate. The two estimates are combined with a Kalman filter and the corrected distribution is used as proposal distribution for the motion model sampling.

Weight update The weight update gives a weight to each particle which is a measure of how well the particle's pose fits with the current observation in its own map. This weight update takes the current observation, the sampled pose and the particle's map. No adjustments need to be made to be fit for HMT maps. Only the features visible in the current map will determine the weight of the particle which means the features which are observed outside the local map will not influence the local accuracy.

Map update The map updater puts the observations in the map. Only the observations within the map boundaries are added to the map. As in the regular occupancy grid mapping algorithm, the observations of the grid are maintained as a running average and the grid will be regarded occupied when the average is above threshold.

Resampling Most of the improvements are made in the resampling step. Also the biggest innovation of this work is made here, the *forced resampling*. The parts of the resampling that are adjusted will be treated in subsection 3-1-1.

3-1-1 Adjustments to RBPF SLAM

Loop closure One of the purposes of a healthy particle spread is that there is at least one particle that can explain the measurement data at a loop closure point 2-5-2. This particle can explain the new observations with it's own map and is therefore able to close the loop. However, the longer distance traveled before closing a loop the more likely it is that the particle that can close the loop is deleted in resampling. This may happen because an other particle might locally have a high weight before the loop is completed, or because of the randomness in the resampling. Therefore in RBPF SLAM the number of particles should be proportional to the size of the loops that need to be closed. An exaggerated example of this situation can be viewed in Figure 3-1, where the particle in (b) sampled the angle a bit differently and did not get deleted yet. Because of this it is now locally correct just around the arrow head because it supplements its own map with measurements, therefore it will not be penalized. Since this is an exaggerated example this situation is not likely to occur, but the less particles there are, the more possible it is that a particle that can later explain loop closure is resampled into a locally high scoring particle which cannot explain a large loop.

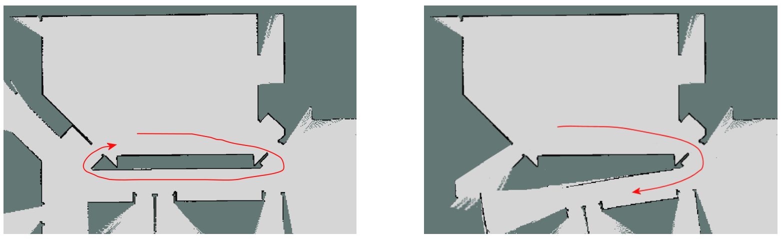

(a) correct path **(b)** incorrect path

Figure 3-1: Two maps, one where the particle has the correct path and can close the loop. The other particle had a biased resampling, halfway it can already be seen that the particle will not be able to close the loop.

In FoRMeT Mapping the loops are not closed by the particle's exact location in a global map, but by looking if there is a local map in the particles history that fits the observed environment. When the particle enters this map, it's pose in the new map is determined by the scan matching algorithm (section 3-3), and afterwards the motion sampling determines the particle's path further. The success of a loop closure is therefore determined not by the chance that one particle in the cloud has the right path, but rather by the match between current observations and the memorized maps of a particle.
Then what determines the amount of particles in HMT SLAM? The answer lays in the philosophy behind HMT mapping: "Local Accuracy in global uncertainty". The amount of particles in HMT SLAM should successfully represent the posterior distribution of pose uncertainty within the local map. FoRMeT is only concerned about the quality of the local map, so the amount of particles should be determined by the accuracy desired in the local map, and the size of the local map for local loop closures. It is hypothesized that since this is the only purpose of the HMT particle cloud, there are less particles needed in FoRMeT than in pure RBPF SLAM.

Forced resampling When we look again at the above mentioned loop closing problem another interesting observation can be made in which one of the major innovations of this framework can be explained. When one particle is locally right, it might not only cause resampling of another particle that can close the loop in the future, but it can also resample a particle that is right about the past. The fact that deletion of a particle that locally scores lower might cause a good map in its history to be thrown away is not in line with the HMT philosophy. The map size (section 3-2-1) is determined in such a way that the assumption of

conditional independence can be made between maps to be able to optimize map location later. It is hard to observe a feature of another map in the current map, then why should the weight of a particle in the current map determine what an area 3 maps back looks like? This HMT framework defines this problem as the *multiple stem* problem.
A healthy particle cloud should be resampled at strategic moments. The deleted particles then take over the history of healthy particles. When visualizing the paths in a particle cloud you can observe a tree like structure with a *stem*, and a *canopy*. See figure 3-2.

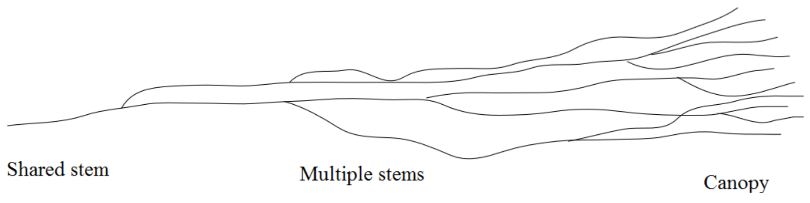


Figure 3-2: Simplified example of particle paths. Most left, all particles share the same path. Most right, all particles have their own pose.

To make sure that the particle spread represents the pose uncertainty in the current environment, and that resampling does not affect previous maps the resampling must be modified to force resampling of multiple stems. Here the stem is defined as the paths which are contained in maps other than the current map, and the shared stem is a single stem containing all the particles paths, see Figure 3-3. FoRMeT Mapping desires a long stem and a short canopy. Multiple stems are only allowed in the current map at the time of traversal i.e. the particles must agree on how they entered the current map when one of the particles wants to exit this map.

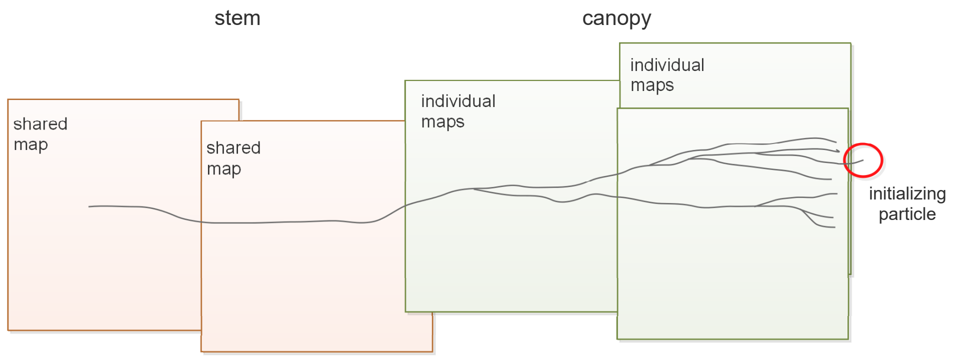


Figure 3-3: Simplified example of particle paths and their maps. Here the particle that initializes the forced resampling is circled in red

When the particle cloud is about to traverse to another map, the configuration of the particles path should be so that no multiple stems are present in the previous map. If there are, forced resampling should be applied. The forced resampling samples stems in stead of particles. The groups of particles sharing a stem in the previous map are identified, and only one group may survive. Forced resampling makes sure that normal resampling only affects the current and previous maps of the particles, since the other maps are shared by all the particles. Hence the forced resampling makes sure every particle has a maximum of two non-shared maps i.e. the current map and the previous map. When the particle exits a map, forced resampling takes place, they agree about how they entered the current map, the previous map is now fixed and can be regarded shared, the initiating particle traverses to the next map, now there are the new/current map and the new previous map which the particles can disagree on (Figure 3-4). The canopy can now be redefined as the maps that contain paths that differ per particle, and the stem can be defined as the paths in the shared maps.

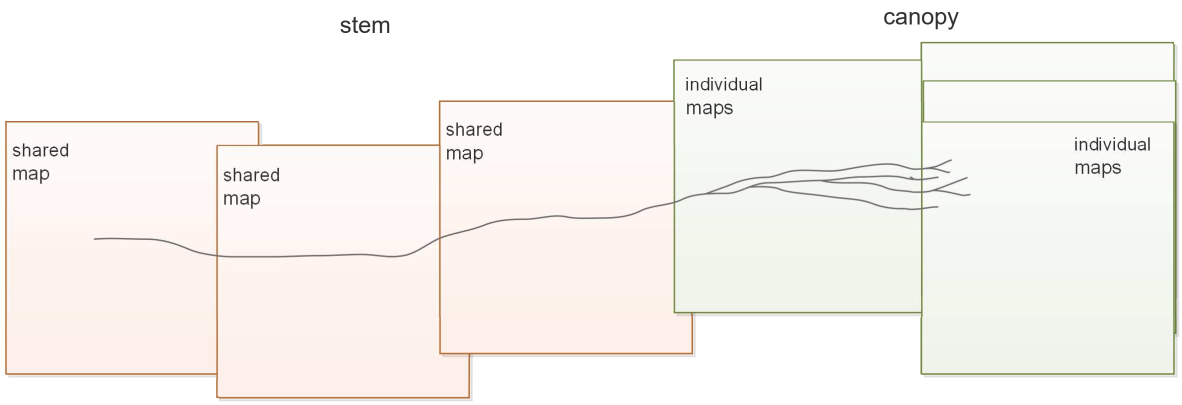


Figure 3-4: Simplified example of particle paths. One step after resampling took place at figure 3-3

Note that the individual maps in the figure may seem just like one, but there are as many maps as particles and per particle they look different because of the different path. The location of the map is determined by where the particle exits the previous map, if all particles have spawned from a single parent particle, all their maps are at the same location, but look different. The shared maps are also with as many as there are particles, all in the same place and they also all contain the same info. This seems like saving a lot of duplicates, this is true, which introduces the next topic: shared maps.

Shared maps Shared maps are maps which contain just one path hypothesis. These are thus the same maps for each particle. Since it would seem a waste of space to keep duplicate maps for each particle a shared map is defined as a single map, the network of shared maps is called the *shared topology*. The particles now only carry their current and previous map and the edge to the shared topology so it knows where all the other maps are to close loops with.
Whenever a particle can close a loop with a shared map this map will be added to the canopy. When multiple particles close this loop, multiple hypotheses about this maps layout exist. It can however only be updated or left the same, but not be deleted, since it is a part of the history of all particles. In fact, whenever a particle enters a shared map a copy of this shared map is added to the particles canopy and will be updated, and when this particle is in the group of particles defining the stem that is kept in forced resampling, the shared map is replaced with the updated map from the canopy.

Particle Ancestry Trees Like duplicate maps in FoRMeT Mapping , Fast SLAM also does not find it useful to save duplicates of records. The particle's path is defined as an ancestry

tree, where the tree points to recorded poses. Whenever a resampling takes place and the particle takes over the history of another particle, the particles tree is pointed to those poses in stead of copying them all. For a practical implementation like GMapping, it means not every particle carries a map, but they carry pointers to poses and laser scans. The map can be constructed from the ancestry tree and the map constructed using the poses in the path and laser scans.
GMapping saves memory because not every particle carries the global occupancy grid map, but carries pointers to the building blocks for map construction. In this framework there is no global map, only a network of shared maps, and two maps per particle in the canopy. It will thus be sufficient just to store these last two maps, these two maps can cheaply be copied to deleted particles and thus it will not be necessary to use an ancestry tree. Because the maps do not need to be reconstructed keeping memory of the total path is also not necessary. Only the current pose is necessary to update the weights and the map, and the paths of the current and previous map are needed to check for multiple stems. Because there is no need for ancestry trees and reconstruction algorithms this method is hypothesized to use less memory and computation than algorithms that do use these methods.

3-2 Map size and separation

Every particle has a *node* associated with it's current location, this is called the current node. This node carries the current local map of the particle. A node is nothing more than a place, however nodes have no global location, they are only constrained with respect to each other with edges. Edges also have no location, they are always defined between two nodes. Edges carry the normally distributed mean transformation $\mu_x, \mu_y, \mu_{theta}$ and covariance Σ between two nodes. This way, if you would set one of the nodes as being the fixed start node, all the other nodes can be expressed with respect to this node.
By defining the maps this way, in stead of a global map, every particle carries its own network of local maps. Whenever a particle travels outside of its local map, it creates a new node, with an empty local map, the particle's pose will be expressed in the new map, and will evolve further.

3-2-1 Map size

The map size will is a designer parameter. A smaller map will need less resources to be updated, but there will be a higher map density and thus more maps to match with and traversal will happen more often. The suggested rule of thumb based on the robots sensors is that the map size is half the maximum range of the observations. This way it is unlikely to be in the current map and observe features from two maps further. Another rule of thumb based on the environment is that the map size should be so large that there are no multiple separate ways of entering a map from one other map. This way the edges represent the overlap of two maps at one point.

3-2-2 Node traversal and map generation

When a particle's pose comes outside of a defined *map traversal threshold* and the particle's pose is directed outwards, a *node traversal* is considered. Traversal is defined as explaining the robot pose in another map which map be a new map, or a map that has already been created. In Figure 3-5 the green poses are eligible for traversal. The set of particle eligible for traversal is stored and will be used later by the matching algorithm (Section 3-3).

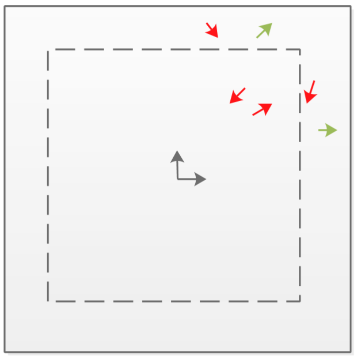


Figure 3-5: The current map with the *map traversal threshold* as a dashed line. Possible particle poses depicted as arrows in which the green poses will be considered for traversal and the red poses not.

3-3 Traversal candidates

When a particle is eligible for node traversal, a few things are done. First Dijkstra's tree (Appendix E) is created to know the shortest paths to each other map. The Dijkstra solution in combination with the edge covariances will form an uncertainty projection. This uncertainty projection represents the uncertainty of each node location as seen from the current node. If the particle is in the projected uncertainty region of one or more of the other nodes, these nodes are considered for traversal.

Topological loop closure Loop closure in global PF SLAM is not as an explicit action as it is often referred to. It is a mere confirmation for the particle if its pose hypothesis is right or not when considering the current observation and map. Actually every time the particle processes new measurements and calculates weights could be considered a loop closure of some sort because it calculates the weights based on correspondence between measurements and the map. In long loops the exact same thing happens, however now the mismatch of the particles is more obvious, and because of the resulting spread in weights the filter will be triggered to resample and most likely the wrong particles die out herewith "closing" the loop. In the HMT framework, the local maps are not big enough to facilitate large loops. The large loop closure that happens in the HMT framework is between maps and thus a *topological loop closure* closure. It happens by recognizing the particles pose and observations can be expressed in one of the particles other maps. By knowing where the other maps are approximately it can scan match to see if it can close a loop with another map. When this is the first time the robot traveled between these nodes an edge is created, the topological loop is closed. When the robot has traveled more often between these nodes the existing edges are updated (section 3-5).

Dijkstra's tree For knowing if there are possible loop closures, the robot must know where the other maps are with respect to its current location. Since the local maps do not have a global reference frame their location can only be calculated by combining all the edges from the current map to the target map. However, multiple combinations of edges can lead to the target, therefore the route taken to target map determines the location and uncertainty of the target map with respect to the current map. The best route to the target map will be the route which has the least uncertainty, then the resulting mean distance and covariate uncertainty will be the particle's best guess of the target node location, keeping the amount of overlapping uncertainties minimal. Since every edge carries a new uncertainty, typically the route with the least edges will be the best, but not necessarily. The route is determined with Dijkstra's algorithm (Appendix E).

Uncertainty projection To find the best route to every target map Dijkstra's algorithm is used. Here the best route to each other node from the source node is calculated by evaluating the costs of the edges. Since we want the route with the least uncertainty the cost of the edge will contain the combined covariance of the traversed edges (2-5-1). The algorithm will return three arrays, one with the nodes sorted by cost from lowest to highest, the next will be the parent node of that node, and at last the cost of traversing to that target vertex. This way the robot knows the cost for every node, and by going up in the parent node tree it knows the best route to this vertex. When the best path is known the uncertainty projection can be calculated as in Eq. (2-3). By combining the edges in the path an estimate of mean and covariance of the target node with respect to the source node can be determined for each target node. This is called the uncertainty projection. The robot can use the uncertainty projection to see with which nodes it can possibly close loops.

Traversal action With the particles current pose it is checked if it's within the uncertainty region of any of the other maps, if so a *scan matcher* will estimate the particle's pose in the other map and the score of this match. The scan matcher uses the current observation with the map of the eligible traversal node to find estimate the robot pose in the new map, and the likelihood score. For each of the possible traversal nodes a score is calculated, and the traversal action of the particle is sampled from this group of possible traversal nodes supplemented with a probability of entering an unexplored area. This way a new node will be generated if there are no possible traversal nodes, or if the unexplored area hypothesis is sampled. Also each particle can have it's own hypothesis about the area the robot has entered, if it is an already mapped area or a new one. It's expected that the wrong particles will be resampled. An example of the traversal action sampling is visualized in Figure 3-6. Note that the uncertainty is only defined to the edge transform, and to know if the particle is in another map, the map size should be added to the node uncertainty.

3-4 Traversal

When a traversal action is sampled the actual traversal can take place. Because the particle is traversing, a check for multiple stems is performed (3-1-1) and if there are multiple stems present forced resampling will be performed. Afterwards all particles should agree about the layout and location of the previous map and it can be transfered to the shared topology. The current map will become the previous map as the traversal map becomes the new current map, and the particle pose will be defined relative to this map frame as estimated by the matcher.

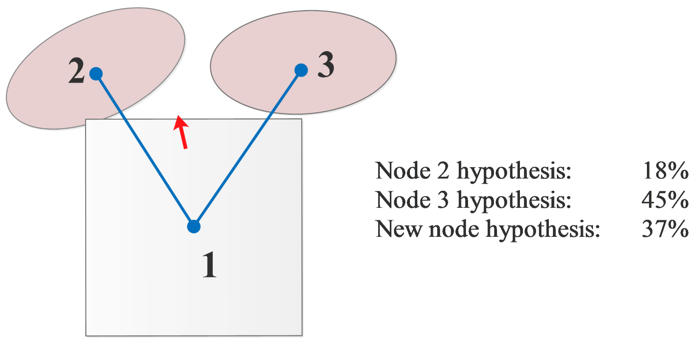


Figure 3-6: Particle in red, with different traversal hypothesis after matching with the candidates 2 and 3 from the uncertainty projection.

Edge covariance With an EKF based local mapping algorithm the uncertainty is captured in the posterior distribution, and when creating a new node from the particles pose the uncertainty of the transform between the two nodes can be expressed with this distribution. With particle filter based mapping this is a bit more complicated. Since only one particle will have the right path at node traversal and the edge transform will be defined by that particle in FoRMeT there is no covariance.
Even though there is no covariance, there are small discrepancies in the local map which may lead to some extra curvature between the maps. Because of this there is an uncertainty about how the maps are configured, otherwise the robot might not be able to close a loop. The uncertainty of the edge must thus represent the degree of discrepancy per map so that loop closure can be performed. When the uncertainty is taken big, the chance a loop can be closed becomes bigger, but because of big uncertainties in the projection the search space for loop closure becomes bigger, and more false nodes will become candidates. This will lead to larger computations and higher chance of false loop closures. When the uncertainty is taken too small, the particles might not recognize the right candidate because it is projected too far away. The edge transform covariance is thus a designer parameter depending on the accuracy of the local mapping system. It should be depending on the traveled path in a map, the odometry noise and the observation noise. It can be tested by seeing if loops can be closed.

3-5 Optimization

The uncertainty about how the graph looks is reduced in two ways in FoRMeT . First the edges are locally updated by a Kalman filter upon re-visitation. Secondly a global optimization algorithm is deployed to minimize the uncertainties in the global network.

3-5-1 Edge updating

Whenever a robot traverses between maps that already exist the matching provides a new estimate of the transform between the edges. The robot is at pose p^1 (homogeneous notation, see Appendix F) in map 1, and that point is matched as pose p^2 in map 2, this provides a constraint between the two maps, see Figure 3-7.
It is clear these two maps do not overlap correctly, as is the intention in this exaggerated

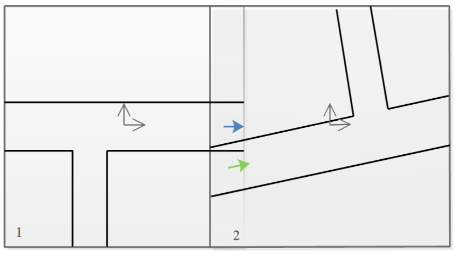


Figure 3-7: Traversal with the robot pose in map 1 in blue and the matched robot pose in map 2 in green

example. The *Edge Updater* will use the new constraint between the two maps to form a new estimate. p^1 and p^2 represent the same pose, but in different coordinate frames, the poses can be regarded as transforms from the map frame to the pose. The transforms will read as $T_1^{p^1}$ and $T_2^{p^2}$. The desired edge transform is T_1^2 and can be easily obtained with the inverted transform of $T_2^{p^2}$ as in (3-1).

$$T_1^2 = T_1^{p1} * T_{p2}^2 \tag{3-1}$$

With the new transform the maps are aligned as in Figure 3-8.

