Master of Science Thesis in Electrical Engineering
Department of Electrical Engineering, Linköping University, 2020

Map Partition and Loop Closure in a Factor Graph Based SAM System

Emil Relfsson

LINKÖPING UNIVERSITY

Master of Science Thesis in Electrical Engineering

Map Partition and Loop Closure in a Factor Graph Based SAM System

Emil Relfsson

LiTH-ISY-EX-20/5350-SE

Supervisor: **Dr. Jonatan Olofsson**
ISY, Linköpings universitet
Jonas Nygårds
FOI

Examiner: **Professor Anders Hansson**
ISY, Linköpings universitet

Division of Automatic Control
Department of Electrical Engineering
Linköping University
SE-581 83 Linköping, Sweden

Abstract

The graph-based formulation of the navigation problem is establishing itself as one of the standard ways to formulate the navigation problem within the sensor fusion community. It enables a convenient way to access information from previous positions which can be used to enhance the estimate of the current position. To restrict working memory usage, map partitioning can be used to store older parts of the map on a hard drive, in the form of submaps. This limits the number of previous positions within the active map. This thesis examines the effect that map partitioning information loss has on the state of the art positioning algorithm iSAM2, both in open routes and when loop closure is achieved. It finds that larger submaps appear to cause a smaller positional error than smaller submaps for open routes. The smaller submaps seem to give smaller positional error than larger submaps when loop closure is achieved. The thesis also examines how the density of landmarks at the partition point affects the positional error, but the obtained result is mixed and no clear conclusions can be made. Finally it reviews some loop closure detection algorithms that can be convenient to pair with the iSAM2 algorithm.

Acknowledgments

I would like to thank Jonas Nygårds for the opportunity to do my master thesis at FOI and Jonatan Olofsson for the coordination and guidance from the perspective of the University during the project.

I would like to thank Jonas Nordlöf for all the help with practical stuff at FOI. I would also like to thank Oskar Karlsson and Max Holmberg for general help during the thesis.

Finally I want to thank all the personnel at FOI in Linköping for a great work environment.

Linköping, November 2020
Emil Relfsson

Contents

Notation

ABBREVIATIONS

abbreviations	Meaning
FOI	Swedish Defence Research Agency
SLAM	Simultaneous Localization And Mapping
GTSAM	Georgia Tech Smoothing And Mapping
GNSS	Global Navigation Satellite System
SAM	Smoothing And Mapping
IMU	Inertial Measurement Unit
LIDAR	LIght Detection And Ranging
ROS	Robot Operating System
PCL	Point Cloud Library
EKF	Extended Kalman Filter
RAM	Random Access Memory
ISAM	Incremental Smoothing And Mapping
TSAM	Tectonic Smoothing And Mapping
GLARE	Geometrical Landmark Relations
ICP	Iterative Closest Point
RMSE	Root Mean Square Error

1 Introduction

1.1 Background and Motivation

In military situations the ability to navigate and localize oneself is crucial for almost any unit. A common way is to use a Global Navigation Satellite System (GNSS) but since GNSS signals can be disrupted a parallel method is desirable.

A common way is to use Simultaneous Localization And Mapping (SLAM) which is a well known method within the sensor fusion and robotics community. SLAM uses a mathematical model of the system together with observations of landmarks with unknown position to estimate its position. At the same time it estimates the position of the landmarks which gives the method its name [1]. A smoothing approach to the SLAM is the Smoothing And Mapping (SAM) which considers the full or partial trajectory instead of only the most recent position for its estimate [2]. Since the trajectory is kept, it grants a convenient way to perform loop closures detection, recognising previously visited places. More information on both SAM and loop closures is found in Chapter 2.

1.2 System Overview

The thesis is performed at the Swedish Defence Research Agency (FOI) who has supplied a SLAM platform which will be used during the thesis. The platform has been developed by FOI to research positioning without GNSS for some general ground vehicle. The SLAM platform, referred to as the platform, can be divided into the hardware system and the software system.

The hardware system of the platform consists of an Inertial Measurement Unit

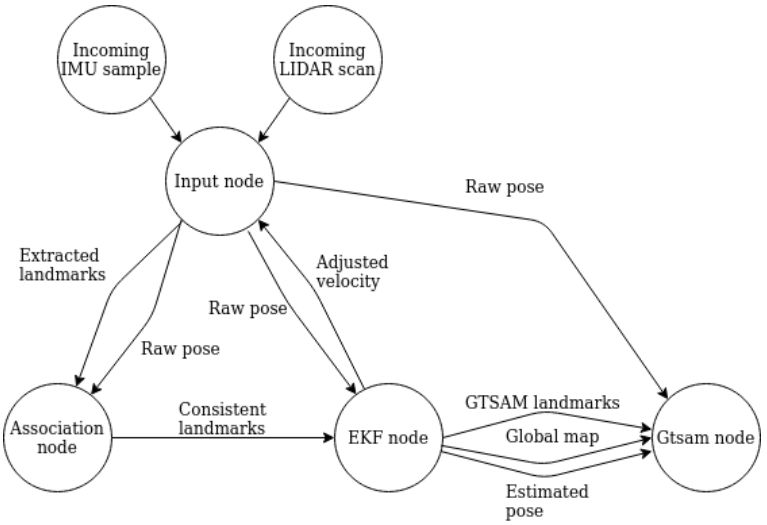

Figure 1.1: *A map over the program's different modules and some of the messages that is passed between them. Both the EKF node and the GTSAM node produces its own estimate of the trajectory and landmarks.*

(IMU), an onboard computer on which the software system is running and a rotating LIght Detection And Ranging (LIDAR) unit mounted on a vehicle.

The software system uses a robot development framework called Robot Operating System (ROS) [3] which manages the internal infrastructure of the software system. It uses Point Cloud Library (PCL) [4] to represent the LIDAR data and the Georgia Tech Smoothing And Mapping (GTSAM) library [5] to perform SAM.

The software system comprises four modules, called nodes in ROS, that are communicating via message passing. These four nodes are: Input node, Association node, Extended Kalman Filter (EKF) node and GTSAM node as seen in figure 1.1. The first three nodes are considered frontend and are used to get an initial estimate of the SLAM platform's position. The GTSAM node is considered as backend and is used to improve the initial estimate.

- The Input node is responsible for receiving data from the LIDAR and IMU. It extracts landmarks from the LIDAR scans and creates a raw estimate of the pose by integrating the IMU samples which it sends to the Association node, EKF node and GTSAM node. It also adjusts the velocity that it receives from the IMU with an estimated velocity from the EKF node.

- The Association node creates landmarks from different LIDAR scans and determines which landmarks are consistent and sends them to the EKF node.

- The EKF node is performing SLAM using an extended Kalman filter (EKF-

SLAM) to estimate the position of the platform as well as its global map. The pose estimated by the EKF node, referred to as the filtered pose is sent to the GTSAM node. The landmarks observed at the current moment by the platform, referred to as GTSAM landmarks and the global map are also sent to the GTSAM node.

- The GTSAM node uses a SAM algorithm to further improve the estimate of the position of the platform and the observed landmarks.

The Input node, Association node and EKF node are described in depth by [6].

1.3 Problem Formulation

The platform is able to estimate a map and position itself in that map but run into trouble when the platform is turned on for longer periods of time. The Random Access Memory (RAM) of the onboard computer fills up and the computer eventually crashes. To be able to handle this, data from the RAM needs to be transferred to the hard drive during runtime, thus freeing up space in the RAM. The part of the software system that uses the most RAM is the GTSAM node since it not only saves the position of the landmarks but also all the previous platform positions and all measurements of the landmarks. To limit this problem the estimated map in the GTSAM node can be partitioned into submaps. The previous submaps can be stored in on the hard drive which often is larger than the RAM. This thesis will explore what effect map partitioning has on the performance of the SAM algorithm — by itself and with a simple loop closure. The thesis aims to investigate the following questions:

- How can map partitioning be integrated into the SAM algortihm?
- How does the size of the partitioned submaps affect the estimated position of the platform, with and without loop closures?
- How does the number of landmarks at the partition point affect the estimated position with and without loop closures?
- Which loop closure detection techniques are reasonable to consider with respect to the platform?

1.4 Limitations

To be able to define the problem fully and to focus on the problem formulations, some limitations have been put on the project:

- The pre-existing platform uses factor graphs to represent the surroundings. Hence, the map partitioning will only explore how to partition a map that consists of a factor graph.

- When examining map partitioning with respect to loop closures, identification of the loop closure will be performed by hand to be able to fit within the time frame of the project.
- The submap is considered to be a set of landmarks and a trajectory. The thesis does not explore the possibility of storing isolated landmarks and positions.
- Only single loop closures will be considered when examining the effect of different map partitionings during loop closure.

2

Theory

2.1 Introduction

This chapter gives a theoretical background to the master thesis. It will touch upon three areas. The first part will present some background theory. The second part will examine map partitioning and a method to shift a position and its uncertainties to a different coordinate which will be needed for map partitioning. The third will introduce loop closure detection.

2.2 Background Theory

This section will present the Simultaneous Localization And Mapping problem, factor graphs, Bayesian trees and different Smoothing And Mapping algorithms.

2.2.1 Simultaneous Localization And Mapping

One of the main problems in the mobile robotics community is to localize oneself within an unknown environment. This can be achieved by incrementally creating a map and simultaneously use that map to estimate one's position. This challenge is referred to as the Simultaneous Localization And Mapping (SLAM) problem [7].

A SLAM system is considered a dynamic system since the output of a dynamic system is not only dependent on the input to the system at current sample but also of the input from previous samples. The information of the system at sample k, which can be used to predict the effect of inputs at different k is defined as the *state* of the system [8]. The current state is written as x_k and previous states of the system as x_{k-1}, x_{k-2}... A series of states is referred to as a trajectory.

The general problem formulation for SLAM can be expressed in the following way [1]:

$$x_{k+1} = f(x_k, u_k, v_k), \tag{2.1}$$

$$m_{k+1} = m_k, \tag{2.2}$$

$$y_k = h(x_k, m_k, u_k) + e_k. \tag{2.3}$$

- Equation (2.1) describes the motion model of the system and how the system moves between iterations. x_k is the state of the system and u_k the control signal. v_k is the modelling error and k the iteration index.
- Equation (2.2) describes the dynamics of the map with landmarks. In this case the map is considered static since the next iteration of the map is modelled as the current one.
- Equation (2.3) describes the measurement of the landmarks that ties together the map with the current state. The measurement model $h(x_k, m_k, u_k)$ describes the relation between the current state x_k, the map m_k, any input u_k and the measurement y_k. The measurement noise is represented by e_k.

Two key methods for solving this problem in the nonlinear case is to use Extended Kalman filters (EKF-SLAM) or particle filters (FastSLAM) which are described in both [1] and [7]. These methods will not be treated in this thesis.

A different variant of the SLAM problem is the SAM problem which is to estimate the trajectory of the system instead of just the current state. Apart from allowing more advanced state estimation, this also gives the advantage of having a dynamic linearization point which filter approaches do not have. Filter approaches have a static linearization point which can result in inconsistency in large scale problems [9]. The SAM problem can be formulated as a large scale inference problem, which can be solved using factor graphs [9].

2.2.2 Factor Graphs

Factor graphs are part of a family of probabilistic graphical models to which among others Bayesian networks and Markov random fields belong. This family of probabilistic graphical models are well known from literature on statistical modelling and machine learning. Factor graphs provide a powerful abstraction tool that can be used to solve large scale inference problem. They make it easier to think of and formulate solutions and to write modular and effective software to solve the problem [9].

A factor graph is made up of variable nodes, edges and factor nodes. The variable nodes can represent different variables of the system, such as the positions in the trajectory or observed landmarks. The variable nodes are connected via

lines, referred to as edges. On the edges, factor nodes are located which symbolize the statistical interactions between the variable nodes such as a measurement of a landmark [9].

A example of a factor graph is shown in Figure 2.1. The variable nodes that represent the position of the robot are illustrated by circles with a description key written within them. The variable nodes that represent the landmarks are illustrated by squares, also with a description key within them and the factor nodes as black dots between them. The figure shows a simple robot trajectory where x_i is the robot positions and l_j is the observed landmarks. A prior factor node is connected to to the first position x_0 to add prior information to the factor graph.

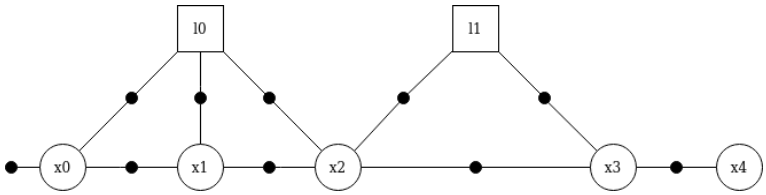

Figure 2.1: *A simple factor graph of a robot trajectory. The robot is transitioning from state x_0 through x_1, x_2, x_3 to x_4. The landmark l_0 is observed at state x_0, x_1 and x_2. Landmark l_1 is observed at state x_2 and x_3.*

A factor graph is closely related to Bayesian networks and conversion between them are a key operation for manipulations. One of the main differences between Bayesian networks and factor graphs are that Bayesian networks are described by directed graphs and can only handle proper, normalized, probabilities whereas factor graphs are undirected and can handle any factored function over the specified variables [9]. In a Bayesian network, the variable nodes contain the probability of the variables whereas in a factor graph, each factor node contains the statistical relation between two variables [9].

The conversion of a factor graph to a Bayesian network can be seen as a way to determine evaluation ordering of the inference for the graph [9]. The operation is referred to as elimination. Figure 2.2 shows the resulting Bayesian network produced by elimination of the example factor graph shown in Figure 2.1. For a linear system the elimination of factor graphs is equivalent to sparse matrix factorization such as QR-factorization [9]. The method is shown in Algorithm 1.

Input: Factor Graph
Result: Baysian network.
while *nodes* $\Theta \in$ *factor graph* **do**
1. Chose a variable node Θ_j to be eliminated. Disconnect all factor nodes $f(\Theta_j, \Theta_i)$ connecting Θ_j to its neighbour variables Θ_i, where i is the subscript of the connected neighbours. Define the *separator node* S_j as the connected variables Θ_i.
2. Form the (unnormalized) joint density $f_{joint}(\Theta_j, S_j) = \prod_i f(\Theta_j, \Theta_i)$ as the product of the disconnected factor nodes.
3. Using the chain rule, factorize the joint density $f_{joint}(\Theta_j, S_j) = P(\Theta_j|S_j)f_{new}(S_j)$. Add the conditional $P(\Theta_j|S_j)$ as a variable in the Bayes network and the factor $f_{new}(S_j)$ back into the factor graph replacing the disconnected factor nodes $f(\Theta_j, \Theta_i)$.

end
Algorithm 1: The elimination algorithm for converting a factor graph to a Bayes Network [10].

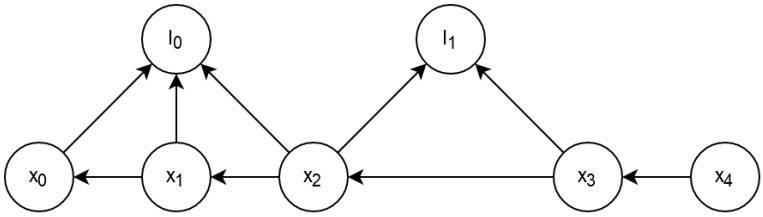

Figure 2.2: *The Bayesian network resulting from the elimination of the simple factor graph of a robot trajectory shown in Figure 2.1. The landmarks have been eliminated first, then the states in rising order from x_0 to x_4.*

2.2.3 SAM Through Inference

In SAM problems the objective is to estimate the trajectory of the system and the position of its surroundings. The problem can be seen as a factor graph, where the placement of the variable nodes Θ is desired. The problem becomes:

$$P(X, L, Z) \propto f(G) = \prod_{j \in g} \prod_{i \in n_j} f(\Theta_i, \Theta_j) \tag{2.4}$$

where P(X,L,Z) is the probability function of the system's positions X and the landmark's positions L given the measurements Z. G is the factor graph, Θ_i and Θ_j are variable nodes in the graph, f(Θ_i, Θ_j) the factor node connecting Θ_i and Θ_j, g is the set of variable nodes in the factor graph and n_j is the set of neighbours to variable node Θ_j [11]. If Gaussian noise is assumed and -log is applied to

Equation (2.4), it can be expressed as:

$$\sum_{j\in g}\sum_{i\in n_j} -\log f(\Theta_i,\Theta_j) \propto \Big(\sum_k \|f_k(x_{k-1})-x_k\|^2_{\Lambda_k}\Big)+\Big(\sum_m \|h_m(x_k,l_m)-z_{m,k}\|^2_{\Sigma_{m,k}}\Big) \tag{2.5}$$

where $f(x_{k-1})$ is the motion model, x_k is the system's state, $z_{m,k}$ is the measurement of landmark m, subscript k is the current sample, l_m is the landmark m and h_m is the sensor model for landmark m [11]. The norm notation stands for the squared Mahalanobis distance with the covariance matrix for the motion Λ_k and the covariance for the measurement $\Sigma_{m,k}$ [11]. An estimate Θ^* of Θ can be obtained by maximizing the joint probability $P(X,L,Z)$ which is the same as minimizing Equation 2.5. The estimate is given by:

$$\Theta^* = \underset{\theta_i}{\operatorname{argmin}} \sum_i \sum_{i+j} -\log f(\Theta_i,\Theta_j) \tag{2.6}$$

where Θ^* is the optimal node placements which corresponds to the optimal placements for the trajectory and landmarks [12].
Equation (2.5) and (2.6) yield the following equation through linearization:

$$\Delta^* = \underset{\Delta}{\operatorname{argmin}} \sum_k \|F_k^{k-1}\Delta x_{k-1} - G_k^k \Delta x_k - a_k\|^2_{\Lambda_k} + \sum_m \|H_m^k \Delta x_k + J_m \Delta l_m - c_m\|^2_{\Sigma_{m,k}} \tag{2.7}$$

where Δ is the change to the current linearization point θ_{lin}, F_k^{k-1} is the Jacobian of $f_k(x_{k-1})$ from k-1 to k, G_k^k is an identity matrix, H_m^k and J_m are the Jacobians of h with respect to x_k and l_m respectively. a_k is the state prediction error $a_k = x_k^0 - f_k(x_{k-1})$ and c_m is the measurement prediction error $c_m = z_{m,k} - h_m(x_k^0, l_m^0)$, where x_k^0 and l_m^0 are linearization points for state k and landmark m [2]. By collecting all the components into one large linear system the following equation can be obtained:

$$\Delta^* = \underset{\Delta}{\operatorname{argmin}} \|A\Delta - b\|^2 \tag{2.8}$$

A is the combined Jacobian matrix from F_k^{k-1}, H_m^k and J_m. b is a_k and c_m collected in one term. For full derivation of Equation (2.8), see [2]. Equation (2.8) can be solved through QR factorization [11]. The step by step QR factorization is shown in Appendix A.1.

