Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Even Skjellaug

Feature-Based Lidar SLAM for Autonomous Surface Vehicles Operating in Urban Environments

Master's thesis in Cybernetics and Robotics
Supervisor: Edmund Brekke
June 2020

NTNU
Norwegian University of Science and Technology

Preface

This master thesis is written as part of the Cybernetics and Robotics study program at the Norwegian University of Science and Technology (NTNU). I would like to thank my supervisor Edmund F. Brekke for giving me the possibility to work on this project, and for taking the time to counsel me and guiding me on the correct path. I would also like to thank Annette Stahl for the help and feedback she has provided.

This thesis builds on the work done in two previous master's theses [1], [2]. It is also preceded by a specialization project [3], which looked into what keypoint extractors and descriptors that should be used to perform laser odometry. A special thanks goes out to Marius Strand Ødven, who recorded the data used in this thesis as a part of his project [1].

This master project is instantiated under the Autoferry Project at the Centre of Autonomous Marine Operations and Systems (AMOS) at NTNU. The goal of this project is to develop a feature-based lidar SLAM system utilizing iSAM2 for milliAmpere.

Abstract

In this thesis a feature-based Simultaneous Localization And Mapping (SLAM) system utilizing the iSAM2 framework for Autonomous Surface Vehicles (ASV)s operating in urban environments has been developed. This system can be used as a back-up system for the Global Navigation Satellite System (GNSS), if it were to be jammed, or if signals are lost. The complete system, using the ISS keypoint extractor and the SHOT descriptor, both available in Point Cloud Library (PCL), is implemented using the iSAM2 framework available in the Georgia Tech Smoothing and Mapping (GTSAM) library, and is validated on real lidar data recorded on-board the autonomous ferry prototype milliAmpere.

This thesis will show how the system is developed, how loop closure is performed, and how the Inertial Measurement Unit (IMU) and GNSS are fused into the system. The map created by the system, and the uncertainty of the estimates will be presented alongside the resulting trajectory.

The resulting trajectory achieved by this system is compared to the ground truth recorded by a Real Time Kinematics (RTK)-GNSS, and the results are more accurate than a standard GNSS receiver, both for the two dimensional xy-plane, and for the z-direction. The loop closure is also shown to be consistent, making this one of the first feature-based lidar SLAM systems which utilize keypoints and descriptors in order to perform loop closure. The system is also shown to be a great improvement to the systems that have been implemented for milliAmpere in the two previous master's theses.

Sammendrag

I denne oppgaven har et feature-basert lokaliserings og kartleggings (SLAM) system som bruker iSAM2 rammeverket for autonome overflatefartøy (ASV)s i urbane miljø blitt utviklet. Dette system kan bli brukt som et back-up system for satellittbasert navigasjon (GNSS) i tilfelle det blir jammet, eller hvis signalet skulle gå tapt. Systemet bruker ISS keypoint ekstraktoren og SHOT deskriptoren som begge er tilgjengelige i Point Cloud Library (PCL), og er implementert ved hjelp av iSAM2 rammeverket, som er tilgjengelig i Georgia Tech Smoothing and Mapping (GTSAM) biblioteket. Systemet er validert på ekte lidar data som har blitt tatt opp på den autonome fergeprototypen milliAmpere.

Denne oppgaven vil vise hvordan systemet er bygd opp, hvordan stedsgjenkjenning er implementert, og hvordan bevegelsessensorer (IMU) og GNSS er integrert inn i systemet. Kartet som systemet lager og usikkerhetsestimatene vil bli presentert sammen med den resulterende trajektorien.

Resultatene som dette systemet oppnår er sammenliknet med ground truth data som er tatt opp av en RTK-GNSS, som er veldig nøyaktig satellittbasert navigasjon. Resultatene som er oppnådd er mer nøyaktig enn hva en standard GNSS mottaker er spesifisert til å være, både for lokalisering i xy-planet, men også med tanke på høyden. Stedsgjenkjenning som er implementert er også vist å være konsistent, som fører til at dette er et av de første feature-baserte lidar SLAM systemene noensinne som bruker keypoints og deskriptorer for å implementere stedsgjenkjenning. Dette systemet er også en klar forbedring fra de systemene som har blitt implementert på milliAmpere i de foregående masteroppgavene.

List of Tables

List of Figures

List of Algorithms

Acronyms

2D SC 2D Shape Context. 45

3D SC 3D Shape Context. 3, 43, 45, 46

ASV Autonomous Surface Vehicles. i, iii, 1, 3, 53, 58, 66, 78

BLAM Berkeley Localization And Mapping. 3, 91

CCOLAMD Constrained Column Approximate Minimum Degree. 38

CNN Convolutional Neural Network. 22

COLAMD Column Approximate Minimum Degree. 38

DFS Depth-First Search. 31

DOF Degrees of Freedom. 27, 30

DoG Difference-of-Gaussian. 42

DTAM Dense Tracking and Mapping. 12

DVL Doppler Velocity Log. 5

EKF Extended Kalman Filter. 2, 21

FAB-MAP Fast Appearance-Based Mapping. 12

FastSLAM Factored Solution To SLAM. 2, 21

FPFH Fast Point Feature Histograms. 3, 43, 45

GLONASS Global Navigation Satellite System. 18

GNSS Global Navigation Satellite System. i, iii, 1, 3, 4, 10, 16, 18, 50, 51, 53, 58, 62, 63, 65–74, 78–83, 91

GPS Global Positioning System. 18

GTSAM Georgia Tech Smoothing and Mapping. i, iii, 12, 13, 15, 24, 48, 61, 62, 92

ICP Iterative Closest Point. 1, 3, 6, 30, 57, 91

IMU Inertial Measurement Unit. i, iii, ix, 3–5, 10, 16, 18, 24–27, 31, 51, 53, 61, 62, 66, 67, 70–74, 77, 79–82, 88

iSAM Incremental Smoothing And Mapping. 13

iSAM2 Incremental Smoothing And Mapping 2. i, iii, 1–3, 13, 35, 38, 39, 53, 58, 61, 62, 69, 70, 72, 73, 79, 80, 91, 92

ISS Intrinsic Shape Signatures. i, iii, 3, 41–43, 53, 57, 65

JCBB Joint Compatibility Branch and Bound. 2, 19

LeGO-LOAM Lightweight and Ground-Optimized Lidar Odometry and Mapping. 2, 3, 13, 27, 29, 30, 91

LOAM Laser Odometry And Mapping. 2, 3, 27–30, 58, 91

LSD-SLAM Large-Scale Direct Monocular SLAM. 2, 13, 33, 60

LSTM Long Short-Term Memory. 22

MAP Maximum a Posteriori. 6, 7, 9, 53

MEMS Microelectromechanical Systems. 18

MSS Minimum Sample Set. 22

NARF Normal Aligned Radial Feature. 3, 41–43, 46

NDT Normally Distributed Transform. 1, 6

NTNU the Norwegian University of Science and Technology. 1, 49

ORB Oriented FAST and Rotated BRIEF. 2, 13, 32–34, 53, 58

PCA Principal Component Analysis. 43, 45

PCL Point Cloud Library. i, iii, 3, 42, 44, 47, 55–57, 60, 92

PDF Probability Density Function. 7

PFH Point Feature Histograms. 3, 43, 45

RANSAC Random Sample Consensus. 2, 21, 57, 58, 60, 61, 65

RNN Recurrent Neural Network. 22

ROS Robot Operating System. 3, 27, 47

RSE Root Square Error. 18, 65–67, 70–74, 77, 79, 81, 83

RTK Real Time Kinematics. i, iii, 1, 18, 50, 62, 63, 68, 79

SHOT Signatures of Histograms of OrienTations. i, iii, viii, 3, 43, 44, 46, 53, 57, 58, 61

SIFT Scale Invariant Feature Transform. 3, 28, 41–44, 46

SLAM Simultaneous Localization And Mapping. i, iii, 1–3, 5–13, 19, 21, 22, 27, 28, 30, 32–35, 38, 39, 47–49, 53, 55, 57, 58, 60, 61, 75, 78–82, 91, 92

SPFH Simplified Point Feature Histogram. 45

SVO Semi-Dense Visual Odometry. 34

UKF Unscented Kalman Filter. 3, 91

USC Unique Shape Context. 3, 43, 46

Table of Contents

Chapter 1

Introduction

1.1 Background

Autonomous Surface Vehicles (ASV)s are autonomous vehicles that operate on the surface of the water. The first ASV was designed by MIT in 1993 for various missions, but it was mainly used to collect bathymetry data [15]. Since then, there has been extensive research on ASVs and their applications [16]. One of these applications are autonomous ferries, which can be used for transportation.

Autonomous ferries do not only have the potential to be more cost effective and environmentally friendly than manned ships, but they also have the potential to be safer. Human errors cause more than 60% of ferry accidents, and are accountable for more than 70% of the fatalities in these accidents [17].

The Autoferry project at the Norwegian University of Science and Technology (NTNU) is a project to develop a fully autonomous ferry to transport passengers between Ravnkloa and Brattøra as a replacement for the Fløtmann, which used to row across the passage until 1965 [18]. The ferry will be fully electrical and should be able to transport people with the push of a button. milliAmpere is the demonstration platform for the Autoferry project, and it will be described in more detail in Chapter 3.

For milliAmpere to be fully autonomous, it needs to be able to localize itself in its environment. To do this, quality sensors are needed, and milliAmpere is therefore equipped with an RTK-GNSS, which is accurate down to a few cm. RTK-GNSS is accurate, but it is expensive, and can be jammed, both intentionally or unintentionally. A back-up or replacement system is therefore important for the ASV to be fully autonomous. One such back-up system could be a Simultaneous Localization And Mapping (SLAM) system, as it can be accurate, it can not be jammed, and it will create a map of the surroundings.

SLAM can be performed either direct or indirect. The direct methods for lidar SLAM usually utilize scan-matching, while the indirect methods are feature-based. Some state-of-the-art scan-matching algorithms are Iterative Closest Point (ICP) [19] and Normally Distributed Transform (NDT) [20]. Scan-matching takes two full point clouds and tries to calculate the transformation between them by matching each individual point from one

point cloud to the next. Feature-based methods, on the other hand, extract keypoints from the full point cloud, to make a sparser point cloud. Feature-based SLAM has the potential to be both faster and more accurate than scan-matching based SLAM. However, this requires both stable and repetitive keypoints, which are hard to extract.

Developing a feature-based SLAM system is not trivial, as there are currently few lidar SLAM systems supporting loop closure. Four state-of-the-art lidar SLAM systems are Hector-SLAM [21], LOAM [22], LeGO-LOAM [23] and Google Cartographer [24]. These state-of-the-art lidar SLAM systems include both indirect and direct methods, as LOAM and LeGO-LOAM are feature-based, and Hector-SLAM and Google Cartographer use scan-matching. Of these systems, only LeGO-LOAM and Google Cartographer support loop closures, and both of these systems perform the loop closures by scan-matching.

These systems focus mainly on ground vehicles, and among these papers, only [21] reports an implementation on a surface vehicle. However, this surface vehicle was not operating in a harbour environment, but rather a lake with dense vegetation. In contrast to surface vehicles, there are large datasets available online for ground vehicles, for example the KITTI dataset [25], as many autonomous cars use lidars.

Factor graphs [26] have become the de facto standard for the formulation of SLAM problems. Using factor graphs, it is possible to solve the full SLAM problem online using onboard computers, while also providing an insightful visualization of the problem. Several state-of-the-art SLAM system utilize the factor graph frameworks, ORB-SLAM [7] and LSD-SLAM [10] use the g2o framework [27] for their back-end, while LeGO-LOAM [23] uses the Incremental Smoothing And Mapping 2 (iSAM2) framework [5].

iSAM2 is a fully incremental, graph-based framework. It utilizes the Bayes tree data structure, incremental reordering and fluid relinearization to obtain a fully incremental algorithm. This algorithm is used for graph-based SLAM by combining the advantage of the graphical model with sparse linear algebra. Such a factor graph based system also makes it possible to fuse in additional information from other sensors in a structured way. New sensors can be fused into the system by adding new nodes to the factor graph, and it is as effective as fusing in sensors using conventional filtering techniques, like the Extended Kalman Filter (EKF) [28].

One of the major challenges in performing SLAM in harbour environments is that a lot of the information is lost due to the fact that the lidar does not reflect water as well as solid ground surfaces. As the area is not contained, information is not only lost due to lacking reflection of the water, but a lot of the rays will not be reflected as they do not hit any object. This results in a sparser point cloud than what is usually obtained by ground vehicles, or vehicles in contained environments.