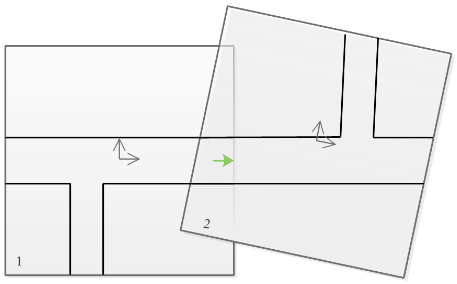


Figure 3-8: map 1 and map 2 constrained with transform from matching.

If there existed no edge between these maps, a new one will be created with the new transform and a loop is closed. If the edge existed already it must be updated with the new estimate. In the example it's clear that the old transform is a bad estimate of the map alignment, therefore the old edge will have a big uncertainty. The new alignment seems to fit perfect, but in reality some small matching errors and noise in the map will also make that the match transform $T_{1,match}^2$ has some uncertainty. The new estimate of the edge transform mean and covariance will be determined with the Kalman matrix as the weighted mean of the $T_{1,old}^2$ and $T_{1,match}^2$. More about the Kalman matrix can be found in appendix A.

3-5-2 Graph optimization

The edge updater only looks at a local fit at the point where the maps connect and calculates a linear solution. If the edges would be added up, the interconnected maps may still not all align, this will especially be true for larger networks with more loops. However this will not be a direct problem for the robot. The uncertainty projection will keep the inconsistencies as far away as possible from the robots current map, the robot does not need to know the exact distance to a target map, but just how to get from map to map, and that's where the updated edges work perfectly. However, it might cause complications when the graph becomes very large and interconnected. Identifying which maps are possibly nearby when the robot wants to traverse might become a large operation when the maps are projected with a lot of uncertainty. Look at the example situation in Figure 3-9. The map is not optimally explored and thus is the graph not optimally interconnected. The robot is approaching the edge of the current map, and uses the uncertainty projection to check which possible loop closures are available. For two target maps the uncertainty is drawn out, these are quite large uncertainties because they contain all the covariances between the nodes on their shortest paths (green). It can be seen that the robots pose is in the uncertainty of these two maps, and perhaps even more maps, and to find the best loop closure it should match with all nodes with overlapping uncertainty.

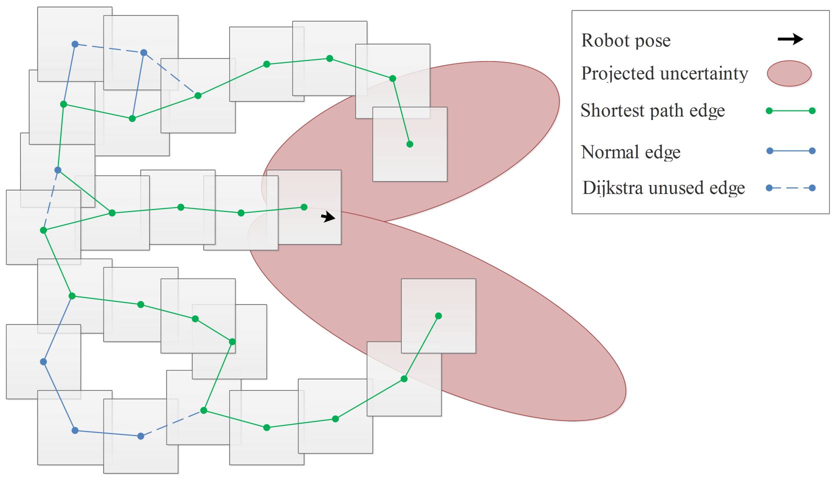


Figure 3-9: An unoptimized interconnected network of maps. The green lines are the shortest paths to the target maps and the dashed ones have no use in the Dijkstra algorithm because these nodes can be better reached with other edge combinations. For simplification no rotated maps are shown. A Dijkstra unused edge is an edge not present in Dijkstra's solution tree.

The graph is overdefined because of multiple loops. When there are multiple ways to go through the network there are loops present. If all the maps would be visualized by their hard transformations (without taking covariance into account) not everything would fit. Specifically in the case where the locations are projected over de Dijkstra solution, the misfit would be at the dashed lines in Figure 3-9. Thats why there is an uncertainty embedded in each edge. For such an overdefined network an optimization algorithm could determine an optimal fit satisfying as many constraints as possible by reducing a norm related to the covariances. The optimized network is dependent on the uncertainties between each edge, which are not visualized in the example, but the solution might look like Figure 3-10.
It can be seen that both of the target maps are no longer loop closure candidates because of the reduced uncertainties. The robot could now spend its resources matching with other more certain maps, or create a new map. Another example of pre and after optimization map layouts is visible in Figure 2-9. For more maps it becomes more important that the edges are optimized, since there will be more uncertainties that could overlap. For a large scale mapping and localization framework graph optimization is thus essential to make sure the computation time does not increase at larger networks. In this framework optimization

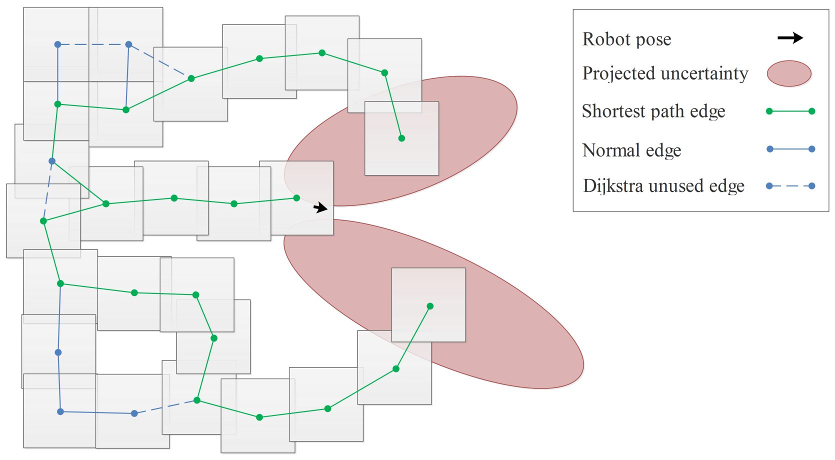


Figure 3-10: An optimized interconnected network of maps. The green lines are the shortest paths to the target maps and the dashed ones have no use in the Dijkstra algorithm because these nodes can be better reached with other edge combinations.

is performed each time an edge is created by loop closure, because only at that time there is a new over definition available. However, a specific type of optimization algorithm is not suggested by the framework, and can be chosen by the user.

Chapter 4

The Implemented FoRMeT Framework

The FoRMeT framework is a general framework for scalable mapping. The implementation used to test the framework is explained in this chapter. The implementations of the different parts are elaborated on in the following subsections. Any bold printed word is a tunable parameter, the parameters are grouped in section 4-9.

4-1 Sample from motion model

The poses of all the particles are initiated at $x_0 = [0, 0, 0]^T$. With a designer specified **sampling frequency** new odometry measurements are taken. The difference between every odometry measurement is used to update the particle poses. The particle poses should be sampled according to the prior probability distribution to accurately describe the pose uncertainty. To do this an approximate displacement model is used which can be viewed in Figure 4-1.

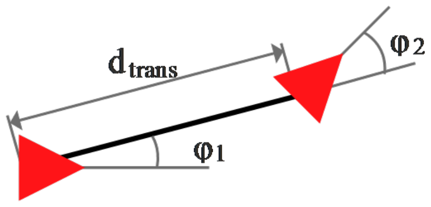


Figure 4-1: Model used to describe odometric evolution.

The transform between poses is expressed in a rotation ϕ_1, a translation d_{trans}, and a second rotation ϕ_2. When the distance between transform measurements is kept small enough, this model approaches reality.
To sample in the prior distribution, normally distributed noise terms are added to the rotations and translations. It is up to the designer to choose these **odometry noise** terms so that the sampled distribution appropriately describes the real odometric uncertainty.
The translations and rotations with added noise terms can now be added to the particle pose. With a higher sampling frequency the displacement model will come closer to the real odometry.

4-2 Filter update

The motion model is sampled with the specified sampling frequency until either a **linear or angular threshold** is reached. When this threshold is reached a filter update is executed, for each particle an observation correction is executed, the weights are determined, possibly resampling takes place, the map is updated and a node-traversal function will check if any topological action must be handled. First lets look at the weight update.

4-3 Weight update

The weight update determines each particle's weight according to the measurement model. In FoRMeT Mapping this means a score is calculated for each particle using the particle pose, laser measurements and local grid map. This score is calculated with the *likelihood field model*. The likelihood field model gives a measure of how well the laser scan fits in the map if projected from the particle pose, which acts as the measurement model in the Bayes filter $P(z_t|x^t, z^{t-1})$. This way the particle weight can be determined and the updated particle cloud will represent the posterior distribution.

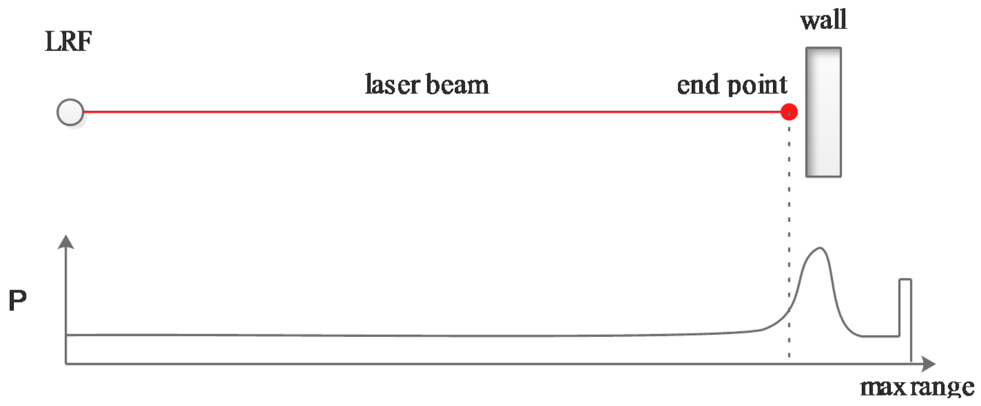


Figure 4-2: Likelihood field describing the measurement model.

The distribution of the likelihood field model looks like Figure 4-2. It consists of four parts, the probability of a random measurement as a uniform distribution, the probability of measuring something before the wall, the probability of a wall measurement as a normal distribution and the probability of a failed measurement which results in a max range reading. The endpoint of the depicted beam falls just before the wall due to a noisy measurement. The probability of this beam falls on the left side of the approximated normal distribution.
FoRMeT filters out all max range measurements and approximates the resulting distribution by a uniform distribution plus a normal distribution around the true hit point. From the laser range measurements a **number of beams** is selected to calculate the weight. More beams means a more accurate result, but more computation is required. In a search space called the **kernel** around every beam endpoint the algorithm searches for the closest match in the grid map. In Figure 4-3 (b) the closest match is depicted as a green dot.
The likelihood of the beam is determined by the distance between the beam end point and

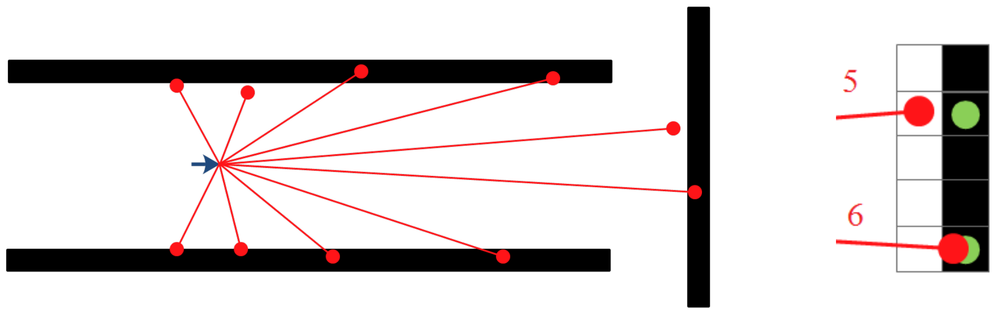


Figure 4-3: Grid map laser beam example for likelihood determination. (a) Beams used for weight calculation. (b) End points of beam 5 and 6 and their closest match in the grid map

matched point in the normal distribution. The normal distribution is approximated by the function in Eq. (4-1), where d represents the distance from the hitpoint to the center of the closest matched grid cell and σ represents the standard deviation. The weight of the particle at this filter update is the mean likelihood of all treated beams. In the example beam 6 will have a higher likelihood than beam 5, the maximum likelihood is 1 when the hit point matches exactly. And when all laser points would align with the map perfectly the weight of the particle will be 1.

$$s = e^{-\frac{d^2}{\sigma}} \tag{4-1}$$

4-3-1 Pose optimization

The framework prescribes that a correction from the motion model is applied to the particles using the current observation and a key frame observation (3-1). To do this the Iterative Closest Point (ICP) algorithm is the standard. More about the ICP algorithm in Appendix G. However without proper pre filtering of the beams, the usage of ICP to compare two laser scans will not have an outcome that improves the odometry. For this implementation the choice is made to compare the current laser scan with the local map per particle. In the particle weighing function the hit points and corresponding matched points are stored in an array and afterwards fed to the ICP algorithm. The algorithm returns an optimized pose where the laser points overlap better with the map. This implementation of FoRMeT Mapping will thus not have an improved pose estimate when there are no features present in the map. The used ICP algorithm comes from the Mobile Robot Programming Toolkit (MRPT) and is open source available[1].

4-4 Map update

The map is updated with the grid mapping algorithm. Each grid cell keeps two values, the ratio of occupied observations α, and the number of observations n. A grid cell corresponding to the endpoint of a beam will be supplemented with one occupied observation and each cell that the beam goes through will be supplemented with a free

[1]MRPT's ICP is available at http://www.mrpt.org/

observation. The equations to determine α are given in Eq. (4-2) and Eq. (4-3), with the latter one being the used recursive formula.

$$\alpha = n_{occ}/n \tag{4-2}$$

$$\alpha_k = \frac{(n-1)*\alpha_{k-1} + obs_k}{n} \tag{4-3}$$

The grid map will contain α values between 0 and 1. At visualization a threshold value is set at which a cell is regarded occupied. An example is given for a grid map updated with two observations in Figure 4-4.

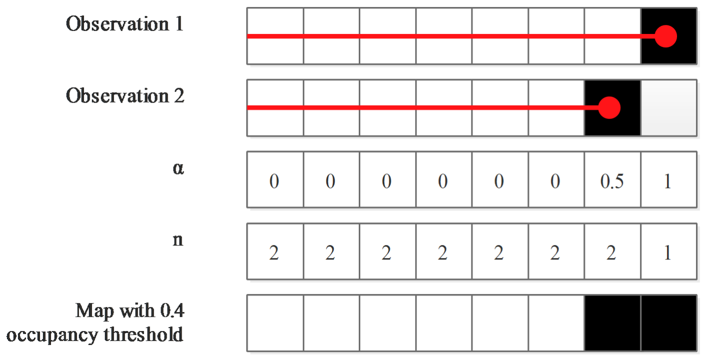


Figure 4-4: An example of grid cells updated with two observations and visualized with an occupancy threshold of 0.4.

When a laser beam is at maximum distance, it means this beam did not return to the laser from a hit object. The beams with maximum distance are filtered out. Also the grid cells further a **max usable range** of the laser beam will not be updated. This because the endpoints further away will have more variance and are thus less accurate. Also filling the grid map only up to a certain range will reduce computing power.

4-5 Resampling

Every filter update the *number of effective particles* (N_{eff}) is determined. If the N_{eff} is under a **resample threshold**, resampling will take place. The N_{eff} is a measure of the amount of particles that have an effect on the map. It becomes lower when the variance of the particle weights becomes bigger. Herewith it is an interpretation of how good the particles represent the target distribution. This is a common way to determine the resampling moment in PF-SLAM algorithms.

$$N_{eff} = \frac{1}{\sum_{k=1}^{n}(w_k^2)} \tag{4-4}$$

When resampling takes place a cumulative table of weights is created using the weight of each particle, starting at zero and ending at the sum of all weights. This table is then normalized and for each particle a random number between zero and one is generated to draw from the table. The new set of particles is then filled with the drawn particles. The probability of a particle to be resampled is thus proportional to its weight, thus it is likely

that low scoring particles will not be resampled. This way a healthy particle cloud is maintained.
For a large set of particles and a large map size the resampling step will be very expensive. Fortunately FoRMeT only keeps the current and the previous map in the particle's memory and therefore the amount of particles can be kept small and during resampling only the latest two maps have to be copied.

4-6 Forced resampling

This framework claims to be lightweight because only the current and previous nodes, and thus local maps, are stored in the particles memory. All the other nodes and edges are shared in the shared topology. To make sure the particle's agree on the maps in the shared topology forced resampling is implemented. When one of the particles leaves its local map, all the particles should agree on how they entered this map. This way there can only be different hypotheses over two maps.
Whenever a particle is traversing first the stems in the previous map are analyzed. This will result in a group of particles in each stem. Only the most healthy stem will survive, so all the particles in the other stems are sampled into particles of the most healthy stem. The forced resampling has now taken place and if there are still particles that want to traverse, the node traversal steps can take place.

4-7 Node traversal

The node traversal function connects the local PF-SLAM to the topological part. Every filter update it is checked for each particle if the pose is outside the **map generation threshold**, and if so the particle will either generate a new map or traverse to a known map. Because it is desirable to find the right match when it is available this implementation gives each particle more opportunity to match by introducing a **map traversal threshold**. The matching threshold is slightly smaller than the traversal threshold, and when the particle crosses this threshold it will already check for a match, but if it is not found, it will stay in the current map. Now there are three traversal states a particle can be in before the node traversal function:

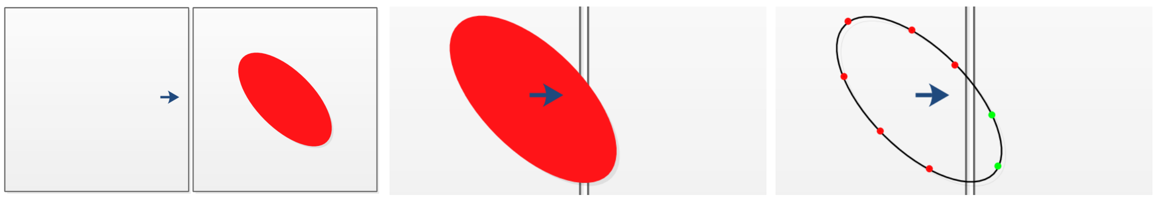
1. no traversal
2. only matching
3. matching or new node generation

When a particle is in the second or third state the node traversal will perform the following steps:

- Selection of traversal candidates
- Sampling of match points
- Matching of each match point
- Sampling of traversal action
- Execution of traversal action

4-7-1 Selection of traversal candidates

For each map it is checked if the particle is in the uncertainty area of that map Figure 4-5 (a). This implementation only regards the x and y components of the covariance for selection of candidates and sample points. Then checking if the particle is in the uncertainty area becomes equivalent to checking if this uncertainty area around the particle overlaps with the map Figure 4-5 (b). To check this samples are drawn on the edge of the uncertainty ellipse Figure 4-5 (c), if the samples are within the map area, this map is a candidate, otherwise not.



(a) a potential candidate **(b)** uncertainty from particle **(c)** ellipse edge samples

Figure 4-5: Determination of candidate map. This map becomes a candidate because of the green sample points falling inside the map square

4-7-2 Sampling of match points

If the considered map is a traversal candidate, the particle can match the current laser scan in the uncertainty area in the candidate map. For the map matching ICP is used (Appendix G), however since ICP is a linear solution it could converge to a local minimum. Also if the uncertainty region is really big the ICP may not find the solution if it is too far away. Therefore multiple sample points are drawn in the overlapping uncertainty area Figure 4-6. The samples are drawn with a specified **sample distance**, this is a designer parameter depending on the available computing power and the reach of the used ICP algorithm. The match point sampling will result for each particle in a list of traversal candidates with associated sampled match points.