2.2.4 Incremental Smoothing and Mapping

In the post-processing case when all measurements are available a standard SAM can be applied but when the measurements are incrementally updated, the full optimization of the graph at each iteration quickly becomes computationally costly. [11] presents a method to incrementally perform SAM, called incremental Smoothing And Mapping (iSAM). It uses the so called Givens rotation to obtain the QR factorization. Though Givens rotation is not the preferred way to perform QR factorization it gives the ability to extend a system equation with new

measurements [11]. QR factorization results in the upper triangular information matrix R, which can be extended with new lines from new measurements. Givens rotation can then be applied to the extended matrix to make it upper triangular again. The linear system can then be efficiently solved, using back substitution, to attain Θ^*. The Givens rotation matrix is given by

$$G(\theta) = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}. \tag{2.9}$$

Since more measurements are constantly added, the linearization point of the system will become more and more uncertain. To limit the number of relinearizations of the system equation an iteration count is used to determine if relinearization is necessary [11].

2.2.5 Bayes Trees

The iSAM algorithm operates in a factor graph environment but the estimation part of the algorithm is implemented using matrix algebra. To be able to stay in the graph environment and better capture the algebra, [13] have introduced a new graph structure. It is derived from the Bayesian network that is formed from elimination of a factor graph but has a tree structure and is called a Bayes tree [13].

The Bayes tree is made up by nodes connected by edges and has an upside down tree structure. It starts at the top with the first node called the root (node). The root usually represents the most recent node and contains the latest eliminated node from the factor graph (often the current state). The root has children which are nodes located beneath it in the tree, see figure 2.3. The root is referred to as its children's parent. A child can in turn have children and be their parent. This pairwise relationship continues throughout the tree down to the bottom layer, see figure 2.3.

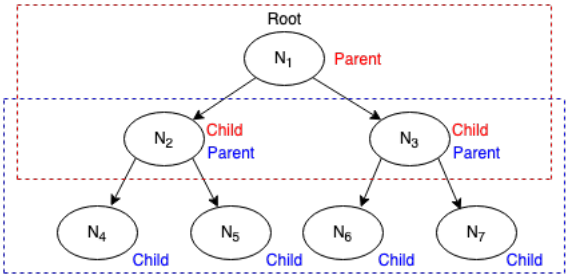

Figure 2.3: *The tree structure of a Bayesian tree. N_1 is the root and parent to N_2 and N_3 which in turn are children to N_1. N_2 is parent to N_4 and N_5 which in turn are children to N_2. N_3 is parent to N_6 and N_7 which in turn are children to N_3.*

The nodes in the Bayes Tree are called *cliques*, C_i which correspond to sets of nodes in the Bayesian network which are grouped together using the maximum cardinality search algorithm presented in [14]. Two examples of cliques are shown in Figure 2.4 and 2.5. When nodes from the Bayesian network are grouped inside a clique C_i in a Bayes tree they are referred to as the clique's variables. Each C_i has a conditional probability which is the product of the conditional probabilities from its variables. It forms the product

$$P(C_i) = \prod_k P(F_k|S_k) \tag{2.10}$$

where S_k is the *separator* and F_k is the *frontal variable*. The separators are the nodes/variables from the Bayesian network that separate the clique C_i from its parent but are part of both the C_i and its parent. The frontal variables are the remaining nodes that are not shared with its parent. Conversion of a Bayesian network to a Bayes tree is shown in Algorithm 2 and Figure 2.6 shows the Bayesian tree resulting from the Bayesian Network shown in Figure 2.2.

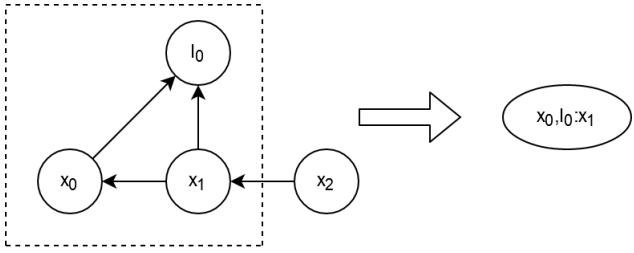

Figure 2.4: *A clique formed from x_0, x_1 and l_0 in the Bayesian network. x_1 is the separator since it separates x_0 and l_0 from the next node x_2.*

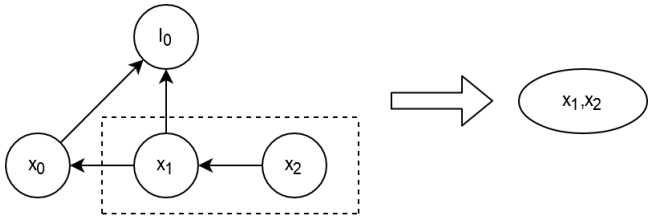

Figure 2.5: *A clique formed from x_1 and x_2 in the Bayesian network.*

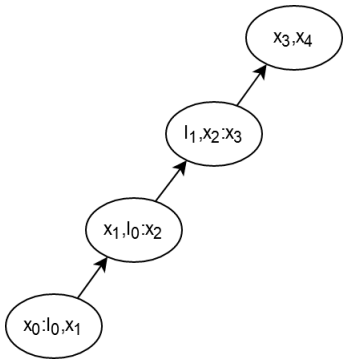

Figure 2.6: *The Bayes tree resulting from the conversion of the Bayesian network in Figure 2.2.*

Input: Bayesian network
Result: Bayes tree
for *Conditional density* $P(\Theta_j|S_j)$ *of the Bayes network, in reverse elimination order:*
do
  if *No parent* ($S_j = \{\}$) **then**
    Start a new root clique F_r containing Θ_j
  else
    Identify parent clique C_p that contains the first eliminated node of S_j as a frontal variable
    if *nodes* $F_p \cup S_p$ *of parent clique* C_p *are equal to separator nodes* S_j *of conditional*
    then
      insert conditional into clique C_p
    else
      start new clique C' as child of C_p containing Θ_j
    end
  end
end

Algorithm 2: Creating a Bayes tree from a Bayesian Network resulting from elimination of a factor graph [10].

When a Bayes tree is edited, the only part of the tree that is affected are the cliques from the edited clique up to the root. The cliques below are not affected which is a key property of the Bayes tree [13]. A Bayes tree has this property because variables in a child clique are eliminated before variables in its parent clique, making the variables of the child clique unaffected by the variables in the parent clique [13].

2.2.6 iSAM2

To evolve the iSAM algorthm, [10] uses the new data structure, Bayesian tree, to perform its optimization in a graph environment instead of using a matrix environment. This makes it easier to understand and write efficient and modular code. The new algorithm is called iSAM2 [10].

Besides staying in a graph environment iSAM2 differs from the previous iSAM in two major ways. The first one is that the relinearization is done in a more dynamic and efficient way. iSAM2 keeps track of all the factor graph nodes' linearization points and compares them to the current solution to see if relinearization is needed. If any linearization point for a variable moves too far from the current solution the variable is marked. At the next measurement update, all marked variables are relinearized. If a marked variable is part of a clique in the Bayes tree, all the variables within that clique are relinearized. All the parent cliques from the affected clique to the root clique are also relinearized. It is referred to as fluid relinearization [10] and shown in Algorithm 3.

Input: linearization point θ_{lin}, Solution Δ for linearized tree.
Result: Updated linearization point θ'_{lin}, marked cliques M.
1. Mark variables in Δ above threshold β: $J = \{\Delta_j \in \Delta \mid |\Delta_j| \geq \beta\}$
2. Update linearization point for marked variables: $\theta_{lin,J} := \theta_{lin,J} \oplus \Delta_J$
3. Mark all cliques M that involve marked variables $\theta_{lin,J}$ and all their ancestors.

Algorithm 3: Fluid relinearization [10].

The second difference from iSAM is that iSAM2 updates its solution for the Bayes tree only partially each iteration instead of fully. It starts at the root of the tree and compares the difference between the previous solution with the new solution for all the variables in the clique. If the change in difference exceeds a small given threshold α between two iterations, it updates the solution and continues to the clique's children. If the change in difference doesn't exceed α , the update stops and does not recurse to update the clique's children. This update is referred to as partial state update [10] and shown in Algorithm 4.

Input: Bayes tree
Result: Updated solution Δ to current linearization point θ_{lin}
Starting from the root clique $C_r = F_r$:
1. For current clique, $C_k = F_k{:}S_k$, compute updated Δ_k of frontal variables F_k from local conditional density $P(F_k|S_k)$.
2. For all variables Δ_{k_j} in Δ_k that change by more than threshold α: recursively process each descendant containing such a variable.

Algorithm 4: Partial state update: Solving the Bayes tree in the nonlinear case returns an update solution Δ to the current linearization point θ_{lin} [10].

The full algorithm works as follows: It starts by collecting the measurements

and organizing them into a factor graph. The new factor graph is linearized around the initial linearization point θ_{lin}. An elimination order is then calculated with an algorithm called "constrained COLAMD" [15] to conserve sparsity before the factor graph is eliminated into a Bayesian network using Algorithm 1. The Bayesian network is in turn converted into a Bayes tree using Algorithm 2. This Bayes tree, representing the optimization in Equation (2.6), can be evaluated using back propagation to find the optimal solution Δ which, when added to Θ_{lin}, solves Eq. (2.6) for the given time.

New measurements update the factor graph at the next time step by adding new linearized factors to it. The new factors are eliminated to the Bayesian Network and added to the Bayes Tree. Algorithm 3 is used to mark any linearization points that differ too much from the current solution. The top of the Bayes tree containing the marked variables is redone with Algorithm 5 and the linearization point θ_{lin} is updated. A new state update $\Delta_{updated}$ is obtained with Algorithm 4. The updated solution is then given by $\Delta_{updated} + \theta_{lin}$ [10].

Input: Bayes tree T, nonlinear factors F, affected variables J
Result: Modified Bayes tree T'
1. Remove top Bayes tree:
   a, For each affected variable in J, remove the corresponding clique and all parents up to the root.
   b, Store orphaned sub-trees T_{orph} of removed cliques.
2. Relinearize all factors required to recreate top.
3. Add cached linearized factors from T_{orph}.
4. Re-order variables.
5. Eliminate the factor graph (Alg. 1) and and create new Bayes tree (Alg. 2).
6. Insert the T_{orph} back into the new Bayes tree.

Algorithm 5: Updating the Bayes tree inclusive of fluid relinearization by recalculating all affected cliques [10].

2.3 Map Partition and Manipulations

This section will present different ways to perform map partitioning, different ways to link partitioned maps together and how to optimize them. The partitioning of the map is a crucial operation since it enables the system to store submaps inactive in long term memory which is necessary for longer runs.

The aim of the available methods is to optimize the solution time for big maps by partitioning them into submaps, under the assumption that the system can store the full map in the RAM. The objective of this thesis is rather to use map partitioning to limit the size of the the continuously growing map by storing inactive submaps on the hard drive of the system. The methods can however be seen as inspiration and with some modification suited for this thesis.

2.3.1 Tectonic Smoothing And Mapping

In [16] a method for solving more complex and bigger maps is presented. It is called Tectonic Smoothing And Mapping (TSAM). The main idea is to partition the map into smaller submaps which are easy to optimize and then assemble the solved submaps into the full map. The information from the solved submaps together with a reduced version of the full map is used to optimize the full map.

The method starts by partitioning the map into submaps. The nodes within the submaps are divided into two sets, the internal nodes and the separators. The internal nodes are the nodes that are only connected to nodes within the submap and the separators are the nodes that are connected to nodes within other submaps. The submaps are linearized but only the internal nodes are optimized, using the measurements which are associated to features of that submap. The separators are cached for the global optimization. Further a node is added to each submap, referred to as a base node (see figure 2.7). The optimized internal nodes are parameterized relative to the base node so that the internal structure and linearization point of the submap is kept intact when global optimization is performed.

The global set of all separators is extended with the base nodes and optimized. The global optimization of separators and the base nodes can be seen as an alignment of the submaps.

The final step of the algorithm is to update each submap with the updated separators and to update the internal nodes in relation to the separators.

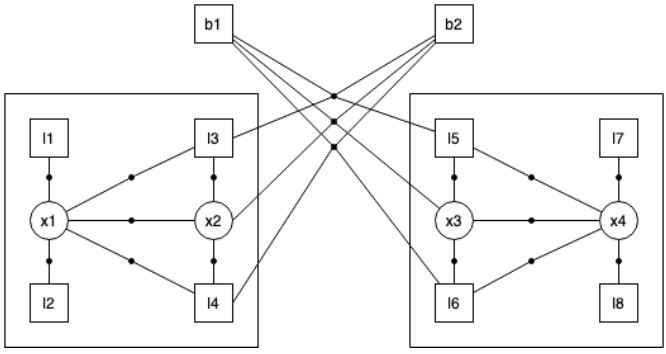

Figure 2.7: *A divided factor graph with added base nodes. The positions are marked x1 to x4 and the landmarks l1 to l8 . The base node b1 summarizes the left submap and the base node b2 summarizes the right submap.*

2.3.2 TSAM2

In [12], improvements are proposed to enhance the TSAM algorithm. The idea is to use a nested dissection to recursively divide the map into smaller submaps

where direct optimization methods can be used.

Consider the original graph G, which through nested dissection partitioning can be divided into the three subsets A_i, B_i and C_i. The partition is performed so that A_i and B_i don't share any factors. Those are called *frontal variables* whereas C_i shares factors with both A_i and B_i, and is called *separator*. A_i and B_i can in turn be seen as disjoint graphs which can be be further partitioned into smaller submaps. This recursive partitioning can continue until the resulting submaps are small enough.

When the partitioning is finished, the submaps at the lowest level can easily be solved with direct methods. For each solved submap a base node is added from which the other nodes' relative position and orientation will be stored. The base node will then be added to the parent submap instead of the nodes of the submap, representing the position and orientation of those nodes. The parent submap will be optimized and replaced with a base node in its parent submap and so on. This is performed all the way up to the root.

Once the optimization is complete, the submaps can be optimized again but this time the base nodes stay fixed within each submap resulting in an optimization equivalent to a traditional SAM optimization [2].

2.3.3 Condensed Measurement

Similar to TSAM, Condensed Measurements [17] is also a divide-and-conquer method where the total map is divided into smaller submaps which can be solved separately. [17] proposes a method to reduce the number of nodes in the submaps. The method then compresses the information from the reduced nodes into virtual measurements referred to as condensed measurements. The condensed measurements are then connected to the remaining nodes of the submaps. The reduced submaps will be much smaller and, when assembled to a sparse factor graph, allows a global optimization to be performed. Again, this global optimization can be seen as an alignment of the submaps. The submaps are expanded to their original number of nodes with the configuration from the global optimization leading to an approximate solution. The global solution can be further improved if needed by optimizing the full factor graph using the approximate solution as initial estimate. The following sections will explain the method more thoroughly.

Partitioning the Map

Since the maps are represented by factor graphs, a partitioning of the factor graph is needed, see Figure 2.8a. All the information in the factor graphs is located in factors and to avoid using global information multiple times the factor graphs are partitioned with respect to the factors. This means that no submap will contain the same factors but can contain the same nodes. The nodes that appear in multiple submaps are referred to as *shared nodes*, x_i, shown in red in figure 2.8b. An

origin node x_g is determined as the position that lies in the middle of the trajectory of each submap, shown in blue in figure 2.8c. The submap optimization and calculation of marginal covariances are performed separately for each submap.

In the submap optimization, only measurements associated to fully observable landmarks are used — the rest are temporarily removed. This can occur when for instance using bearing only measurements which need two measurements of a landmark to become fully observable and the partition point of the map appears between them. The unused measurements can still be used in the global optimization of the full factor graph if their landmarks are fully observable there.

Computing the condensed measurement

When the submaps have been formed and solved, the condensed measurement can be calculated. A family of measurement functions is defined as

$$h^{typeOf(x_i)}(x_g, x_i) \triangleq h(x_g, x_i) \tag{2.11}$$

where h is the measurement function which depends on the type of the shared node x_i. Assuming that the submaps have been solved correctly the following term in the joint probability function for the graph, see Equation 2.5, ideally becomes

$$h(x_g^*, x_i^*) - z_{g,i} = 0. \tag{2.12}$$

To take errors into account, we can define the condensed measurement,

$$z_v \triangleq z_{g,i} = h(x_g^*, x_i^*) \tag{2.13}$$

between origin node x_g and the shared node x_i [17]. The measurement will represent the relation between the origin node and the shared node. The uncertainty of the measurement can be approximated from the marginal covariance of the shared node using e.g. the unscented transform [1] applied to its measurement function. A reduced submap is formed by the origin node x_g and the shared nodes x_i connected by condensed measurements as shown in figure 2.8d.