Another major challenge for SLAM is data association. Some state-of-the-art methods for data association are Joint Compatibility Branch and Bound (JCBB) [29], the Auction algorithm [30], the Hungarian algorithm [31], FastSLAM [32], and Random Sample Consensus (RANSAC) [33]. In the last couple of years there has also been research on how to perform the data association using deep learning [34], [35], [36]. Performing long term data association, or loop closure as it is commonly known, is one of the hardest problem when solving SLAM, and using a suitable method is therefore essential to make sure that there are as few false positives as possible.

The objective of this master thesis is to develop a feature-based lidar SLAM system for

ASV's operating in an urban harbour environment. This is done by using ISS keypoints with SHOT descriptors, and fusing in Global Navigation Satellite System (GNSS) and Inertial Measurement Unit (IMU) data in the iSAM2 framework. The system developed in this thesis might be one of the first 3D lidar SLAM system which utilizes keypoints and descriptors for loop closures, as both LeGO-LOAM and Google Cartographer utilizes scan-matching to perform loop closure.

This thesis builds on two earlier master projects on SLAM using lidar on milliAmpere. The data used in both of these master projects are the same as the one used in this thesis, and it was collected by Marius Ødven [1]. The first project [1] compared Hector-SLAM [21], LOAM [22] and BLAM [37], which are all lidar SLAM or odometry methods available open-source in Robot Operating System (ROS). The thesis also goes into detail about the sensors and equipment used on milliAmpere. It was concluded that none of the methods were good enough to be used as the primary pose estimator for milliAmpere. Hector-SLAM was the most promising one, but it was not accurate enough to give a map for autonomous docking [1].

The second project [2] was more focused on localization using particle filters. The conclusion from this thesis was that even though one can achieve a quite accurate localization, particle filters are not computationally efficient enough to be able to run on the boat in real-time. Both Unscented Kalman Filter (UKF) and Iterative Closest Point (ICP) were considered, and ICP was found to be the better option. This thesis also concluded that even though using particle filters provide a good localization, it is not suitable to be used as the primary pose estimator for milliAmpere as it is not able to run in real-time [2].

In addition to the two earlier master projects, it also builds on the specialization project [3] which developed feature-based laser odometry using keypoint extractors and detectors available in the Point Cloud Library (PCL). A similar comparison has been made when it comes to object recognition [38], [39], [40], [41], however, it has not been done in the context of laser odometry or SLAM. The keypoint extractors implemented were ISS [42], NARF [43], Harris3D [44] and SIFT [45] and these will be further discussed in Chapter 4 along with the keypoint descriptors, NARF [43], PFH [46], FPFH [47], SHOT [12], 3D SC [48] and USC [49]. Based on the specialization project, a conference paper for the 2020 Fusion conference has been written [50]. This conference paper can be seen in Appendix A. In this conference paper, the odometry solution was further improved and integrated in the iSAM2 framework. The conclusion was the the combination of ISS keypoints and SHOT descriptors gave the best results for performing feature-based laser odometry on milliAmpere.

1.2 Thesis outline

This thesis is organised in the following way. In chapter 2, odometry, SLAM and background information will be introduced. Then an introduction to the milliAmpere platform, sensors and software used in the thesis will follow in Chapter 3. Chapter 4 will introduce state-of-the-art SLAM systems for both lidar and visual SLAM, and the iSAM2 framework, while Chapter 5 introduces the keypoints extractors and descriptors that were investigated. Chapter 6 will introduce the SLAM system developed in this thesis. Chapter 7 will then present the results from this SLAM system, first for only the odometry, then for

fusing in the IMU data, the GNSS and including loop closure.

Chapter 2

Background

In this chapter, the background information for a SLAM system will be presented. As odometry is a key part of a SLAM system, it will be presented first, followed by probability theory and factor graphs. After this a survey over state-of-the-art SLAM is presented, and some open-source factor graphs SLAM libraries will be introduced. This is followed by an introduction to some nonlinear solvers, and a short introduction to some highly used sensors in SLAM. After that, several state-of-the-art data association techniques are presented, followed by some information on transformation matrices and an introduction to Lie theory. Lastly, preintegration will be presented.

2.1 Odometry

Odometry is the use of sensors to estimate change in pose over time [51]. This can be done by using IMU, wheels, cameras, lidars, Doppler Velocity Log (DVL) or other sensors. Wheel odometry has been one of the most used because of its simplicity, however it is not as accurate as visual or lidar odometry [52], as it suffers from wheel slippage in slippery and uneven terrain [53]. Wheel odometry is also constrained to ground vehicles and can obviously not be used for surface vehicles. Visual odometry uses one or multiple cameras to estimate changes in pose over time, while laser odometry uses a lidar. One major problem using odometry is drift, and this will affect all forms of odometry. All sensors experience noise and because odometry is built on imperfect data and does not have the possibility to update previous estimates, it will experience drift in its pose estimates. A solution to this is posed by SLAM, which includes long-term tracking in the form of loop closures. Loop closures will update earlier poses in addition to the current one, and it is therefore possible to remove the drift from the system.

2.1.1 Scan-matching vs feature-based laser odometry

As a lidar is used in this thesis, the focus will of this section will be on laser odometry. There are two main ways to perform laser odometry, scan-matching or feature-based.

Scan-matching uses the entire point clouds and tries to calculate a rigid body transformation between them. As it uses the entire point clouds, it is computationally heavy and prone of converging to a local minima. Some popular scan-matching algorithms are ICP [19] and NDT [54].

Feature-based odometry extracts keypoints, usually based on corners, lines or curves, from the point clouds. These keypoints needs to be repeatable, as they need to be tracked over time and compared in order to calculate the transformations. The data association is often done using descriptors, to make sure that the same keypoints are being matched between the point clouds. Matching keypoints substantially decreases the computational complexity as the point clouds will be reduced significantly, in addition, searching over data associations for features are computationally less expensive than searching over the space of rigid-body transformations. Feature-based odometry can be both faster and more accurate than scan-matching, however, it does need distinct and repeatable keypoints to work well.

2.2 Probability theory

Probability theory is often used to solve the SLAM problem, and to understand how factor graphs work. As measurements received by a SLAM system are uncertain, they are often modelled using probability theory. The states will then also be uncertain, and will therefore require to be modelled using probability theory. In order to solve for the most likely states, Maximum a Posteriori (MAP) estimation will be used to provide the most accurate results. In order to represent the uncertainty, multivariate Gaussians are normally used. Multivariate Gaussians are also important when considering sensor fusion, and representing how the uncertainty develops over time in a SLAM system.

2.2.1 Maximum a Posteriori estimator

The true states are not known to a SLAM system, however measurements are received from the sensors. $\mathcal{X}_i$ will here refer to the current state, while $\mathcal{X}$ will refer to all the states of the system. The measurements are often corrupted by noise, $z_i = h_i(\mathcal{X}_i) + \eta_i$. This noise is often assumed to be zero-mean Gaussian noise, and the measurement prediction function is therefore only a function of the true state, $\hat{z}_i = h_i(\mathcal{X}_i)$. The measurement error function is therefore: $e_i(\mathcal{X}_i) = h_i(\mathcal{X}_i) - z_i$. Choosing the objective function to be the squared Mahalanobis distance [55] of the all the errors, it can be written in the following way, $f(\mathcal{X}) = \sum_{i=1}^{n} \|h_i(\mathcal{X}_i) - z_i\|^2_{\sum_i}$. This results in the nonlinear least squares problem:

$$\begin{aligned} \mathcal{X}^* &= \underset{\mathcal{X}}{\operatorname{argmin}} f(\mathcal{X}) = \underset{\mathcal{X}}{\operatorname{argmin}} \sum_{i=1}^{n} \|e_i\|^2_{\sum_i} \\ &= \underset{\mathcal{X}}{\operatorname{argmin}} \sum_{i=1}^{n} \|h_i(\mathcal{X}_i) - z_i\|^2_{\sum_i} . \end{aligned} \tag{2.1}$$

It turns out that the nonlinear least squares solution to this problem is the MAP estimate [56].

The MAP estimate will utilize a likelihood function in order to estimate the most likely state based on the available measurements:

$$\begin{aligned}\hat{\mathcal{X}}_{\text{MAP}} &= \underset{\mathcal{X}}{\operatorname{argmax}}\, p(\mathcal{X}|\mathcal{Z}) \\ &= \underset{\mathcal{X}}{\operatorname{argmax}}\, \frac{p(\mathcal{Z}|\mathcal{X})p(\mathcal{X})}{p(\mathcal{Z})} \\ &= \underset{\mathcal{X}}{\operatorname{argmax}}\, l(\mathcal{X};\mathcal{Z})p(\mathcal{X}) \\ l(\mathcal{X};\mathcal{Z}) &\propto p(\mathcal{Z}|\mathcal{X}).\end{aligned} \tag{2.2}$$

This shows that maximizing the likelihood function, $l(\mathcal{X};\mathcal{Z})$ will give the same result as maximizing the probability function $p(\mathcal{Z}|\mathcal{X})/p(\mathcal{Z})$. The MAP estimate will therefore be a maximization of the likelihood function multiplied with the prior probability $p(\mathcal{X})$. This prior probability, $p(\mathcal{X})$, is often assumed to be a multivariate Gaussian.

2.2.2 Multivariate Gaussian

The multivariate Gaussian is a generalization of the univariate Gaussian, and is one of the key constructions that underlies SLAM and virtually all of sensor fusion [4]. This is because it is the only model that makes it possible to maintain all the useful correlations in a systematic fashion. A multivariate Gaussian distribution can be assigned using a expectation vector μ and a symmetric positive definite covariance matrix P, according to

$$\mathcal{N}(x;\mu,P) = \frac{1}{(2\pi)^{\frac{n}{2}}|P|^{\frac{1}{2}}}\exp\left(-\frac{1}{2}(x-\mu)^T P^{-1}(x-\mu)\right). \tag{2.3}$$

An example of a univariate Gaussian Probability Density Function (PDF) can be seen in Figure 2.1. This is a bell curve where the peak location comes from the expectation value and the spread comes from the covariance. This can be extended to several dimensions, and for the two-dimensional Gaussian, the PDF is a bell surface, where an example can be seen in Figure 2.2. Similarly to the bell curve, the peak of a bell surface is given by the expectation vector and the spread is then given by the covariance matrix.

When studying the Multivariate Gaussian, the exponent of Equation (2.3) is what is studied, as a Gaussian can be specified entirely in terms of this exponent. It is on a quadratic form in the variable x, and includes both the expectation vector and the covariance matrix, and can be written as:

$$q(x) = (x-\mu)^T P^{-1}(x-\mu). \tag{2.4}$$

An alternative representation of Equation (2.3) is to represent the multivariate Gaussian in canonical form. This can be seen in the following equation:

$$\mathcal{N}(x;\mu,P) = \exp\left(a + \eta^T x - \frac{1}{2}x^T \Lambda x\right), \tag{2.5}$$

where

Figure 2.1 A bell curve for a univariate Gaussian.

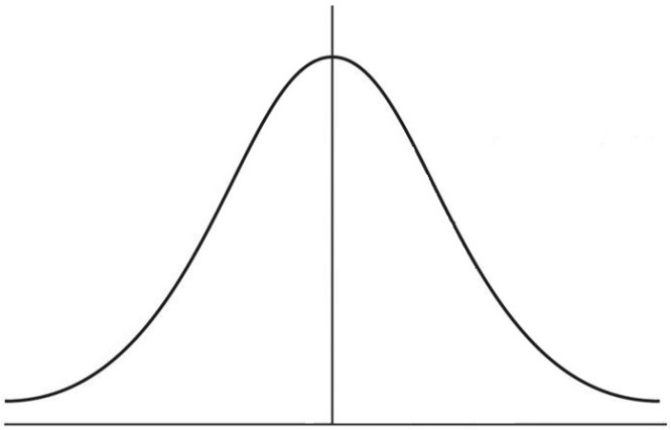
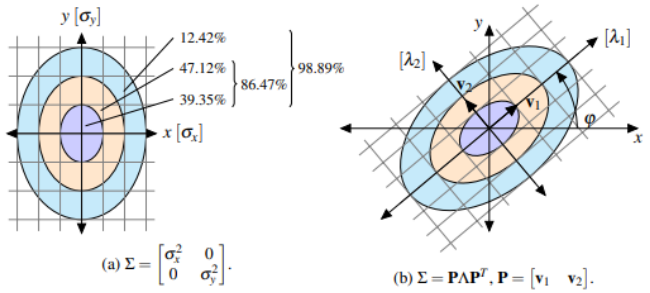


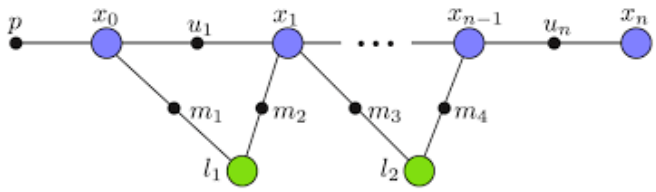
Figure 2.2 Two-dimensional Gaussian distribution. Probability of ellipses. Image and caption taken from [4].



$$
\begin{aligned}
\Lambda &= P^{-1} \\
\eta &= \Lambda\mu \\
a &= -\frac{1}{2}n\ln(2\pi) - \ln|\Lambda| + \eta^T\Lambda\eta.
\end{aligned}
\tag{2.6}
$$

η is often referred to as an information state, while Λ is known as the information matrix, or sometimes the precision matrix. When solving SLAM problems, the information matrix may have a neater structure than the covariance matrix, and the canonical form will also be able preserve the factor-graph structure, which the covariance form is not able to do.