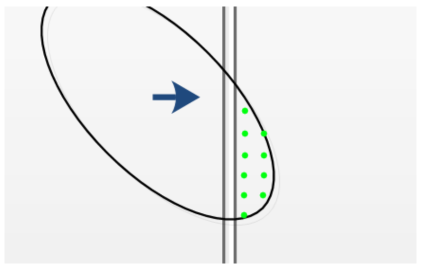


Figure 4-6: Samples drawn in the matchable area

4-7-3 Matching of match points

From each match point the likelihood field model is used in a very similar way to section 4-3. Only now more beams and a larger kernel are used to find the grid cell match of the laser hit point. On the found point pairs ICP is executed to find the best pose around this match point. Then the likelihood field model is applied again, and again ICP is executed on the new matched points. This process is iterated until a maximum amount of iterations, or convergence. Every map is only allowed to have one best matched pose. The score of this pose determines the probability that this candidate map will be sampled in the traversal action sampling.

4-7-4 Sampling of traversal action

Every particle has a list of candidate maps with scores between 0 and 1. To the list of candidate maps a probability of no traversal is added. The traversal action is sampled the same way as the sampling of particles during the resampling in section 4-5. E.g. if there is only one candidate map with a score of 0.8, there is a probability of 0.8 the particle will traverse to this map at the matched pose, and there is a probability of 0.2 that the particle will stay in its map when it has state 2 or that it will generate a new map when it is in state 3.

4-7-5 Execution of traversal action

When the traversal action is known, the particle can traverse. This means its previous node will be transfered to the shared topology, the current node becomes the previous node and the new node will be either taken from the shared topology or a new map will be generated. When the particle traverses to a known map the matching has defined a new edge as explained in section 3-5-1. If there was already an edge present this edge will be updated as a covariance weighted mean using the Kalman matrix. When the edge did not exist yet a loop is closed. The new edge will get a standard covariance, this covariance can be scaled with the **covariance scalar**. This way the uncertainty of the nodes can be set to be bigger or smaller. Bigger uncertainty has more chance to close the loop, but more samples will be drawn in the uncertainty making it computationally more expensive. The covariance scalar should be related to the accuracy of the robots mapping and describe the real node transform uncertainty for the best working algorithm (3-4). After the traversal action, the uncertainty projection must be recalculated for this particle to identify where all the nodes are, and what their uncertainty is with respect to the new current node. This uncertainty projection will be used the next time this particle goes out of the match or traversal threshold.

4-8 optimization

This implementation does not use non linear graph optimization.

4-9 parameters

Many parameters are present in this implementation. They can be varied to increase mapping capabilities with different map size, sensor specifications, processing power and environmental characteristics. Before using this implementation the parameters must be tuned. Below is a list of the important parameters in Table 4-1.

Table 4-1: Parameters in the implementation of FoRMeT Mapping

number of particles	**max usable range**
map size	**linear/angular update threshold**
map resolution	**resample threshold**
map generation threshold	**kernel size**
map traversal threshol	**covariance scalar**
odometry noise	**sample distance**
sample frequency	**number of beams**

Some of the parameters have not yet been discussed in the text.

The **number of particles** influences the accuracy of the localization. With a larger posterior distribution more particles are desired to accurately fill the uncertainty. However most operations are executed per particle, and therefore the amount of particles has a linear effect on the computational complexity.

The **map size** determines the size of the local area that needs to be accurately mapped by the particles. A larger map size will increase computation time because more grid cells need to be updated and also a larger set of particles is necessary to accurately map the local environment. However, when the local map becomes too small there will be a higher map density. The expensive steps in FoRMeT Mapping are the traversal actions, and they will more frequently occur and need to take more candidate maps into account when there is a higher map density. The advised map size lies between 10 and 20 meters. It is also dependent on the size of the rooms in the environment, it is desirable that there are not two separate ways to traverse from one map to another so that the edge will describe only one path between maps.

The **map resolution** determines the amount of grid cells per meter. The more grid cells, the more information can be stored in the grid map. Since the framework performs 2D mapping there is a squared relationship between the map resolution parameter and the computational complexity. Also, the grid size work as a filter on the laser range data. When the grid size is smaller than the noise on the laser beams the noise will be visible in the grid map. For an example of the noise on laser data see Figure 4-7.

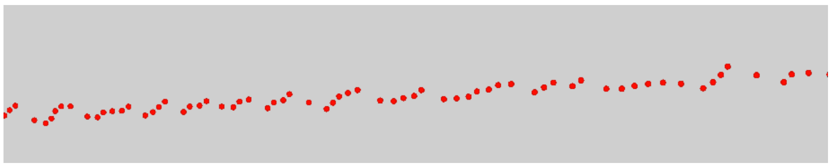


Figure 4-7: Laser endpoints falling on a straight wall. A scan taken from the IDE dataset (5-2-1).

Chapter 5

Experiments

In this chapter the implemented framework is tested against the following hypotheses:

1. Forced resampling in FoRMeT Mapping reduces computational load while maintaining a good representation of the environment.
2. By maintaining the uncertainty in the edges instead of in the particle spread FoRMeT Mapping is able to close large loops.
3. FoRMeT Mapping is capable of mapping large scale environments

Since the framework is so specific and different from other open source available mapping algorithms, the direct comparison with other particle filter based SLAM algorithms like GMapping will not say much. Both because GMapping is a global mapping system, and because this framework is not an extension of GMapping, but based on a particle filter SLAM that is slightly adjusted and solely created for the purpose of proving this thesis. Also GMapping has been designed to be efficient and is a maintained open source piece of code, and this implementation of FoRMeT Mapping is a mere proof of concept. Therefore the tests will be performed as comparison between the proposed framework and both GMapping and FoRMeT with one local map spanning the space of the environment i.e. no new map will be generated. This way a comparison between GMapping and the adjusted PF-SLAM can be made, and from there the comparison between adjusted PF-SLAM to the HMT system.

5-1 Experimental setup

For real world dataset gathering the SAM1 robot is used (Figure 5-1). SAM1 can be driven manually with a remote control. The robot is equipped with wheel encoders and a Hokuyo laser range finder (LRF). The LRF has a maximum range of 30 meters, it takes 1080 measurements from a 270 degree view every 25 ms with an accuracy of $\pm$ 30mm. The wheel rotations are converted into transforms between subsequent measurements and together with the range scans they are recorded into a ROS bag file. SAM1's odometry not regarded as very accurate, but the laser range finder is. The bag file thus contains LRF measurements and transforms to represent the environment and odometry measurements.

They will be played in ROS on the "HP ZBook 15" notebook with a Intel 2.4 GHz Core i7-4700MQ CPU and 4GB RAM. The processing into a map with GMapping and FoRMeT Mapping will also be done in ROS on the notebook. ROS is used as communication between sensors, the framework and visualization tools. ROS bag files are used to record sensor data and to play it back.

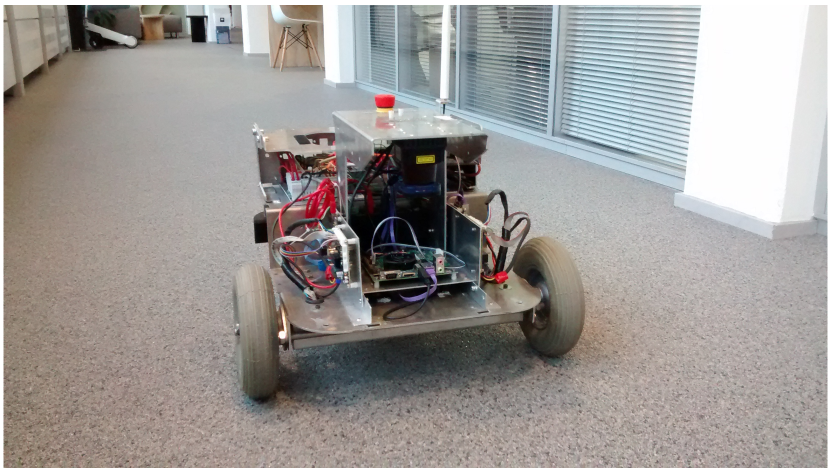


Figure 5-1: The SAM1 robot.

5-2 Experimental design

5-2-1 Datasets

The environments used for the experiments should contain the elements necessary to test the mentioned hypotheses. They should be large and contain a big loop closure. RSS's SAM1 robot has recorded a dataset at the Delft University of Technology where it drives at the Industrial Design (IDE) faculty's first floor to close a large loop (Figure 5-2).

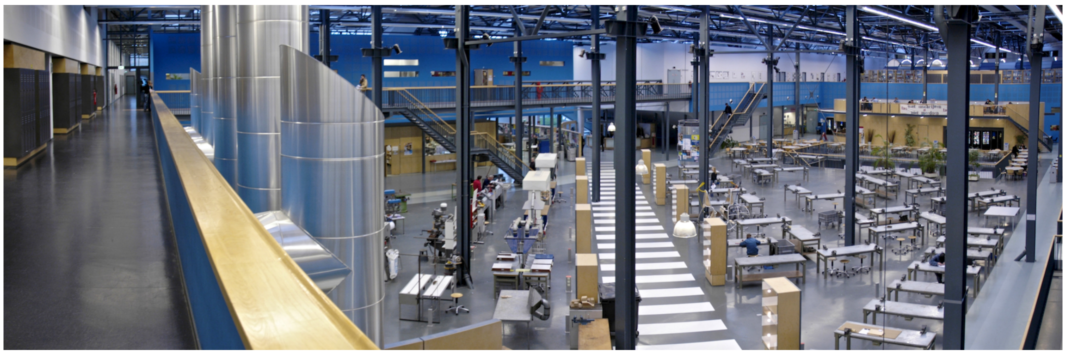


Figure 5-2: The first floor of the IDE faculty at the TU Delft looking down at the central square at the ground floor. [5]

In the floor plan view in Figure 5-3 it can be seen that there are two loops, a small and a big loop. The approximate distances are stated next to the floor plan, the simplified traveled path for loop 1 is visualized in green, the continued path for loop 2 is shown in blue

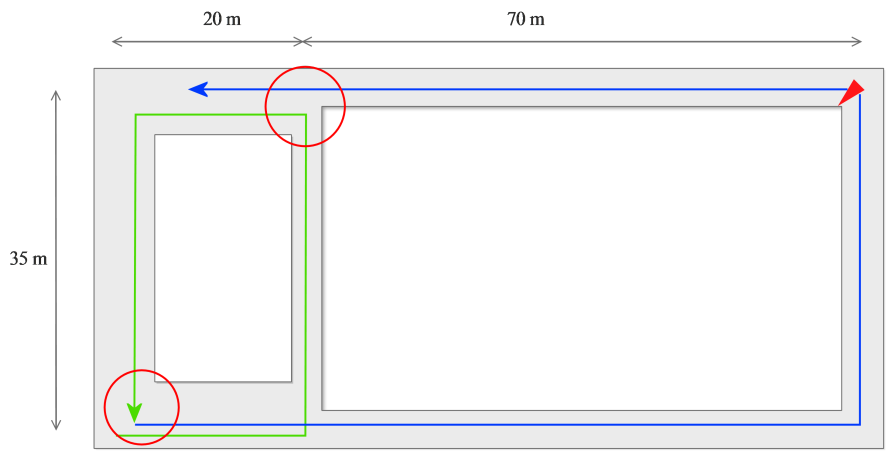


Figure 5-3: A simplified floor plan of the dataset area at the first floor of IDE. The gray areas represent the hallways. A trial indicates where Figure 5-2 is taken. A green arrow indicates the path to close the first loop and the blue arrow indicates the continued path to close the second loop.

and the loop closure points are circled in red. The green path is about 110 meters long, and the combined green and blue path 300 meters. The robot only drives forward and does not use mapping strategies like looking back from time to time. The IDE dataset is used to prove that the algorithm works on real world data, and to show the loop closure capacities. Besides the real world data the algorithms are also tested in simulation. A dataset is recorded in a Gazebo simulation where the environment represents a combination of three "Willow Garage" worlds visible in Figure 5-4. The simulated dataset is used to determine the scalability of the algorithms. The environment is large, but the dataset does not contain large loops.

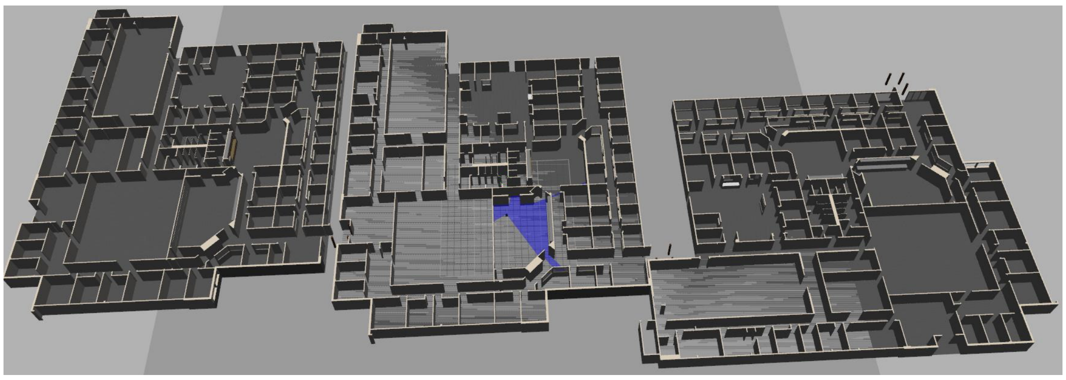


Figure 5-4: The simulated environment with three Willow Garage worlds

5-2-2 Experiments

During the experiments the performance of the algorithms will be quantified with the characteristics of the notebook's processor. Two performance measures are recorded: CPU usage and CPU time. Throughout the experiments, samples are being taken from the CPU which are stored for later processing. The CPU usage is the percentage of processing power that the mapping algorithm uses on one core of the CPU at the moment of the sample. Because the CPU has multiple cores, the notebook can remain functional even when the process takes up 100% of one core. The CPU time is defined as the time that the algorithm has used on the core in CPU operations up to the sample point. Besides these performance metrics, success and failure during loop closure and general map building will be monitored. Three experiments are performed:

1. GMapping vs global and hybrid FoRMeT at the small loop at IDE
2. GMapping vs FoRMeT at the combined small and big loop at IDE
3. GMapping vs FoRMeT in the large scale simulated environment

Experiment 1 will be used to see the relations between the two pure metric mapping methods and the metric and hybrid variant of FoRMeT Mapping . Experiment 2 will be used to see the performance of GMapping when closing the large loop and if FoRMeT is able to close the large loop. In experiment 3 The computational complexity will be regarded with respect to the scalability.

5-2-3 Comparison

The performance of GMapping and FoRMeT Mapping is heavily dependent on the parameters, and the experimental setup. To compare the two algorithms some of the parameters have to be set to similar values. These parameters will be *map resolution*, *odometry noise*, *max usable range*, *linear and angular update*, *kernel size*. For further explanation of these parameters the reader is referred to section 4-9. GMapping's parameters will be tuned in each experiment to get the best loop closure performance with the lowest computation power. FoRMeT 's parameters will be tuned to the most difficult loop closure experiment, experiment 2, and kept constant for comparability to the other experiments.
Both algorithms do a filter update after the set linear and angular threshold, however, when the previous update is not finished yet, the update will be performed when the previous update is finished. The filter update point is thus at or after the distance threshold. Therefore, when the set parameters require heavy calculations, or when the robot speed is too fast the map quality will decrease. Tuning the parameters becomes a delicate job when mapping real time because usually using more resources would increase map quality but now it can reduce the amount of filter updates per distance and thus reduce map quality. Also creation and visualization of the map is a costly operation. FoRMeT creates and visualizes the maps every update, GMapping has a standard visualization rate of 5 seconds. To compare the two algorithms, GMapping's visualization rate is set to 1 second which approximates the update rate of FoRMeT with the speed of the robot.

5-3 Experiment 1: IDE small loop

The small loop at IDE will be mapped by GMapping, global FoRMeT and hybrid FoRMeT Mapping . First GMapping will be regarded with standard parameters. Then it's parameters will be tuned and afterwards it will be compared to global and hybrid FoRMeT . A projection of points from the pure odometry path is included in Figure 5-5 to get an idea of the correction that needs to be performed by the algorithms.

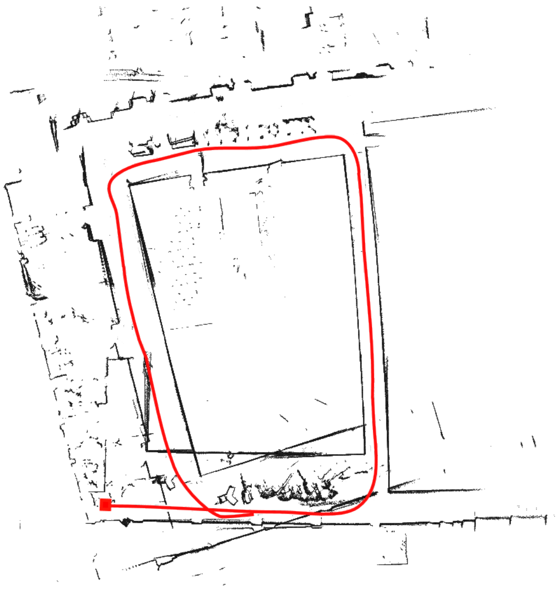


Figure 5-5: A raw point map of the small loop at IDE. Constructed from laser point projection from pure odometry. The trajectory is shown in red with a square at the start point.

5-3-1 GMapping with standard parameters

The map of a run using GMapping with standard parameters which has a map update every 5 seconds is shown in Figure 5-6.

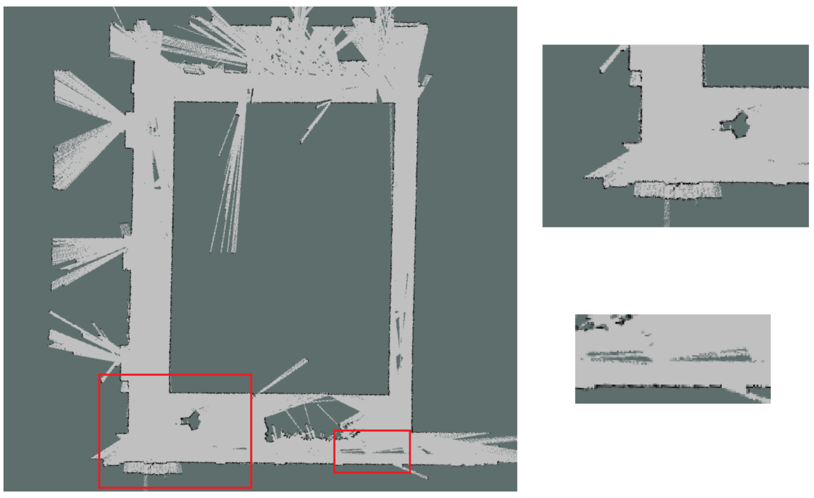


Figure 5-6: A map of the small loop at IDE created by GMapping with standard parameters. Red rectangles indicate the areas of the close-ups which are visualized on the right.

It can be seen that at the loop closure point (in the red rectangle on the left) there is a discrepancy in the map. The best particle thinks it approaches the loop closure with one meter offset compared to the robot in the real world. This is why the observation of the wall is about one meter wrong and a double wall is seen in the image. Afterwards the particles match with the map and their poses are corrected so their observations will fit the map, and they converge to the right location. GMapping can use scan matching to recover from, but not correct a small discrepancy in a loop closure. However, because of the discrepancy at the loop closure, this closure will not be regarded as successful. Also in the second red rectangle a some unobserved grids are present in the middle of the hallway. The length of the unobserved area indicates the space between filter updates in the algorithm. These updates are larger than the 0.5 meter update threshold, indicating that the filter update point is determined by the time it takes to do the previous update. GMapping is thus doing heavy computations which is also visible in the performance results in Figure 5-7. GMapping appears to use about 100% of the processor core until a filter update is finished, then there is a dip in processor use. The total processor time used to map the small loop is 238 seconds.