Calculating the global estimate

When the submaps have been reduced they are added together to a sparse factor graph. The graph is optimized to get the approximate position and orientation of each submap, as in Figure 2.8e. The submaps are expanded according to that configuration, giving an approximate solution to the full global factor graph, see Figure 2.8f. The full global factor graph can be optimized with the approximate solution as initial guess if needed to gain a better estimate.

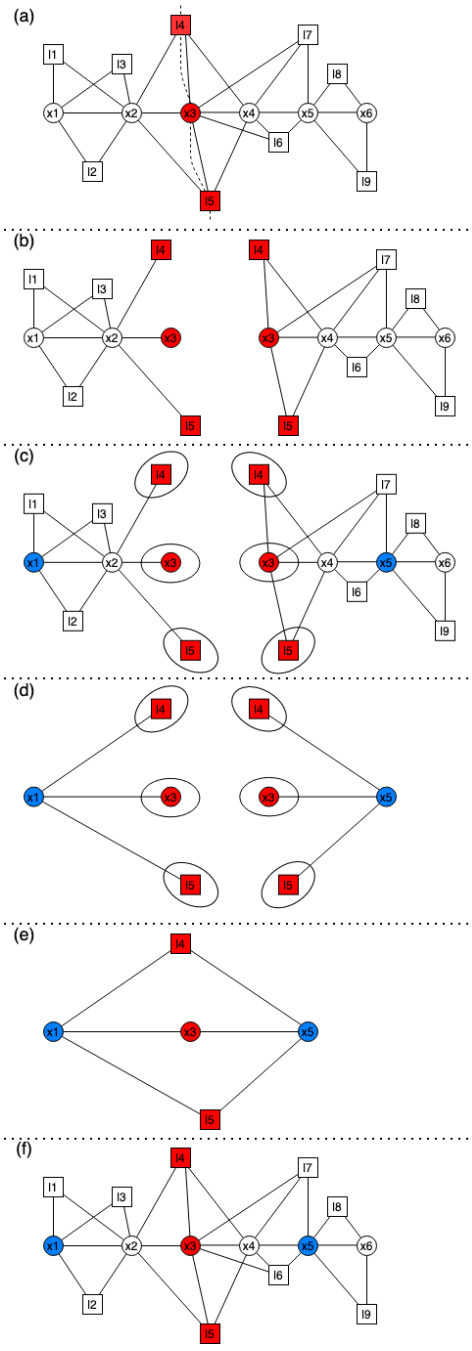

Figure 2.8: *a), The partition of a map. b), The shared nodes shown in red. c), The origin nodes shown in blue. d), The two submaps condensed to submaps only containing shared and origin nodes. e), The two condensed submaps globally optimized to find the approximate position of the origin and shared nodes. f), The submaps reunited using the origin nodes position.*

2.3.4 Uncertain Spatial Relationships

[18] presents an alternative method to the unscented transform that is used when calculating the covariances of the condensed measurements in section 2.3.3. It is a way to describe poses and their uncertainties in different world frames. The method both describes the pose, referred to as p in this section which contains both the position and orientation, together with the covariance, $C(p)$, of that pose in different world frames. The two operators presented by [18] are $\oplus$ and $\ominus$. The $\oplus$ operation adds two 2D poses in different world frames resulting in a pose in the combined world frame, see Figure 2.9. The $\ominus$ flips a 2D pose putting it in the origin of the global frame and putting the origin of its frame in its previous pose, see Figure 2.10. [18] also presents a 3D version of the operators but is not shown here.

Adding two poses

When adding two poses p_{ij} and p_{jk} the $\oplus$ operator is used

$$p_{ik} = p_{ij} \oplus p_{jk} = \begin{bmatrix} x_{jk}\cos\phi_{ij} - y_{jk}\sin\phi_{ij} + x_{ij} \\ x_{jk}\sin\phi_{ij} + y_{jk}\cos\phi_{ij} + y_{ij} \\ \phi_{ij} + \phi_{jk} \end{bmatrix} \tag{2.14}$$

where x, y and ϕ is the x-coordinate, y-coordinate and bearing of the pose p. Besides from adding the two poses the $\oplus$ operator produces the matrix $J_\oplus$ which can be used to approximate the covariances for the new poses

$$C(p_{ik}) \approx J_\oplus \begin{bmatrix} C(p_{ij}) & C(p_{ij}, p_{jk}) \\ C(p_{jk}, p_{ij}) & C(p_{jk}) \end{bmatrix} J_\oplus^T \tag{2.15}$$

where $C(p)$ is the covariance of p and $C(p_1, p_2)$ is the cross covariance of p_1 and p_2. $J_\oplus$ is the Jacobian of the $\oplus$ operator which is given by

$$J_\oplus = \frac{\partial p_{ik}}{\partial (p_{ij}, p_{jk})} = \begin{bmatrix} 1 & 0 & -(y_{ik} - y_{ij}) & \cos\phi_{ij} & -\sin\phi_{ij} & 0 \\ 0 & 1 & -(x_{ik} - x_{ij}) & \sin\phi_{ij} & \cos\phi_{ij} & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}. \tag{2.16}$$

The $J_\oplus$ matrix can be divided into two 3x3 matrices $J_{1\oplus}$ and $J_{2\oplus}$. They can be used separately if p_{ij} and p_{jk} are independent, shown below

$$C(p_{ik}) \approx J_{1\oplus} C(p_{ij}) J_{1\oplus}^T + J_{2\oplus} C(p_{jk}) J_{2\oplus}^T \tag{2.17}$$

resulting in $J_\oplus = [J_{1\oplus} \quad J_{2\oplus}]$.

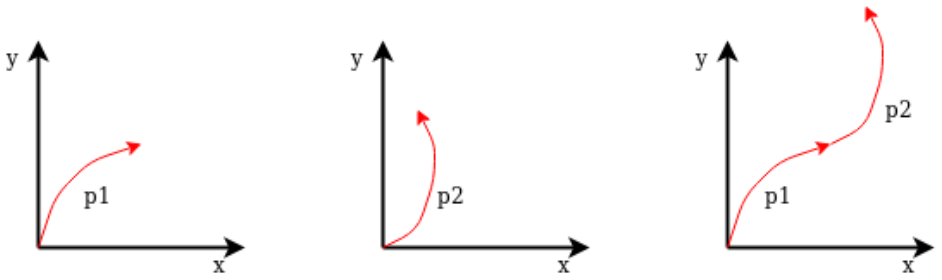

Figure 2.9: *The operation $\oplus$ is performed on two different positions, p1 and p2.*

The inverse of a pose

To be able to subtract a pose from another the $\ominus$ operator is defined as

$$p_{ji} = \ominus p_{ij} = \begin{bmatrix} -x_{ij}\cos\phi_{ij} - y_{ij}\sin\phi_{ij} \\ x_{ij}\sin\phi_{ij} - y_{ij}\cos\phi_{ij} \\ -\phi_{ij} \end{bmatrix} \tag{2.18}$$

The $J_\ominus$ matrix is given by

$$J_\ominus = \begin{bmatrix} -\cos\phi_{ij} & -\sin\phi_{ij} & y_{ji} \\ \sin\phi_{ij} & -\cos\phi_{ij} & -x_{ji} \\ 0 & 0 & -1 \end{bmatrix}.$$

The $J_\ominus$ matrix can be used to calculate the new covariance in the following way

$$C(p_{ji}) \approx J_\ominus C(p_{ij}) J_\ominus^T. \tag{2.19}$$

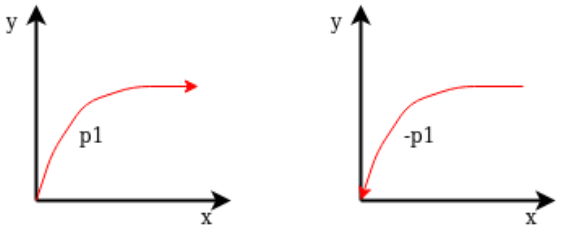

Figure 2.10: *The operation $\ominus$ is performed on position p1 resulting in the reverse p1.*

Adding a pose and the inverse of an another pose

The two operations $\oplus$ and $\ominus$ can be combined to to get the difference of two poses, shown in figure 2.11.

$$p_{jk} = \ominus p_{ij} \oplus p_{ik} \tag{2.20}$$

and the resulting covariance can be approximated as

$$C(p_{ik}) \approx J_{\oplus} \begin{bmatrix} J_{\ominus} C(p_{ij}) J_{\ominus}^T & C(p_{ij}, p_{jk}) J_{\ominus}^T \\ J_{\ominus} C(p_{jk}, p_{ij}) & C(p_{jk}) \end{bmatrix} J_{\oplus}^T. \tag{2.21}$$

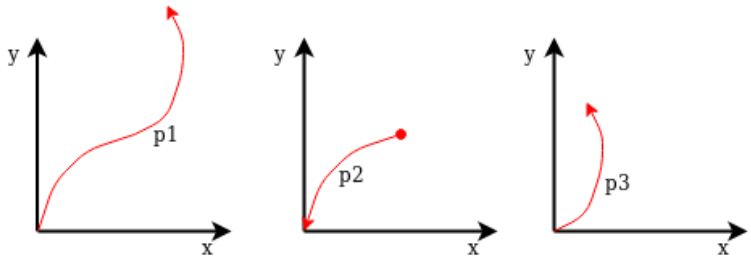

Figure 2.11: *The combined operaton of $\ominus$ and $\oplus$ is performed on pose p1 with the reverse pose p2 resulting in p3.*

2.4 Loop closure

A common way to restrict the ever growing covariance in a SLAM problem is to use loop closure. A loop closure recognizes a previously visited place and uses that information to form a loop. The trajectory that forms the loop can then be optimized with the added loop closure constraint, resulting in a better estimate and lower covariance. Algorithms that can detect loop closures are investigated here as a first step towards full loop closure functionality.

2.4.1 Lidar Histogram Methods

The general LIDAR histogram method for loop closure detection uses histograms as a signature of an area which can be compared with a global set of histograms. If any of the histograms in the global set resembles the given one, they are marked as candidates. A more precise method can then be used to determine which, if any, candidate is the right one. The method projects the landmarks from a LIDAR scan to the same or a lower dimension. The function that projects the landmarks usually calculates some property between landmarks such as absolute distance or some property between a landmark and the system, which also can be absolute distance. The projection is done for all the different configurations of nearby landmarks and added together and put into bins, forming a histogram. The histogram can then be compared via various numbers of metrics to find candidates for detection [19].

1D histogram method

In [19] a full 360 degree LIDAR scan is used to create the histograms. The distance between the system and a measured landmark is put in a 1d bin that quan-

tizes the value into one of b different intervals which gives the quantized value h_b. The quantized value is normalized with the absolute distance between the system and all the landmarks in the scan in the following way

$$h_b^k = \frac{1}{|S|}\{p \in S : v(p) \in I_b^k\} \tag{2.22}$$

where S is the scan, p is a landmark in the scan, v is distance operator and I is the interval for a bin in the histogram [19]. This is done for all the landmarks in the lidar scan for a given area which results in the histogram $H_b = (h_b^0, ..., h_b^{b-1})$, where the quantized values that belong to the same bin are added together. For histogram comparison a simplified version of the so called Wasserstein metric [19] is used which in this configuration becomes

$$W(G_b, H_b) = \sum_i \frac{1}{b} \left| \sum_{j=0}^{i} \left[g_b^i - h_b^j \right] \right| \tag{2.23}$$

where G_b is the current histogram, H_b a candidate, g_b^i is a bin value i for histogram G_b and h_b^j the bin value j for histogram H_b [19]. The metric is used against a threshold to find candidates for loop closure [19].

GLARE

In [20] a 2d lidar histogram method is presented called Geometrical LAndmark RElations. A set of landmarks, $l_1, l_2, ..., l_N$, is chosen to be examined. The relative distances $\rho_{i,j}$ between the landmarks, together with the relative bearing, $\theta_{i,j}$ is obtained from the observed positions prospective. Since the bearing between two landmarks depends on which one is considered first the positive bearing is chosen $\theta_{i,j}^+ = max(\theta_{i,j}, \theta_{j,i})$. The relative distance and positive bearing are quantized and put in a 2d $bin(n_\rho, n_\theta)$ which creates the bin value $h_{i,j}$. A multivariate Gaussian distribution with covariance matrix $\Sigma_{i,j}$ is added to the landmark relation. A bin value is obtained for all the combinations of landmarks and bin values that correspond to the same bin are added together. The set of bins is then collected into a 2d histogram H_A and normalized for that given area. The histogram can then be compared with a set of saved histograms H with a L1-norm to achieve loop closure [20].

GLAROT

A modification to the previous method is presented by [21], making it bearing invariant. This is is achieved by adding the general angle β to the positive bearing $\theta_{i,j}^+$ which results in a new bearing

$$\theta_{new,i,j} = \langle \theta_{i,j}^+ + \beta \rangle_\pi \tag{2.24}$$

where $\langle x \rangle_m$ is the m modulo of x [21].
The comparison between two histograms using L1 norm then becomes a minimization problem

$$L_1(F, G) = \min_{\beta} \sum_{t=0}^{n_\theta - 1} \sum_{r=0}^{n_\rho - 1} |F_{t,r} - G_{\langle t+\beta \rangle_{n_\theta}, r}| \tag{2.25}$$

where n_θ is the number of bearing bins, n_ρ is the number of range bins, F and G are two different histograms and G is rotated to minimize the norm [21].

GLAROT-3D

In [22] an even more comprehensive histogram method is presented. This one uses 3d landmarks instead of 2d which adds extra information to the histogram. Is uses the same distance comparison $\rho_{i,j} = |p_i - p_j|$ between two landmarks that are put in a bin. The bin is extended to handle 3d bearings in polar coordinates using a quantized polar sphere. The polar sphere is divided in six faces, creating a cube. A face, f, can in turn be divided into a l*l grid, shown in Figure 2.12. The face of a bearing is determined by

$$f = \underset{f}{\operatorname{argmin}}\, d_f^T r_{i,j} \tag{2.26}$$

where d_f^T is the normal vector of the face and $r_{i,j}$ is the bearing between two points [22]. When a face is determined a simple 2d quantization is performed to divide the face into a 2d grid. The square in the grid that corresponds to the polar bearing is calculated in the following way:

$$u = l * \left(\frac{2}{\pi} \arctan\left(\frac{u_f^T r_{i,j}}{d_f^T r_{i,j}} + \frac{1}{2} \right) \right) \tag{2.27}$$

$$v = l * \left(\frac{2}{\pi} \arctan\left(\frac{v_f^T r_{i,j}}{d_f^T r_{i,j}} + \frac{1}{2} \right) \right) \tag{2.28}$$

where u is the horizontal coordinate and v is the vertical [22]. The quantized polar bearing and distance are used to form a histogram. A stored histogram can then be compared with the current one using the rotated L_1-norm defined as

$$RL_1(F, G) = \min_{R \in \mathbb{R}} \|F - RG\|_1 \tag{2.29}$$

where F and G are two different histograms and R the rotation to a quantized bearing [22].

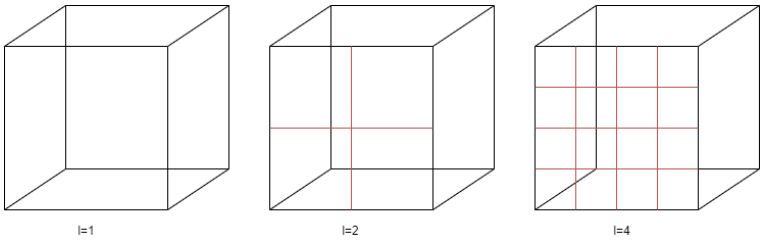

Figure 2.12: *The cube represent the 6 faces of the quantized polar coordinate sphere. Within a face a grid depending on l can be formed.*

Scan Context

[23] proposes a method where the whole point cloud in a LIDAR scan of a location is compared with another location. The idea is to create a matrix that corresponds to the position of points in the point cloud. The perception of the system is divided into 2D bins creating a matrix where one dimension corresponds to the range between a point and the origin of the system and the other to the bearing. When cloud points are detected in a bin, the point with the highest z-coordinate is chosen to represent that bin with its z-coordinate. The result becomes a low resolution height map of the surroundings.
To improve robustness the current point cloud is copied and root shifted into N_{trans} copies. The copies are then also transformed into 2D bin matrices. All the copies and the original 2D bin matrix will be referred to as the query matrices.
A pre-search is then performed to limit the number of candidates for the full matrix. The pre-search starts by adding the rows of the 2D bins' matrices together making a k-vector for each query matrix.

$$k = (\phi(r_1), .., \phi(r_{N_r})) \tag{2.30}$$

where $\phi(r_x)$ is the sum of all bearing bins at range bin r_x [23]. The k-vectors of the query matrices are then compared to find candidates for loop closure. Since all the bearing bins are summed together the comparison becomes bearing invariant. When candidates are found the full matrices are compared between the candidates and current matrix. The cosine distance between the matrices are calculated with respect to minimum shift

$$D(I^q, I^c) = \min_{n \in [N_s]} d(I^q, I^c_n) \tag{2.31}$$

where N_s is the number of bearing bins in the scans, I^q is the current matrix and I^c_n the candidate matrix shifted n number of bearing bins [23]. The cosine distance $d(I^q, I^c)$ is is calculated in the following way

$$d(I^q, I^c) = \frac{1}{N_s} \sum_{j=1}^{N_s} \left(1 - \frac{c^q_j \cdot c^c_j}{\|c^q_j\| \cdot \|c^c_j\|}\right) \tag{2.32}$$

where c^x is a bearing bin in matrix x. If $D(I^q, I^c) \leq \tau$ the candidate is accepted as a loop closure [23].