Figure 2.3 Factor graph where x_i are poses, l_i are landmarks, p is the prior, m_i are measurements and u_i is the odometry. Image taken from [5].



2.3 Factor graphs

A factor graph is a bipartite graph [56], meaning that it has two types of nodes: factor nodes and variable nodes. The factor nodes represent conditional probabilities or likelihoods, while the variable nodes represent the states. In addition to the nodes, there are also edges that connect a factor node with a variable node. The factor nodes can also be understood as measurements, as they connect the states. Some examples of measurements can be wheel odometry between the poses and lidar or camera measurements between the poses and the landmarks. An example of a factor graphs can be seen in Figure 2.3. Here the x's, representing the poses, and the l's, representing the landmarks, are the variable nodes while the u's, representing the odometry, and the m's, representing the measurements, are factor nodes.

Factor graphs have become the de facto standard for the formulation of SLAM problems. Graph-based SLAM can be split up into two tasks, the graph construction, also known as the SLAM front-end, and the graph optimization, also known as the SLAM back-end. The front-end constructs the graph using the raw measurements, while the back-end determines the most likely configuration of poses given the factors of the graph. While the front-end is heavily dependent on the sensor measurements, the back-end is sensor agnostic. The front-end and back-end of SLAM will be discussed more in Section 2.4.

One of the main reasons that the factor graphs have become the de-facto standard for SLAM is how it can be factorized. If one assumes that all the measurements are independent, a factor graph problem can be split into many smaler problems that are easier to solve. This makes it so that any factor graph can be easily evaluated for any given value, by simply evaluating each factor individually and multiplying the results. This will result in sparse problems, which can be solved efficiently using fast linear solvers. Local maxima of the posterior can then quickly be found by optimizing over all the factors. This is a MAP problem as introduced in Section 2.2.1 assuming that all the factors have Gaussian priors and noise with zero mean. This MAP problem can be formulated as a nonlinear least squares problem and can be solved using Gauss-Netwton, Levenberg-Marquardt, Powell's Dogleg or other nonlinear optimization methods.

In addition to being suitable for poses and landmarks, as seen in Figure 2.3, factor graphs are also suitable for sensor fusion, as new sensors can be added as new nodes to the graph. Doing this will not greatly increase the complexity of the problem, as all measurements are considered independent. It will, however, make it easier for the optimization to

find the global maxima instead of a local maxima. One example of a sensor that can be fused in to provide additional information is the IMU. The IMU require bias and velocity nodes, and will help the system estimate the velocities alongside the poses. GNSS is another sensor that can be easily fused into a factor graph, providing useful information on the prior of the pose nodes.

2.4 Simultaneous Localization And Mapping

SLAM can be described as the problem to navigate a body in a previously unknown environment, while building and updating a map of the workspace using on-board sensors and computation [6].

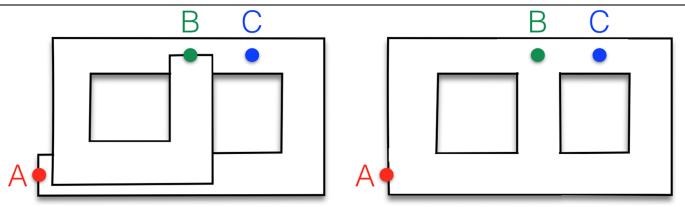
SLAM has been a popular research topic for a long time, but especially in the last decades due to the effect it has for autonomous robots. There are many reasons as to why SLAM is needed in autonomous robots. It is necessary if the robots have to be truly autonomous and not having to be controlled through human input. There are also many situations where the robots cannot rely on external positioning systems like GNSS, this includes when the robots are inside, under water or if the GNSS gets jammed, intentionally or not. The research going forward is focused in how to make SLAM more efficient and to be able to use it for a longer period of time, as this is currently computationally heavy for on-board computers.

SLAM is needed for problems where both the poses and the map of the surroundings are unknown. If the poses were to be known, for example by using GNSS, it would be a mapping problem, where the pose knowledge could be used to build the map. If, on the other side, the map of the area is known, this would be a localization problem where the objective would be to localize inside of the currently known map. In the case of SLAM however, neither the poses of the robot nor the surrounding map are known, and on top of that there will be noise corrupting both the exteroceptive and proprioceptive sensors. Exteroceptive sensors are the sensors that acquire information about the surrounding environment, while proprioceptive sensors measure values internal to the system. SLAM will therefore allow the system to travel in new environments without having to create a map of it beforehand or relying on sensors such as the GNSS to be able to create a map of the environment.

SLAM can be defined in two manners, online SLAM, $p(x_k, m|Z_{1:k}, U_{1:k})$, and full SLAM $p(x_{1:k}, m|Z_{1:k}, U_{1:k})$. Online SLAM will only update the current pose estimate, and will therefore not be able to correct previous poses with the new information gained. Full SLAM, on the other hand, will update all of its pose estimates at every point in time, meaning it will provides a better estimate, as it will correct previous poses as well as the current one. Previously, full SLAM was computationally more complex than online SLAM, however, this has changed due to the sparse nature of the SLAM problems. With efficient sparse solvers [5], it is possible to run full SLAM real-time using on-board computers. Full SLAM will therefore be a better solution when looking to create an accurate map, and when creating a full trajectory over the poses.

As introduced in Section 2.3, SLAM can be divided into a front-end, and a back-end. The front-end performs both short-term and long-term tracking, where short-term tracking is the same as odometry which is described in detail in Section 2.1. Long-term tracking is

Figure 2.4 Left: map built from odometry. The map is homotopic to a long corridor that goes from the starting position A to the final position B. Points that are close in reality (e.g., B and C) may be arbitrarily far in the odometric map. Right: map build from SLAM. By leveraging loop closures, SLAM estimates the actual topology of the environment, and "discovers" shortcuts in the map. Image and caption taken from [6].



the ability to perform loop closures, giving the system the possibility to correct previous poses in addition to the current pose. The back-end is responsible for the mapping, and for performing the state estimation. This estimation will ensure that the pose estimate given from the system is the most likely pose given the observations.

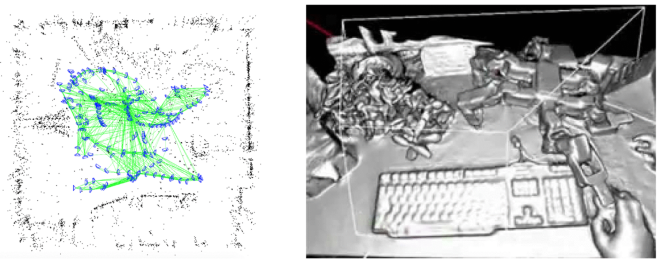
Loop closure is what really separates SLAM from odometry. This is the ability for a system to recognize a place it has been to before, and thereby use the information previously obtained from that location. In feature-based SLAM, this is performed when the same feature is seen at two different occasions, greatly reducing the uncertainty at the second sighting. This is often seen as one of the hardest problems in SLAM, as false positives for full SLAM will degrade the quality of all the previous poses as well as the current one. In order to perform consistent loop closures for feature-based SLAM systems, descriptors are often used, in addition to some of the data association techniques that will be introduced in Section 2.8. In addition to improving the state estimates, loop closure also gives the robot a better understanding of the environment, as odometry sees the world as one long corridor. Implementing loop closure could therefore give a spatial understanding of how to move between points as seen in Figure 2.4.

Loop closures are often performed in one of three different ways [57], map-to-map, image-to-image, or image-to-map. The map-to-map method looks for keypoints in two sub-maps by looking at both the appearance and their relative positions. Image-to-image will look at the latest image and compare it to all previously stored images. This method uses the occurrences of image features to detect whether the two images are from the same part of the world. The third method is image-to-map, and it will compare the latest image to all the features in the map.

A major difficulty when performing loop closures is perceptual aliasing [58]. Perceptual aliasing is that similarly looking scenes at different locations deceive the place recognition. This is a big problem as no current SLAM systems are fail-safe or failure-aware.

SLAM is not only about pose estimation, but as the name suggests, also about building a map of the surroundings. This map can be used for collision avoidance, path planning or to provide visualization for human operators. There are several different mapping methods, with the most popular being topological and metric mapping. Topological maps lack metric information, but it does contain information up to scale, meaning that the scaling

Figure 2.5 Left: feature-based map of a room produced by ORB-SLAM [7]. Right: dense map of a desktop produced by DTAM [8]. Image and caption taken from [6].



between the points are correct. Metric maps on the other hand uses metric information to make sure that the spatial distances are correct. There are several different types of metric maps, for example feature-based metric maps and dense metric maps. Feature-based metric maps uses the keypoints extracted in order to create a map of the surroundings, and is used by for example ORB-SLAM [7]. Dense maps contain more detail, and is used by for example Dense Tracking and Mapping (DTAM) [8]. Examples of metric maps created using these methods can be seen in Figure 2.5. Topological maps has also been used for SLAM systems, and one of the first to utilize it was Fast Appearance-Based Mapping (FAB-MAP) [59].

SLAM has not only been used for a single agent moving around, but it has also been implemented on several agents in order to cooperate and create a map together [60]. This can be very useful for fast mapping of unknown locations, and can therefore be used for rescue operations, fire fighting or even cleaning operations. There are usually two different ways multi-agent SLAM is performed, it can either be centralized or decentralized. For centralized multi-agent SLAM, all the information created by the robots are sent to a central station which performs the computations and optimization [61], [62]. Decentralized multi-agent SLAM does not have a centralized computer, and all the agents needs to communicate to reach a consensus on a common map [63].

A disadvantage of SLAM is that it does not work as well in dynamic environments as they do in static environments. Especially feature-based SLAM relies on the keypoints to be chosen at static locations, so that if they are seen at a later point in time, they will be at the same location. A static world assumption is therefore often made when creating SLAM systems, if this assumption is not made, the system has to detect, discard or track changes [64], [65], [66], [67].

2.5 Factor graph SLAM libraries

There are several factor graphs SLAM libraries online that solves nonlinear optimization problems using the factor graph framework. Many of these libraries are open-source, and most are written in C++ due to the speed and efficiency of C++ code. Some well-known libraries that will be presented here are Georgia Tech Smoothing and Mapping (GTSAM), g2o and SLAM++. Some other libraries that will not be presented in more detail here include, SPA [68], HOG-Man [69] and Ceres [70].

2.5.1 Georgia Tech Smoothing and Mapping

The GTSAM toolbox [71] is an open-source C++ library based on factor graphs, developed at the Georgia Institute of Technology. It implements smoothing and mapping in robotics and vision, using factor graphs and Bayes networks as the underlying computing paradigm rather than sparse matrices. Doing this, it provides state-of-the-art solutions for SLAM problems, such as iSAM [72] and iSAM2 [5]. In order to be computationally efficient, GTSAM exploits the sparsity of factor graphs. Several state-of-the-art SLAM systems utilize GTSAM for its back-end, such as LeGO-LOAM [23].

2.5.2 g2o

The g2o library [27] is an open-source C++ library for optimizing graph-based nonlinear error functions. It has been designed in order to be easily extensible to a wide range of problems, and making it easy to specify new problems quickly. Solutions to several variants of SLAM are provided within this framework. Similarly to GTSAM, g2o also utilizes factor graphs, making it efficient due to the sparsity of the factor graph problem. Several state-of-the-art SLAM systems utilize g2o for its back-end, such as ORB-SLAM [7] and LSD-SLAM [73].

2.5.3 SLAM++

SLAM++ [74] is another open-source C++ library based on factor graphs. SLAM++ is also done incrementally, but unlike GTSAM and g2o, it performs all the matrix operations by blocks. This leads to fast matrix manipulation and arithmetic operations. It is applicable to solve several SLAM problems, while also being efficient and easy to implement.