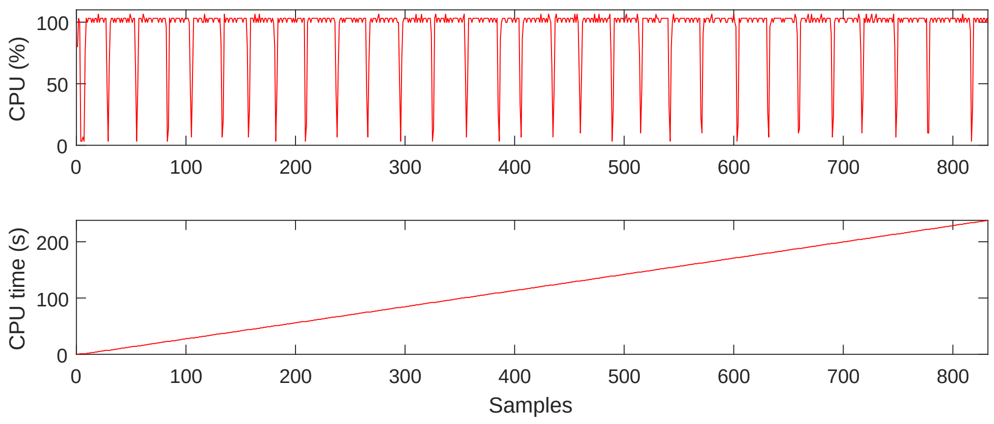


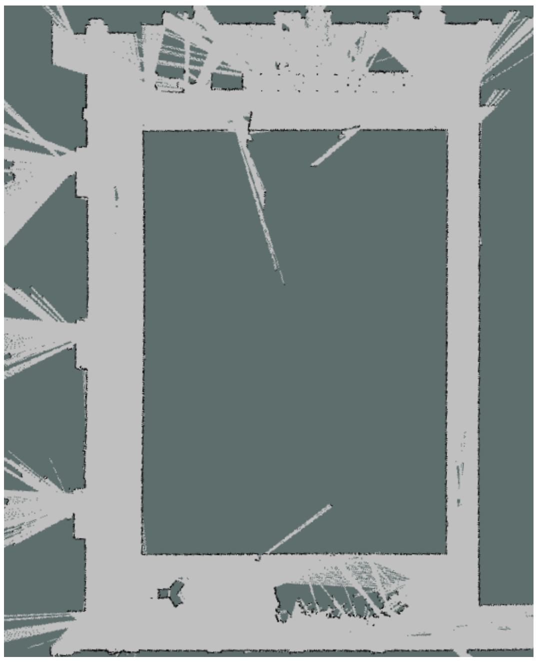
Figure 5-7: performance results of GMapping with standard parameters

5-3-2 GMapping with tuned parameters

With the current setup GMapping was not able to close the small loop without discrepancies regardless of parameter tuning. Changing the parameters to more accurate mapping using more computation would lead to less filter updates and eventually a worse map. The sweet spot could not be found with this setup. To close the loop it was necessary to play back the dataset at half the rate that it was recorded in, thus allowing GMapping to use twice as much time to process the odometry and laser scans. Now the parameters could be tuned again. It was even necessary to lower the rate of map updates to come to robust loop closure with minor discrepancies. A run where the result approaches a perfect loop closure is depicted in Figure 5-8.

5-3-3 FoRMeT Mapping

FoRMeT Mapping has been tuned for experiment 2 (5-4-2). Using the settings for robust HMT loop closure, the small loop was mapped by global FoRMeT with a map size as big as the small loop size so no new map generation takes place. The resulting map can be viewed in Figure 5-8. The resulting map of hybrid FoRMeT can be viewed in Figure 5-15 in the next section as the 6 maps on the left representing the small loop.


(a)

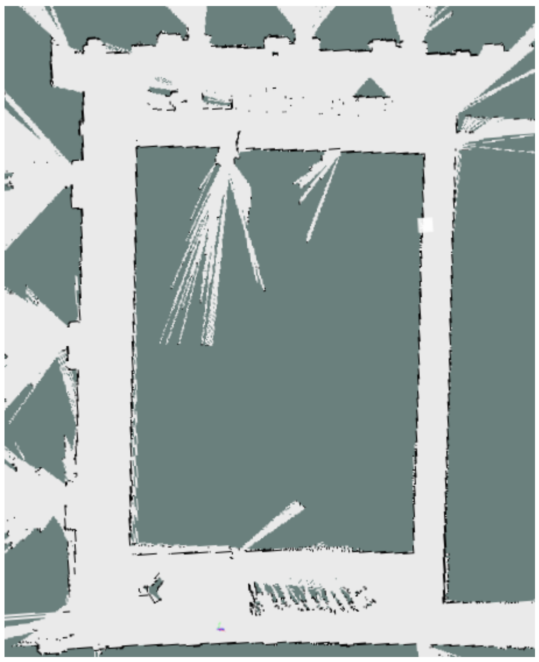

(b)

Figure 5-8: Maps created at the IDE small loop. (a) GMapping with tuned parameters. (b) global FoRMeT Mapping tuned for hybrid performance.

It can be seen that global FoRMeT is not able to close the loop without some discrepancies, and this is a visualization of one of the better runs. However the local parts in the map are locally accurate and the algorithm is thus fit for HMT mapping. More interesting than the visual results are the performance results visible in Figure 5-9.
GMapping uses about 100% of the core until it is finished with a filter update, then there is a brief dip. Global FoRMeT has uses about 75% of the core and briefly goes to 100% every time a resampling takes place. At the end more resampling takes place because the particles cannot explain the map at after the loop has not been closed correctly. Recall that resampling in FoRMeT copies the entire map to the sampled particles and is thus not meant for global mapping. Also note that GMapping uses a dataset played at half rate and half as many samples of the CPU are taken. This combined with a CPU usage about 1.5 times larger than global FoRMeT explains a cumulative CPU time which is more than two times as long. GMapping uses 442 seconds of CPU time, global FoRMeT 181 seconds and hybrid FoRMeT 104 seconds.

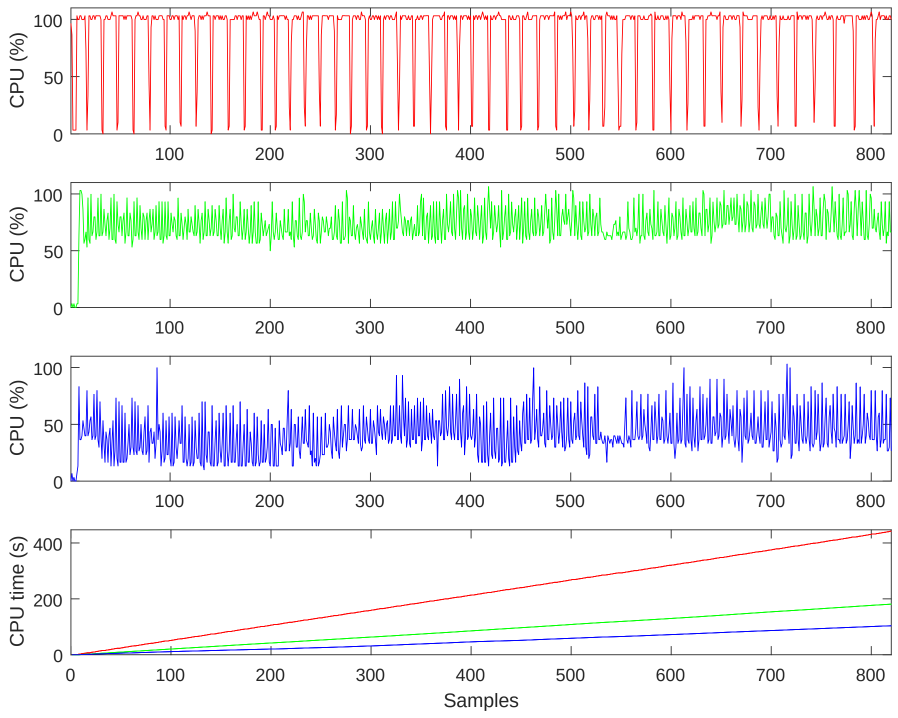


Figure 5-9: Performance plot of experiment 1. GMapping in red, global FoRMeT in green and hybrid FoRMeT Mapping in blue

5-4 Experiment 2: IDE large loop

In this experiment the whole path is traveled, first the small loop and then the big loop. A projection of laser points from the pure odometry path is included in Figure 5-10 to get an idea of the correction that needs to be performed by the algorithms. It can be seen that especially at turns the odometry has a deviation to the right.

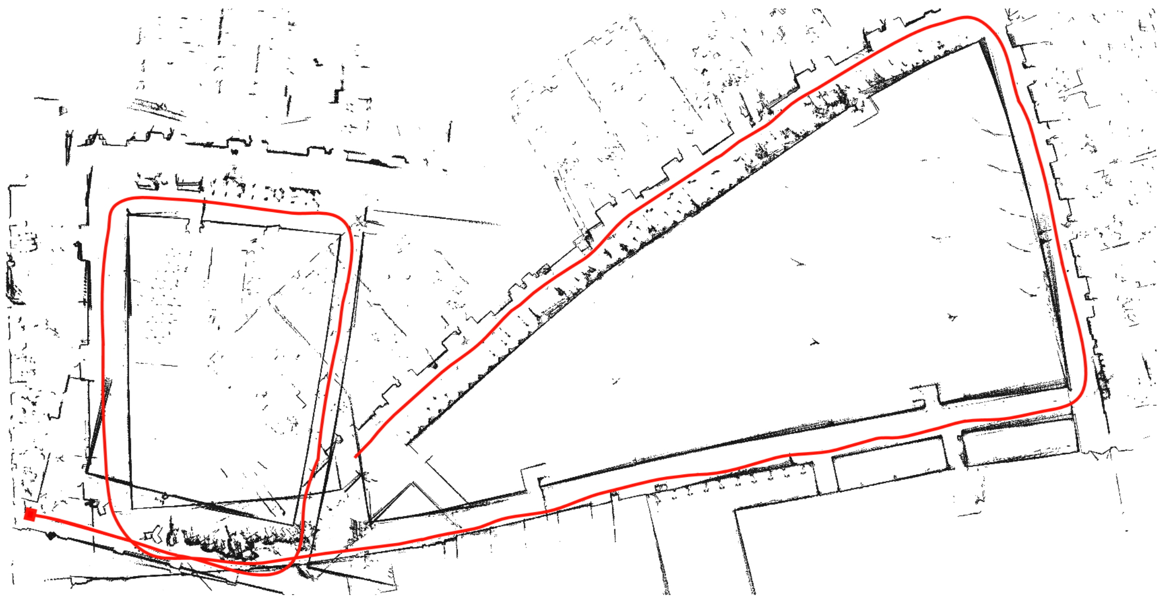


Figure 5-10: A raw point map of the small and large loop at IDE. Constructed from laser point projection from pure odometry. The trajectory is shown in red with a square at the start point.

5-4-1 GMapping

First GMapping is tested. The resulting maps are very straight, but the accumulated small errors make it very hard for GMapping to close the loops at IDE. GMapping is tuned with parameters yielding results closest to good loop closure. However, when doing five trials with these settings, GMapping could only close the large loop once, and not without discrepancies. A typical run is visualized in Figure 5-11, and close ups of the two loop closure points are shown in Figure 5-12. There are some minor discrepancies at the first loop closure in (a), and at the second loop closure (b) GMapping is about 1.5 meters off. The rest of the map looks straight without significant discrepancies.
The map of the run that successfully closed the loop is visualized in Figure 5-13. The first loop closure, visible in Figure 5-14 (a), is near perfect. Hardly any discrepancies are present. At the second loop closure Figure 5-14 (b), at first there is a small offset, but it is small enough for GMapping to recover to the right pose in the map. Only the two double wall artifacts are left in the map.
The CPU performance of the test with successful loop closure is shown in Figure 5-18 together with the results of FoRMeT Mapping . It is very similar to the performance in experiment 1 with a processor usage of about 100% until each filter update is over. The total time spend by the core is 1162 seconds. Since there are many samples the performance cannot easily be observed in the plot. Therefore a close up of samples 200-400 at the

beginning, and a close up of the 200 samples around the large loop closure are shown in respectively Figure 5-19 and Figure 5-20. It can be seen that GMapping can do a filter update approximately every 17 samples in the beginning and every 21 samples at the end. This indicates that GMapping needs more time to process a filter update when the mapped environment becomes larger.

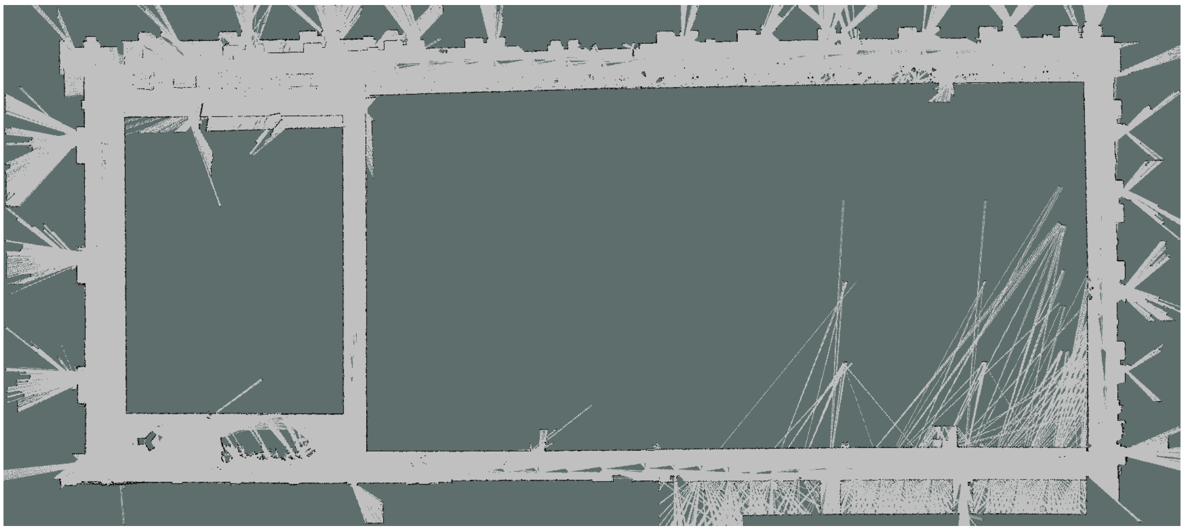
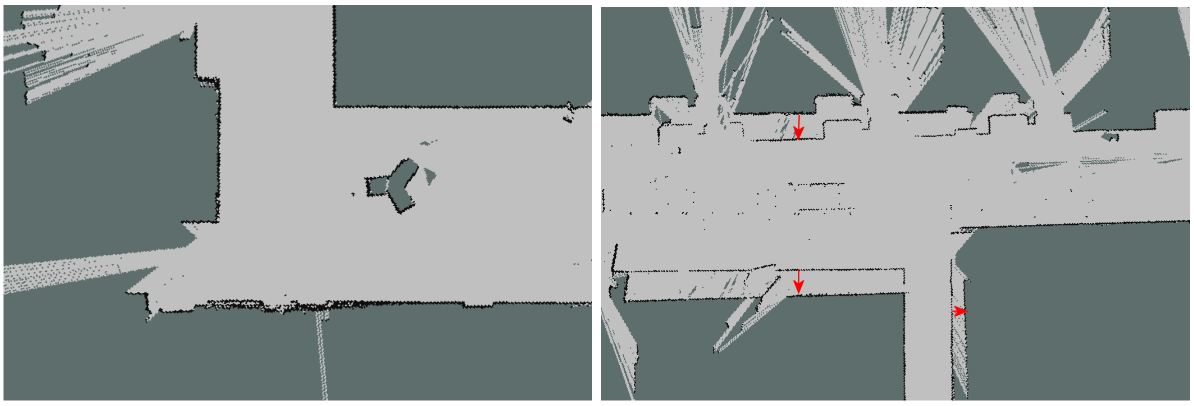


Figure 5-11: GMapping in experiment 2, unsuccessful loop closure.



(a) closure point 1 **(b)** closure point 2

Figure 5-12: Close ups of Figure 5-11 loop closure points. Red arrows indicate misalignments.

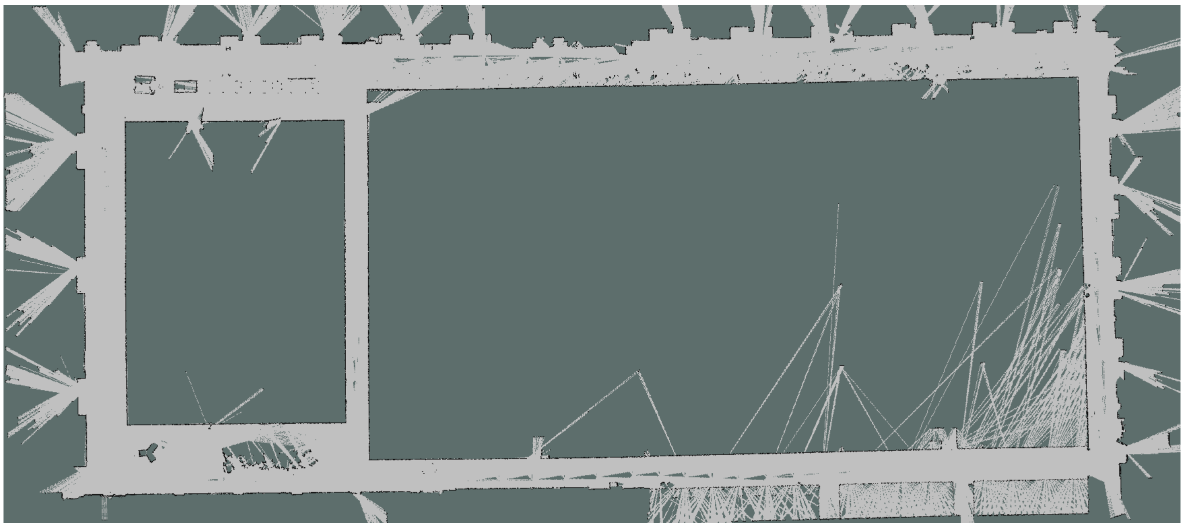
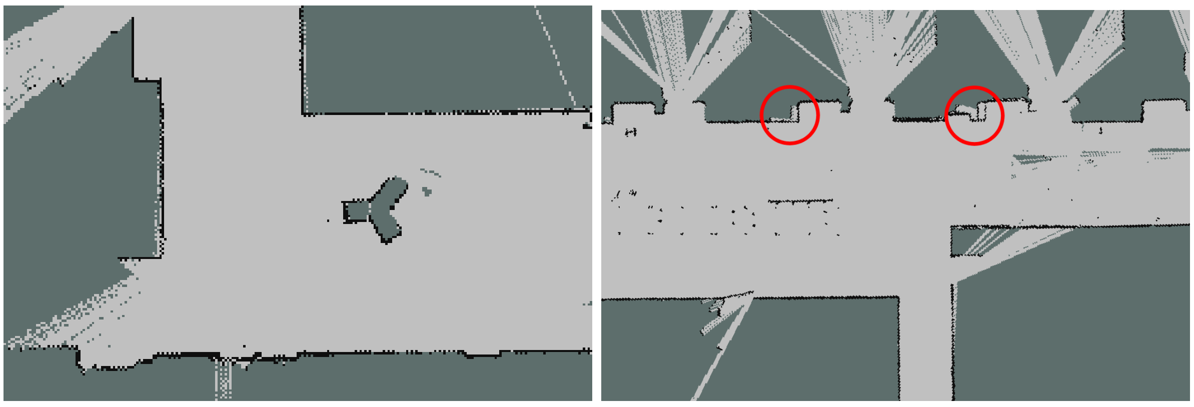


Figure 5-13: GMapping in experiment 2, successful loop closure



(a) closure point 1 **(b)** closure point 2

Figure 5-14: Close ups of Figure 5-13 loop closure points. Discrepancies circled in red.

5-4-2 FoRMeT Mapping

Also for FoRMeT Mapping it was a hard loop to close. It was not uncommon that the large loop was closed correctly, but it took a lot of parameter tuning to come to a robust loop closure with five out of five trials with correct closure. It turned out the algorithm needed to lower the odometry sample rate and use more beams for pose optimization. This can be related to the fact that SAM's odometry is significantly less accurate than its laser range finder and to correct the odometry more information from the laser range finder is necessary. These parameters have been used in experiment 1 and 3 as well.
The resulting map of one of the best runs with optimized parameters is displayed in Figure 5-15. Because it is the first time a visualization of FoRMeT is shown in this work, the different parts are explained. Multiple maps are visible in the figure, 16 to be exact. These maps can overlap each other, and a unexplored gray part could be over an explored white part in another map. This is why the maps have some opacity. The red ellipses show the uncertainty projected from the node that the best particle is currently in. A close up of the current map is given in Figure 5-16. Here the traversal and map generation thresholds can be seen as gray lines, the particle cloud as blue arrows under the best particle's coordinate frame shown with a red x and green y axis. The particle cloud is almost not visible because the robot is very certain about it's location since this is a previously mapped area. The small white square is a visualization bug. The edges are shown as lines between the uncertainty ellipses. The green edges are created by loop closure and the blue edges are created by new map generation. Dashed lines indicate the edges which are not used in the uncertainty projection i.e. the shortest paths to the two nodes use other edges. De corrected loop closure inconsistency is most visible at the dashed edge. To illustrate this a run with a larger loop inconsistency is visualized in Figure 5-17. Here it can be seen at the dashed edge in the bottom right corner the maps do not align. Also sometimes maps connected with a regular edge seem to have some rotational misalignment. This is because there is nothing to match when a particle enters a new map and it could easily go of course without being penalized. This is a downside of the current implementation, but the edge will correct this when a particle re-enters the map.
Though it might be hard for the human eye, this is a very usable result for the robot. The local maps are all locally accurate, and there are no misalignments in the loop closure area. The processor results are plotted next to GMapping's results in Figure 5-18. There are some peaks at the moments new maps are generated and one longer peak at the moment of the large loop closure around sample 1900. Also around sample 550 and 1500 the robot has stood still for a while, no peaks are visible there because no filter updates where performed. The cumulative time used by the core ends at 299 seconds, which is quite contrasting to GMappings 1162 seconds. A close up is made at the beginning and around the large loop closure point, these plots are visible in respectively Figure 5-19 and Figure 5-20. The filter update frequency seems to be similar in both close up plots, however at the end the peaks are a bit higher which can be explained by the multitude of maps that need to be visualized. Also the prolonged 100% peak around sample 2080 is clearly visible, this peak shows the processor usage at the large loop closure. This takes a long time both because there was a big uncertainty so many sample points needed to be matched, and because after loop closure the algorithm has to process all odometry readings missed in the time the matching took place. However, note that it's not much longer than GMapping's filter update peaks.