2.4.2 Association in LIDAR Data

Though LIDAR histogram methods perform well as a candidate finder a more exact method is preferred for validation of the different candidates. For that end a point-to-point association method can be used.

Correspondence graph

The correspondence graph presented in [21] solves the association problem by creating two graphs P^S and P^T. P^S represents the current submap and is made up by nodes n^S corresponding to the landmarks within the submap. P^T represents the loop closure candidate with nodes n^T, corresponding to its landmarks. The nodes within the graphs are connected by edges $\epsilon^S_{i,j}$, where i and j are the index of two different nodes. The edges are assigned the value of the Euclidean distance between the two nodes that it connects. The nodes n^S_i and n^S_j in graph P^S are then connected to the nodes n^T_i and n^T_j in P^T if the edges of the nodes have similar value by some threshold τ shown in Equation (2.33) [21].

$$|\epsilon^S_{i,j} - \epsilon^T_{i,j}| \leq \tau \tag{2.33}$$

This is done for all the different combinations for both graphs which results in a combined graph $P^{S,T}$. The size of the clique in the combined graph then corresponds to reliability of the point-to-point association and can be compared between the candidates [21].

Hough Data Association

The Hough Data Association is also presented in [21] and is based on comparing the position of landmarks with each other. A landmark compared with another gets a parameter vector that contains the positions $[t_x, t_y]$ at some given rotations t_θ. For computational reasons the three components are quantized and bounded within a given span $[max_x, min_x] * [max_y, min_y] * [max_\theta, min_\theta]$. When applied to two sets of landmarks $p^S_i \in P^S$ and $p^T_i \in P^T$ where T stands for target and S for source the following expression is calculated:

$$\begin{bmatrix} t_{x_{i,j}} \\ t_{y_{i,j}} \end{bmatrix} = p^T_j - R(\theta_{i,j}) p^S_i \tag{2.34}$$

where $R(*)$ is a 2d rotation matrix [21]. The calculation is performed for all the quantized θ within the bounded limit $[max_\theta, min_\theta]$ which will result in a vector of positions and rotations $[t_x, t_y, \theta]$. If the vector is drawn it will look like a helix. The optimal matching is then given by the subsets of matching pairs with maximum cardinality [21].

Generalized-ICP

A common algorithm in data association is Iterative Closest Point (ICP). The standard ICP computes the two following things:

1. The correspondence between two sets of points.
2. The transformation T that minimizes the distance between two corresponding sets of points.

The full algorithm is shown in Algorithm 6 where d_{max} is the threshold for the error function $d_i = \|Tb_i - m_i\|^2$ that is used to filter out overlapping points.

Input: Two point clouds $A = a_i$, $B = b_i$ and initial transform T_0
Output: Estimated transform T, which aligns A and B.
while *not converged* **do**
  for *i = 1:N* **do**
    m_i = *findClosestPointInA*($T^* b_i$)
    if $\|Tb_i - m_i\|^2 \leq d_{max}$ **then**
      $w_i = 1$;
    else
      $w_i = 0$;
    end
  end
  $T^* = \text{argmin}_T\{\sum_i w_i \|Tb_i - m_i\|^2\}$
end

Algorithm 6: Standard ICP [24].

To improve the performance of the standad ICP the point-to-plane variant of the ICP is proposed by [24] which has proven to be more robust and accurate. The algorithm works in a 2.5d environment and minimizes the error function along the surface normal $T* = \text{argmin}_T\{\sum_i w_i \|\eta T b_i - m_i\|^2\}$ where η is the surface normal of m_i [24].

The Generalized-ICP [24] is a further improvement of the standard where a stochastic model is used in the error function instead of a deterministic. A Gaussian distribution is added to points in A and B resulting in $a_i \sim \mathcal{N}(\mu_{a_i}, \sigma_{a_i})$ and $b_i \sim \mathcal{N}(\mu_{b_i}, \sigma_{b_i})$. The distribution of the error function then becomes:

$$d_i^{(T)} \sim \mathcal{N}(\mu_{b_i} - (T^*)\mu_{a_i}, \sigma_{b_i} + (T^*) * \sigma_{a_i}(T^*)^T) = \mathcal{N}(0, \sigma_{b_i} + (T^*) * \sigma_{a_i}(T^*)^T) \quad (2.35)$$

where full correspondence is assumed: $\mu_{b_i} = T^* \mu_{a_i}$ [24].
The computation of T* becomes:

$$T^* = \underset{T}{\text{argmin}}\{\sum_i d_i^T (\sigma_{b_i} + T\sigma_{a_i}(T)^T)^{-1} d_i\} \quad (2.36)$$

[24].

3

Software System Modules

This section will describe the developed software system and tools used in the master thesis to investigate the problem formulation.

3.1 Visualization Tool

As a first step in the thesis a visualization tool was developed to visualize the performance of different map partitions and get a way to evaluate them. The visualization tool is also developed to gain hands-on experience of the software system and data that it produces.
The visualization tool is developed in Python and uses the library matplotlib to visualize the output of the software system. It plots different kinds of trajectories and landmarks to form a local map that is either the final output of the software system or sent internally between different nodes. The trajectories are made up of a series of estimated positions that is represented by a custom data structure. The estimated position of the landmarks is represented by a point cloud from the C++ library Point Cloud Library.
The visualization tool is able to plot the following different kinds of trajectories and landmarks:

- The raw trajectory that is the trajectory of the platform before the EKF-estimation.
- The absolute trajectory and total map which is the EKF-estimated trajectory and landmarks.
- The trajectory and landmarks estimated by the GTSAM node.
- The GTSAM landmarks.

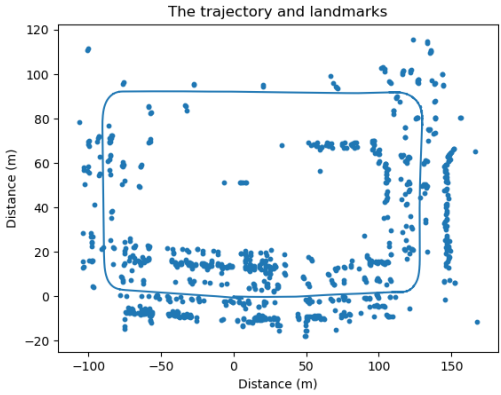

Figure 3.1: *An example of a trajectory and landmarks of the platform when driving around a residential block in an urban environment. Dots are landmarks and the line is the trajectory.*

- The divided submaps which is the divided GTSAM-estimated trajectory and landmarks.

An example of how the trajectories and landmarks can be plotted is shown in Figure 3.1. It is also able to plot the absolute error between the estimated trajectory and a reference trajectory against time which is shown in Figure 3.2.

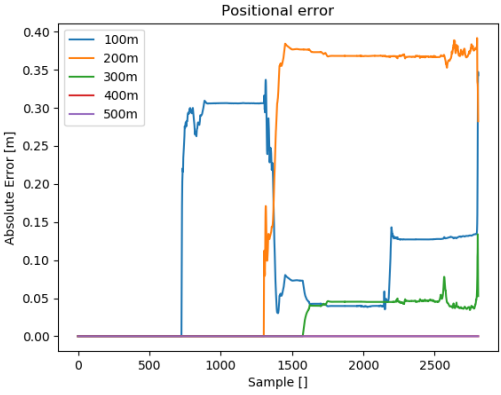

Figure 3.2: *An example of the positional error for different submap sizes when driving around a residential block in an urban environment.*

3.2 The Frontend System

The frontend system consists of three ROS nodes: The Input node, the Association node and the EKF node. It is used to collect and process the data received from the rotating LIDAR in form of point clouds and from the IMU unit in form of linear accelerations and rotational velocities. Landmarks are extracted from the point cloud and are processed together with IMU data by a SLAM algorithm. The algorithm builds a map of its surroundings and locates the platform within it. The map and the position estimate is sent to the GTSAM node for refinement. This section will describe the three frontend nodes in short. For more information, see [6].

3.2.1 Input Node

The Input node is responsible for collecting incoming IMU measurements into a pose estimate referred to as raw pose. It also receives point clouds from the rotating LIDAR and extracts features from the point cloud. The type of features that can be extracted are edge features from edges such as house corners and circular features from circular landmarks such as thicker tree trunks. A flow chart of the node is shown in Figure 3.3. The Input node has the functionality to improve the raw pose by using a GNSS receiver to calculate its velocities. This functionality is only used as a ground truth reference and when it is enabled the platform is referred to as the GNSS aided platform.

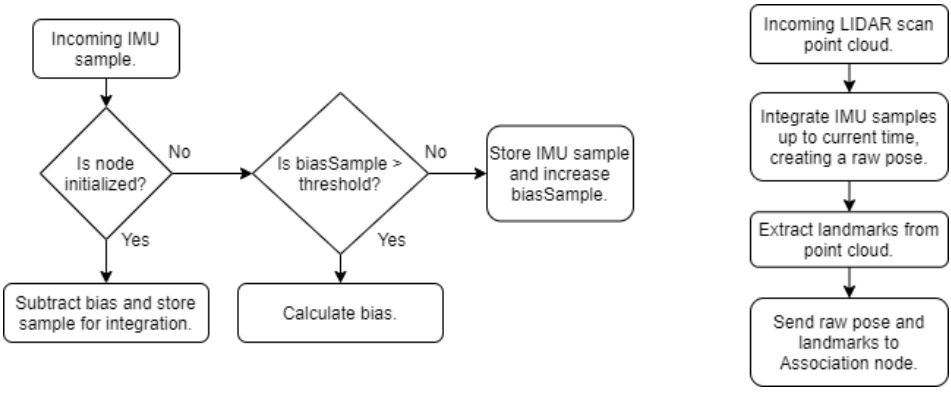

Figure 3.3: *A flow chart for the Input node.*

3.2.2 Association Node

The Association node associates new landmarks with old ones and determines which ones are consistent. The consistent landmarks are stored in a vector called consistent landmarks. The landmarks that are consistent get an id key and are sent to the EKF Node. A flow chart for the Association node is shown in Figure 3.4.

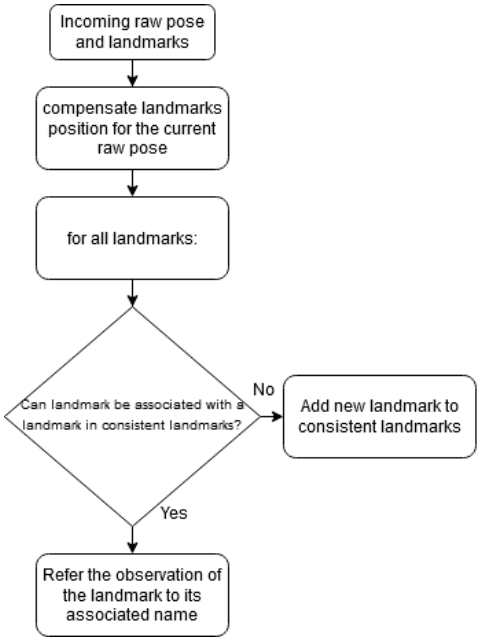

Figure 3.4: *A flow chart for the Association node.*

3.2.3 EKF Node

The EKF node performs SLAM using an Extended Kalman filter. It takes the constant landmarks produced by the Association node and creates a global map in which it localizes the platform. It also performs a χ^2-test as a sanity check on the incoming landmarks discarding landmarks too close to each other. Figure 3.5 shows a flow chart of the node.

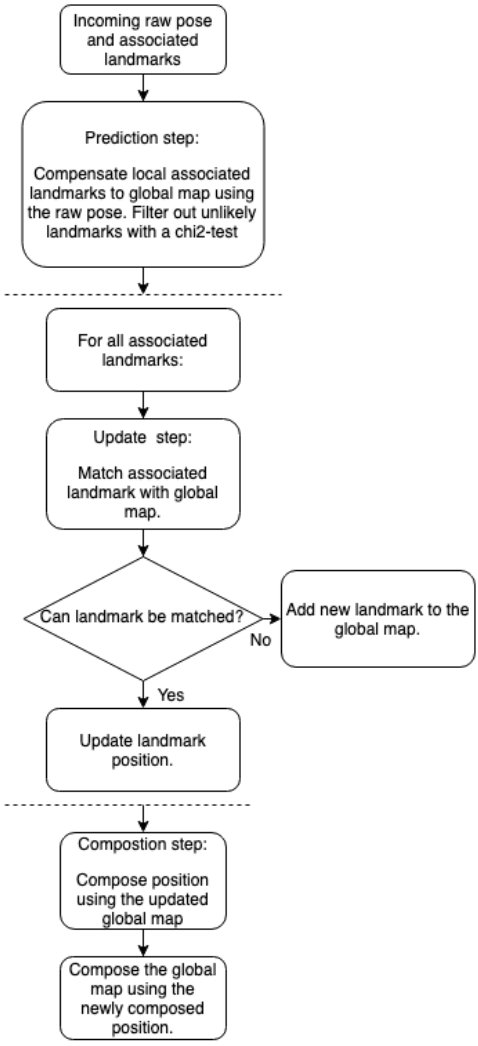

Figure 3.5: *A flow chart for the EKF node.*

3.3 The Original GTSAM Node

To be able to understand the modifications to the GTSAM node the original GTSAM node is described in the following section.

The original GTSAM node is used to improve the initial estimate performed by the frontend EKF node. It uses the GTSAM C++ library to calculate an estimate for the full trajectory and observed landmarks. It builds a factor graph from measurements which are added to the iSAM2 estimator together with an initial estimate from the EKF node. iSAM2 estimator refines the initial estimate according to section 2.2.6. The current pose in the estimate is saved as a pose in the trajectory. Figure 3.6 shows a flow chart of the node.

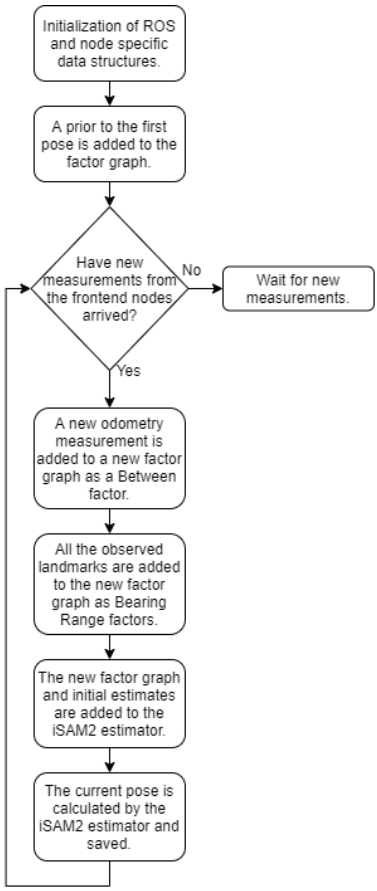

Figure 3.6: *The flow chart for the current GTSAM node is shown.*

3.4 The Modified Software System

The software system is modified with added map partitioning functionality to be able to handle larger areas. The partitioned submaps are reduced according to a modified version of the Condensed Measurement described in section 2.3.3. The modified Condensed Measurement method uses $\oplus$ and $\ominus$ operations, described in section 2.3.4, instead of the unscented transform to estimate the covariances of the condensed measurements. An overview of the modified software system is shown in figure 3.7.

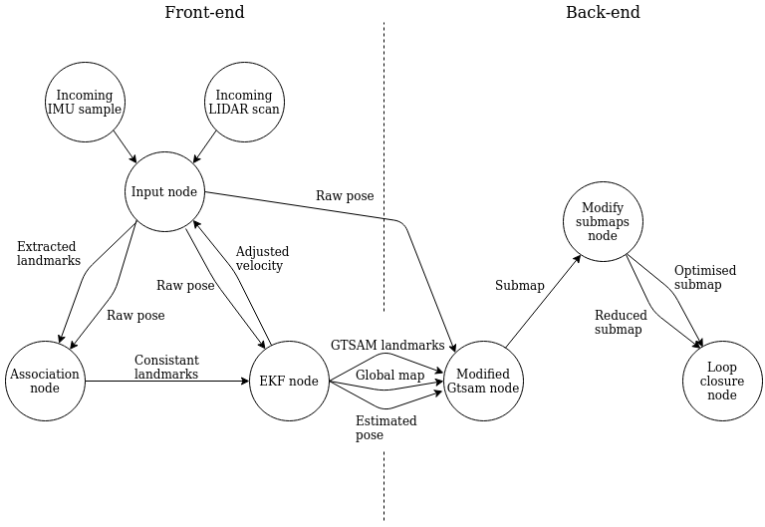

Figure 3.7: *A map over the modified software system. To the left is the unmodified front-end system. To the right the back-end system composed of the modified GTSAM node together with the two new nodes, the Modify submaps node and the Loop closure node.*

3.4.1 The Modified GTSAM Node

The modified GTSAM node has the same basic structure as the current GTSAM node but with the added functionality of partitioning the map. The main difference is that when the total distance travelled within a map exceeds a given threshold the loop is broken and the map is partitioned into two submaps. The second goes back into the loop as a prior and the first is stored, see figure 3.8.