2.6 Nonlinear solvers

When using factor graphs, the state estimation can be reduced to a nonlinear least square problem. These problems cannot be solved directly, and it is hard to determine if there even exist a solution. Heuristic algorithms are therefore used, as they often work well in practice and give good estimates, especially if the initial estimate is good [56] [9]. The GTSAM library, which is introduced in Section 2.5.1 and will be used in this thesis, provides three different nonlinear solvers, Gauss-Newton, Levenberg-Marquardt [75] and Powell's Dogleg [76].

2.6.1 Gauss-Newton

The Gauss-Newton algorithm is an iterative algorithm to solve nonlinear least squares problems. It is based on a linear approximation to the components of the objective function $f(\mathcal{X})$. For a small iteration τ^G, the Taylor expansion of $f(\mathcal{X})$ can be written as:

$$f(\mathcal{X} + \tau) \approx l(\tau^G) = b(\mathcal{X}) + A(\mathcal{X})\tau^G. \tag{2.7}$$

Inserting this into the nonlinear least squares problem introduced in Section 2.2.1 and assuming no correlation, one can deduce the following equation:

$$\begin{aligned} F(\mathcal{X} + \tau^G) \approx L(\tau^G) &= \frac{1}{2} l(\tau^G)^T l(\tau^G) \\ &= \frac{1}{2} b^T b + \tau^{G^T} A^T b + \frac{1}{2} \tau^{G^T} A^T A \tau^G \\ &= F(\mathcal{X}) + \tau^{G^T} A^T b + \frac{1}{2} \tau^{G^T} A^T A \tau^G. \end{aligned} \tag{2.8}$$

From this, it can easily be seen that the gradient of $L(\tau)$ is $L'(\tau) = A^T b + A^T A\tau$ and the Hessian is $L''(\tau) = A^T A$. From this, Gauss-Newton finds the extrema by setting the gradient equal to 0, and the following problem needs to be solved:

$$(A^T A)\tau^G = A^T b. \tag{2.9}$$

This algorithm will be able to obtain almost quadratic convergence to a minimum, given that the linearization happens near the solution, and thereby having a good initial condition. If this linearization is poor, the convergence might be much slower, and even diverge. This algorithm does however require convexity, which is often violated in practical application, as the Hessian, $A^T A$, may fail to be invertible. If the problem is not convex, it is possible to create inexact Gauss-Newton algorithms that replace the Hessian with an approximation that is invertible [77]. The full Gauss-Newton algorithm can be seen in algorithm 1.

Algorithm 1 Gauss-Newton Algorithm

Data: An objective function $f(\mathcal{X})$ and a good initial state estimate $\hat{\mathcal{X}}^0$
Result: An estimate for the states $\hat{\mathcal{X}}$
for t = 0, 1, ..., t^{max} **do**
  A, b $\leftarrow$ Linearize $f(\mathcal{X})$ at $\hat{\mathcal{X}}^t$
  $\tau^G \leftarrow$ Solve the linearized problem $A^T A \tau^G = A^T b$
  $\hat{\mathcal{X}}^{t+1} \leftarrow \hat{\mathcal{X}} \oplus \tau^G$
  if $f(\hat{\mathcal{X}}^{t+1})$ is very small or $\hat{\mathcal{X}}^{t+1} \approx \hat{\mathcal{X}}^t$ **then**
    $\hat{\mathcal{X}} \leftarrow \hat{\mathcal{X}}^{t+1}$
    return
  end
end

2.6.2 Levenberg-Marquardt

Levenberg-Marquardt [75] utilizes trust regions in order to improve the convergence of the Gauss-Newton algorithm. The step τ^L will be defined similarly to Gauss-Newton in Equation 2.9 with a variable step size, λ,:

$$(A^T A + \lambda I)\tau^L = A^T b. \tag{2.10}$$

This variable step size has several effects. First of all, it allows the coefficient matrix $(A^T A + \lambda I)$ to be positive definite, ensuring that the next step provides an improvement. If the step size is very large, it will be a step towards the steepest descent direction, as the current approximation is far from the solution:

$$\tau^L \approx A^T b/\lambda. \tag{2.11}$$

If the step size it small, the approximation is close to the solution, and Levenberg-Marquardt will be very close to the Gauss-Newton algorithm, making it possible to get close to a quadratic convergence near the solution.

As the variable step size allows the coefficient matrix to be positive definite, unimodality is required instead of convexity, in order to find the global extrema. This variable step size makes the Levenberg-Marquardt guaranteed to converge, unlike the Gauss-Newton algorithm. The full Levenberg-Marquardt algorithm can be seen in Algorithm 2.

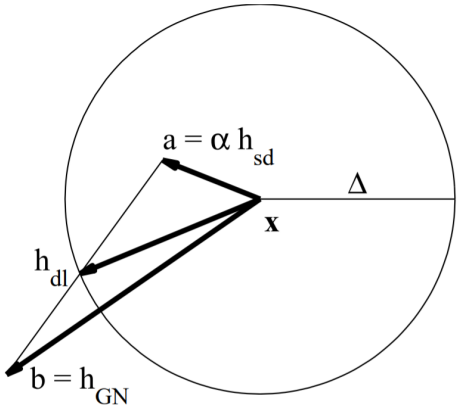
Algorithm 2 Levenberg-Marquardt Algorithm

Data: An objective function $f(\mathcal{X})$ and a good initial state estimate $\hat{\mathcal{X}}^0$
Result: An estimate for the states $\hat{\mathcal{X}}$
Initialize λ
for t = 0, 1, ..., t^{max} **do**
  A, b $\leftarrow$ Linearize $f(\mathcal{X})$ at $\hat{\mathcal{X}}^t$
  $\tau^L \leftarrow$ Solve the linearized problem $(A^T A + \lambda * diag(A^T A))\tau^L = A^T b$
  if $f(\hat{\mathcal{X}} \oplus \tau^L) < f(\hat{\mathcal{X}})$ **then**
    Accept update, increase trust region
    $\hat{\mathcal{X}}^{t+1} \leftarrow \hat{\mathcal{X}} \oplus \tau^L$
    Decrease λ
  else
    Reject update, reduce trust region
    $\hat{\mathcal{X}}^{t+1} \leftarrow \hat{\mathcal{X}}$
    Increase λ
  end
  if $f(\hat{\mathcal{X}}^{t+1})$ is very small or $\hat{\mathcal{X}}^{t+1} \approx \hat{\mathcal{X}}^t$ **then**
    $\hat{\mathcal{X}} \leftarrow \hat{\mathcal{X}}^{t+1}$
    return
  end
end

2.6.3 Powell's Dogleg

The third method available in the GTSAM library is Powell's Dogleg method [76]. Powell-s Dogleg algorithm is a mixture between the Gauss-Newton algorithm and the steepest descent algorithm, resulting in a step of τ^D. Similarly to Levenberg-Marquardt, it uses trust regions to make sure that the algorithm does not diverge, and due to this it does not require convexity as the Gauss-Newton algorithm does.

Figure 2.6 This image shows the steps taken by Powell's dogleg, compared to the steepest descent and Gauss-Newton. h_{sd} corresponds to τ^S, h_{dl} corresponds to τ^D and h_{GN} corresponds to τ^G. Image taken from [9].



The Gauss-Newton step is calculated by Equation (2.9), and the steepest descent step is given by:

$$\tau^S = A^T b. \tag{2.12}$$

The step direction is then chosen to first go towards the Steepest Descent update, followed by a sharp bend towards the Gauss-Newton update, stopping at the trust region bounds, Δ. The trust region bound will be changed using ρ, which is calculated using Equation (2.13), as a large ρ indicates that the linear model is a good approximation, resulting in a larger Δ. Powell's Dogleg algorithm can be more efficient than Levenberg-Marquardt [78], as it does not need to refactor the information matrix every time an update is rejected. The full Powell's Dogleg algorithm can be seen in Algorithm 3.

$$\rho = \frac{g(X^t) - g(X^t + \Delta)}{L(0) - L(\Delta)} \tag{2.13}$$

2.7 Relevant sensors

Sensor fusion is about combining data from different sensors in order utilize all the available information. Doing this will result in information that is more reliable than what would be possible if the sensors were used individually [79]. Some commonly used sensors for sensor fusion for autonomous vehicles are camera, GNSS, IMU, lidar and radar [80], [81], [82].

Algorithm 3 Powell's Dogleg Algorithm

Data: An objective function $f(\mathcal{X})$ and a good initial state estimate $\hat{\mathcal{X}}^0$
Result: An estimate for the states $\hat{\mathcal{X}}$
Initialize λ
Initialize Δ
for t = 0, 1, ..., t^{max} **do**
  A, b $\leftarrow$ Linearize $f(\mathcal{X})$ at $\hat{\mathcal{X}}^t$
  $\tau^S \leftarrow$ Solve the linearized problem $(\lambda * diag(A^T A))\tau^S = A^T b$
  $\tau^G \leftarrow$ Solve the linearized problem $(A^T A)\tau^G = A^T b$
  if $\left\|\tau^G\right\| \leq \Delta$ **then**
    $\tau^D \leftarrow \tau^G$
  else if $\left\|\tau^S\right\| \geq \Delta$ **then**
    $\tau^D \leftarrow \frac{\Delta}{\|\tau^S\|}\tau^S$
  else
    $\tau^D \leftarrow \tau^S + \lambda(\tau^G - \tau^S)$
  end
  Calculate ρ using Equation (2.13)
  if $\rho > 0$ **then**
    $\hat{\mathcal{X}}^{t+1} \leftarrow \hat{\mathcal{X}}^t \oplus \tau^D$
  end
  if $\rho < 1/4$ **then**
    Reduce Δ
  else if $\rho > 3/4$ **then**
    Increase Δ
  end
  if $f(\hat{\mathcal{X}}^{t+1})$ is very small or $\hat{\mathcal{X}}^{t+1} \approx \hat{\mathcal{X}}^t$ **then**
    $\hat{\mathcal{X}} \leftarrow \hat{\mathcal{X}}^{t+1}$
    return
  end
end

2.7.1 Intertial Measurement Uni

An IMU is an electronic device that measures and reports a body's acceleration, angular rate, and sometimes the orientation of the body, using a combination of accelerometers, gyroscopes, and sometimes magnetometers. The price of an IMU has dropped significantly due to Microelectromechanical Systems (MEMS) technology. MEMS makes it possible to have an IMU on a small electrical chip. These low cost sensors are not very accurate, and can have a large drift, meaning an expensive IMU is needed for accurate measurements. However, due to sensor fusion, a high-cost IMU might not be needed, and even a low-cost will be able to provide useful information.

2.7.2 Blobal Navigation Satellite System

GNSS is satellite navigation systems that provide autonomous geo-spatial positioning with global coverage. There are several GNSS systems used across the world. Global Positioning System (GPS) was created by United States, Global Navigation Satellite System (GLONASS) by Russia, Galileo by the EU, BeiDou by China and there also exists several regional systems. GNSS provides global coverage and a regular receiver is low-cost and has a specified Root Square Error (RSE) of about 4 meters. It is possible to get more accurate measures utilizing Real Time Kinematics (RTK)-GNSS, however they are significantly more expensive. RTK-GNSS uses measurements of the phase of the signal's carrier wave in addition to the information content of the signal and relies on a single reference station to provide real-time corrections. This provides accuracy down to a few centimeters.

2.7.3 Camera

Cameras were one of the first sensors used in autonomous vehicles and is still one of the most used sensors. Using cameras helps the vehicles and the operator to visualize the surroundings, and it will therefore make it easier for the operator to understand the decisions made by the vehicles. Cameras are widely available, and more affordable than radar and lidar. However, the camera require high computational power to process all the data, as it can provide millions of pixels per frame, and 30 to 60 frames each seconds, leading to multi-megabytes of data needed to be processed in real-time.

2.7.4 Radar

Radar is short for Radio Detection and Ranging and is often integrated into maritime vehicles to detect other ships and land obstacles. It can also be integrated into ground vehicles, however these are usually much smaller and provides more primitive information. Radar does not only measure distance and bearing, but has the ability to measure velocity directly using the Doppler effect. A marine radar can have a range of close to 50 nautical miles, and could therefore be used at a much greater range than the lidar and camera. It is also more efficient in bad weather than the lidars or cameras, but it has less angular accuracy and generates less data. However, since less data is generated, it requires less computational power to process it.