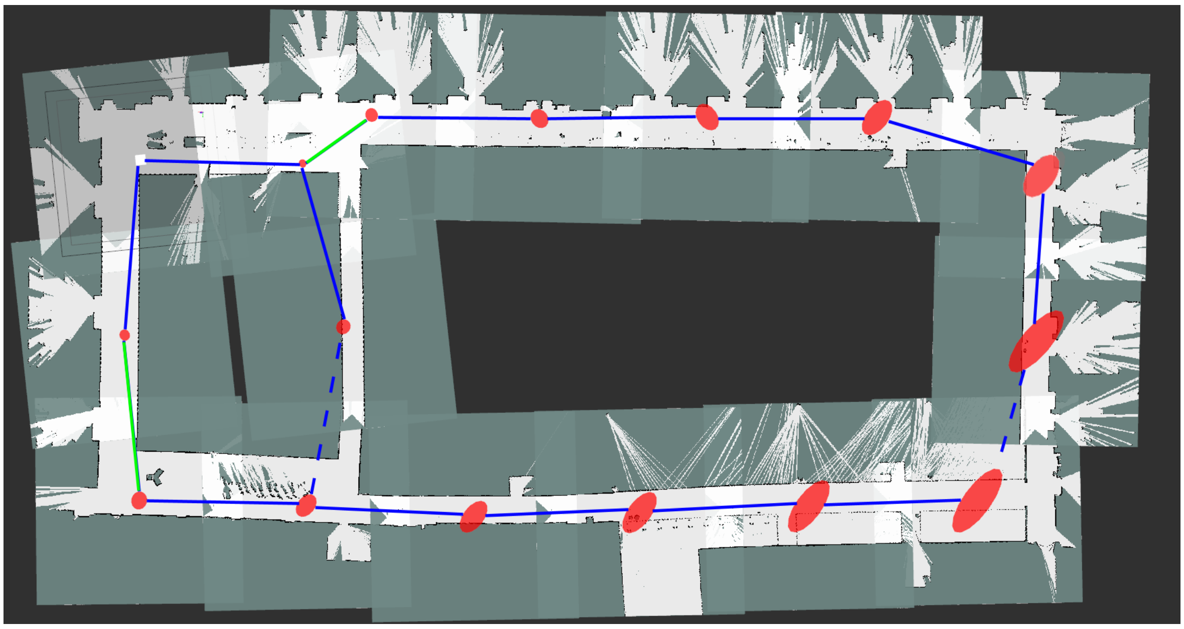


Figure 5-15: FoRMeT Mapping in experiment 2

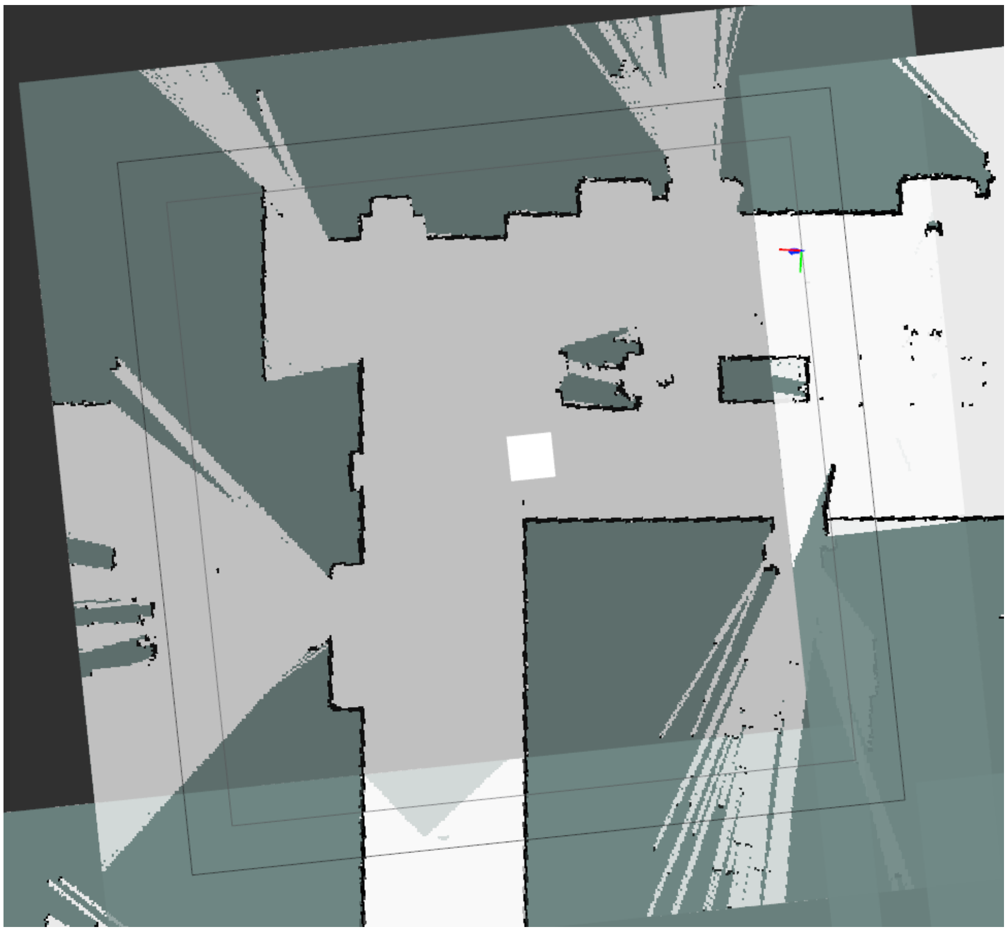


Figure 5-16: The current map from Figure 5-15.

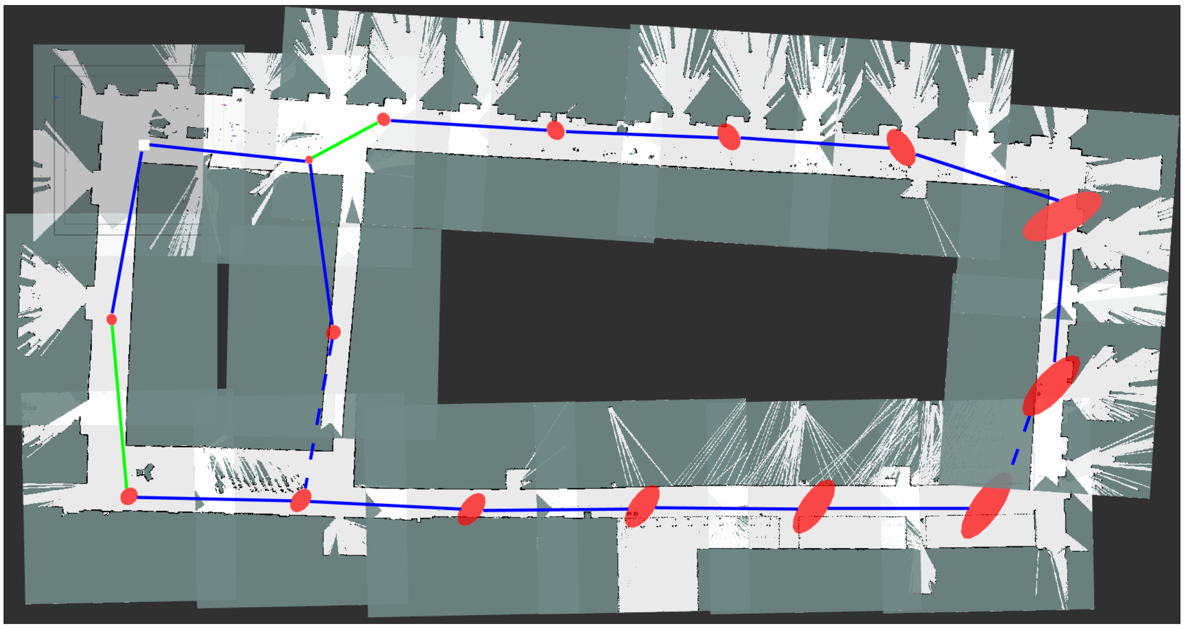


Figure 5-17: FoRMeT Mapping in experiment 2, a run where a larger mismatch was corrected.

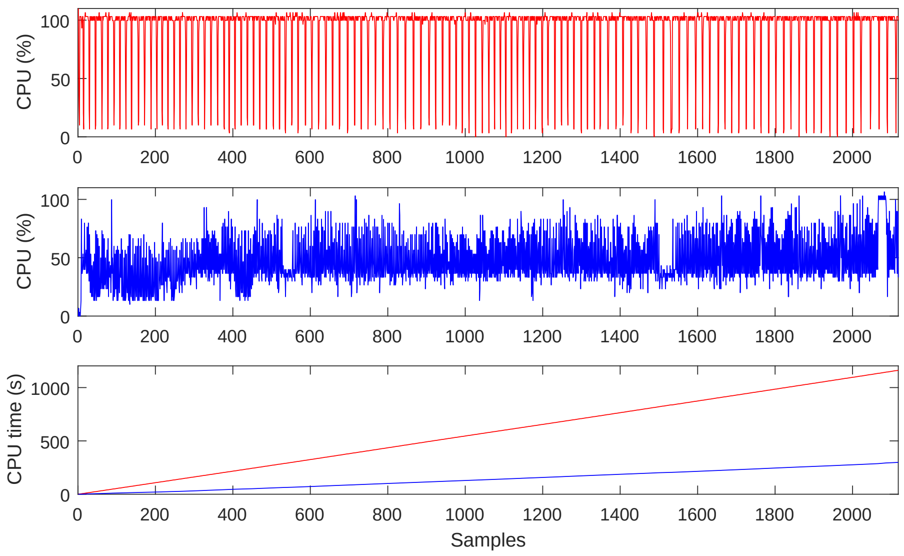


Figure 5-18: Performance plot of experiment 2. GMapping in red and FoRMeT in blue.

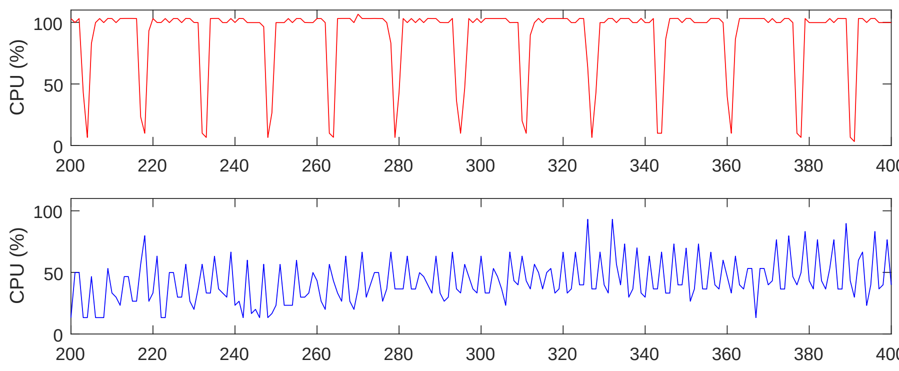


Figure 5-19: A close up of CPU samples 200-300 in experiment 2. GMapping in red and FoRMeT in blue.

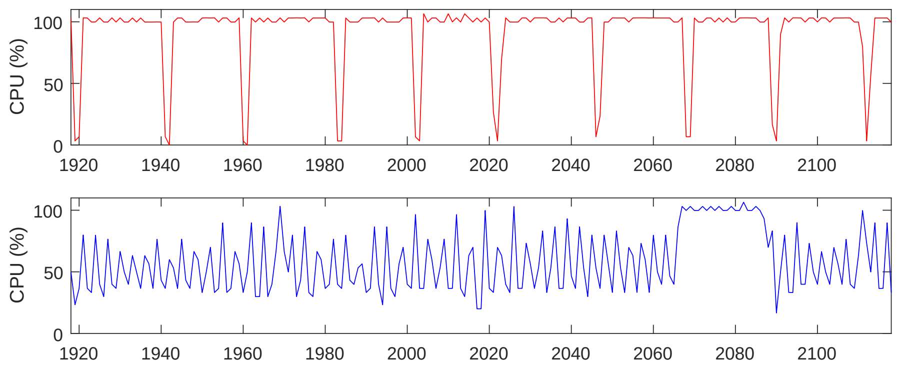


Figure 5-20: A close up of the last 200 CPU samples in experiment 2. GMapping in red and FoRMeT in blue.

5-5 Simulated large scale experiments

The IDE dataset has one large loop, but is not sufficiently large to test the scalability. The third dataset is recorded in a simulated environment. This dataset does not have as much noise as the odometry and laser scans of the SAM1 robot, but the path is about 250 meter longer (± 550 meter) and seven loops with path lengths between 30 and 80 meter need to be closed. A view of the simulated environment, the robot path and the loop closure points is visible in Figure 5-21.

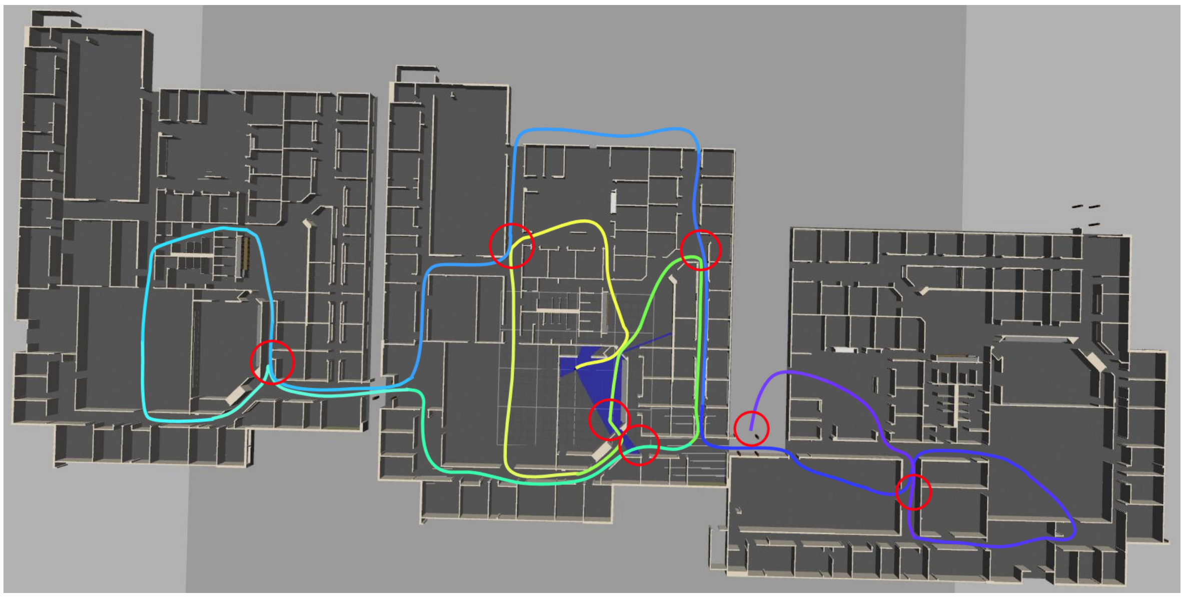


Figure 5-21: The Gazebo environment with the three Willow Garage worlds. The robot is in the middle and the traveled path is shown as a line with color changing from yellow at the start to purple at the end. There are seven loop closure points present, they are shown as red circles.

For FoRMeT Mapping the settings of the previous experiments resulted in a good map. Therefore no adjustments needed to be made to the parameters, which is good for the comparability to experiment 2. GMapping was not able to map for more than 20 meters without totally misaligning parts of the map due to wrong scan matching. Because there is less noise in this dataset GMapping's parameters could be tuned to lighter settings at which it performed reasonable. GMapping was able to map real time with this dataset, there was no need to decrease the playback rate.

5-5-1 GMapping

The resulting map of GMapping can be seen in Figure 5-22. It performs well. Usually the hallways and rooms are mapped straight and next to each other. Loop closures also do not seem to be a problem usually the observations at loop closure points fit good. However, during the last part, on the right of the figure, GMapping becomes slower and starts to make mistakes. The rooms have a slight angular tilt opposed to the rest of the map, and the last two loop closures are more than a meter off. A closer look to the failed last part can be viewed in Figure 5-23.
The processor performance is plotted in Figure 5-26. Again GMapping uses about 100% of

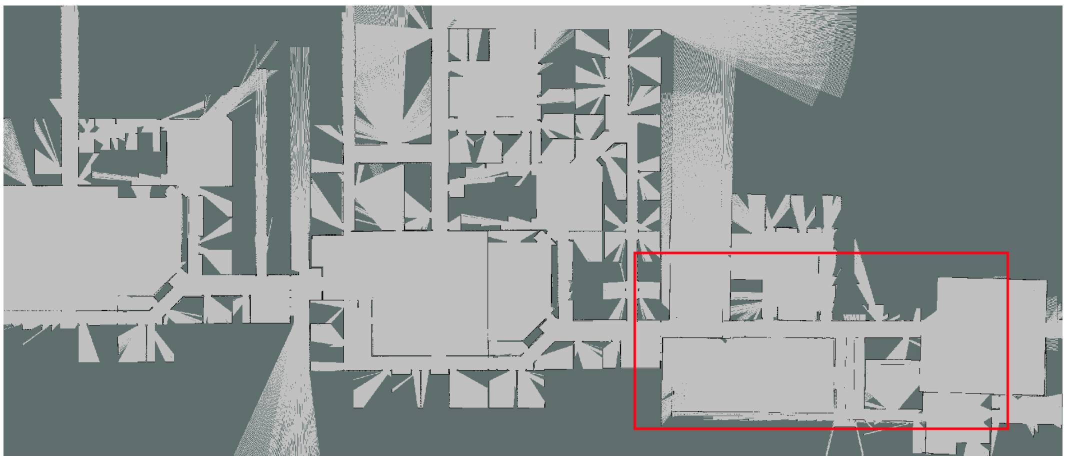


Figure 5-22: A GMapping created map of the simulated environment in experiment 3. The close up of Figure 5-23 is located in the red rectangle

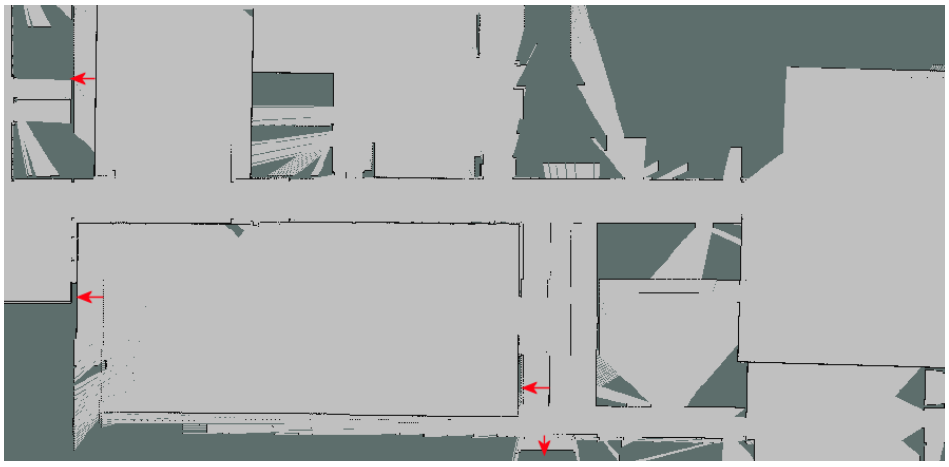


Figure 5-23: A partial map created by GMapping, the local environment in Figure 5-22. The loop closure offsets are visualized with red arrows.

the core to do filter updates and has a small dip between updates. GMapping uses a total of 1645 processor seconds to map the simulated environment. Figure 5-27 and Figure 5-28 zoom in on respectively samples 800-1000 and 5000-5200 to view processor performance near the beginning and the end. It can be seen that GMapping can do a filter update every 18 samples in the beginning, but at the end, when the map becomes larger each filter update uses more computation power and can only update every 35 samples.