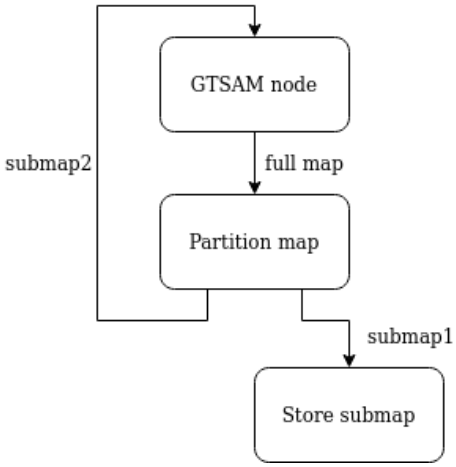

Figure 3.8: *Overview of the modified GTSAM node.*

Partitioning of the map

The partitioning is performed after a given travelled distance which will correspond a position node in factor graph referred to as a partition node. The partition node will be copied and belong to both submaps and considered a shared node. Any landmarks observed by both the partition node and the node after the partition node will also be considered a shared node, see Figure 3.9. All the shared nodes are marked when the map is partitioned to grant easy access for the calculation of the condensed measurement.

The partition point of the map can also be selected depending on how many landmarks that are observed at that point. This feature will make it possible to determine the impact of landmarks at the partition point by forcing the software system to partition the map at either a landmark dense or a landmark sparse area.

The functionality for map partitioning in landmark sparse areas is implemented in the following way: At a given sample the software system stores the number of landmarks. When the platform has travelled a predetermined distance a preliminary partition point is set and a function is called to determine the number of landmarks observed by the platform. A mean of the number of landmarks observed in the 16 latest samples is compared to an upper threshold. If the mean exceeds the threshold, indicating that the platform is located in a landmark dense area, the partition point is pushed one sample forward and thereby lengthening the distance travelled before partitioning the map. The same check will then be performed next sample and continue until the mean does not exceed the threshold. If the mean does not exceed the threshold the partition point is kept as it is. The functionality for map partitioning in a dense area is implemented in the corresponding way but with a lower threshold instead.

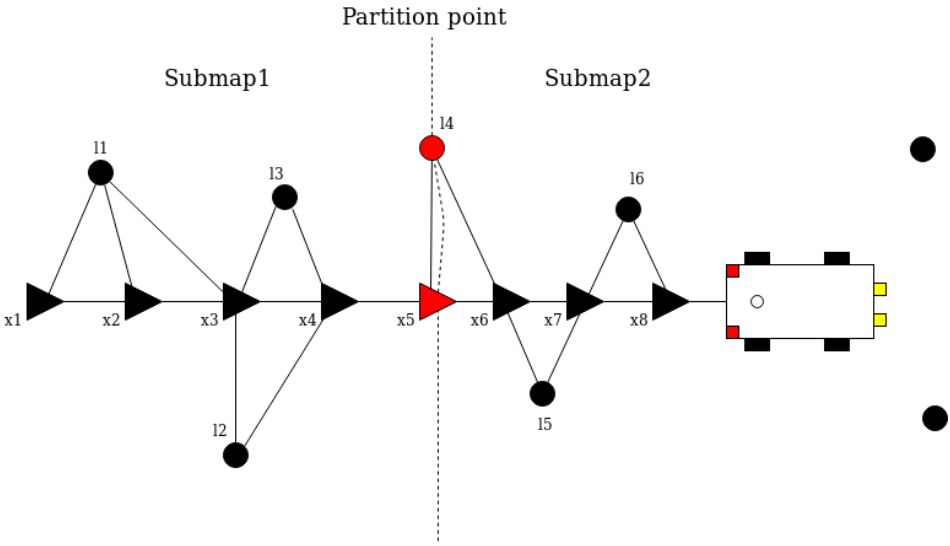

Figure 3.9: *The partitioning of a map into two submaps. The shared nodes are marked in red.*

Storing the submaps

When the software system has partitioned the map into two submaps the first submap is stored. This is performed by breaking apart the factor graphs into factors. The factors are in turn broken apart into measurements, covariances and stored together with identification keys of the two nodes that are connected by the factor. The current estimation of each landmark and position is also stored together with a vector with the keys of the shared nodes. A stored submap can then be loaded by rebuilding the factor graph with the stored factor components and initiated with the stored estimate.

3.4.2 Modify Submaps Node

To compute the condensed measurements for each submap a ROS node named Modify submaps was created. It starts by optimizing the submaps from the GTSAM node individually without any influence from each other. The optimized submaps are used to compute the condensed measurements for each submap, creating reduced submaps, which are sent to the Loop closure node together with the optimized submap.

The condensed measurements are computed in the same way as they are in [17] with the exception that it uses the $\oplus$ and $\ominus$ operators from [18] to calculate the spatial relations between the shared nodes and the origin node, both for the poses and the uncertainties. It starts by getting the pose of the origin node p_o and its marginalized covariance $C(p_o)$ together with the pose of the shared nodes p_i and

the marginal covariance for those poses $C(p_i)$. The joint marginal covariance for the origin node and the shared nodes $C(p_o, p_i)$ is also obtained. For every shared node, the following calculations are performed:

$$p_{o,i} = (\ominus p_{e,o}) \oplus p_{e,i} \tag{3.1}$$

and

$$\begin{aligned} C(p_{o,i}) \approx \begin{bmatrix} J_{1\oplus} & J_{2\oplus} \end{bmatrix} \begin{bmatrix} J_\ominus C(p_{e,o}) J_\ominus^T & C(p_{e,o}, p_{e,i}) J_\ominus^T \\ J_\ominus C(p_{i,e}, p_{e,o}) & C(p_{e,i}) \end{bmatrix} \begin{bmatrix} J_{1\oplus}^T \\ J_{2\oplus}^T \end{bmatrix} = \\ J_{1\oplus} J_\ominus C(p_{e,o}) J_\ominus^T J_{1\oplus}^T + J_{1\oplus} C(p_{e,o}, p_{e,i}) J_\ominus^T J_{2\oplus}^T + \\ J_{2\oplus} J_\ominus C(p_{i,e}, p_{e,o}) J_{1\oplus}^T + J_{2\oplus} C(p_{e,i}) J_{2\oplus}^T \end{aligned}$$

where $J_{1\oplus}$ and $J_{2\oplus}$ are parts of the Jacobian matrix for the $\oplus$-operation. $J_\ominus$ is the Jacobian for the $\ominus$-operation. Subscript e stands for entrance node, o for origin node and i for the current shared node. The $p_{o,i}$ is used as the measurement in the condensed measurement and the $C(p_{o,i})$ is the uncertainty of that measurement. These calculations are made for all the shared nodes of the submap except for the entrance node (the first pose node in the submap) for which the following calculations are performed:

$$p_{e,o} = (\ominus p_e) \oplus p_{e,o} \tag{3.2}$$

and

$$\begin{aligned} C(p_{o,i}) \approx \begin{bmatrix} J_{1\oplus} & J_{2\oplus} \end{bmatrix} \begin{bmatrix} J_\ominus C(p_e) J_\ominus^T & C(p_e, p_{e,o}) J_\ominus^T \\ J_\ominus C(p_{o,e}, p_e) & C(p_{e,o}) \end{bmatrix} \begin{bmatrix} J_{1\oplus}^T \\ J_{2\oplus}^T \end{bmatrix} = \\ J_{1\oplus} J_\ominus C(p_e) J_\ominus^T J_{1\oplus}^T + J_{1\oplus} C(p_e, p_{e,o}) J_\ominus^T J_{2\oplus}^T + \\ J_{2\oplus} J_\ominus C(p_{o,e}, p_e) J_{1\oplus}^T + J_{2\oplus} C(p_{e,o}) J_{2\oplus}^T \end{aligned}$$

where p_e is the marginal covariance for the entrance node.

3.4.3 Loop Closure Node

A ROS node that performs manual loop closure is implemented as a first concept for loop closure. It is also used to examine how the partitioning of the map influences the estimate of the trajectory and landmarks when loop closure is achieved. It is called The Loop closure node and uses the optimized submaps together with the reduced submaps from the Modify submaps node to perform its estimate.

The node begins by adding together the condensed measurements of the submaps that are in the loop, creating a sparse factor graph of the full trajectory. A last factor is added between the first and the last node of the factor graph, closing the loop. The measurement of the last factor is manually calculated from a GNSS aided reference trajectory and its uncertainty is selected to an arbitrarily low value compared to the other factors in the graph, forcing it to close the loop. The sparse factor graph is then batch optimized and a global solution is

obtained. The optimized submaps are then once again optimized but with constraints on the nodes that coincide with the global solution of the sparse factor graph. This forces the submaps to obtain the position, orientation, twist and scale of its reduced counterpart in the sparse factor graph. The resulting submap is then stored and referred to as a loop closed optimized submap.

3.5 Tools and Software

The software system is working in an open environment based on ROS, distribution release Melodic Morenia and is developed using C++. It uses, among others the two C++ libraries, GTSAM and Point Clouds library (PCL). Most of the software system is written in C++ since it uses some C++ libraries but also because C++ is a mid-level programming language, able to write direct low level functions. The software system is run on a Dell Precision Workstation T3500 with a Intel Xeon CPU on 2.80 GHz.
For the data recording a Gigabyte Ultra compact PC is used together with the Lidar and IMU. The platform and the recording setup is mounted on a Toyota Land Cruiser. Git is used for version control.

4

Initial Investigation

An initial investigation was performed to limit the number of test cases to be examined in the thesis. It was mainly used to give some background to how many landmarks the vehicle should observe in an area in order for that area to be considered a landmark sparse area and how many should be considered as a landmark dense area. During the initial investigation the platform used the original GTSAM node. The different test cases will be presented followed by the results. The last section will present some discussion and conclusion of the investigation.

4.1 Test Cases

The first test examines the absolute error change between the GNSS aided platform, which is considered ground truth, and the platform. The absolute error between two samples is not of interest since it could be accumulated from previous samples and does not therefore give a good measure of the current sample. The change in errors is proposed as a better measure and should give a crude estimate of how the platform performs at a given sample observing a number of landmarks. The absolute error change is calculated using the expression shown in Equation (4.1).

$$\dot{e} = \left| \frac{x[k] - x_{ref}[k] - (x[k-1] - x_{ref}[k-1])}{2} \right| \tag{4.1}$$

where $x[k]$ is the position of the platform at sample k and $x_{ref}[k]$ is the ground truth position of the GNSS aided platform at sample k. The second test examines the change in error as a histogram over samples with a given number of landmarks. The test is performed to get a notion of how a given number of landmarks affects the absolute error of the platform.

4.2 Results of the Investigation

Three different runs were made during the investigation where the first two had the same route. The first run started in a residential block in an urban environment and headed out to the countryside. It is referred to as *City to country 1*. The second run had the same route as the first and will be referred to as *City to country 2*. The last run was a little longer drive at the countryside and will be referred to as the *Countryside*.

4.2.1 Routes

The following routes were followed during the investigation: *City to country 1* is shown in Figure 4.1, *City to country 2* is shown in Figure 4.2 and *Countryside* is shown in Figure 4.3.

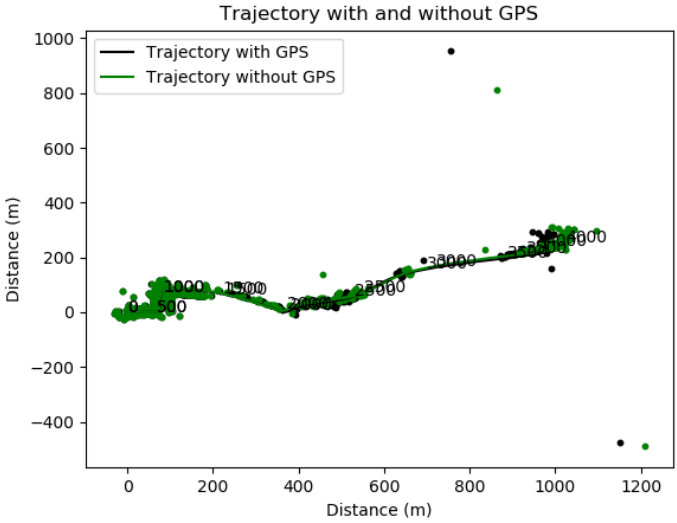

Figure 4.1: *The trajectory and estimated landmarks for run City to country 1. The lines are the estimated trajectories and the dots landmarks. The numbers show the current sample number of that position.*

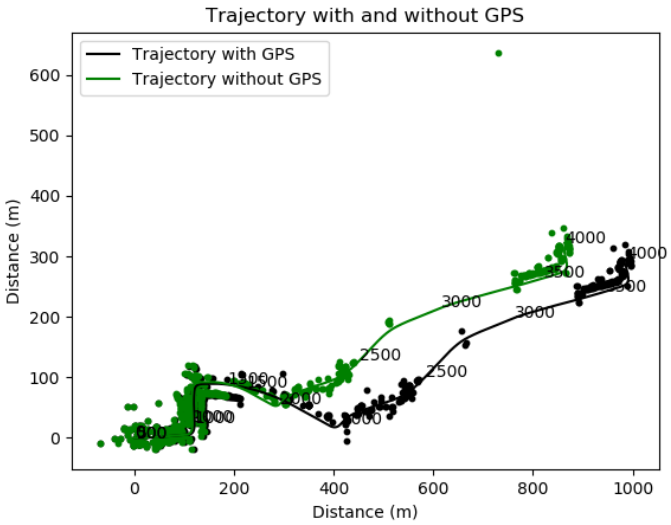

Figure 4.2: *The trajectory and estimated landmarks for run City to country 2. The lines are the estimated trajectories and the dots landmarks. The numbers show the current sample number of that position.*

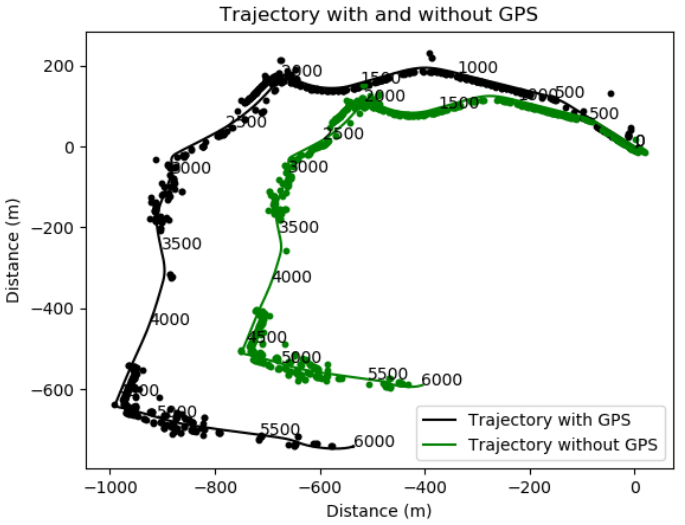

Figure 4.3: *The trajectory and estimated landmarks for run Coutryside. The lines are the estimated trajectories and the dots landmarks. The numbers show the current sample number of that position.*

4.2.2 Absolute Error Change

The following figures shows how the absolute error change depends on the number of observed landmarks in a given sample. *City to country 1* is shown in Figure 4.4, *City to country 2* is shown in Figure 4.5 and *Countryside* is shown in Figure 4.6.

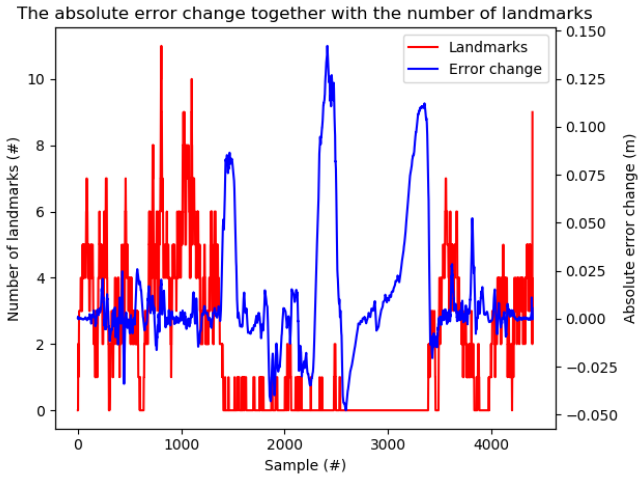

Figure 4.4: *The absolute error change together with the number of landmarks at a sample for run City to country 1.*

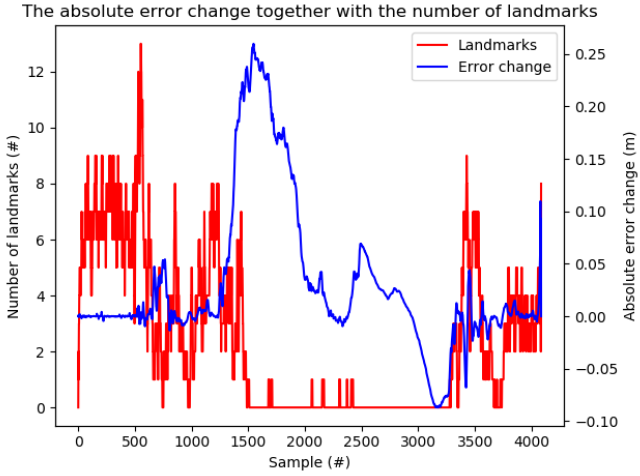

Figure 4.5: *The absolute error change together with the number of landmarks at a sample for run City to country 2.*

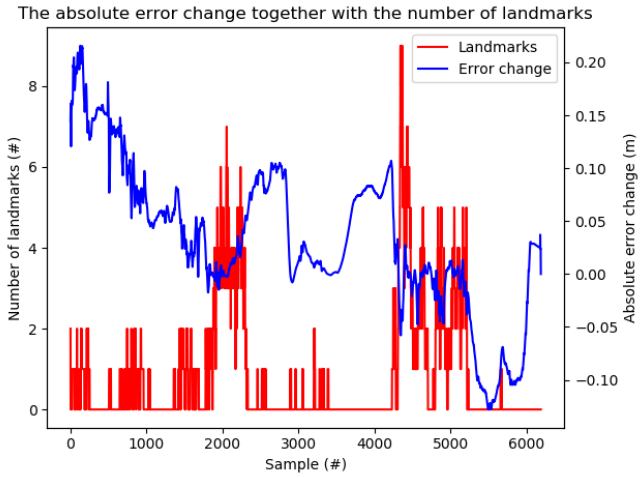

Figure 4.6: *The absolute error change together with the number of landmarks at a sample for run Countryside.*

4.2.3 Histogram of the Absolute Error Change

Histograms of the absolute error change at a given sample is shown in the following figures. *City to country 1* is shown in Figure 4.7, *City to country 2* is shown in Figure 4.8 and *Countryside* is shown in Figure 4.9.