2.7.5 Lidar

Lidar is short for Light Detection and Ranging and it uses an infrared laser beam, usually in the 900 nm wavelength range, to determine the distance between the sensor and a nearby object. In most lidars, a rotating swivel scans the laser beam across the field of view. The lasers are pulsed and the pulses are reflected by objects. These reflections return a point cloud that represents these objects. Lidar has a much higher spatial resolution than radar because of the more focused laser beam, the larger number of scan layers in a vertical direction, and the high density of lidar points per layer. Lidars are not able to directly measure velocity, and will therefore need to rely on the different positions between two scans to estimate it. Lidars are also more affected by weather conditions and by dirt on the sensor than the radar is. One of the limits with the lidar in maritime environment is that a lot of the information is lost due to the fact that the lidar does not reflect water as well as solid ground surfaces. They are, in addition, much more expensive than cameras, with prices ranging from 50 KNOK for a Velodyne VLP-16 lidar, to close to 1 MNOK for a Neptec high-grade lidar.

2.8 Data association

One of the major challenges when performing SLAM is data association. Data association is the matching of keypoints between point clouds, and also over time in order to perform loop closures. Some state-of-the-art methods to perform data association will be presented here. Poor data association might cause the trajectory to diverge, as the loop closures will be incorrect, and will greatly reduce the accuracy of the short term tracking as well.

2.8.1 Joint Compatibility Branch and Bound

Joint Compatibility Branch and Bound (JCBB) [29] performs data association using joint compatibility of the keypoints. In order to do this, the algorithm reconsiders all the established matches at every step in order to reduce the amount of false positives. The goal is also to get as many matches as possible, as the probability of new false positives decreases with the number of matches. Reconsidering all matches at every step has several advantages over sequential compatibility. Firstly, it can be tested without recomputing the state, which is an $O(n^2)$ operation, where n is the total number of keypoints. Secondly, the inversion of a growing covariance matrix is avoided. Lastly, most of the elements involved in the calculation have been previously computed to assert individual compatibility.

The JCBB algorithm uses an interpretation tree [83], which is a search tree where the nodes represent all possible joint correspondences. This tree is traversed in search of the hypothesis with the largest number of nonnull jointly compatible matches. As this is monotonically nondecreasing, it can be used to bound the search in the interpretation tree. The quality of a node at a certain level can be defined as the number of nonnull matches that can be established from the node. The nodes with a lower quality than the best available hypothesis are therefore not explored. Nearest neighbor rule using the Mahalonobis distance can be used as heuristic for branching, in order to explore hypothesis with a higher degree of joint compatibility first.

2.8.2 Hungarian Algorithm

The Hungarian algorithm [31] is another data association algorithm. It consists of seven steps as seen in Algorithm 4, where D is the cost matrix. The Hungarian algorithm is a popular algorithm, even though it is not the most efficient one, as the run-time is $O(n^3)$.

Algorithm 4 Hungarian Algorithm

1) **Row preprocessing:** From each row of D, find the row's minimum, and subtract it from all elements in that row.
2) **Column preprocessing:** From each column of D, find the column's minimum, and subtract it from all elements in that column
3) **Covering zeros:** Cover all zeros in D by crossing out the minimum number of rows and columns in D
4) If all rows of D are crossed out, we are done, and go to step 7
5) **Adjust D:**
   a) Find the minimum in D that is not crossed out
   b) Subtract the minimum from the elements that are not crossed-out in D
   c) Add the minimum to elements that are crossed out twice in D
6) Return to step 3 with the adjusted matrix D
7) Solutions are zero elements of D. Go first for the zero element which is unique in its row and column. Then, delete that row and column from D.

2.8.3 Auction Algorithm

The Auction algorithm [30] is yet another algorithm for data association. It has gotten its name as it is based on the same principles as an auction is. Like in the Hungarian algorithm, a cost matrix is used in order to perform the data association. The algorithm splits the features into customers and items, for the new measurements, and for the features in the map. The auction is built from two cases. The first case is if the customer is not matched yet. Then the customer will set a price for a specific item, like in an auction. This completed the (k+1)st iteration of the algorithm, so that all the customers that needs a match have received a price and a reward. The second case is the bidding part of the algorithm, where all the customers try to bid for the items in order for all to maximize their profits while not buying the same item. This can create bidding wars that are inefficient as the number of customers and items increases. It was shown in the original article that while using 100 items, 75$\%$ of the iterations were used to assign the last two or three items. This has however been improved in later research by adding in a reverse auction as well [84]. The Auction algorithm has the same worst-case run-time as the Hungarian algorithm of $O(n^3)$, but has been shown to have a better average case.

2.8.4 FastSLAM

Factored Solution To SLAM (FastSLAM) [32] [85] is a state-of-the-art SLAM method, and it is based on particle filtering. Using particle filtering for SLAM was the first successful attempt at solving the full SLAM problem, and by using the Rao-Blackwellized particle filter [86], FastSLAM manages to scale logarithmically with the number of keypoints in the map. The state-of-the-art data association created by FastSLAM is known as per-particle data association.

Operating with a per-particle data association allows FastSLAM to pursue multiple data association hypothesis at the same time, as the posterior is represented by multiple particles. This allows the data association to work in the presence of only a few good features, where other data association techniques might have fatal errors. There are several ways to sample over the possible data associations.

The first is to do a per-particle maximum likelihood data association, which is a deterministic approach. This principle is adopted from the maximum likelihood assignment procedure used by the EKF, but on a per-particle basis. Particles that have the correct data association will receive high probabilities, while the others will receive low probabilities and will consequently be removed in future resampling steps. Using per-particle data association has two clear advantages. The first is that the accuracy of the data association will not be affected by the robot motion noise, as long as an appropriate number for particles are chosen. This will significantly improve the results in situations where there is a substantial amount of noise. The second advantage it provides is delayed decision-making, as it will at any point receive plausible, but wrong data associations. This means that future observations can prove the earlier ones wrong, and thereby give the observations low probability and consequentially remove them from the filter.

Another way to sample is using Monte-Carlo data association, which is a stochastic approach. The Monte-Carlo data association will provide each particle with a randomly associated weight, instead of assigning each particle with the most likely data association. In the events of measurement ambiguity, this will have a small positive effect on estimation accuracy. However, if the measurement error is high, it will induce a high number of plausible data associations, which for the same scenario with known data association would require exponentially more particles to achieve.

In order to further improve the sampling, mutual exclusion could be included. Mutual exclusion is used to make sure that each keypoint is only assigned to one measurement. In FastSLAM, this can be achieved in a greedy fashion, meaning that each observation is associated with the most likely keypoint in each particle that has not received an observation yet. This calls for the need of more particles, as the mutual exclusion will sometime apply the constrains incorrectly. This will however make the process of adding new keypoints a much simpler problem, as it will create new keypoints instead of incorrectly assigning it to a previous observation.

2.8.5 Random Sample Consensus

Random Sample Consensus (RANSAC) [33] is a popular algorithm within the computer vision community, and it is commonly used to find inliers among correspondences. RANSAC creates its candidate solutions based on a small amount of data, and tries to find the amount

of inliers within each hypothesis. Then it repeats the process as seen in Algorithm 5.

Algorithm 5 Random Sample Consensus algorithm

1) Select randomly a Minimum Sample Set (MSS) with just enough correspondences required to determine the model parameters.
2) Solve for the parameters of the model.
3) Determine how many correspondences from the set of all correspondences fit with a predefined tolerance ϵ.
4) If the fraction of the number of inliers over the total number correspondences in the set exceeds a predefined threshold τ, re-estimate the model parameters using all the identified inliers and terminate.
5) Otherwise, repeat steps 1 through 4 (maximum of N times).

2.8.6 Deep Learning

Recently, there has been several research papers using deep learning to perform data association [34], [35], [36]. These methods all use deep neural networks, but they use different networks in order to perform the data association. The first paper uses a Convolutional Neural Network (CNN) [87] in a tracking-by-detection framework. The second paper uses a Recurrent Neural Network (RNN) [88] to directly map raw sensor input to object tracks in sensor space without any form of feature engineering or system identification. The third mentioned paper uses a bi-directional Long Short-Term Memory (LSTM) [89] network with a projection layer. This shows that there are several different ways to create a deep neural network that can perform state-of-the-art data association, and some of these might be available for SLAM systems in the future.

2.9 Transformation matrices

A homogeneous transformation matrix is a 4×4 matrix, which includes both a rotation matrix, $\mathbf{R}$, and a translation in three directions, $\mathbf{t}$. The transformation matrix is in the special Euclidean group in 3D, $SE(3)$, and can be seen in the following equation:

$$SE(3) = \left\{ \mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \in \mathbb{R}^{4x4} | \mathbf{R} \in SO(3), \mathbf{t} \in \mathbb{R}^3 \right\}. \tag{2.14}$$

The rotation matrix is used to present the roll, pitch and yaw of a vehicle, and it is a part of the special orthogonal group in 3D, $SO(3)$, which can be seen in the following equation:

$$\mathbf{R} \in SO(3), SO(3) = \{\mathbf{R} \in \mathbb{R}^{3x3} | \mathbf{R}\mathbf{R}^T = \mathbf{I}, \det(\mathbf{R}) = 1\}. \tag{2.15}$$

In order to calculate the transformation over time, the different transformation matrices are concatenated. This will track both the rotation and translation from the beginning to

the current time. This multiplication is performed as seen in the following equation, where $\mathbf{T}_a$ is the transformation from the beginning to the previous point cloud, and $\mathbf{T}_b$ is the transformation from the previous point cloud to the current point cloud:

$$\mathbf{T}_a\mathbf{T}_b = \begin{bmatrix} \mathbf{R}_a\mathbf{R}_b & \mathbf{R}_a\mathbf{t}_b + \mathbf{t}_a \\ \mathbf{0}^T & 1 \end{bmatrix}. \tag{2.16}$$

2.10 Lie theory

Orientations and poses lie on manifolds in higher-dimensional spaces [90]. This makes it complicated to add increments, represent uncertainty and perform differentiation, as shown in Equations (2.17) and (2.18), where $\delta\mathbf{R}$ and $\delta\mathbf{T}$ represent perturbations. However, these operations can be performed using Lie algebra as both orientations and poses are matrix Lie groups. A matrix Lie group is a group on a smooth manifold and Lie theory describes the tangent space around elements of a Lie group, in order to define the exact mappings between the tangent space and the manifold. The tangent space has the same dimension as the number of degrees of freedom of the group transformations.

$$\begin{aligned} \mathbf{R} &\in SO(3) \\ \delta\mathbf{R} &\in \mathbb{R}^{3x3} \\ \mathbf{R} + \delta\mathbf{R} &\notin SO(3) \end{aligned} \tag{2.17}$$

$$\begin{aligned} \mathbf{T} &\in SE(3) \\ \delta\mathbf{T} &\in \mathbb{R}^{4x4} \\ \mathbf{T} + \delta\mathbf{T} &\notin SE(3) \end{aligned} \tag{2.18}$$

In order to be a Lie group, certain criteria needs to be satisfied. The group $(\mathcal{G}, \circ)$ with set $\mathcal{G}$ and composition operation $\circ$ needs to satisfy the following axioms:

$$\begin{aligned} \text{Closure under } \circ &: \mathcal{X} \circ \mathcal{Y} \in \mathcal{G} \\ \text{Identity } \mathcal{E} &: \mathcal{E} \circ \mathcal{X} = \mathcal{X} \circ \mathcal{E} = \mathcal{X} \\ \text{Inverse } \mathcal{X}^{-1} &: \mathcal{X}^{-1} \circ \mathcal{X} = \mathcal{X} \circ \mathcal{X}^{-1} = \mathcal{E} \\ \text{Associativity } &: (\mathcal{X} \circ \mathcal{Y}) \circ \mathcal{Z} = \mathcal{X} \circ (\mathcal{Y} \circ \mathcal{Z}) \\ \text{Action of } \mathcal{X} \in \mathcal{M} \text{ on } v \in \mathcal{V} &: \mathcal{X} \cdot v \end{aligned} \tag{2.19}$$

Both $SO(3)$ and $SE(3)$ fulfill the criteria axioms presented above. For $SO(3)$, the inversion is achieved with transposition $\mathrm{R}^{-1} = \mathrm{R}^T$, composition with matrix multiplication $\mathrm{R}_a \circ \mathrm{R}_b = \mathrm{R}_a\mathrm{R}_b$ and the group action on vectors is given by the product R · x = Rx.