5-5-2 FoRMeT Mapping

FoRMeT Mapping has no trouble mapping the simulated environment real time and accurate using the settings of experiment 2. There are some angular differences between the maps, but all local maps are locally accurate, and all loops are closed correctly.
It can be seen that all the uncertainties are correct, small near the robot, and larger far away from the robot. Also sometimes the mapping went a bit wrong and a straight hallway

turned a bit slanting, however this was corrected in the edge during loop closure. The effect of this is visible in the local maps which are slightly rotated to make the mapped path connect well. This is excellent HMT behavior. The processor results are also good. In Figure 5-26 it can be seen that it is uncommon to reach the 100%. Some higher percentages are reached when the local space has a lot of grids to map e.g. the open wide space in the upper middle section of Figure 5-24 (around sample 4000), and some lower percentages when the robot travels through already mapped areas (samples 4400-5000). The total time the processor spends is 590 seconds. In the close ups of Figure 5-27 and Figure 5-28 it is visible that the filter keeps on updating in real time with the same frequency. The peaks in CPU usage however have become larger. This is because the larger the mapped environment becomes, and the maps the robot visualizes each update. Also during traversal more uncertainty projections need to be calculated.

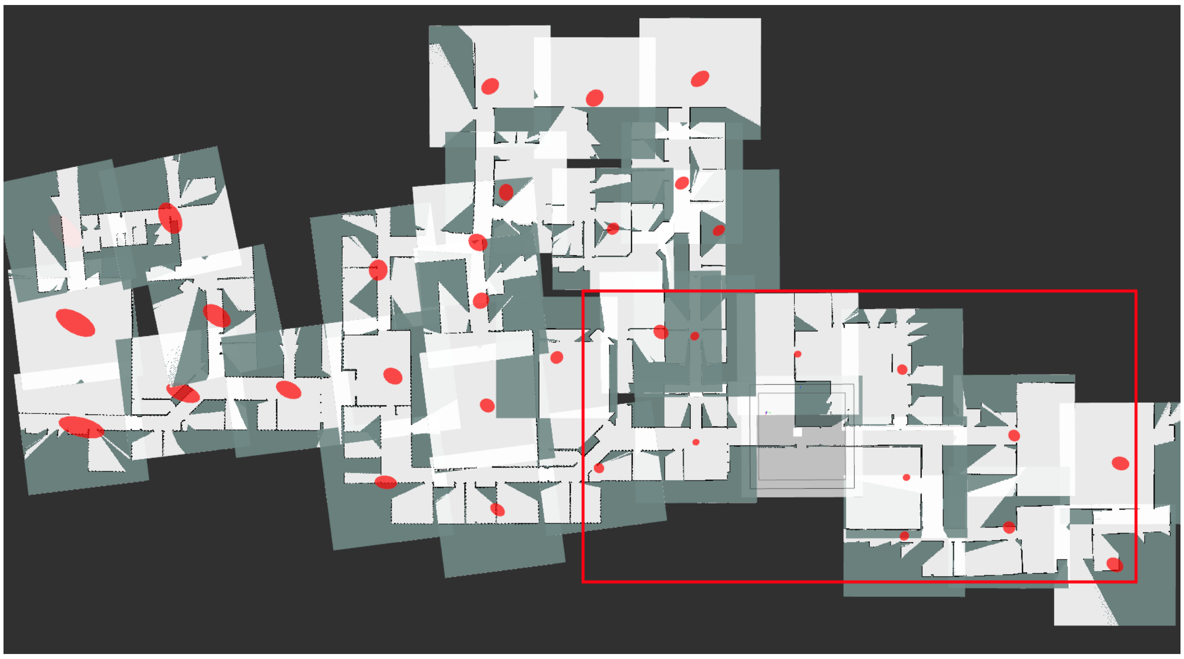


Figure 5-24: A FoRMeT created map of the simulated environment in experiment 3. The uncertainties projected from the current local map are indicated as red ellipses. The Figure 5-25 close up is indicated with a red rectangle.

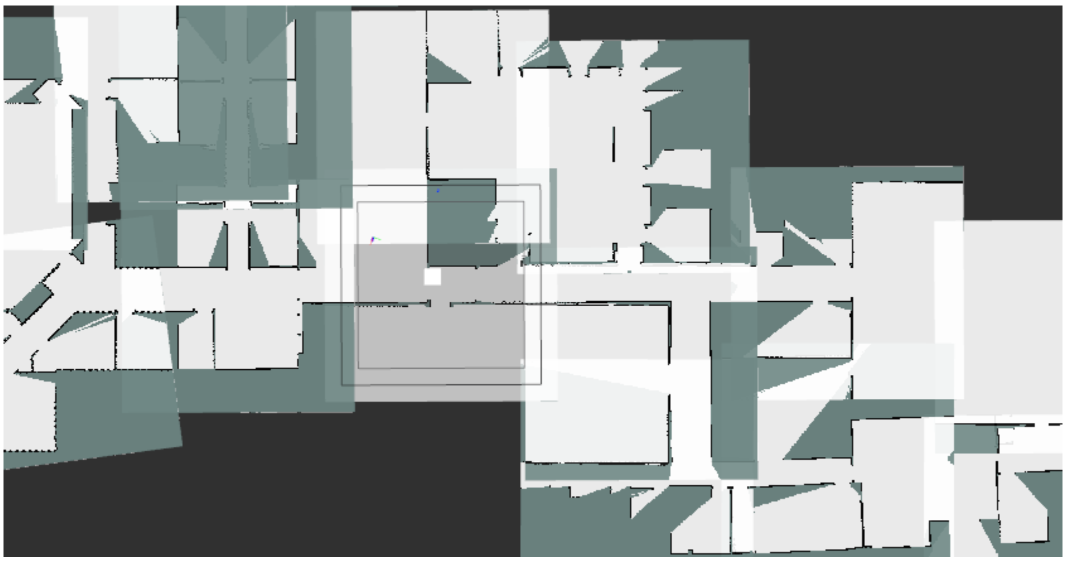


Figure 5-25: A part of the map created by FoRMeT Mapping , the local environment in Figure 5-24.

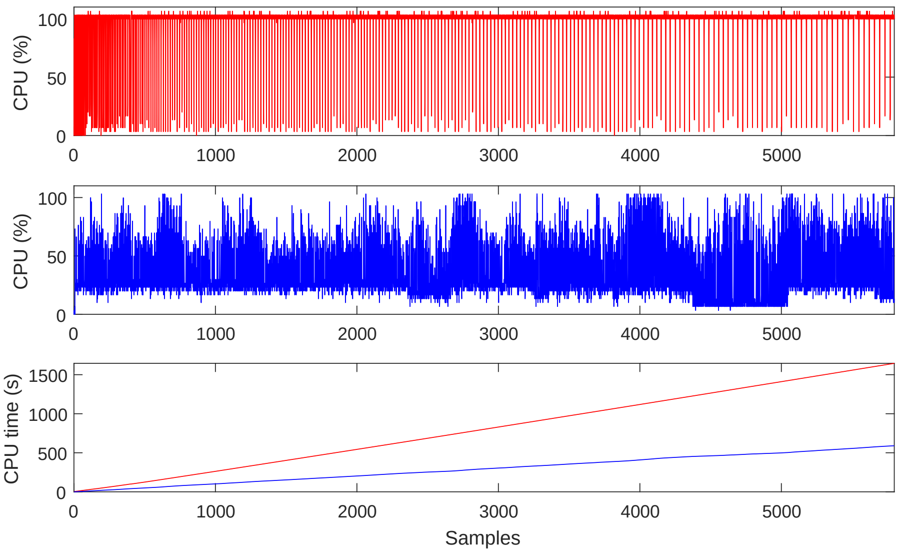


Figure 5-26: Performance plot of experiment 3. GMapping in red and FoRMeT in blue.

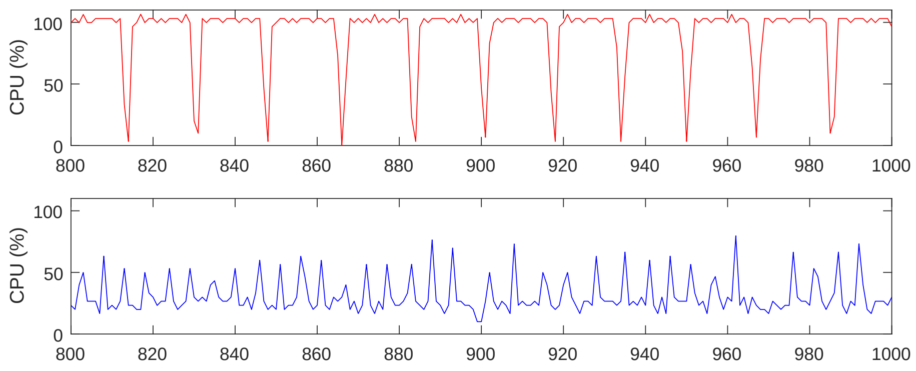


Figure 5-27: A close up of CPU samples 800-1000 in experiment 3. GMapping in red and FoRMeT in blue

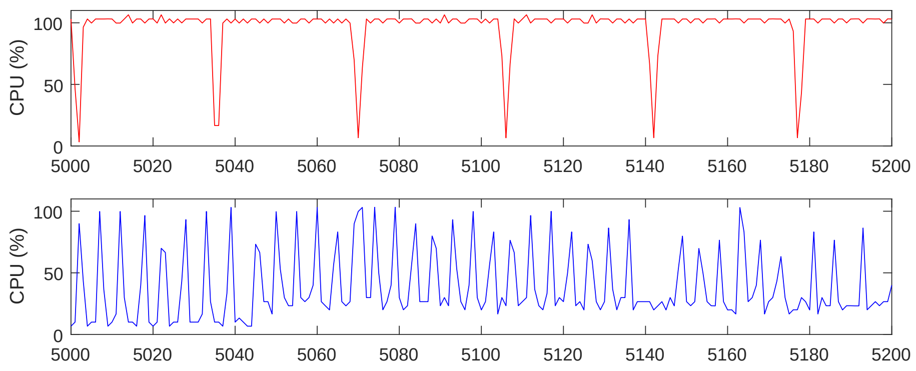


Figure 5-28: A close up of CPU samples 5000-5200 in experiment 3. GMapping in red and FoRMeT in blue

Chapter 6

Conclusion

The FoRMeT framework has been introduced and a version has been implemented to test it's capabilities. Conclusions are drawn about the performance of the algorithm with respect to the hypotheses and requirements.

The forced resampling strategy ensures the particles only need to remember two maps. This makes resampling of a particle a cheaper action. From the results of FoRMeT Mapping in the experiments it does not seem that the forced resampling has a negative effect on the quality of the local map. Herewith it can be concluded that the strategy has worked as hypothesized in the beginning of Chapter 5.

In the real world dataset FoRMeT Mapping had no trouble closing the small loop with slightly biased and inaccurate odometry and a good laser range finder. The large loop could also be closed but it was a challenge where more computational resources was needed. It can be said that a large loop of at least 210 meter can be closed using this setup, while the robot uses no mapping strategies like looking back from time to time.

In the larger simulated dataset FoRMeT Mapping had no trouble mapping all the observed locations. It had correctly and quickly closed all seven loop closures with sizes ranging from 30 to 80 meters in path size. Though it was simulated and thus contained very little noise still the scalability could be tested. 35 local maps with a size of 12 meter, spanning more than 3500 m^2 where kept in FoRMeT 's memory and still it ran each filter update well within time for the next one, and each loop closure within the time of two filter updates. It appeared like the 550 meter path in the simulated environment was no challenge for the algorithm.

GMapping performed worse than FoRMeT Mapping in all experiments. GMapping was not able to finish a filter update before a new update threshold was reached, even not when given twice the time in the real world dataset. With optimized parameters it was sometimes able to close the small loop without discrepancies and otherwise recovered its position while leaving a small discrepancy. The large loop was only closed once, however not without some small discrepancies, and with the same settings it failed the four other trials. In the simulated dataset it also uses a lot of computational recourses, but the map looks very straight and all loops could be closed up to a path length of 450 meter. Afterwards the scale of the map requires too much computational recourses for GMapping to accurately continue. It can be concluded that FoRMeT can close larger loops and is better scalable than GMapping while using significantly less resources.

The items from the design specifications (1-3) which are not discussed yet will be treated now. Item 2 mentions that the map should be globally consistent by approximation. Clearly the resulting FoRMeT map is less intuitive to understand than a global map from i.e. GMapping. However, all the local environments are approximately where they should be and to the practiced eye the network of maps can be interpreted. Item 5 specifies the framework should not limit itself to a specific environment. Though it has only been tested on one real world and one simulated dataset, in theory the framework has not limited itself to a specific environment and it remains flexible by having tunable parameters like map size.

If the user has no problems with the visual aspects of the network of maps e.g. projected loop inconsistencies and overlapping maps where areas are mapped twice, the FoRMeT framework will be an excellent solution for mapping large scale environments.

Chapter 7

Discussion and Recommendation

This chapter will be used to discuss the performance of GMapping and the limitations of the framework. Also recommendations for improvements and related future research are done.

7-1 GMappping performance

If GMapping is an acknowledged algorithm in the field of SLAM, why did it work suboptimal? Let's start by saying GMapping worked well in simulation. It could not stay within its filter update threshold, but it created a straight map in real time. Only when the map became larger GMapping started to fail, which is common for global mapping algorithms. In the real world GMapping also produced a map that looked reasonably straight to the human eye but at loop closure points the metric alignment proved to be to far off to be captured by the particle spread. When giving GMapping more time it could make its local areas straighter, but it was often still not satisfactory to close the loop. The odometry of SAM1 is not very accurate, therefore GMapping has to put a lot of effort to correct this in the filter and it uses a lot of computational resources to try to do so. Also GMapping might work good as an offline mapping method. Now the data was played back at a specific rate and GMapping has to keep up with this rate. If GMapping would have read the dataset from the point it left off at the previous filter update, it would take the time it needed for each update never lag behind and it might provide a very consistent map. However, in this work a real time mapping algorithm is required.

7-2 Similarity to HMT-SLAM

This framework takes a lot of inspiration from HMT-SLAM and Atlas. However it is most similar to HMT-SLAM because of the particle filter strategy. It is however a different framework. In HMT-SLAM the maps size is based on overlap of observations which is determined with a segmentation algorithm. In FoRMeT Mapping the map size is fixed and a new one is started when the robot arrives at a threshold. Also HMT-SLAM first creates the map and then matches with other maps to see if the area was not already known in hindsight. FoRMeT matches before it exits its current map. Besides these arguments this work also introduces forced resampling which makes FoRMeT unique.

7-3 Limitations

7-3-1 Network of maps

The robot is able to work with the network of maps very well, but in the end it may be up to humans to give the robot a task and possibly indicate where this task must be carried out. Humans should thus be able to work with the hybrid map view, which is not always straightforward, or at least not as intuitive as one global map. This can be seen as a downside of this HMT framework opposed to global mapping.

7-3-2 In between maps

In a specific case the robot move drive straight upwards between the map match threshold and the map border of both maps as indicated in Figure 7-1.

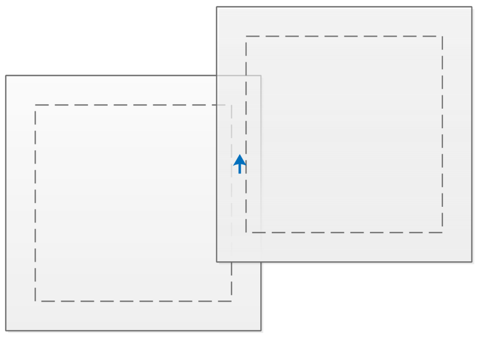
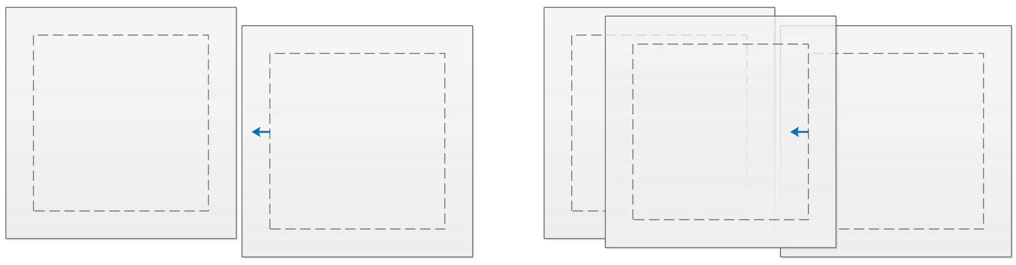


Figure 7-1: Robot traveling between the map match threshold and the map border of both maps.

Since the particles are sampled with noise some of them will have an outward and some of them will have an inward direction. The ones with an outward direction will start matching and traversing (see 3-2). This is desired behavior. However in the other map the particle will again be directed straight between the maps by matching. Due to the noise in the sampling some particles might again look outwards and start matching again with the previous map. An oscillating effect where forced resampling and matching will take place each filter update. This sounds bad, but it's not as bad as it seems. Apparently the other map existed already and possibly the edge as well, or otherwise the edge will be created at first traversal. Then the uncertainty will be small, matching will go easily and the forced resampling will not decrease the map quality because the map is already filled with previous observations. The downside is thus that some more computational resources are used during oscillating traversal.
In another case the current map might end just before a loop can be closed with another map as indicated in Figure 7-2 (a).
A new map will now be generated which largely overlaps with the already mapped area as shown in Figure 7-2 (b). For the robot this will be no problem. As soon as the robot goes out of the generated map it will match with the mapped area and connect the maps. The double mapped area will not cause any problems in the framework, depending from where the robot comes it will either use the older or the newer map until it is at the traversal threshold. After more frequent traversal the newer map will be connected to any neighboring maps. Only when the robot arrives from such a direction that it could match with both



(a) before map generation **(b)** after map generation

Figure 7-2: Robot driving between the map match threshold and the map border of both maps

maps it will take a longer to traverse because more potential map matches are present. The performance of the framework is determined by the size of the projected uncertainty. When the uncertainty is larger the matching will use more computational resources and the peak computations are performed during matching. This is amplified when multiple candidates are available to match with. Long loop closure will not mean missed data is lost, because the algorithm will catch up on the missed data during matching. So if the processor is not fully utilized during normal operation the algorithm will catch up to the real time situation.

7-3-3 Map exploration condition

Besides reaching a map end before a particle could match with another map, it could also happen that a map cannot be matched because the correct location is unexplored in the candidate map. To create a network of maps with a minimum of overlapping double mapped areas it is desirable to first explore a local area as good as possible before continuing to the next map.

7-3-4 Empty maps

The current framework cannot match with empty maps. When there are no landmarks present in a local map a particle cannot determine its location in this map. The current implementation does not even allow traversal to empty maps. This is a downside of the current framework and implementation.

7-4 Improvements

7-4-1 Tilted environment in new maps

The biggest reason for large offsets in loop closures is because of tilted environments new maps. Whenever a particle enters a new map there is nothing to match and thus there is little penalization for particles who's odometry is sampled far from reality. Due to this it could happens that i.e. a hallway is suddenly mapped slanted with respect to the previous map. This will be corrected when the robot will traverse the edge again, but it is better to be avoided in the first place. An improvement could be made where the last few observations that already fall in the new map while the particle was in the previous map are used for matching in the new map.

7-4-2 Uncertainty ellipses

The larger an uncertainty is the more matching needs to take place when one or more particles want to match with the associated map. The performance is therefore largely dependent on the size of uncertainties and it is thus important to accurately represent the uncertainty. Often the real uncertainty is not an ellipse, but some nonlinear shape which cannot be expressed in a Gaussian. Therefore it would be better to express the uncertainty as particles, multiple hypotheses about the relative location of nodes. The particles could be projected over all edges into a resulting particle cloud representing a more accurate distribution. The traversal function now does not have to create match points in an uncertainty, but the projected particles themselves act as the match points. The distribution is now more accurate, and the step of creating an ellipse and sampling from an ellipse can then be omitted.

7-4-3 Code improvements

The implementation is far from optimized. Code improvements using different programming strategies could be implemented to speed up the computations. Also double actions could be identified which can be omitted from the process. An example of this is the matching after forced resampling at traversal point. Many particles will be identical to others and have exactly the same pose after forced resampling. The result of map matching will thus also be identical. For each group of identical particles only one matching action would suffice, possibly saving significant amounts of computation time.