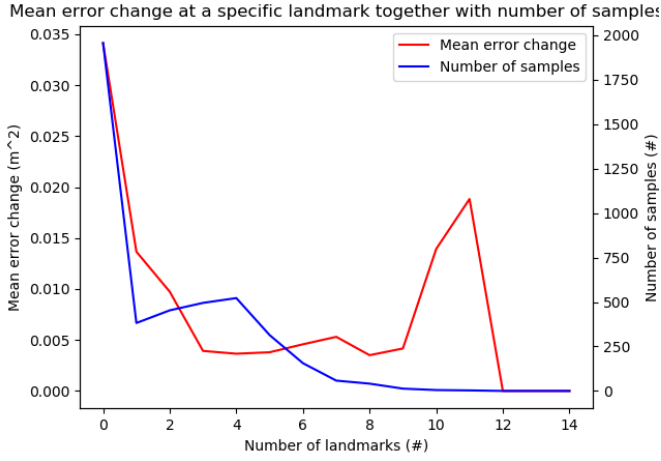

Figure 4.7: *The mean error change at a specific number of landmarks together with the number of samples at that number of landmarks for the City to country 1.*

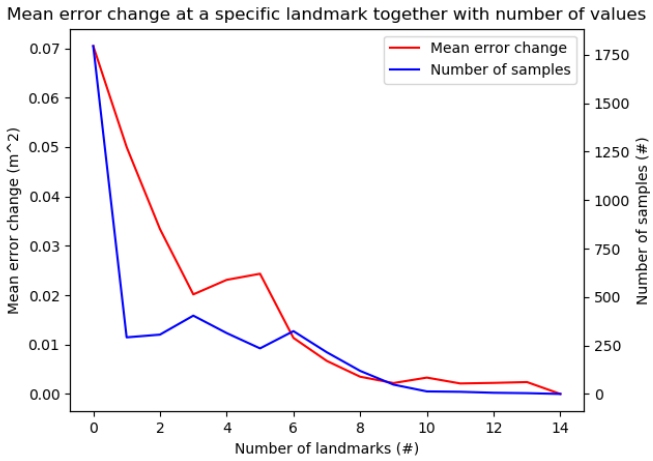

Figure 4.8: *The mean error change at a specific number of landmarks together with the number of samples at that number of landmarks for the City to country 2.*

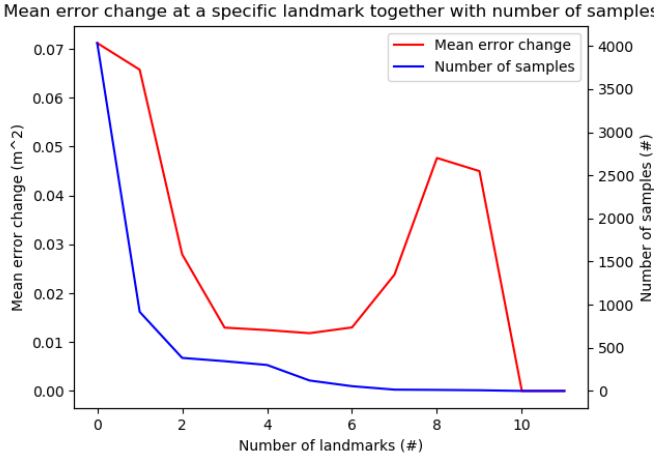

Figure 4.9: *The mean error change at a specific number of landmarks together with the number of samples at that number of landmarks for the Countryside.*

4.3 Discussion and Conclusion of the Investigation

The results of the investigation give some indication of how the number of observed landmarks affects the change in positional error. In Figures 4.4, 4.5 and 4.6 the absolute error change is presented for the three runs. There is a clear rise in absolute error change when the platform is located in a landmark sparse area compared to a landmark dense area. The most obvious behaviour that contradicts this conclusion is that in both the dataset *City to country 2* and *Countryside* the rise in error change starts in a fairly landmark dense area at one point each. In *City to country 2* this occurs around sample 1300 and in *Countryside* around sample 2000. The *Countryside* dataset is overall more fluctuating as can be seen between sample 4250 and 5250 but the behaviour is clear in *City to country 2*.

When looking at the histograms, the same trend is present. In Figure 4.7, 4.8 and 4.9 the absolute error change histograms are shown. The error change is clearly largest when the platform does not observe any landmarks and falling with increasing number of landmarks. This can be said about all the three runs with one added note. Both *City to country 1* and *Countryside* show a rise in absolute error change when the number of observed landmarks becomes more than 9 for *City to country 1* and 6 for *Countryside*. The number of samples with that many observed landmarks is very low though, making it hard to draw any conclusions from that note.

Disregarding this and the irregular behaviour in the absolute error change of run *City to country 2*, the overall trend is that the platform has a lower absolute error change when the number of observed landmarks is high and higher when it is low. When examining the histograms there seems to be a rapid decrease in the mean absolute error change between zero landmarks and three landmarks. This trend is seen as motivation to consider a landmark sparse area to be less than one to three landmarks. The subsequent number of landmarks, four to six, is somewhat stable when compared to one to three. This trend is seen as a motivation to consider a landmark dense area to have more landmarks than four to six. These configurations of dense and sparse areas are more closely looked into in the next chapter.

5

Map Partitioning Results

This chapter presents the result of map partitioning and loop closure performed by the platform using the modified software system described in Chapter 3. The scenarios tested were chosen with respect to the problem formulation in Chapter 1. The issues studied were as follows:

- The effect that partitioning the map with different static submap sizes has on the estimated trajectory.
- The effect that partitioning the map with different static submap sizes during loop closure has on the estimated trajectory.
- The effect that partitioning the map with dynamic submap sizes depending on the number of landmarks observed at the partition point has on the estimated trajectory.
- The effect that partitioning the map with dynamic submap sizes depending on the number of landmarks observed at the partition point during loop closure has on the estimated trajectory.

5.1 Static Partitioning of the Map without Loop Closure

When different static submap sizes were examined, the following results were obtained. The results show the effect of partitioning the map with different submap sizes compared to a full map. The platform was working on the same data that was recorded in advance which resulted in the same conditions for all the different submap sizes. The previously recorded data that the platform used was played at 0.5 real-time speed which affected the different time measures taken.

5.1.1 Urban Route

The platform was tested in an urban environment with a route around a residential block. The result is presented in Figure 5.1, where the trajectories of different submap sizes are shown and in Figure 5.2, where the error between the different submap sizes and an uncut map is shown.

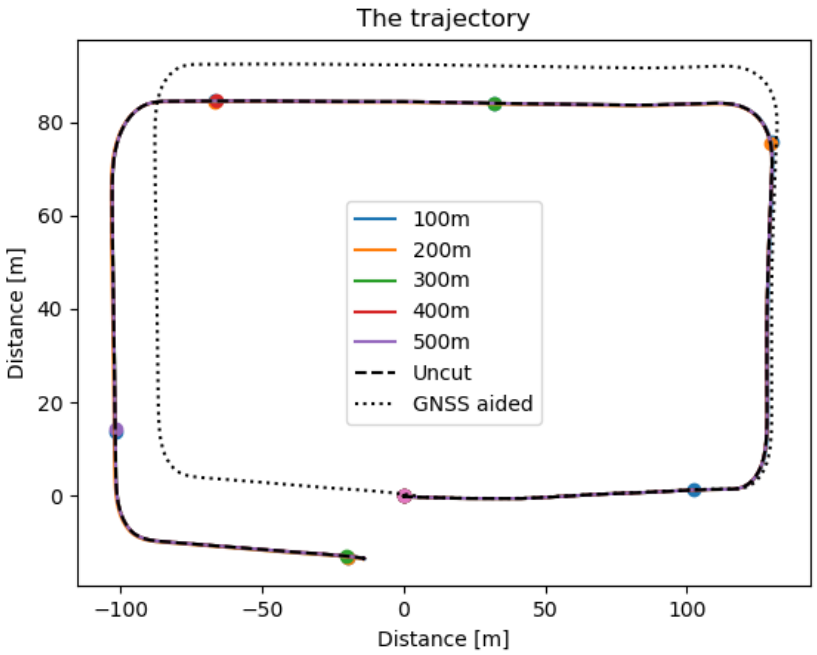

Figure 5.1: *The trajectories with different submap sizes when the platform was driven around a residential block. The dots represent partition points.*

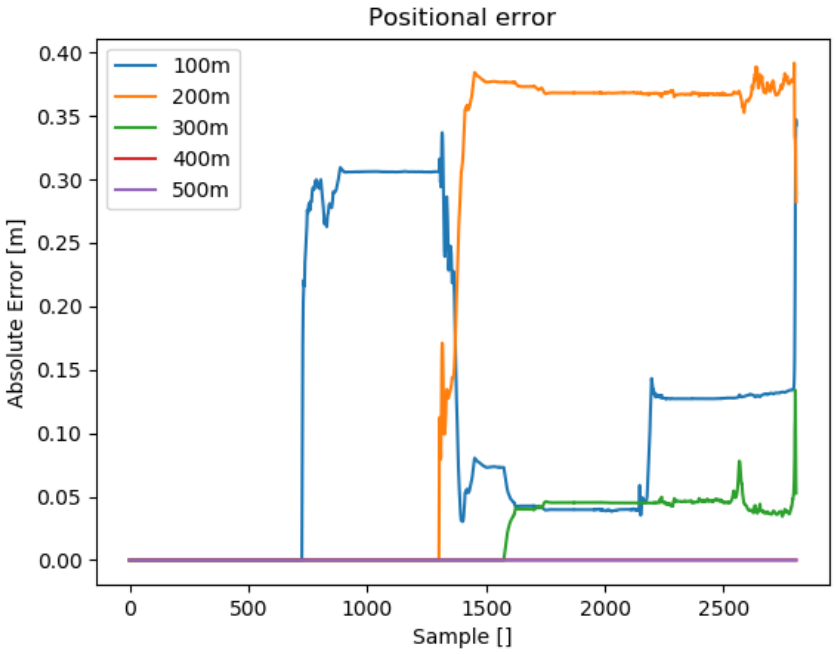

Figure 5.2: *The positional error between trajectories when different submap sizes where used and an uncut trajectory.*

The Root Mean Square Error (RMSE) between the differently partitioned trajectories and the uncut trajectory was calculated and is shown in Table 5.1. The table also shows the mean partition time, which was the time the map partitioning function took for the different submap sizes. It also shows the mean iteration time, which was the time one iteration in the GTSAM node took for the different submap sizes.

Table 5.1: *The RMSE, mean partition time and mean iteration time at different static submap sizes.*

Submap size	RMSE	Mean partition time	Mean iteration time
100m	0.1569m	0.0239s	0.0014s
200m	0.2637m	0.0390s	0.0038s
300m	0.0295m	-	-
400m	-	-	-
500m	-	-	-

Figure 5.1 shows that on a larger scale the difference between the uncut trajec-

tory and a partitioned one is quite small when compared to distance travelled. It also shows that in this short drive the platform quickly accumulates a quite large error when compared to the GNSS aided trajectory.

In Figure 5.2 the positional errors appear to change in steps which seems to coincide with the partition points for the submaps. The 400m and 500m submaps do not partition in this short drive which is why they don't show any errors. The steps seem to either increase or decrease the positional errors at each map partitioning but with such a short route compared to the submap length it is hard to draw any conclusions for the submap sizes. One remark that can be said is that the map partitioning error seems to be negligible compared to the overall positional error.

The RMSE shown in Table 5.1 seems to be somewhat stable for 100m and 200m and then drop by one order of magnitude for 300m. Both the Mean Partition Time and the Mean Loop Time seems to be increasing with larger maps but since the result only shows two samples it is hard to say much about the time consumption.

When the platform was used it needed to stand still for 20 seconds so the IMU could calibrate. Since the partitioning was governed by distance travelled, the first submap became much larger than the rest creating an outlier for the time measurements. The last submap became much smaller than the rest since the platform was turned off in the middle of it which also creates an outlier for time measurements. These two samples were excluded from the calculation of the mean time. The 300m submap therefore does not have any mean partition time and mean iteration time because it only consisted of two submaps.

5.1.2 Urban to Countryside Route

The platform was tested on a route that went from the urban environment in previous section to the countryside. The result is presented in Figure 5.3 where the trajectories of different submap sizes are shown and in Figure 5.4 where the error between the trajectories with different submap sizes and a trajectory with a full map is shown.

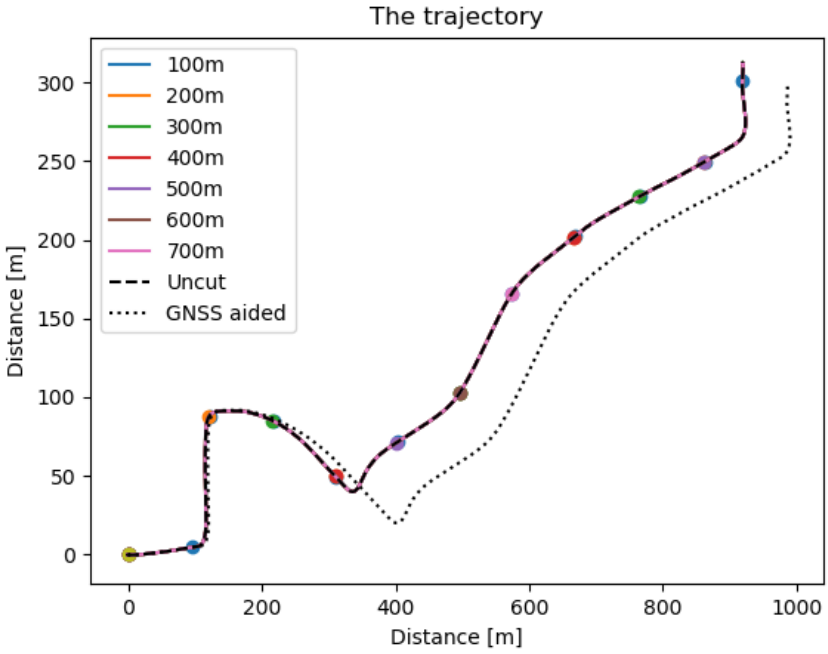

Figure 5.3: *The trajectories for different submap sizes when the platform was driven from an urban area to the countryside. The dots represent partition points.*

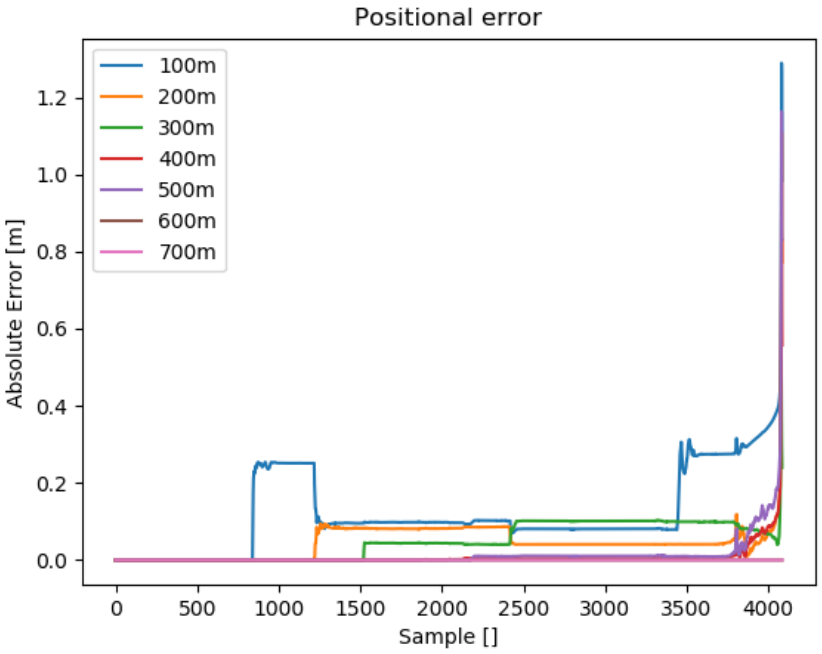

Figure 5.4: *The absolute error for different partition points when the platform was driven from an urban area to the countryside.*

Table 5.2: *The RMSE, mean partition time and mean iteration time at different static submap sizes.*

Submap size	RMSE	Mean partition time	Mean iteration time
100m	0.0261m	0.0134s	0.0012s
200m	0.0044m	0.0229s	0.0009s
300m	0.0045m	0.0262s	0.0007s
400m	0.0012m	-	-
500m	0.0027m	-	-

The result for the Urban to countryside route is quite similar to the Urban route. The map partitioning doesn't affect the trajectory noticeably on a larger scale as shown in Figure 5.3.

The same step errors can be seen in Figure 5.4 as in the urban route and the RMSE seems to get smaller with larger submaps. The mean partition time seems to get longer with bigger submaps. The mean iteration time looks somewhat stable or

even shorter with larger submaps.