The tangent space at the identity $\mathfrak{m} = \mathcal{T}\mathcal{M}_\mathcal{E}$ is called the Lie algebra of $\mathcal{M}$. The Lie algebra is a vector space with elements $\tau^\wedge \in \mathfrak{m}$, where the hat operator, $(\cdot)^\wedge$, is used to map a matrix as a vector. The vee operator, $(\cdot)^\vee$, is the inverse operation, and can be used

to map a vector to a matrix. In order to transform elements of the Lie algebra to elements of the group, an exponential map is used:

$$\exp : \mathfrak{m} \to \mathcal{M}; \quad \mathcal{X} = \exp(\tau^{\wedge}) \tag{2.20}$$

Transforming the elements of the group to elements of the Lie algebra is then the inverse operation, and the logarithmic map is used:

$$\log : \mathcal{M} \to \mathfrak{m}; \quad \tau^{\wedge} = \log(\mathcal{X}) \tag{2.21}$$

For notational convenience, a vectorized version of the exponential and logarithmic maps are adopted by capitalization:

$$\begin{aligned} \text{Exp} : \mathbb{R}^m \to \mathcal{M}; \mathcal{X} = \text{Exp}(\tau) = \exp(\tau^{\wedge}) \\ \text{Log} : \mathcal{M} \to \mathbb{R}^m; \tau = \text{Log}(\mathcal{X}) = \log(\mathcal{X}^{\vee}) \end{aligned} \tag{2.22}$$

Perturbations can then be performed on the manifold expressed as tangent space vectors by combining one Exp/Log operation with one composition, and then transform it back to the group.

2.11 Preintegration

Preintegration was first introduced in [91] and it consists of combining measurements from the IMU between two keyframes into one relative motion constraint. This work was taken further by [14], where the preintegration theory was extended to the $SO(3)$ rotation group and framed into a factor graph model. This is what is currently implemented in GTSAM.

An IMU usually includes both a 3-axis accelerometer and a 3-axis gyroscope, resulting in measurements of the rotation rate, ω, as seen in Equation (2.23), and the acceleration, a, as seen in Equation (2.24). The B subscript denotes that it is in expressed in the body frame, while the W subscript denotes that it is expressed in the world frame. The following WB subscript describes a transformation from the world frame to the body frame, R^T_{WB}, or it describes the angular velocity of the body frame relative to the world frame, ${}_W\omega_{WB}$. The b is the bias of the measurement, while η is the noise of the measurement, and the superscripts denoted if the measurement is from the accelerometer, a, or the gyroscope, g.

$${}_B\tilde{\omega}_{WB}(t) = {}_B\omega_{WB}(t) + b^g(t) + \eta^g(t) \tag{2.23}$$

$${}_B\tilde{a}(t) = R^T_{WB}({}_Wa(t) - {}_Wg) + b^a(t) + \eta^a(t) \tag{2.24}$$

The goal of preintegration is to be able to calculate the motion from the IMU measurements. The following kinematic model is therefore introduced:

$$\begin{aligned} \dot{R}_{WB} &= R_{WB} \cdot_B \omega^{\wedge}_{WB}, \\ {}_W\dot{v} &= {}_Wa, \\ {}_W\dot{p} &= {}_Wv. \end{aligned} \tag{2.25}$$

In order to obtain the state at time $t + \Delta t$, Equation (2.25) is integrated:

$$\begin{aligned} R_{WB}(t+\Delta t) &= R_{WB}(t)\text{Exp}\left(\int_t^{t+\Delta t} {}_B\omega_{WB}(\tau)d\tau\right), \\ {}_W v(t+\Delta t) &= {}_W v(t) + \int_t^{t+\Delta t} {}_W a(\tau)d\tau, \\ {}_W p(t+\Delta t) &= {}_W p(t) + \int_t^{t+\Delta t} {}_W v(\tau)d\tau + \int\int_t^{t+\Delta t} {}_W a(\tau)d\tau^2. \end{aligned} \tag{2.26}$$

If one assumes that the accelerometer and gyroscope measurements are constant in the time interval $[t, t + \Delta t]$, the states can be written as a function of the measurements:

$$\begin{aligned} R(t+\Delta t) &= R(t)\text{Exp}\left(\left(\tilde{\omega}(t) - b^g(t) - \eta^{gd}(t)\right)\Delta t\right), \\ v(t+\Delta t) &= v(t) + g\Delta t + R(t)\left(\tilde{a}(t) - b^a(t) - \eta^{ad}(t)\right)\Delta t, \\ p(t+\Delta t) &= p(t) + v(t)\Delta t + \frac{1}{2}g\Delta t^2 + \frac{1}{2}R(t)\left(\tilde{a}(t) - b^a(t) - \eta^{ad}(t)\right)\Delta t^2. \end{aligned} \tag{2.27}$$

In Equation (2.27) the subscripts are dropped for readability. Now, the system has to be extended so that it is possible to combine all IMU measurements between two keyframes into a single factor node, which is how preintegrated IMU measurements are worked into a factor graph. All Δt intervals between the two consecutive keyframes $k = i$ and $k = j$ are concatenated, which results in:

$$\begin{aligned} R_j &= R_i \prod_{k=i}^{j-1} \text{Exp}\left(\left(\tilde{\omega}_k - b_k^g - \eta_k^{gd}\right)\Delta t\right), \\ v_j &= v_i + g\Delta t_{ij} + \sum_{k=i}^{j-1} R_k(\tilde{a}_k - b_k^a - \eta_k^{ad})\Delta t, \\ p_j &= p_i + \sum_{k=i}^{j-1}\left[v_k\Delta t + \frac{1}{2}g\Delta t^2 + \frac{1}{2}R_k\left(\tilde{a}_k - b_k^a - \eta_k^{ad}\right)\Delta t^2\right]. \end{aligned} \tag{2.28}$$

In this equation $\Delta t_{ij} = \sum_{k=i}^{j-1} \Delta t$ and $(\cdot)_i = (\cdot)(t_i)$. This will provide an estimate of the motion between t_i and t_j, but it has to be repeated every time the linearization point changes. In order to avoid this recomputation, the following relative motion increments are defined:

$$
\begin{aligned}
\Delta R_{ij} &= R_i^T R_j = \prod_{k=i}^{j-1} \text{Exp}\left(\left(\tilde{\omega}_k - b_k^g - \eta_k^{gd}\right)\Delta\right), \\
\Delta v_{ij} &= R_i^T (v_j - v_i - g\Delta t_{ij}) = \sum_{k=i}^{j-1} \Delta R_{ik}(\tilde{a}_k - b_k^a - \eta_k^{ad})\Delta t, \\
\Delta p_{ij} &= R_i^T \left(p_j - p_i - v_i \Delta t_{ij} - \frac{1}{2}\sum_{k=i}^{j-1} g\Delta t^2\right), \\
&= \sum_{k=i}^{j-1} \left[\Delta v_{ik}\Delta t + \frac{1}{2}\Delta R_{ik}\left(\tilde{a}_k - b_k^a - \eta_k^{ad}\right)\Delta t^2\right].
\end{aligned}
\tag{2.29}
$$

Here $\Delta R_{ik} = R_i^T R_k$ and $\Delta v_{ik} = R_i^T(v_k - v_i - g)\Delta t_{ik}$. With these equations, it is possible to compute the results directly from the measurements received from the IMU. These equations require knowledge of the bias, however, this is assumed to remain constant between two keyframes:

$$
b_i^g = b_{i+1}^g = ... = b_{j-1}^g, \quad b_i^a = b_{i+1}^a = ... = b_{j-1}^a \tag{2.30}
$$

More details on how Lie algebra is integrated in the preintegrated IMU measurements for the rotational group SO(3) can be found in [14].

Chapter 3

State-of-the-art SLAM

3.1 Lidar SLAM systems

In order to create a feature-based lidar SLAM system, it is essential to see what other methods are out there and how they have solved the problem. In this section, Hector-SLAM, LOAM, LeGO-LOAM and Google Cartographer will be discussed in some more detail.

3.1.1 Hector-SLAM

Hector-SLAM [21] is one of the earlier SLAM methods created for lidars, and is available open-source in ROS. Hector-SLAM consumes low computational resources and can therefore be used on low-weight, low-power and low-cost processors, which are what is commonly found on autonomous vehicles.

The approach taken in Hector-SLAM is a combination of a 2D laser odometry system and an integrated 3D navigation system based on an IMU. As the system integrates in 3D information, it is able to estimate the full 6 Degrees of Freedom (DOF) state, consisting of both rotation and translation.

Hector-SLAM uses an occupancy grid to represent the real world. As the lidar platform can exhibit 6DOF motion, the scans need to be transformed into a local stabilized coordinate frame. After this coordinate frame is extracted, the scan is converted into a point cloud of scan endpoints and pre-processed, both to remove outliers and to downsample the number of points. The endpoint z-coordinate is then used to filter, so that only the endpoints that are within a certain threshold are used in the scan matching process.

As occupancy grids are discrete by nature, they will limit the precision and will also not allow direct computation of derivatives or interpolated values. An interpolation scheme allowing sub-cell accuracy through bilinear filtering is employed for both estimating occupancy probabilities and derivatives. The grid map cell values can therefore be viewed as samples of an underlying continuous probability distribution.

Scan-matching is then used for the motion estimation. The approach utilized in this method is based on the optimization of the alignment of beam endpoints with the map. The basic idea is inspired by other methods in computer vision and uses a Gauss-Newton approach. This approach makes it so that there is no need for a data association search between beam endpoints, or an exhaustive pose search, because scans are aligned with the existing map. The goal of the scan-matching algorithm is to find the rigid transformation, ξ, that minimizes

$$\xi^* = \min_{\xi} \sum_{i=1}^{n} \left[1 - M(S_i(\xi))\right]^2 . \tag{3.1}$$

Here, S_i are the world coordinates of the scan endpoints, while $M(S_i)$ is a function that returns the map value at the coordinates given by S_i. As the starting ξ is given, the goal is to estimate $\Delta\xi$, which will be the change in the transformation, by minimizing the error measure according to

$$\sum_{i=1}^{n} [1 - M(S_i(\xi + \Delta\xi))]^2 \to 0. \tag{3.2}$$

In order for the scan-matching algorithm to not get stuck in local minima, Hector-SLAM uses a multi-resolution map representation. This is similar to the image pyramid approach that is used in computer vision. An example of where the image pyramid is used is for Scale Invariant Feature Transform (SIFT) [45]. Several occupancy grids are used, where each coarser map has half the resolution of the preceding one. These maps are however not generated as they usually are for computer vision, by applying Gaussian filtering and downsampling. Instead, different maps are at all times kept in the memory and they are all updated using the pose estimates generated by the alignment process. This method will make sure that maps are consistent across scales, while avoiding expensive downsampling operations. A positive side-effect is the immediate availability of coarse grained maps which can be used for path planning or other operations.

3.1.2 LOAM

Laser Odometry And Mapping (LOAM) [22] performs SLAM by combining two different algorithms. One algorithm performs odometry at a high frequency to estimate the velocity of the lidar, while the other algorithm runs at a 10 times lower frequency and performs fine matching and mapping.

As LOAM is a feature-based method, keypoints needs to be extracted in order to perform the motion estimation. Keypoints are chosen as points on sharp edges and planar surface patches. The following equation will extract these keypoints:

$$c = \frac{1}{|\mathcal{S}| \cdot ||X^L_{(k,i)}||} || \sum_{j \in \mathcal{S}, j \neq i} (X^L_{(k,i)} - X^L_{(k,j)}) ||. \tag{3.3}$$

Here *i* is a point in the point cloud $\mathcal{P}_k$, and $\mathcal{S}$ is the set of consecutive points around *i* returned by the lidar in the same scan. X^L are the coordinates of the points.

In order for the keypoints to be evenly distributed throughout the scan, each scan divided into four subregions. Each subregion can then provide no more than two edge points and four planar points. An edge point is chosen if the value of c is above a given threshold, and a planar point is chosen if he value of c is below another given threshold. While selecting keypoints, there are certain points that should be avoided. If a local planar surface is roughly parallel to the laser beams, the points will be unreliable. The same is true for points that are on boundary of occluded regions. Lastly, keypoints should not be surrounded by other keypoints, in order to get a good distribution of keypoints throughout the scan.

After keypoints are chosen, the next step is to find correspondences in order to calculate the odometry. Edge lines are found in order to find the correspondences for the edge points, and planar patches as the correspondences for the planar points. The edge line is represented by two points, the closest edge point neighbor in the reprojected point cloud, and the closest edge point neighbor between the two scans. This will form an edge line as the correspondence of an edge point. The planar patch is represented by three points. As with the edge line, the closest neighbor in the reprojected point cloud is found. The closest neighbor in the same scan, and between the scans will be the two other points needed for the patch. This will ensure that the three points are non-collinear.