7-4-4 Improved localization and optimization

The odometry correction using the transform between two observations (Section 3-1) has not been implemented yet as described in Subsection 4-3-1 where an ICP algorithm on two laser scans is suggested. Some more research is necessary to accurately improve the robots odometry this way. Also the mentioned non linear optimization (Subsection 3-5-2) could still be developed for an even more scalable implementation.

7-5 Future research

7-5-1 Continuous mapping

The current framework describes how to create a network of maps in real time. For a fully functional mobile robot it is necessary to be able to perform tasks using this network of maps. The current implementation could likely be used for continuous mapping because it does not use a lot of computational resources. Commonly mapping happens once and afterwards a localization algorithm is used in the fixed map. This is mainly because mapping is an expensive action and localization in a known map is cheap. FoRMeT Mapping has, like any SLAM framework, already a localization built in because of the pose estimate. Whenever a map is explored sufficiently a future version of FoRMeT could only update newly observed items in a map, and when entering an unexplored area it could continue as a full mapping system. This way it can be used to map continuously and keep up to date with the latest environment configuration.

When continuous mapping is not desired a localization framework should be designed that can function in the network of local maps. It would be like any other localization framework, but it is now necessary to take traversal into account. Also for well functioning navigation it is recommended to have the topological graph as connected as possible. Thats why it is advised to update and create edges in the localization framework when traversing between maps.

7-5-2 Navigation

For a fully functional mobile robot navigation is essential. The current framework does not include any navigational aspects. However, robots using a network of maps to travel to a designated location have to use it with a hybrid type of navigation. Because the local area's are arranged in a graph it could make path planning easier. A global route over the topology, from node to node, could be quickly calculated. Afterwards a local path planner over the grid cells could be used for each node in the local map.

7-5-3 Dynamic maps

Local maps are perfect to keep place related information. As discussed in the continuous mapping subsection, a future framework could keep on mapping while the robot is being used for other tasks. During the continued mapping it could keep track of the dynamic elements in the local surroundings and relate them to the associated node. This way the map is kept up to date to prevent future mismatches in loop closures and general localization. Currently when the robot travels through an explored area it uses less computational resources, so there is room for extra processing in these areas.

7-5-4 Autonomous exploration

To relieve humans from the task of coordinating the robot during map creation an autonomous exploration framework could be devised. This would especially be useful in larger environments. Such a framework should keep certain HMT exploration strategies in mind like first exploring the current local map before going to the next, creating as many node interconnections as possible and keeping the loop size to a maximum to prevent large calculations.

Appendix A

Bayes Filter

A-1 Bayes Theorem

The Bayes theorem Eq. (A-1) expresses the probability of one variable given another variable. In this case the probability of A when B is given.

$$p(A|B) = \frac{p(B|A)p(A)}{p(B)} \propto p(B|A)p(A) \tag{A-1}$$

$p(A)$ is called the *prior* which contains information about A before B is known, we would like to infer the quantity of A given B. $p(A|B)$ is called the *posterior* which is the final estimate with the prior and information about B incorporated. The division by $p(B)$ is used to normalize, hence the proportionality with the most right side. The Bayes theorem thus splits the estimation of a conditional variable into a normalized prior and posterior estimation.
In the sense of pose estimation this would mean the probability of the robot's pose given a measurement equals the probability of the measurement given the pose, times the prior estimate of the pose, divided by the probability of having the measurement. The Bayes theorem is used in the Bayes filter, which recursively estimates $p(A)$ using the posterior of the previous time step as the prior of the current one. The Bayes filter is treated later this chapter in section (A-2).

A-2 Bayes filter

The probabilistic definitions of the previous sections all come together in the Bayes Filter, which is used as basis for localization and mapping. Before the Bayes filter can be introduced the *motion* and *observation model*, concept of *belief* and the *Markov assumption* must be discussed.

A-2-1 Motion and observation model

The estimation of the robot state can be estimated with the Bayes filter using two models, the *motion* and *observation model.* The motion model describes the relative motion of the robot and is given as a distribution of the pose given all previous poses and the control inputs Eq. (A-2).

$$p(x_t|x^{t-1}, u^t) \tag{A-2}$$

$$p(z_t|x^t) \tag{A-3}$$

The measurement model describes how a measurement is related to the state estimate Eq. (A-3). The next section describes how a Bayes filter uses these models to estimate the states.

A-2-2 Beliefs

The robot has no access to the true state, but can only form a belief by estimation based on measurements. The internal belief the robot has about the state is appositely called the *belief*, abbreviated as *bel.* The belief is a distribution function assigning a probability to the state hypothesis', and is more or less a way abbreviate probability distributions.

$$bel(x_t) = p(x_t|z^t, u^t) \tag{A-4}$$

The belief in Eq. (A-4) is the posterior of x, it estimates x based on all measurements up to time t. A belief that does not include the latest measurement called the prior, is denoted as $\overline{bel}$.

$$\overline{bel}(x_t) = p(x_t|z^{t-1}, u^t) \tag{A-5}$$

A-2-3 Markov assumption

The Markov assumption is an important assumption which allows us to use the Bayes filter. It hypothesizes that all previous measurement information can be disregarded in the estimation of the current state, if one has the previous state estimation given. Herewith it states that there is a *conditional independence* between the current state and the previous measurements. The assumption can be made because the previous measurements are incorporated in the previous state estimates. In practice the Markov assumption looks like Eq. (A-6).

$$p(x_t|x^{t-1}z^{t-1}, u^{t-1}) = p(x_t|x_{t-1}) \tag{A-6}$$

The assumption can be made because x_{t-1} has been estimated using all measurements up to $t-1$. The Markov assumption allows recursive state estimation.

A-2-4 Bayes filter

The Bayes filter uses the previous probability theorems to devise a prediction of the current state given the previous state and the control input, and afterwards a correction of that prediction using the current measurement. The equations are formed by writing Markov assumptions into the beliefs of the motion and sensor models. The Bayes prediction is given by equation Eq. (A-7).

$$\overline{bel}(x_t) = \int p(x_t|u_t, x_{t-1})bel(x_{t-1}) \tag{A-7}$$

It can be seen that the belief prediction of x_t is dependent on the motion model in which the previous measurements have been substituted by x_{t-1} due to the Markov assumption. This prediction functions as a prior in the corrected belief equation Eq. (A-8).

$$bel(x_t) = \eta p(z_t|x_t)\overline{bel}(x_t) \tag{A-8}$$

The correction step uses the measurement model to correct the prediction with a normalizing term η. Here the underlaying Bayes Rule Eq. (A-1) can be seen. Together these equations give the believe of the state at time t.

A-3 Kalman filter

The Kalman filter is a Bayes filter. In particular, the Kalman filter is the Bayes filter used for linear systems with Gaussian distributions. It will converge to the optimal solution if the motion and observation model are of the following structure Eq. (A-9).

$$\begin{aligned} x_t &= Ax_{t-1} + Bu_t + \epsilon \\ z_t &= Cx_t + \delta \end{aligned} \tag{A-9}$$

A describes how the state evolves based on the previous state, B describes how the control input affects the state and ϵ is an error term introduced by motion noise. C describes how the measurement is related to the state and δ is an error term introduced by measurement noise. The Kalman filter steps are described in algorithm 1.

Algorithm 1: Kalman Filter

```
Input: Mean $\mu_{t-1}$, covariance $\Sigma_{t-1}$, control $u_t$, observation $z_t$
Output: $\mu_t, \Sigma_t$
                                  /* Prediction */
1 $\bar{\mu} = A_t\mu_{t-1} + B_t u_t$
2 $\bar{\Sigma} = A_t\Sigma_{t-1}A_t^T + R_t$
                                  /* Correction */
3 $K_t = \bar{\Sigma}_t C_t^T(C_t\bar{\Sigma}_t C_t^T + Q_t)^{-1}$
4 $\mu_t = \bar{\mu}_t + K_t(z_t - C_t\bar{\mu}_t)$
5 $\Sigma_t = (I - K_t C_t)\bar{\Sigma}_t$
6 return $\mu_t, \Sigma_t$
```

By calculating the Kalman matrix K_t, the state and covariance can be estimated at each new measurement. First a prior estimate of the state and covariance (barred) is calculated with the state space matrices A and B, the input u and the previous estimation μ_{t-1} while incorporating the motion noise R_t covariance. Then the Kalman matrix K_t is calculated with the measurement noise covariance Q_t and used to form the posterior mean and covariance which incorporate the measurement z_t.
The result of the Kalman filter is a new weighted mean and covariance. The steps in a one dimensional situation would look like the example in Figure A-1. Since the covariance in an estimate is a measure of accuracy the most accurate state estimate will have the most influence on the final estimate.

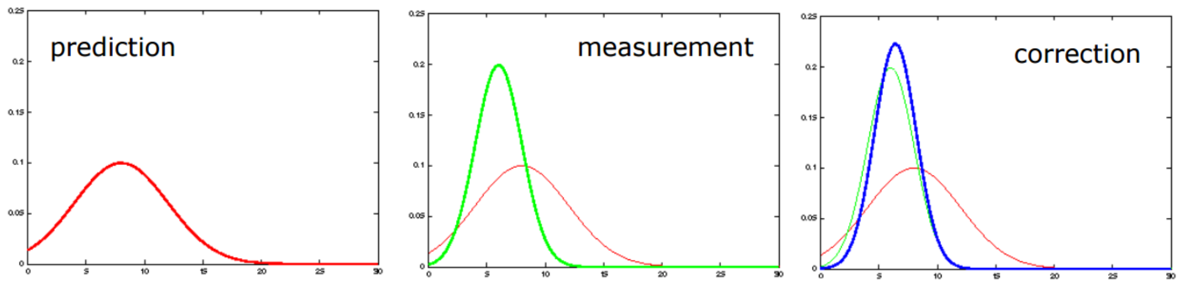


Figure A-1: The Kalman filters prediction, measurement and correction step

It can be seen that the measurement (green) has a smaller variance then the prediction (red). Therefore the corrected (blue) estimate has a mean that lies closer to the measurement than the prediction. Also note that the correction has an even smaller variance than the measurement, it is more certain because it is based on two separate estimates.

Appendix B

Introduction to Kalman Filter SLAM

The traditional SLAM methods are based on the Kalman Filter (KF) which estimates the state of the map and the robot simultaneously. The Kalman Filter estimates the poses of the robot and the landmarks as normally distributed means and covariances. A *state vector* μ_t at time t contains the estimated mean pose of the robot x_{robot} and the estimated poses of the landmarks L_i in the map and a *covariance matrix* Σ contains the covariances between all elements in the state vector Eq. (B-1).

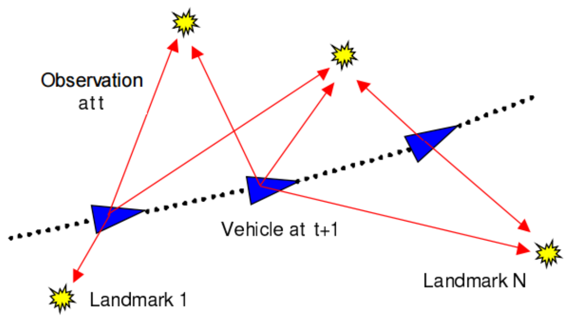


Figure B-1: The robot pose at different time steps and corresponding landmark observations

When the pose of the robot is two dimensional it contains $x_{robot} = [x, y, \theta]^T$. The total state adds the poses of each landmark $l_i = [l_x, l_y]^T$. The composition of μ and Σ in the Kalman filter are given in equation Eq. (B-1)

$$\mu = \begin{bmatrix} x \\ y \\ \theta \\ l_1 \\ \vdots \\ l_n \end{bmatrix} \qquad \Sigma = \begin{bmatrix} \sigma_{x^2} & \sigma_{xy} & \sigma_{x\theta} & \sigma_{xl1} & \cdots & \sigma_{xln} \\ \sigma_{yx} & \sigma_{y^2} & \sigma_{y\theta} & \sigma_{yl1} & \cdots & \sigma_{yln} \\ \sigma_{\theta x} & \sigma_{\theta y} & \sigma_{\theta^2} & \sigma_{\theta l1} & \cdots & \sigma_{\theta ln} \\ \sigma_{l1x} & \sigma_{l1y} & \sigma_{l1\theta} & \sigma_{l1^2} & \cdots & \sigma_{l1ln} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \sigma_{lnx} & \sigma_{lny} & \sigma_{ln\theta} & \sigma_{lnl1} & \cdots & \sigma_{ln^2} \end{bmatrix} \tag{B-1}$$

B-1 EKF SLAM

EKF stands for Extended Kalman Filter [11]. It is used because the assumptions a standard Kalman filter makes are violated in most cases for mobile robots. The typical motion and observation models are non linear Eq. (B-2) and i.e. the prediction with a motion model can look like the distributions in Figure B-2 . These shapes are not Gaussian and with every subsequent pose another nonlinear shape is added to this one, so the real distribution after several time steps can have an arbitrary shape.

$$x_t = g(x_{t-1}, u_t) + \epsilon_t \qquad , \qquad z_t = h(x_t) + \delta_t \tag{B-2}$$

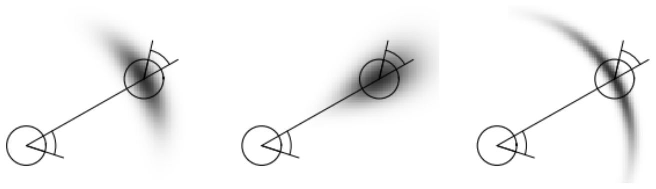


Figure B-2: Typical distributions from motion models. Adopted from [2]

When a Gaussian distribution is propagated through a nonlinear function, the result is typically non-Gaussian. The EKF works by making an approximation by linearizing the model with a first order tailor expansion Eq. (B-3).

$$\begin{aligned} g(u_t, x_{t-1}) &\approx g(u_t, \mu_{t-1}) + g'(u_t, \mu_{t-1})(x_{t-1} - \mu_{t-1}) \\ h(x_t) &\approx \bar{h}(\bar{\mu}_t) + h'(x_t - \bar{\mu}_{t-1}) \end{aligned} \tag{B-3}$$

B-2 Data association

The algorithm estimates the poses of the robots and landmarks with their respective covariances. In Figure B-3 the map with landmarks and robot trajectory of a run with an EKF based SLAM algorithm [1] is visualized next to the related covariance matrix.

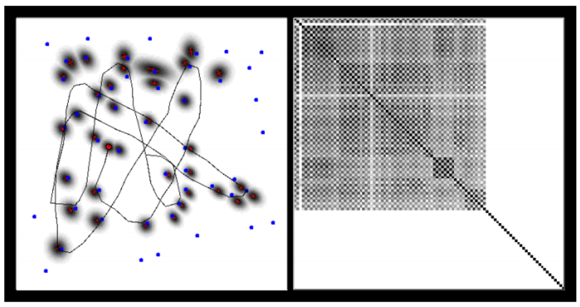


Figure B-3: Left: estimated robot and landmark poses with covariance, right: covariance matrix

Newly observed landmarks are inserted as the next entry in the state and covariance matrix, and therefore there is an empty area with the size of the amount of unobserved landmarks in the covariance matrix. The EKF algorithm can only function when the landmarks are uniquely identifiable, or with known *data association*. If landmark 5 is observed again, you need to know that is landmark 5 to be able to update the matrices. If you regard it as a new landmark the map will not likely converge to the true map which in turn makes localization difficult. Besides, the covariance matrix would grow unlimited if every seen landmark is inserted as new. The Kalman filter assumes the robot uses an algorithm to identify the landmarks uniquely.

B-3 Loop closure

Whenever an area that is already mapped is recognized we can speak of a *loop closure*. A loop closure is a moment of feedback for the SLAM process since the uncertainty of being somewhere is reduced when seeing an already visited place. It can be regarded as an extra correction step by the observation model. However, all the means and covariances are recalculated, and the uncertainties are reduced with the highest reductions the closest to the loop closure point. A visualization of a loop closure is given in Figure B-4.

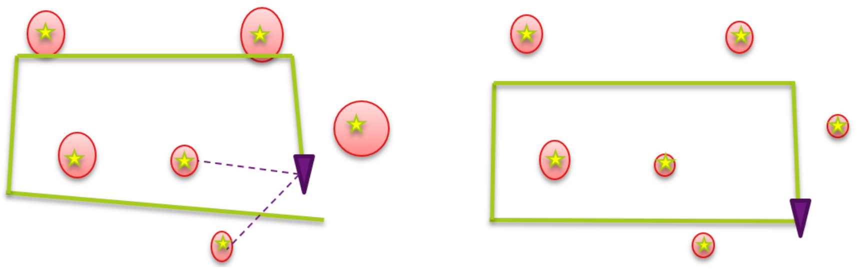

(a) just before loopclosure **(b)** after loop is closed

Figure B-4: The uncertainties of the landmarks and the estimate of the robot poses before and after loop closures.

When the robot is back at an already mapped place its internal representation might say that it is somewhere else because of small error accumulation. The robot in Figure B-4 re-observes two landmarks and can now correct its location by performing a loop closure. Note that the landmarks furthest away from the closure point have the greatest uncertainty. Also note that the uncertainty of the trajectory is omitted for convenience, but normally present. Just like in regular data association, the association with the right features at the right closure point is desirable. If a loop is closed with a wrong data association, the map will likely not converge to the true map. Detection of a loop closing point can be highly ambiguous. When two areas have very similar features one should be very careful not to close the wrong loop.

B-4 Computational complexity

The EKF algorithm has it's limitations, because the computational complexity can become quite high. The extended KF variant has to linearize the model at each time step. Also every time a new landmark is observed, an entry is added to the state vector and covariance matrix. Furthermore, with every observed landmark, all the covariances of that landmark with the other state variables must be recalculated. And with every loop closure every entry must be updated. This makes it a more time consuming algorithm when more landmarks are observed. The computational complexity becomes quadratic ($\mathcal{O}(n^2)$) with n, the number of landmarks. To improve these issues more variants of the KF are available. The Unscented Kalman Filter (UKF) does not use linearization by Tailor expansion, but by propagating the covariance from points which represent an abstraction of the uncertainty. The Sparse Extended Information Filter (SEIF) is a linearization of the information filter which uses the inverse of the covariance matrix, by ignoring almost 0 entries in the information matrix the calculations can be sparsified.

consideration Depending on the purpose, a different method may be used. EKF for nonlinear models, UKF for a bit better quality estimate, SEIF for a decreased computational cost. However for most robotic purposes any KF version only gives a linear estimation of the distributions, yielding suboptimal results. Also the computation time is often too big for large scale environments due to the increasing number of landmarks.

Appendix C

Introduction to Particle Filters

The Particle Filter (PF) is a non parametric implementation of the Bayes algorithm [2]. Particle filters represent the posterior by a set of weighted samples or particles, where each particle corresponds to a hypothesis about the robot pose. Besides a pose, particles have a weight, which can be derived from it's value in a distribution. The resulting set of weighed particles represents a sample-based approximation of the continuous posterior distribution. An advantage of using particles is that a particle cloud can take on different distributional shapes. Since in particle filters multiple hypothesis' can exist alongside each other, a multimodal Gaussian distribution for example can be easily approximated with a set of particles.
A set of particles denoted as $\mathcal{X} := x^{[1]}, x^{[2]}, \ldots, x^{[M]}$ that approximates the belief $bel(x)$ is calculated each step (Algorithm 2). It recursively constructs the new particle population $\mathcal{X}_t$ from the previous $\mathcal{X}_{t-1}$ while including control inputs and observations. Similar to the Kalman filter, first a temporal particle set $\bar{\mathcal{X}}_t$ is constructed by propagating all particles from $\mathcal{X}_{t-1}$ through the motion model. It will represent the prior belief $\overline{bel}(x_t)$, which is called the *proposal distribution*. Afterwards this set is resampled according to the *target distribution* which is based on the observations to represent the posterior belief $bel(x_t)$.