5.2 Static Partitioning of the Map with Loop Closure

When the different submap sizes under loop closure were examined the following results were obtained. The result shows the effect of partitioning the map with different submap sizes when loop closure is performed compared to a GNSS aided estimate of the full map. The uncut trajectory and GNSS aided trajectory were kept as reference and have not been loop closed. The platform was working on prerecorded data which created the same conditions for all the different submap sizes. The prerecorded data was played at 0.5 real-time speed.

5.2.1 Urban Route

The platform was tested in an urban environment with a route around a residential block. The result is presented in Figure 5.5 where the trajectories with different submap sizes are shown and in Figure 5.6 where the absolute error between the trajectories with different submap sizes and a trajectory with GNSS reference is shown.

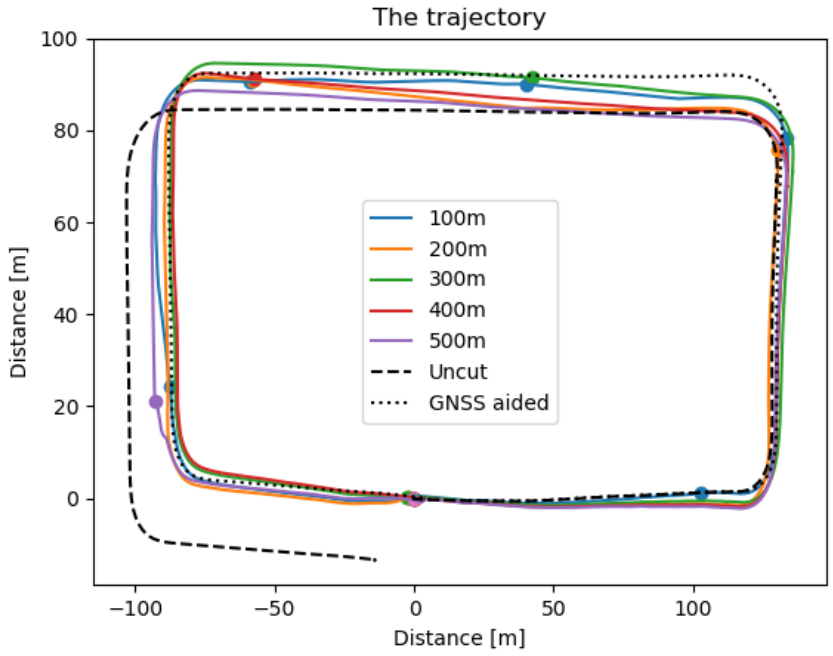

Figure 5.5: *The trajectory for different submap sizes with loop closure when the platform was driven around a residential block in an urban environment. The dots are the partition points.*

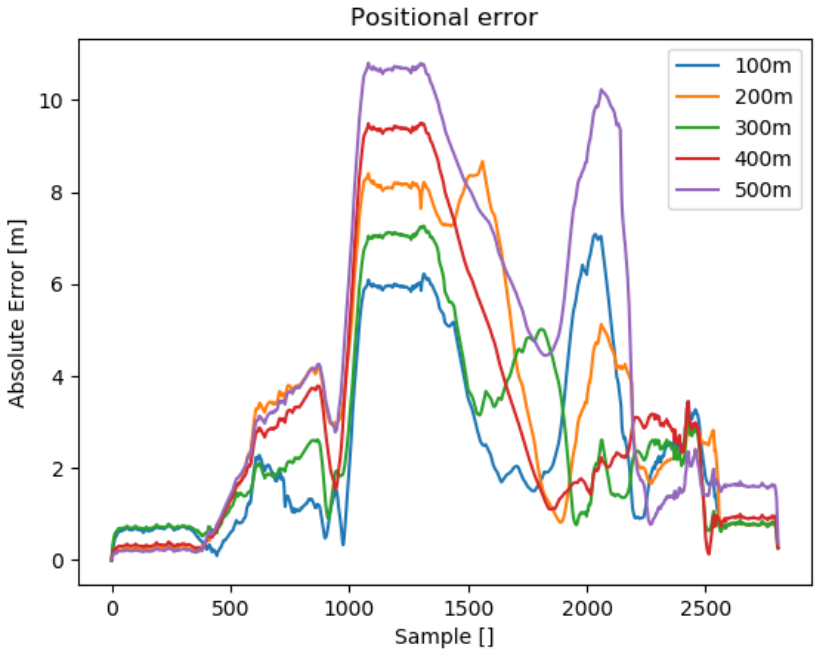

Figure 5.6: *The positional error for different submap sizes with loop closure when the platform was driven around a residential block in an urban environment.*

Table 5.3: *The RMSE for different static submap sizes.*

Submap length	RMSE
100m	3.1915m
200m	4.4185m
300m	3.4109m
400m	4.2739m
500m	5.7739m

The partition points of submaps seem to be quite on point with the GNSS reference in Figure 5.5 and the accumulated error is rather a result of the trajectory between partition points. This error might be solved by either increasing the number of iterations in the final optimization of the submaps or by changing the initial estimate of the submaps to match the loop closed partition points. Either way this shows that the loop closed reduced submap greatly decreases the positional error and that the final optimized submap could be improved for

this configuration and route. Shorter submaps that create more partition points should be preferred since the reduced loop closed submap, containing only the partition points, performs quite well. Trajectories comprised of shorter maps should therefore contain more accurate nodes, the partition points, which is seen in the RMSE. The RMSE appears to get lower with smaller submaps apart from the 200m sample.

The positional errors shown in Figure 5.6 don't have the same step appearance as the open trajectories. The starting point and the finish point have a low absolute error since they were forced to the same location by the loop closure. The middle of the trajectories have the highest absolute error as they were the farthest away from the loop closure point.

5.3 Dynamic Partitioning of the Map without Loop Closure

To investigate how the number of observed landmarks affects the map partitioning, a new requirement was put on the partition points. The requirement was that the partition points were forced to a landmark dense or a landmark sparse area. The submaps were initially set to a size of 100m but could be lengthened depending on the density of landmarks at the partition points. Six different cases were tested.

- The first case forced the submaps to be lengthened if the number of observed landmarks were more than or equal to 1 at the partition point. The submaps were therefore lengthened until the partition point ended up in an area with less than 1 observed landmark.
- The second case forced the submaps to be lengthened if the number of observed landmarks were more than or equal to 2 at the partition point, forcing the partition point to an area with less than 2 observed landmarks.
- The third case forced the submaps to be lengthened if the number of observed landmarks were more than or equal to 3 at the partition point, forcing the partition point to an area with less than 3 observed landmarks.
- The forth case examined the opposite scenario, forcing the submaps to be lengthened if the number of observed landmarks were less than or equal to 4 at the partition point, forcing the partition point to an area with more than 4 observed landmarks.
- The fifth case forced the submaps to be lengthened if the number of observed landmarks were less than or equal to 5 at the partition point, forcing the partition point to an area with more than 5 observed landmarks.
- The sixth case forced the submaps to be lengthened if the number of observed landmarks were less than or equal to 6 at the partition point, forcing the partition point to an area with more than 6 observed landmarks.

The number of current observed landmarks was determined by a mean over the last 16 samples of observed landmarks as explained in Chapter 3. The platform was working on the same prerecorded data creating the same conditions for all the different requirements on the partition points and played at 0.5 of real-time speed.

5.3.1 Urban Route

The platform was again tested on the urban route. The result is presented in Figure 5.7, showing the trajectories of submap with different requirements put on the partition points and in Figure 5.8 where the absolute error between those trajectories and an uncut trajectory is shown.

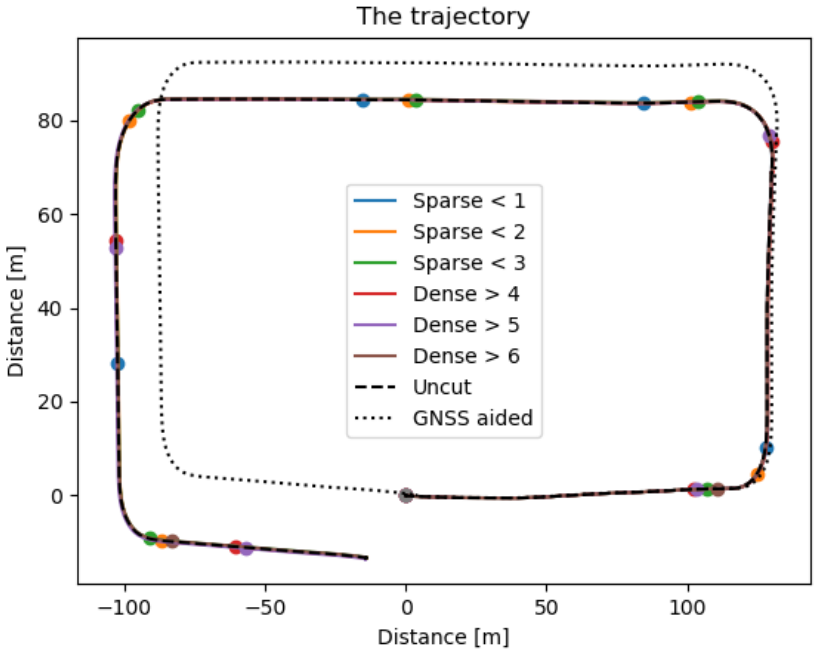

Figure 5.7: *The trajectories with different partitioning requirements put on the partition points when the platform was driven around a residential block in an urban environment. The dots show the partition points.*

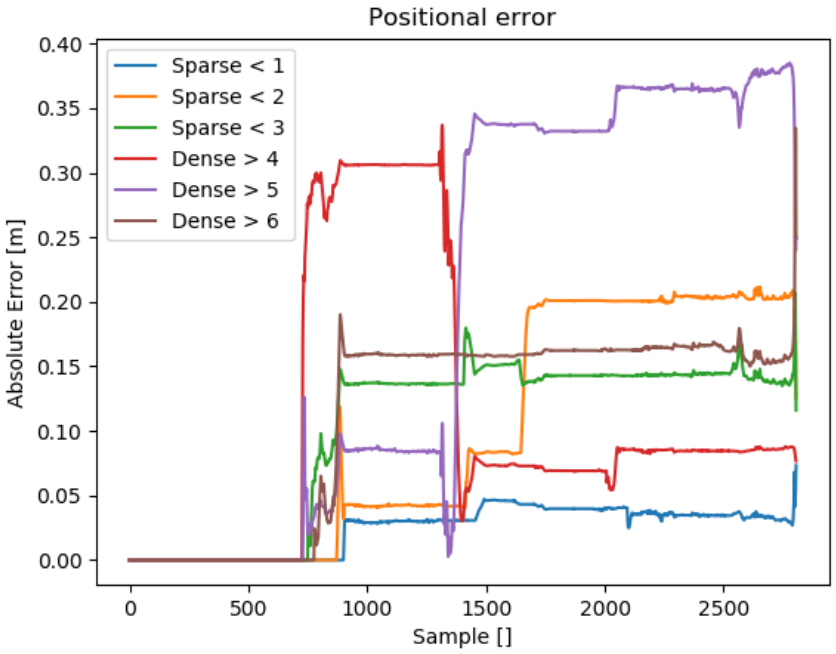

Figure 5.8: *The absolute error between the trajectories with different requirements put on their partition points and the uncut trajectory when the platform was driven around a residential block in an urban environment.*

Table 5.4: *The RMSE, mean partition time and mean iteration time and average submap size at different dynamic submap requirements.*

Partition point	RMSE	Mean partition time	Mean iteration time	Average submap size
Sparse < 1	0.0297m	0.0267s	0.0015s	121.6m
Sparse < 2	0.1326m	0.0208s	0.0014s	101.3m
Sparse < 3	0.1194m	0.0222s	0.0013s	101.4m
Dense > 4	0.1523m	0.0351s	0.0013s	121.7m
Dense > 5	0.2526m	0.0366s	0.0004s	121.7m
Dense > 6	0.1342m	-	-	202.7m

In Figure 5.7 the difference between submaps and the uncut map seems to be relatively small. In Figure 5.8 the same step appearance is present as with the statically partitioned submaps shown in Figure 5.2 and 5.4

When looking at the RMSE measure, it seems quite constant except for Sparse < 1 which is 1 order of magnitude smaller. The Mean Partition Time is a little bit longer for the partition point in dense areas and the Mean Loop Time is quite constant.

5.3.2 Urban to Countryside Route

The platform was again tested on the urban to countryside route. The results are presented in Figure 5.9 where the trajectories of submaps with different requirements put on the partition points are shown and in Figure 5.10 where the absolute error between those trajectories and the uncut trajectory is shown.

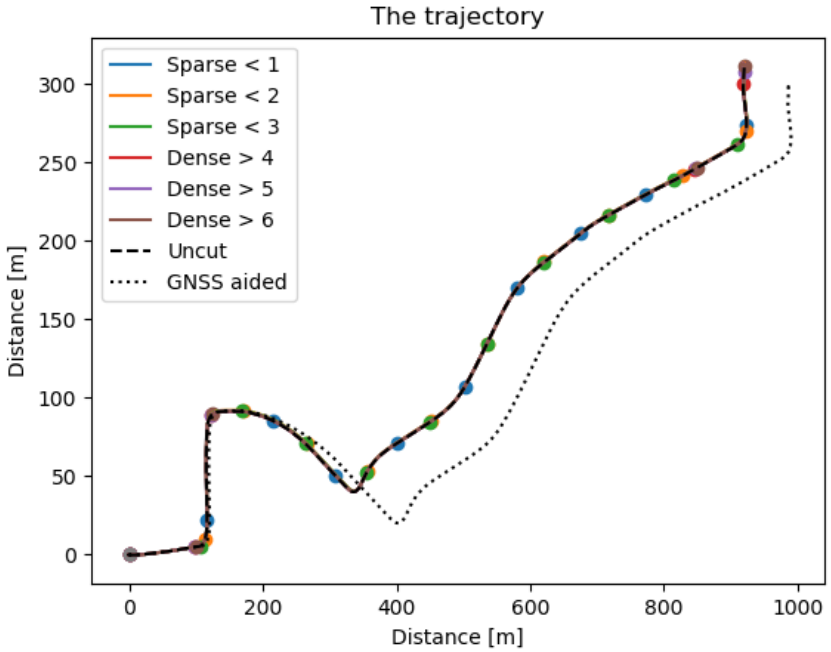

Figure 5.9: *The trajectories with different requirements put on the partition points when the platform was driven from a residential block to the countryside. The dots are partition points.*

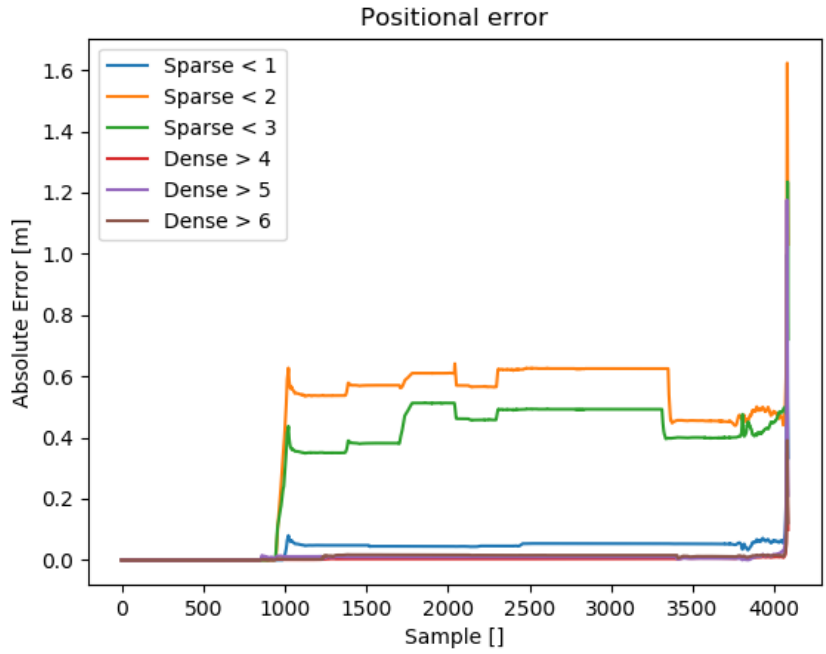

Figure 5.10: *The absolute error between the trajectories with different requirements put on their partition points and an uncut trajectory when the platform was driven from a residential block in an urban environment to the countryside.*

Table 5.5: *The RMSE, mean partition time and mean iteration time and average submap size at different dynamic submap requirements.*

Partition point	RMSE	Mean partition time	Mean loop time	Average submap size
Sparse < 1	0.1997m	0.0137s	0.0004s	111.2m
Sparse < 2	0.6562m	0.0131s	0.0014s	101.1m
Sparse < 3	0.5809m	0.0135s	0.0013s	101.0m
Dense > 4	0.0626m	0.0447s	0.0008s	222.2m
Dense > 5	0.0989m	0.0449s	0.0007s	222.3m
Dense > 6	0.1075m	0.0475s	0.0010s	222.2m

Figure 5.7 show the same trends as the results from the Urban route. The RMSE measure seems overall smaller for partitioning in a dense area with a

higher Mean partition time than for a sparse area. The Mean loop time is relatively constant. The Average submap size for partitioning the map in a landmark sparse area is around 100m and for partitioning the map in a landmark dense area around is 200m.

5.4 Dynamic Partitioning of the Map with Loop Closure

In this section, we present the results from the different cases of dynamic partitioning when combined with loop closure. The cases are the same as Section 5.3. The results show the effect of partitioning the map in either a landmark dense or a landmark sparse area when loop closure was performed, compared to a GNSS aided estimate of the full map. The uncut trajectory and GNSS aided estimate were kept as reference and have not been loop closed. The platform was working on prerecorded data which resulted in same conditions for all the different configurations.

5.4.1 Urban Route

The platform was tested on the urban route. The result is presented in Figure 5.11, picturing the trajectories of submaps with different requirements put on their partition points and in Figure 5.12 where the absolute error between those trajectories and the GNSS reference is shown.