Now that the correspondences are found, the distance from a keypoint to its correspondence will be calculated. This is done by minimizing the overall distances of all the keypoints. Using the information from these correspondences, the motion estimation is calculated.

The mapping algorithm matches and registers the reprojected point cloud in world coordinates. Keypoints are extracted the same way as above, however, ten times as many keypoints are used. The point clouds are stored on the map in order to find correspondences for the keypoints. After the correspondences are found, distances between all correspondences are minimized using the Levenberg-Marquardt method.

LOAM does not support long time loop closures, and will therefore drift over time, as all odometry methods do.

3.1.3 LeGO-LOAM

Lightweight and Ground-Optimized Lidar Odometry and Mapping (LeGO-LOAM) [23] is an extension of the LOAM method presented above. It is divided into five different modules. The first module, segmentation, projects a single scan onto a range image to perform segmentation. The segmented point clouds is then sent to the keypoint extraction module. The keypoints are then sent to the lidar odometry module which uses the keypoints to find the transformation between consecutive scans. The keypoints are then sent to the lidar mapping module, registering them to a global point cloud. Lastly, the transformation integration module fuses the pose estimation results from the lidar odometry and lidar mapping modules, outputting the final pose estimate.

The segmentation module transforms the point clouds into a range image with a resolution of 1800 by 16, since these are the number of outputs provided by the Velodyne VLP-16 lidar. Each pixel will have the range value associated to the point it represents. A column-wise evaluation of the range image is conducted in order to extract the ground point before performing segmentation. After the ground points have been removed, an

image-based segmentation method is applied in order to segment the range image into different clusters. Clusters with less than 30 points are omitted, as they can form trivial and unreliable keypoints. After this process, only the points that are part of the segmentation are preserved for further processing.

The keypoints are extracted in a similar manner as in LOAM using Equation (3.3) to get a value for c, which is used to extract edge points that do not belong to the ground, and planar points that do belong to the ground. The range-image is then split into six different sub-images instead of four as in LOAM, giving sub-image resolution of 300 by 16.

The lidar odometry module estimates the motion between two consecutive scans. The transformation between the two scans is found by performing point-to-edge and point-to-plane scan-matching. The correspondences are found the same way as in LOAM, however a few changes are made to improve the efficiency and matching accuracy. Since labels were extracted in the segmentation part, it will be used in order to only find correspondences with the same label. The second improvement is using a two step Levenberg-Marquardt method for the distance vectors to find the minimum-distance transformation between two consecutive scans. This two step algorithm will in the first step optimize for the z-translation, pitch and roll, and in the second step optimize for the x- and y-translation and the yaw. Fusing these will provide the full 6 DOF transformation. This will provide similar accuracy, while reducing computation time by about 35%.

The lidar mapping module works similarly to the one in LOAM. The main difference is however how the final point cloud is stored. LOAM saves all point into a single point cloud map, while LeGO-LOAM saves each individual feature set. This will make it possible to only use a local map when estimating the motion. Pose-graph SLAM can then be integrated into LeGO-LOAM, and it can be further extended to include loop closures. In order to detect loop closure, ICP is used.

3.1.4 Google Cartographer

Google Cartographer [24] is a lidar SLAM method which, unlike LOAM and Hector-SLAM, does provide long term loop closure. The original method presented was created for 2D, however it has been extended and is available open-source in ROS for both 2D and 3D.

Google Cartographer is, similarly to LeGO-LOAM, split into two different modules, one local and one global approach. The local approach is used to find the transformation between consecutive scan against a small part of the world. This is done using scan-matching.

Sub-maps are created by an iterative process of aligning scans and sub-map coordinate frames. The pose of the scan frame in the sub-map is represented by the transformation between the scan frame and the sub-map frame. Each sub-map is built by a few consecutive scans, and they are created in the form of probability grids. For each grid point, the corresponding pixel is defined by all points that are closest to that grid point. Whenever a new scan is added to the sub-map, these probabilities are updated using the hits and misses of the scan.

In order to find the transformation between the scans, a Ceres-based [70] scan-matcher is used. This scan-matcher maximizes the probabilities at the scan points in the sub-map, casting it as a nonlinear squares problem. Doing this often gives better precision than the

resolution of the grid, however it is prone to local minima. IMU is therefore fused in order to have a good initial estimate.

The second module is the global approach. This is done by creating many small sub-maps. The poses of all scans and sub-maps are then optimized using Sparse Pose Adjustment [92]. The relative poses where the scans are inserted, are stored in memory for use in the loop closing optimization.

Loop closure optimization is also formulated as a nonlinear least squares problem, meaning it can also be solved using Ceres. A loss function is first applied in order to reduce the influence of outliers, which can appear when the scan-matching adds incorrect constraints to the optimization problem.

As performing loop closures can be computationally expensive, a branch-and-bound scan-matching algorithm is used in order to reduce the search. In addition to this, step sizes between the scans are chosen in order to reduce the space where the scan-matcher will search. An angular step size is also set so that the scan points does not move too much between the scans. Doing this will form a search window around an estimate placed in the center. To search for loop closures over a larger window, a branch and bound algorithm is used. The main idea is to represent subsets of possibilities as nodes in a tree where the root node represents all possible solutions within the window.

To arrive at Algorithm 6, Google Cartographer used the methods of node selection, branching and computation of upper bounds. For the node selection, the algorithm uses Depth-First Search (DFS) as the default choice. The efficiency of the algorithm depends on two things: a good upper bound, and a good current solution. DFS will help providing a good current solution as it can quickly evaluate many lead nodes. A score threshold from below is also added so that no poor matches are validated as a loop closure. Regarding the order in which the children nodes are visited during the DFS, an upper bound on the score of each child is computed, and DFS will visit the child with the largest bound first.

For the branching to be performed, all nodes in the tree will be described by the x- and y-translation, the yaw angle, and the height. The height refers to the height of the node in the tree, meaning that all the leaf nodes have height equal to zero. The root node encompasses all feasible solutions, but it does not explicitly appear in the algorithm and branches into a set of initial nodes $\mathcal{C}_0$ at a fixed height covering the search window. At a given node c with a height $c_h > 1$, we branch into up to four children of height $c_h - 1$.

The remaining part of the algorithm is then to efficiently compute the upper bounds at inner nodes, both in terms of computational effort and in the quality of the bounds. This score is calculated by the following equation:

$$score(c) \geq \max_{j \in \bar{\mathcal{W}}_c} \sum_{k=1}^{K} M_{nearest}(T_{\xi_j} h_k). \tag{3.4}$$

To compute the maximum efficiently, a set of pre-computed grids are used. Pre-computing one grid per possible height, c_h, allows us to compute the score with effort linear in the number of scan points.

Algorithm 6 Depth-first search branch and bound scan-matcher

best_score ← *threshold_score*
Compute and memorize a score for each element in $\mathcal{C}_0$.
Initialize a stack $\mathcal{C}$ with $\mathcal{C}_0$ sorted by score, the maximum score at the top.
while $\mathcal{C}$ is not empty **do**
  Pop c from the stack $\mathcal{C}$.
  if *score(c)* > *best_score* **then**
    if c is a leaf node **then**
      match ← ξ_c
      best_score ← *score(c)*
    else
      Branch: Split c into nodes $\mathcal{C}_c$
      Compute and memorize a score for each element in $\mathcal{C}_c$.
      Push $\mathcal{C}_c$ onto the stack $\mathcal{C}$, sorted by score, the maximum score last.
    end if
  end if
end while
return *best_score* and *match* when set.

3.2 Visual SLAM systems

Visual SLAM is a more mature research field than lidar SLAM. Two state-of-the-art visual SLAM systems are therefore shortly introduced here in order to see how they differ from the current state-of-the-art lidar SLAM systems, and if any of the information can be used for a lidar SLAM system. ORB-SLAM and LSD-SLAM are chosen as they are quite different, but both achieve state-of-the-art results.

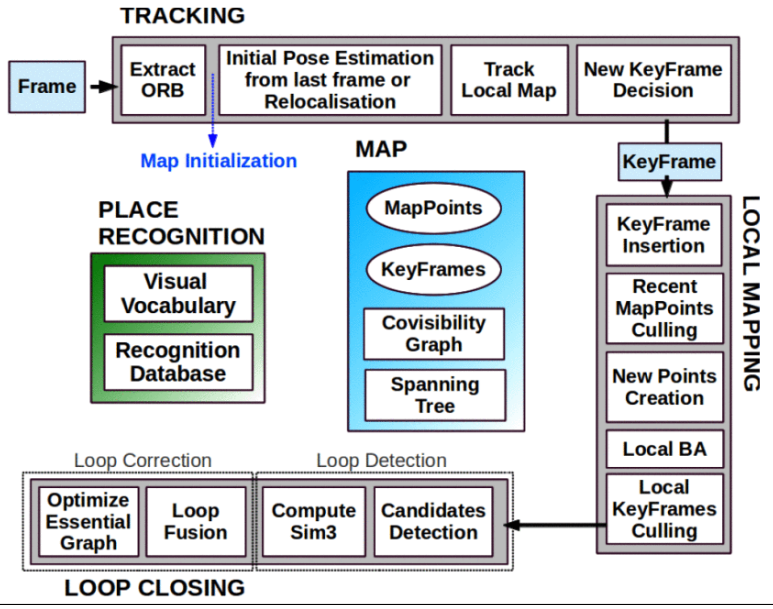
3.2.1 ORB-SLAM

ORB-SLAM [7] is a feature-based monocular visual SLAM system operating in real time. Oriented FAST and Rotated BRIEF (ORB) features [93] are used because they are very fast and had already shown good performance for place recognition [94]. ORB-SLAM uses the same features for all tasks: tracking, mapping, relocalization, and loop-closing, making the system efficient, simple and reliable.

ORB-SLAM consists of three threads that run in parallel: tracking, local mapping, and loop closing as seen in Figure 3.1. The tracking thread localizes the camera with every frame and will also decide when to add new keyframes. This is done by an initial feature matching between two frames and then performing motion-only bundle adjustment. If this tracking is lost, the place recognition module is used to perform global relocalization. After the camera pose has been localized and the features have been matched, a local visible map is retrieved using the covisibility graph of keyframes. The matches from the local points will then be searched by reprojection, and the camera pose will be optimized again with the new matches.

The second thread is the local mapping, this thread processes the new keyframes and performs local bundle adjustment. Performing local bundle adjustment will optimize the

Figure 3.1 ORB-SLAM system overview, showing all the steps performed by the tracking, local mapping, and loop closing threads. The main components of the place recognition module and the map are also shown. Image and caption taken from [7].



reconstruction in the surroundings of the camera pose. The ORB features in the new keyframes are then matched and connected to other keyframes in the covisibility graph, in order to triangulate the new points. After some time, a point culling policy is applied to make sure that only high quality points are retained. If the keyframe is redundant, the local mapping thread is also responsible for the removal.

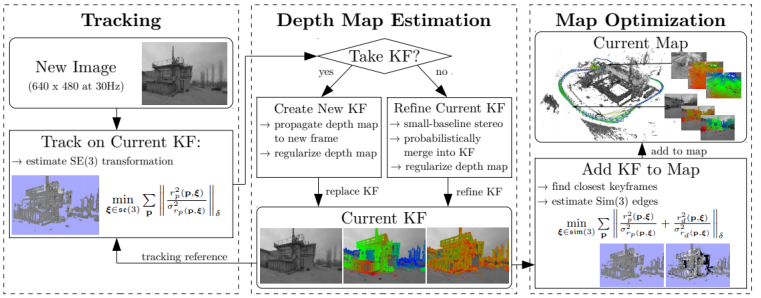
The last thread is the loop closing thread, which search for loop closures every time a new keyframe is added. If the system detects a loop it will compute a similarity transform to calculate the drift accumulated in the loop. After this, both sides of the loop are aligned and the keypoints are fused. The pose graph is then optimized over similarity constraints to achieve global consistency. This is done using the g2o library [27] and Levenberg-Marquardt [75] as the nonlinear solver.

3.2.2 LSD-SLAM

Large-Scale Direct Monocular SLAM (LSD-SLAM) [10] is a direct monocular visual SLAM system operating in real time. Similarly to ORB-SLAM, it consists of three different threads, however for LSD-SLAM they are tracking, depth map estimation and map optimization as seen in Figure 3.2.