Algorithm 2: Particle Filter

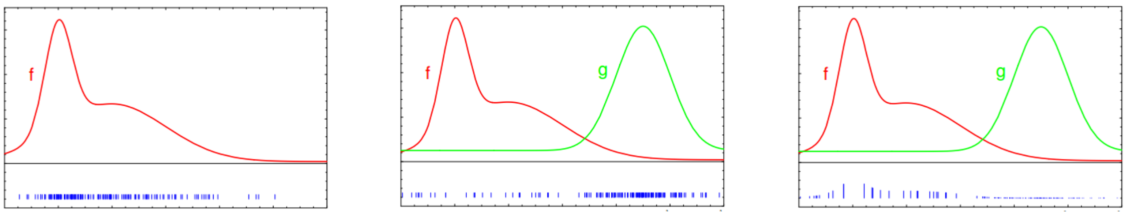
```
Input: previous state particle set $\mathcal{X}_{t-1}$, control $u_t$, observation $z_t$
Output: $\mathcal{X}_t$
1  $\bar{\mathcal{X}} = \mathcal{X} = \emptyset$
2  for $m$=1 to $M$ do
3      sample $x_t^{[m]} \sim p(x_t|u_t, x_{t-1}^{[m]})$
4      $w_t^{[m]} = p(z_t|x_t^{[m]})$
5      $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
6  end
7  for $m$=1 to $M$ do
8      draw $i$ with probability $\propto w_t^{[i]}$
9      add $x_t^{[i]}$ to $\mathcal{X}_t$
10 end
11 return $\mathcal{X}_t$
```

Line 3 samples each particle from their state transition distribution $p(x_t|u_t, x_{t-1})$. Line 4 determines the weight of each sample based on the posterior believe $p(z_t|x_t)$. And line 5 adds each sample to the sample population, which is noted with a bar $\bar{\mathcal{X}}$ indicating its dependency on the prior belief distribution. In the last for loop *resampling* takes place. Since the distribution of $\bar{\mathcal{X}}_t$ is based on the prior, the posterior will be included by resampling the particles based on their weights. The particles with low weights, or 'weak' particles, will be redrawn from the set of particles with high weights, or 'fit' particles. This introduces a 'survival of the fittest' concept which incorporates the posterior into the new set $\mathcal{X}_t$.

C-1 Resampling

The resampling step is a crucial part in the functioning of the algorithm. The particles in the proposal distribution should be corrected by the target distribution. In a Kalman filter this is a simple mathematical procedure with the mean and covariance. In the particle filter however there are samples from which some fit well in the proposal distribution, and others fit less. The particles should be weighed to how good they fit into the target distribution. Therefore the importance weights are chosen to represent the mismatch between the the proposal distribution $g(x)$ and the target distribution $f(x)$: $w(x) = \frac{f(x)}{g(x)}$. Intuitively one might see how an $x \in g(x)$ can be represented in $f(x)$ by multiplying it with $w(x) = \frac{f(x)}{g(x)}$.

$$x \in g(x) \rightarrow \frac{f(x)}{g(x)} \rightarrow x \in f(x) \tag{C-1}$$



(a) target distribution $f(x)$

(b) proposal distribution $g(x)$

(c) particles from $g(x)$ amplified with their weights

Figure C-1: particles sets (in blue) with their distribution shapes

In Figure C-1 the green distribution is the proposal distribution. The samples are drawn as can be seen below Figure C-1b. The weights can be derived by dividing the nonlinear target by the proposal distribution at the sample location. Note that the proposal distribution is clearly not Gaussian. The weighted particles can be seen at the bottom of Figure C-1c.

Appendix D

Monte Carlo Localization

D-1 Monte Carlo localization

Often the map of an environment is created once and assumed static. Afterwards the robot uses a localization algorithm to determine its pose in the map. One of the more popular algorithms based on the particle filter (Appendix C) is the Monte Carlo Localization (MCL) algorithm [12]. The algorithm places particles in a known map and determines the probability of its location being the location of the particle. Each particle evolves over time with the motion and measurements of the robot and gets a higher or lower probability of having the correct location. Going through this algorithm the belief about the robot pose likely converges to the true pose.

Example A robot is moving through one of two almost identical hallways that have been mapped before in Figure D-1. The MCL particles are distributed equally over both hallways, based on the observation the probability of being in either hallways is the same.

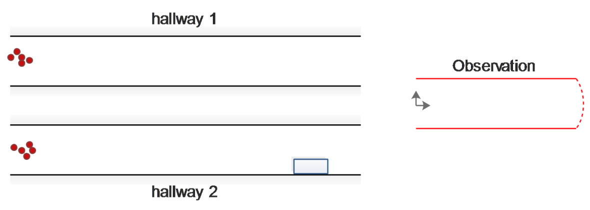


Figure D-1: Particle distribution over two similar hallways, and robot observation.

When the robot moves further in Figure D-2 it observes a feature that is present in only one of the hallways. The probability of being in hallway 1 has now becomes very low, and the hallway 2 hypothesis is now very likely. The particles in hallway 2 have a higher weight and are favored in the resampling step.

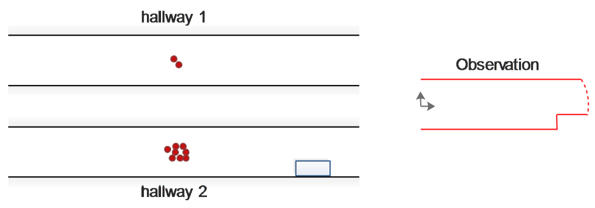


Figure D-2: Particle distribution over two similar hallways, and robot observation including box.

An advantage of MCL is that multimodal beliefs are possible, in the example it can be in both hallways at the same time. Also the resampling is based on the weight of the particle. Because the resampling is random and the chance of sampling a particle is proportional to its weight, it is also possible, but unlikely, that a particle with low weight is sampled. This is a strength of the particle filters since a particle might temporarily score low, but may still be the right particle. If someone would move the box, and the rest of the environment has remained the same, these two particles in the example that have a low weight will have a higher weight later on and the filter can escape this false local maximum. A particle filter works better with more particles, since the distribution it describes will go towards the real continuous distribution when the amount of particles goes to infinity. However the more particles used the more computation time. It is can be considered a design parameter. Too little particles: likely to get stuck in false local maxima, too many particles: high computation time.

D-2 Adaptive MCL

Kidnapped Robot When a the particles have converged around the robots true pose, and the robot would be "kidnapped" and placed in another known environment but without initial pose estimate we speak of a *kidnapped robot* situation. The robot does not know where it is and has to figure this out by moving around and observing the local features. The MCL still has it's previous pose estimate, and is not capable to recover, because it samples its new pose around the old pose in the old area. Adjustments have been made [2] where extra random particles are injected when the localization performance is low. i.e. when all particle weights are low, the pose is likely not around, more random particles will be spawned. This way the robot can recover from being kidnapped, or just from a resampling step where the correct particles where discarded during resampling.

Adaptive MCL Adaptive Monte Carlo Localization (AMCL) uses a statistical bound on how far off the particle based posterior is from the real distribution. Based on this the number of particles needed to represent this posterior is estimated, and the sample size is adapted to this number. Thus, in AMCL the number of particles is adjusted to the estimated accuracy of the particle based posterior.

Appendix E

Shortest Path Algorithms

An important purpose of knowing the map and the robot location is to do navigation. When the robot is asked to travel from a start to an end point it will use a *path planning* algorithm to determine an optimal route. This optimal route can be the fastest, shortest or safest, whichever arbitrary property that makes a path optimal can be used.

E-1 Dijkstra's algorithm

The *Dijkstra's shortest path algorithm* [13] is often used to do path planning. It uses a graph representation with nodes connected with edges, often also called vertices and arcs. The nodes are places in the environment and the edges represent the cost of traversing form one node ot another. The algorithm will determine the costs of different routes by accumulating the costs of all traversed edges. The route with the smallest cost will be the solution to Dijkstra's algorithm. In Figure E-1 the optimal path will be S-C-D-E with a cost of 6.

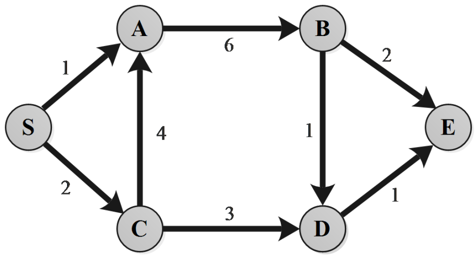


Figure E-1

The algorithm will begin from a start node and look the cost of traversing to each neighboring node. From there it will go to the first neighbor node and analyze for the cost of traversing to each of that nodes neighboring nodes, if not analyzed yet. While doing this it sums up the costs of going from the start node to each node in the graph, when the cost is already present it will update a new cost only if its lower. The algorithm continues until all nodes have been regarded. The result of Dijkstra's algorithm is a list of nodes together with the cost of traveling to that node and it's parent node e.g. the node one needs to travel to before traversing the last edge to the target node. From this the cheapest cost of traversing to each node can be seen and the path to arrive to each node can be reconstructing by following each parent node to the start node. The example shows a directed graph e.g. the edges can only be traversed in one direction. It is also possible to have an undirected graph where the edges can be traversed in both directions which do not necessarily need to contain the same cost. Pseudo code of Dijkstra's algorithm is shown in Algorithm 3. It requires a graph G containing vertices V, neighbors N for each vertex and a start vertex v_s.

Algorithm 3: Dijkstra's algorithm

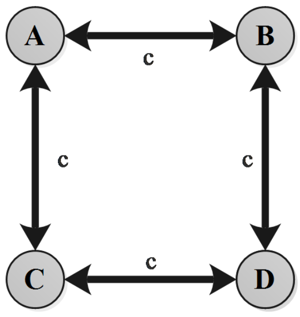
```
Input: $G(V, N), v_s$
$dist[v_s] \leftarrow 0$
$parent[v_s] \leftarrow v_s$
foreach $v \in V - \{v_s\}$ do
    $dist[v] \leftarrow \infty$
    $parent[v] \leftarrow \varnothing$
end
$S \leftarrow \emptyset$
$Q \leftarrow V$
while $Q \neq \emptyset$ do
    $u \leftarrow mindist(Q)$
    $S \leftarrow S \cup \{u\}$
    foreach $v \in N[u]$ do
        if $dist[v] > dist[u] + w(u, v)$ then
            $dist[v] \leftarrow dist[u] + w(u, v)$
            $parent[v] = u$
        end
    end
end
return $dist, parent$
```

E-2 Dijkstra for grid maps

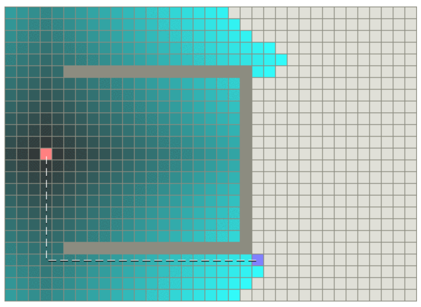
For grid maps Dijkstra's algorithm can be used by regarding every free grid as a node. The adjacent free grids are connected by edges with a constant cost. The grids can now be evaluated to find the path with the lowest cost. However, this method could become a large problem to solve for larger grid maps. Therefore Dijkstra's algorithm typically expands around the start point, exploring cells until it has found the end point, and finds the shortest route with the grids in between as shown in Figure E-2.

E-3 Other algorithms

Less computation time is used when less grids need to be evaluated, this is especially important for planning in grid maps. Some other algorithms like *Breadth-first search* and *A** based on Dijkstra's algorithm have been developed which have different policies to evaluate less grids. They are often more efficient, but this depends on the type of environment.



(a) a graph representing 4 grid cells



(b) shortest path and search space [14].

Figure E-2: Dijkstra's algorithm for grid maps

Appendix F

Homogeneous coordinates

Homogeneous coordinates can be used to describe points in coordinate frames and transforms between coordinate frames. Homogeneous coordinates make use of projective geometry e.g. adding an extra dimension, which allows points and transforms to be written in vectors and matrices on which operations can easily be performed. In this appendix only 2D operations are discussed and the operations necessary for understanding this work.

F-1 Notation

In Figure F-1 two coordinate frames and a point are seen in a 2D world. A point is regarded as a translation from a coordinate frame. The value of point p is thus coordinate frame dependent. Point p in frame 1 will be written as p^1. Point $p^1 = [4, 1]^T$ in the example. Between coordinate frames exist translations and rotations. A coordinate frame can only be expressed in another coordinate frame to get a value. This expression uses these rotations which contain x and y offsets and rotation θ. In the example the translation from frame 1 to 2 can be expressed as $T_1^2 = [2, 2, -\frac{1}{4}\pi]^T$. The transform from frame 2 to 1 is written as T_2^1 or $(T_2^1)^{-1}$, because it is actually the inverse of T_1^2.

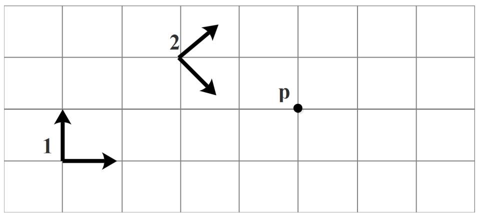


Figure F-1: A point and two coordinate frames in a 2D world

F-2 Operations

The operations that are disused are rotations, translations and combinations of these. To a 2D point or transform in projective geometry an extra dimension is added to be able to work with homogeneous operations. The transform can be written as a matrix containing a rotation $R(\theta)$, translation $t(x,y)$ and an extra dimension.

$$p^i = \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \tag{F-1}$$

$$T_i^j = \begin{bmatrix} R_i^j(\theta) & t_i^j(x,y) \\ \bar{0} & 1 \end{bmatrix} = \begin{bmatrix} cos(\theta) & -sin(\theta) & x \\ sin(\theta) & cos(\theta) & y \\ 0 & 0 & 1 \end{bmatrix} \tag{F-2}$$

This extra dimension allows transforms to be added to each other by means of multiplication. Let's say for example the transform T_1^2 and the point p^1 are known. If the point expressed in frame 2 is desired this can be calculated by doing in inverse of the transform between frame 1 and 2 to the point expressed in frame 1 (Eq. (F-3)). So by knowing the transform between frames points can be expressed in these frames using the transforms.

$$p^2 = T_2^1 * p^1 \tag{F-3}$$

As discussed before T_2^1 is actually the inverse of T_1^2. In homogeneous coordinates there is a simple method of calculating the inverse of a transform. Since it is a rotation in the opposite direction and a backwards translation which is dependent on the rotation of the coordinate frame it can be written in terms of R and t.

$$(T_i^j) = \begin{bmatrix} R_i^j(\theta)^-1 & R_i^j(\theta)^-1 * t_j^i(x,y) \\ \bar{0} & 1 \end{bmatrix}, \ with\ R(\theta)^{-1} = \begin{bmatrix} cos(\theta) & sin(\theta) \\ -sin(\theta) & cos(\theta) \end{bmatrix}, t_j^i = -t_i^j \tag{F-4}$$

Using these building blocks from homogeneous geometry points and coordinate frames can be rotated and translated with respect to a coordinate frame.

Appendix G

Iterative Closest Point

The Iterative Closest Point (ICP) algorithm can be used to find a Euclidean alignment of 2D or 3D point sets. The algorithm takes a alignment $\mathcal{A}$ and a reference $\mathcal{R}$ point set as input. It returns a transform which should align the alignment point cloud with the reference point cloud. There are many different versions of this algorithm, one of the better known, early, fully developed algorithms using ICP is created by Besl et al. in 1992 [15]. The steps of the most basic version will be explained in this appendix.
For each point $a_i \in \mathcal{A}$ the algorithm finds the closest point $r_i \in \mathcal{R}$ and it will pair these two points. The result is a set of paired points and the algorithm will try to find the optimal transform T_a^{r*} to minimize an error norm between the paired points.

$$f(T_a^{r*}) = min \sum_{i=1}^{n} ||r_i - a_i|| \tag{G-1}$$

Different versions of the ICP algorithm have different methods to find this transformation based on the paired point set. The found transform is applied on all points in $\mathcal{A}$. One iteration of the algorithm has now been executed. The next iteration the algorithm will again look for the closest pair point, this is necessary because the first found pair points might not actually be close to the alignment points after optimization. The closer the alignment set comes to the reference set the more likely it is the found pair are actually valid pairs in hindsight. The algorithm continues until an exit flag is reached, this might be a maximum amount of iterations or a minimum error norm to be reached. An example with two iterations is visualized in Figure G-1.
The ICP algorithm is a linear solution to the alignment problem, and it could easily get stuck in a local minimum. To be more sure to find the global minimum it is advised to use a nonlinear optimization method at a higher level. Often an estimate of the transform is requested a priori. By running the ICP algorithm various times with slightly varying initial transforms different solutions could be found and the transform with the smallest error could be used.
Different ICP algorithms have supplemented the basic steps with i.e. filtering outliers caused by noise in the point sets, or using other methods like point to plane in stead of point to point.

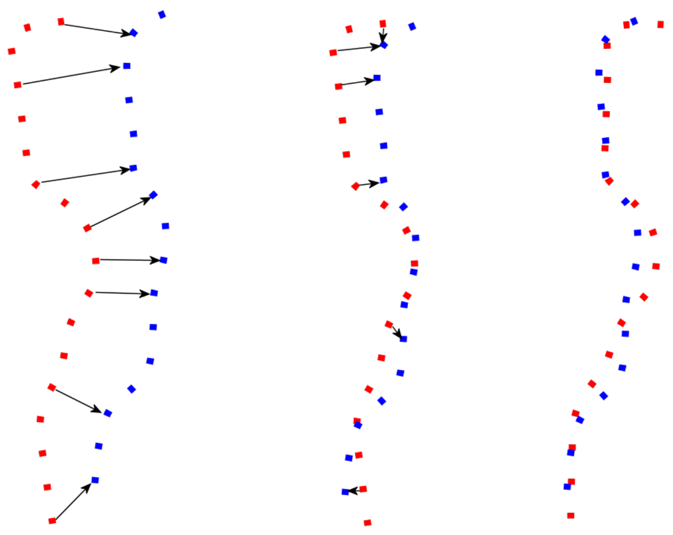


Figure G-1: Two iterations of the ICP algorithm. Alignment set in red, reference set in blue, pairs indicated with arrows

Bibliography

[1] C. Stachniss, “Course material for SLAM Course University of Freiburg.” http://ais.informatik.uni-freiburg.de/teaching/ws12/mapping/index_en.php, 2013. [Online; accessed Sep-2016].

[2] D. F. Sebastian Thrun, Wolfram Burgard, *Probabilistic Robotics.* MIT Press, 2005.

[3] M. Bosse, P. Newman, J. Leonard, M. Soika, W. Feiten, and S. Teller, “An atlas framework for scalable mapping,” in *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, vol. 2, pp. 1899–1906, IEEE, 2003.

[4] J.-L. Blanco, J.-A. Fernández-Madrigal, and J. Gonzalez, “Toward a unified bayesian approach to hybrid metric–topological slam,” *Robotics, IEEE Transactions on*, vol. 24, no. 2, pp. 259–270, 2008.

[5] M. Trauttmansdorff, “The interior of the faculty of industrial design engineering in delft..” http://michael.trauttmansdorff.ca/photoblog/archive/2013/03/23/ide-faculty/, 2013. [Online; accessed Sep-2016].

[6] M. Montemerlo, S. Thrun, D. Koller, B. Wegbreit, *et al.*, “Fastslam: A factored solution to the simultaneous localization and mapping problem,” in *Aaai/iaai*, pp. 593–598, 2002.

[7] M. Montemerlo and S. Thrun, “Fastslam 2.0,” *FastSLAM: A scalable method for the simultaneous localization and mapping problem in robotics*, pp. 63–90, 2007.

[8] G. Grisetti, C. Stachniss, and W. Burgard, “Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling,” in *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pp. 2432–2437, IEEE, 2005.

[9] H. Holendrecht, “A metro map of amsterdam.” http://hubholendrecht.nl/holendrecht-alle-metrotijden/, 2013. [Online; accessed Sep-2016].

[10] Niertransplantatie.info, “Ziekenhuis gang - leeg.” https://www.niertransplantatie.info/ervaringsverhalen-donoren/ik-ben-blij-dat-ik-mijn-dochter-kon-helpen.html, 2015. [Online; accessed Sep-2016].

[11] P. Cheeseman, R. Smith, and M. Self, “A stochastic map for uncertain spatial relationships,” in *4th International Symposium on Robotic Research*, pp. 467–474, 1987.

[12] F. Dellaert, D. Fox, W. Burgard, and S. Thrun, “Monte carlo localization for mobile robots,” in *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, vol. 2, pp. 1322–1328, IEEE, 1999.

[13] E. W. Dijkstra, “A note on two problems in connexion with graphs,” *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[14] A. Patel, “Dijkstra-trap.png.” http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html, 2016. [Online; accessed Oct-2016].

[15] P. J. Besl and N. D. McKay, “Method for registration of 3-d shapes,” in *Robotics-DL tentative*, pp. 586–606, International Society for Optics and Photonics, 1992.

Glossary

List of Acronyms

KF	Kalman Filter
EKF	Extended Kalman Filter
HMT	Hybrid Metric-Topological
MCL	Monte Carlo Localization
PF	Particle Filter
RBPF	Rao Blackwellized Particle Filter
SEIF	Sparse Extended Information Filter
SLAM	Simultaneous Localization and Mapping
UKF	Unscented Kalman Filter
AMCL	Adaptive Monte Carlo Localization
FoRMeT	Forced Resampling hybrid Metric Topological
ICP	Iterative Closest Point
ROS	Robot Operating System

Index