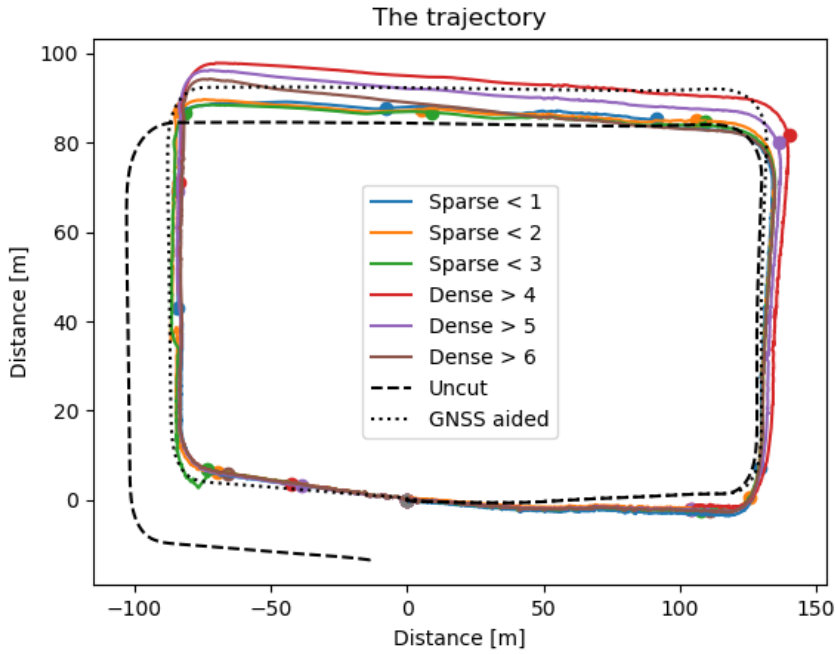

Figure 5.11: *The trajectories with different requirements put on their partitioning points when the platform was driven around a residential block in an urban environment.*

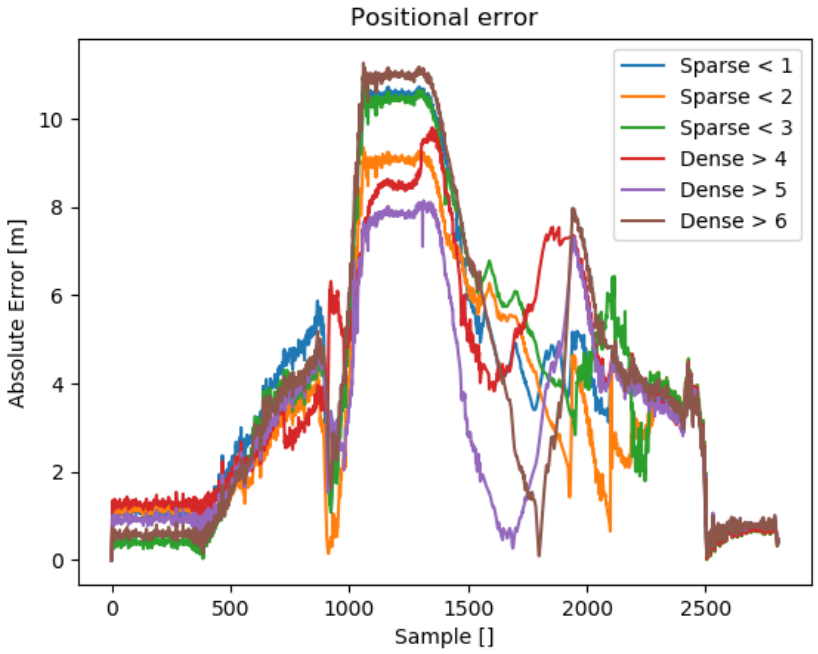

Figure 5.12: *The absolute error between the trajectories and the GNSS reference when the platform was driven around a residential block in an urban environment.*

Table 5.6: *The RMSE and average submap size for different dynamic submap requirements.*

Partition point	RMSE	Average submap size
Sparse < 1	5.1349m	139.4m
Sparse < 2	4.4431m	118.4m
Sparse < 3	5.1750m	118.7m
Dense > 4	4.8201m	144.4m
Dense > 5	4.0145m	143.2m
Dense > 6	5.2813m	232.1m

In Figure 5.11 the partition points do not follow the GNSS aided trajectory as well as with the static submaps. This is especially clear in the top side of the rectangle. Figure 5.12 shows a similar picture as for the static submaps except that it has a lot more noise. The RMSE shows no clear trend.

6

Discussion

In this chapter a discussion is presented about the results from Chapter 5 and the chosen methods of the master thesis. The presented loop closure detection algorithms are also discussed. Finally some conclusions and suggestions of further research is presented.

6.1 Result

When looking at static partitioned submaps, the different submap sizes do not appear to affect the positional error very much when compared to the overall error. The trend seems to be that longer submaps have a smaller positional error which would agree with conclusions from a pure theoretical point of view. When partitioning the map into submaps the smoothing becomes closer to the plain filtering case as the submaps gets smaller. This limits the information the algorithm has access to and limits its performance. More specifically this limits the ability of the algorithm to handle the growing linearization error that appears with larger maps, which for iSAM2 is handled by Algorithm 3 in Chapter 2.

When looking at the placement of the partition points, it seems hard to draw any conclusion about if it is better to partition the map in a dense or sparse area. In both the unclosed routes the Sparse < 1 seems to have a low positional error leading to the conclusion that it might be preferable. This result is contradicted by the result shown in the route Urban to Countryside where the partition points in dense areas have a lower positional error. This can be the result of larger submaps as a consequence of a shortage of landmark dense areas along the route, which can be seen in Table 5.5. This behaviour is not present in the Urban route though, shown in Table 5.5.

The partition points for the submaps seem to be quite on point with the GNSS reference when closing the loop as discussed before. This implies that smaller submaps with more partition points would be advantageous. However, it is difficult to verify this idea since no RMSE measure is gathered at only the partition points. A thought is that, when just looking at the geometrical placement of the partition points, it would be preferable to partition the map in sharp turns and corners of the trajectory. This remark needs to be further explored to be able to conclude anything and is somewhat contradicted by the dynamic loop closed submaps which don't have all the partition points on the GNSS reference.

One aspect of submap sizes and loop closure is that the reduced loop closed factor graph will at some point be too large to optimize. This event will happen later when using larger submaps, since more nodes will be reduced, making it possible to form longer loops. The event will eventually occur however, on the assumption that larger and larger loops are formed. One way to solve this is to recursively reduce the submaps but this solution needs to be further explored.

When looking at submap sizes from a loop closure detection point of view, using the GLARE, GLAROT or GLAROT 3D, the chosen number of landmarks could correspond to a submap for convenience, though this is not a requirement. Smaller submaps would in that case be quicker than a larger map, whereas a larger map would grant fewer candidates that needs to be further validated. A configuration with smaller submaps could also detect loop closure more easily when just crossing a previous trajectory than one with larger submaps. The overlapping area of two candidates would be bigger compared to the full size of those candidates.

6.2 Test Setup

When looking at the test setup, such as chosen routes, algorithms etc. a lot of remarks can be made. First of all, the chosen routes turned out to be too short with respect to the chosen submap sizes. This resulted in missing RMSE, Mean Partition Time and Mean Loop Time for some results. It also resulted in quite small sample sizes for the different submap sizes.

Another remark about the chosen submap sizes is that a larger range of sizes could have been chosen. Larger steps between the static submap sizes might result in bigger differences between large and small submaps.

The Condensed Measurement algorithm was chosen since it seemed easier to combine with submap storage and loop closure than the TSAM algorithms. The TSAM algorithms seemed more focused on solving big factor graphs using map partitioning instead of storing them.

When optimizing the separate submaps for calculation of the Condensed mea-

surements the optimization was fixed to 100 iterations. The reason for this was that the normal threshold bound optimization would not optimize the submaps more than one iteration for unknown reason. The impact of this iteration bound optimization might be that with larger submaps optimization does not have the time to converge assuming that larger submaps are more difficult to optimize. This in turn would increase the positional error for larger submaps.

6.3 Loop Closure Detection

Loop closure detection was planned to have a bigger part of the thesis but due to shortage of time only a short theory chapter was added. The methods presented were chosen to fit the software system modules and their interfaces. When choosing a candidate algorithm GLARE, GLAROT or GLAROT 3D seems most promising due to simplicity and that they seem to be easy to integrate with the current platform. Since they have a rising complexity, with GLARE being the simplest and GLAROT 3D being the most complex, a good idea might be to start with an implementation of GLARE and continue with more complex methods if needed. The Scan Context method can be an alternative to GLARE methods but needs LIDAR scans to work. It seems more bound to roads since it is more dependent on the origin of the scan and might not work as well in open terrain.

When looking at the different validation methods presented, the Generalized-ICP appears to be a good starting point for implementation, mostly due to the fact that an open implementation of the algorithm already exists in PCL [4]. It also produces a transform between the two submaps which is needed to close the loop. The Correspondence graph only compares the Euclidean distances and does not produce a direct estimate of the transform between the two submaps which makes it less attractive. This is true for the Hough Data Association as well though it is considering the rotation between the submaps during its calculation which might make it easier to estimate a submaps transform from it.

The Scan Context method can also be considered as a validation candidate since it uses LIDAR scans and is therefore independent of landmarks. This would make the platform more robust towards faulty loop closure detection due to bad landmark estimates.

6.4 Conclusion

This master thesis set out to investigate the effect of map partitioning and loop closure in a factor graph based SAM system. To concretize the problem, four questions were formulated which are stated in Chapter 1.

The first question is:

- How can map partitioning be integrated into the GTSAM algorithm?

Map partitioning could be integrated into the GTSAM algorithm in mainly two direct ways. Either by partitioning of the factor graph which has been done in this thesis or by partitioning the Bayes tree which was considered but discarded in this thesis due to a more complicated implementation. There are a lot more ways map partitioning can be achieved but since the main data structures used by the GTSAM algorithm are a factor graph and a Bayes tree these are considered the main ones. The partitioning of the factor graph breaks apart the latest part of factor graph until the partition point is reached and saves the components. The components are then put together into a new factor graph and map partitioning has been achieved.

Continuing with the second question:

- How does the size of the partitioned submaps affect the estimated position of the platform, with and without loop closures?

When partitioning the map without loop closure a bigger submap seems preferable with respect to positional error. On the other hand when partitioning the map and loop closure is achieved smaller submaps seem preferable with respect to positional error.

The third question is:

- How does the number of landmarks at the partition point affect the estimated position with and without loop closures?

No clear conclusion can be drawn in the thesis about the landmark density at the partition points during this thesis. The result shows in one case that partitioning the map in really landmark sparse areas might be preferable and in another that partitioning the map in landmark dense areas might be preferable.

The last question is:

- Which loop closure detection techniques are reasonable to consider with respect to the platform?

When comparing the chosen loop closure detection and validation algorithm the GLARE algorithm together with the Generalized-ICP seems like a good starting point. If more complex algorithms are needed GLAROT and GLAROT 3D might be considered together with the Scan Context algorithm as validation method.

6.5 Further Research

In the scope of this master thesis a method for partitioning the map has been presented together with some loop closure detection theory. It is one step towards achieving full scalability and automatic loop closure detection but to accomplish that further work is required.

One way to continue the project would be to investigate how to implement an efficient loop closure detection algorithm. This could be coupled with some sort of database implementation to efficiently keep track of the stored submaps which the algorithm would compare against its current map.

Another way to continue the research is to investigate some sort of recursive implementation of the map partitioning. Since the reduced factor graph used to obtain loop closure can become too large to optimize one would have to reduce it in order to handle bigger loops. The new reduced factor graph could be considered its own submap consisting of smaller submaps. This direction could also be coupled with an implementation of a database to keep track of the submaps at different submap levels. If this can be achieved then the platform would obtain full scalability.

A third suggestion would be to research how to use the stored submaps, either explored previously by the platform itself or explored by another system? To determine if the factor graph of a submap can be used to improve the estimated position, by predicting the previously observed landmarks when revisiting that area. Submaps that are revisited a lot would need more and more memory since not only the position of a landmark is saved but also every measurement of the landmark that are gathered within the submap. All the previous positions of the platform together with the transitions between them would also be saved. A pruning of the revisited submap would therefore be needed. One way could be to fuse sets of connected factors into one factor that would summarize their information. A method for fusing factors together is presented in [25] which could be a direction to explore.

Appendix

A

Linear Algebraic Operations

Appendix A shows how different algebraic operations are performed.

A.1 QR Factorization

This section shows how QR factorization is performed on a linear system. This is equivalent to the elimination algorithm presented in Chapter 2.
The factorization is performed to avoid performing inverses of the sparse measurement Jacobian matrix $A \in \mathbb{R}^{mxn}$. A is divided as follows

$$A = Q \begin{bmatrix} R \\ 0 \end{bmatrix} \tag{A.1}$$

Where $R \in \mathbb{R}^{nxn}$ is the upper triangular information matrix and Q is an orthogonal matrix. ($QQ^T = I$). When used to solve equation $y = Ax$, where $x \in \mathbb{R}^{mx1}$ and $y \in \mathbb{R}^n$ the solution is given by

$$\begin{bmatrix} y_0 \\ y_e \end{bmatrix} = Q^T y = Q^T Q \begin{bmatrix} R \\ 0 \end{bmatrix} x \tag{A.2}$$

where $(y_e y_e^T)$ becomes the residual of the least square problem. The solution can be gained from

$$y_0 = R\hat{x} \tag{A.3}$$

where simple back substitution can be used to solve for $\hat{x}$. [11] [1]

Bibliography

[1] Fredrik Gustafsson. *Statistical Sensor Fusion*. Studentlitteratur, third edition, 2018.

[2] F. Dellaert and M. Kaess. Square root sam: Simultaneous localiza- tion and mapping via square root information smoothing,. *Intl. J. of Robotics Research*, 25, 2006.

[3] Robot operating system. `https://www.ros.org/`. Accessed: 2019-11-29.

[4] Point cloud library. `http://pointclouds.org/`. Accessed: 2019-11-24.

[5] Georgia tech smoothing and mapping. `https://borg.cc.gatech.edu/`. Accessed: 2019-11-24.

[6] Max Holmberg. Development and evaluation of a robocentric slam algorithm. Master's thesis, Uppsala University, 2018.

[7] Hugh Durrant-whyte and Tim Bailey. Simultaneous localisation and mapping (slam): Part i the essential algorithms, 2006.

[8] Torkel Glad and Lennert Ljung. *Reglerteknik, Grundläggande teori*. Studentlitteratur AB, Lund, fourth edition, 2014.

[9] Frank Dellaert and Michael Kaess. *Factor Graphs for Robot Perception*. now publishers inc., first edition, 2017.

[10] Kaess M, Johannsson H, Roberts R, Ila V, Leonard JJ, and Dellaert F. *iSAM2*: Incremental smoothing and mapping using the bayes tree. *The International Journal of Robotics Research*, 2011(1), 2011.

[11] Michael Kaess, Ananth Ranganathan, and Frank Dellaert. *iSAM*: Incremental smoothing and mapping. *IEEE Transactions on Robotics*, 24(6), 2008.

[12] Kai Ni and Frank Dellaert. Multi-level submap based slam using nested dissection. *The 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010.

[13] Michael Kaess, Viorela Ila, Richard Roberts, and Frank Dellaert. *Algorithmic Foundations of Robotics IX, pp 157-173: The Bayes Tree: An Algorithmic Foundation for Probabilistic Robot Mapping*. Springer, Berlin, Heidelberg, first edition, 2011.

[14] Robert E. Tarjan and Mihalis Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *Society for industrial and applied mathematics*, 13, 1984.

[15] Timothy A. Davis, John R. Gilbert, Stefan l. Larimore, and Esmond G. Ng. A column approximate minimum degree ordering algorithm. *ACM Transactions on Mathematical Software (TOMS)*, 30, 2004.

[16] Kai Ni, Drew Steedly, and Frank Dellaert. Tectonic sam: Exact, out-of-core, submap-based slam. *IEEE International Conference on Robotics and Automation*, 2007.

[17] Giorgio Grisetti, Rainer Kümmerle, and Kai Ni. Robust optimization of factor graphs by using condensed measurements. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012.

[18] Randall Smith, Matthew Self, and Peter Cheeseman. Estimating uncertain spatial relationships in robotics. *Machine Intelligence and Pattern Recognition*, 5, 1988.

[19] Timo Röhling, Jennifer Mack, and Dirk Schulz. A fast histogram-based similarity measure for detecting loop closures in 3-d lidar data. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2015.

[20] Marian Himstedt, Jan Frost, Sven Hellbach, Hans-Joachim Böhme, and Erik Maehle. Large scale place recognition in 2d lidarscans using geometrical landmark relations. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014.

[21] Fabjan Kallasi and Dario Lodi Rizzini. Efficient loop closure based on falko lidar features for online robot localization and mapping. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2016.

[22] Dario Lodi Rizzini. Place recognition of 3d landmarks based on geometric relations. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2017.

[23] Giseop Kim and Ayoung Kim. Scan context: Egocentric spatial descriptor for place recognition within 3d point cloud map. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[24] Aleksandr V. Segal, Dirk Haehnel, and Sebastian Thrun. Generalized-icp. *Robotics:Science and Systems*, 2009.

[25] Luca Carlone, Zsolt Kira, Chris Beall, Vadim Indelman, and Frank Dellaert. Eliminating conditionally independent sets in factor graphs: A unifying perspective based on smart factors. *IEEE International Conference on Robotics & Automation (ICRA)*, 2014.

Index