The tracking thread will continuously track new camera images, and estimate the rigid body pose with respect to the current keyframe. This thread will use the previous frame as initialization. To do this, a direct image alignment method is used, where the photometric error is minimized based on the following equation:

Figure 3.2 Overview over the complete LSD-SLAM algorithm. Image and caption taken from [10].



$$
\begin{aligned}
E_p(\xi_{ji}) &= \sum_{p \in \Omega_{D_i}} \left\| \frac{r_p^2(p, \xi_{ji})}{\sigma^2_{r_p(p,\xi_{ji})}} \right\|_\delta, \\
r_p(p, \xi_{ji}) &= I_i(p) - I_j(w(p, D_i(p), \xi_{ji}), \\
\sigma_{r_p(p,\xi_{ji})} &= 2\sigma_I^2 + \left(\frac{\partial r_p(p, \xi ji)}{\partial D_i(p)} \right)^2 V_i(p).
\end{aligned}
\tag{3.5}
$$

Here, ξ is the relative 3D pose, I are the images and p the pixel. The subscripts i and j denotes the location of each pixel on the image. D is an inverse depth image, while V is the variance of the inverse depth. The depth map and its variance are only defined for a subset of pixels, containing all image regions in the vicinity of sufficiently large intensity gradients. w is a 3D projective warp function, which projects an image point and its inverse depth into a plane. $||\cdot||_\delta$ is the Huber norm applied to the normalized residual:

$$
||r||_\delta = \begin{cases} \frac{r^2}{2\delta} & \text{if} |r| \leq \delta \\ |r| - \frac{\delta}{2} & \text{otherwise.} \end{cases}
\tag{3.6}
$$

The depth estimation thread has three main tasks: keyframe selection, depth map creation, and depth map refinement. New keyframes are selected if the camera has moved further than a threshold since the last keyframe was added to the map. When a new keyframe is added, a depth map is initialized by projecting points from the previous keyframe into the new keyframe, and then removing outliers and performing one iteration of spatial regularization. The depth map is then scaled to have a mean inverse depth of one, to make sure it is directly incorporated into the camera pose. The previous keyframe is then replaced by the new one for further usage. If the frame is not a keyframe it will be used to refine the current keyframe. In order to do this the same filtering approach as used in Semi-Dense Visual Odometry (SVO) [73] is applied.

The last thread, map optimization, will continuously optimize the pose graph. Just like as in ORB-SLAM, the g2o library [27] is used for the back-end optimization.

3.3 iSAM2

iSAM2 [5], [11] is a fully incremental, graph-based version of the iSAM algorithm [95]. iSAM2 utilizes the Bayes tree data structure, incremental reordering and fluid relinearization to obtain a fully incremental algorithm without the need of periodic batch steps. This algorithm is used for graph-based SLAM by combining the advantage of the graphical model with sparse linear algebra. The creation of the Bayes tree, the incremental variable reordering, the partial state updates and the fluid relinearization are the main reasons why iSAM2 is one of the fastest full graph-SLAM methods used today. The iSAM2 algorithm is summarized in Algorithm 7.

Algorithm 7 One step of the iSAM2 algorithm, following the general structure of a smoothing solution

In/out: Bayes tree $\mathcal{T}$, nonlinear factors $\mathcal{F}$, linearization point Θ, update Δ
In: new nonlinear factors $\mathcal{F}'$, new variables Θ'
Initialization: $\mathcal{T} = \emptyset, \Delta = \emptyset, \mathcal{F} = \emptyset$

1. Add any new factors $\mathcal{F} = \mathcal{F} \cup \mathcal{F}'$
2. Initialize any new variables Θ' and add $\Theta = \Theta \cup \Theta'$
3. Fluid relinearization which yields affected variables $\mathcal{J}$
4. Redo top of Bayes tree
5. Solve for delta Δ
6. Current estimate given by $\Theta \oplus \Delta$

3.3.1 Bayes tree

iSAM2 uses the Bayes tree [72] graphical model. Bayes trees are similar to junction trees [96], however, the Bayes trees are directed and encodes the factored probability density more similar to the Bayes net [97]. In order to get from a regular factor graph to a Bayes tree, elimination needs to be performed, an example of how this elimination is performed can be seen in Figure 3.3. From this it can be seen that when using the Bayes tree, it becomes obvious how to reoder the variables in the tree structure to create the matrix representation. This elimination encodes the Bayes tree with clique structures and creates a chordal Bayes tree. The nodes of the Bayes tree encode the conditional probability distribution corresponding to the variables that are eliminated in the clique, and these conditionals directly correspond to rows in the square root information matrix, represented by R in Figure 3.3.

When adding new nodes to the factor graphs, the Bayes tree needs to be updated with the new information. An example of how this is done can be seen in Figure 3.4. This shows how adding in new factors and variables only affects part of the Bayes tree, and not the full tree.

Figure 3.3 (a) The factor graph and the associated Jacobian matrix A for a small SLAM example, where a robot located at successive poses x1, x2, and x3 makes observations on landmarks l1 and l2. In addition there is an absolute measurement on the pose x1. (b) The chordal Bayes net and the associated square root information matrix R resulting from eliminating the factor graph using the elimination ordering l1, l2, x1, x2, x3. The last variable to be eliminated, here x3, is called the root. (c) The Bayes tree and the associated square root information matrix R describing the clique structure in the chordal Bayes net. A Bayes tree is similar to a junction tree, but is better at capturing the formal equivalence between sparse linear algebra and inference in graphical models. The association of cliques and their conditional densities with rows in the R factor is indicated by color. Image and caption taken from [11].

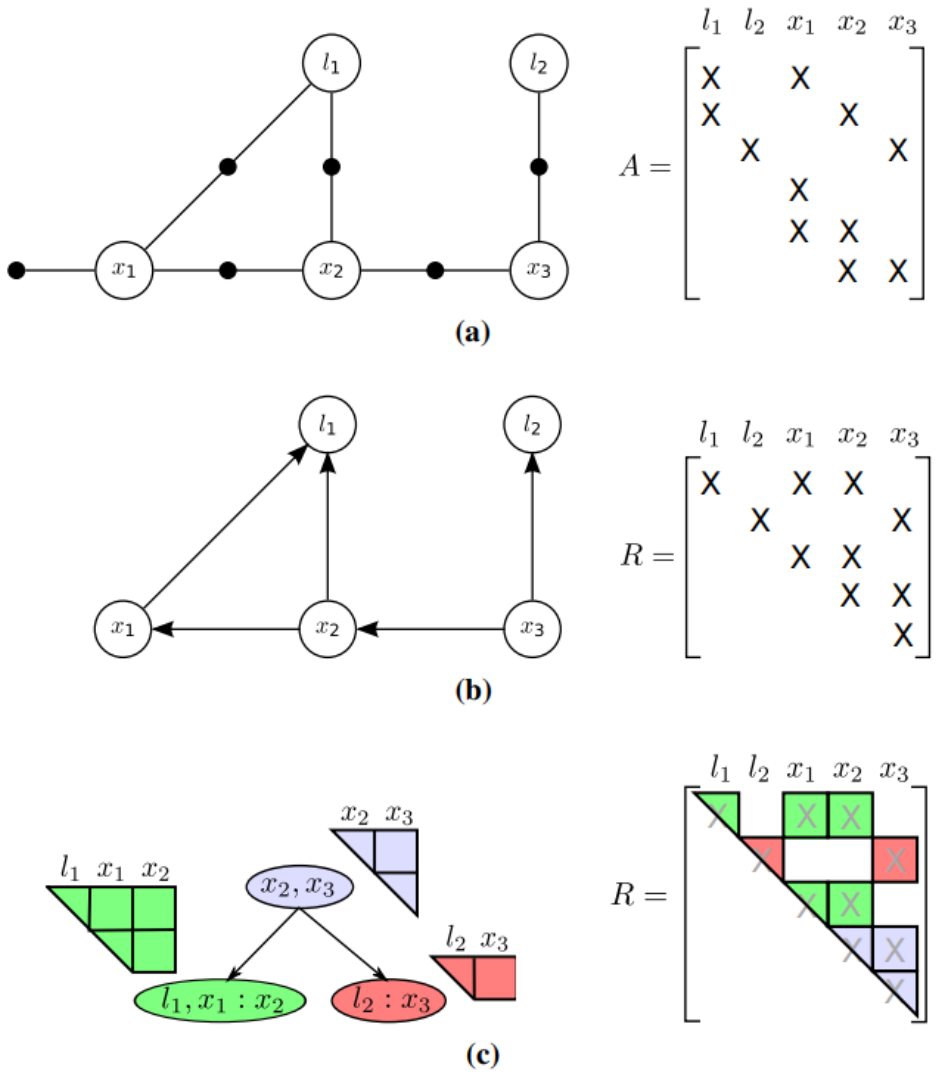
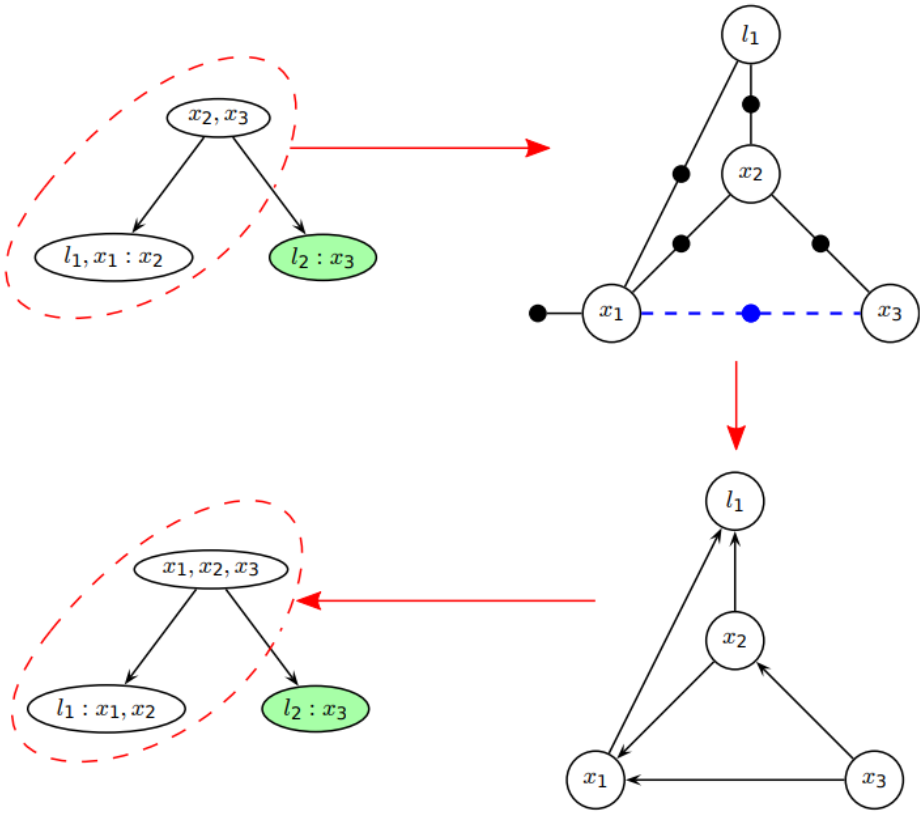


Figure 3.4 Updating a Bayes tree with a new factor, based on the example in Fig. 3.3c. The affected part of the Bayes tree is highlighted for the case of adding a new factor between x1 and x3. Note that the right branch is not affected by the change. (top right) The factor graph generated from the affected part of the Bayes tree with the new factor (dashed blue) inserted. (bottom right) The chordal Bayes net resulting from eliminating the factor graph. (bottom left) The Bayes tree created from the chordal Bayes net, with the unmodified right "orphan" sub-tree from the original Bayes tree added back in. Image and caption taken from [11].



3.3.2 Incremental Variable Ordering

A good variable ordering is important so that sparse matrix solutions can be found efficiently. The goal for the reordering is to create a sparse matrix, while also not reordering too often as it can be computationally expensive. In the Bayes tree, fill-in can be seen as the size of the cliques, as more computations are required for larger cliques. Finding the variable ordering that leads to the minimum fill-in is NP-hard [98], and heuristic algorithms are therefore used. One such algorithm that has been proven to yield good results for SLAM is Column Approximate Minimum Degree (COLAMD) [99]. COLAMD was used for the iSAM algorithm, however as the variable reordering should happen incremental and not periodically using batch reordering, it was found that Constrained Column Approximate Minimum Degree (CCOLAMD) yielded better results. CCOLAMD will force the most recently accessed variables to the end of the ordering, as they are the ones that are the most likely to need reordering. Doing this makes sense for the SLAM problem, as the new state should be similar to the previous state.

3.3.3 Partial State Updates

Instead of updating all states for every iteration, iSAM2 performs partial state updates. This will results in a nearly exact solution in every step, as new measurements often only have local effect. iSAM2 will therefore only solve the variables that actually change for each iteration, usually those close spatially. Full backsubstitution, which is used in iSAM, starts at the root and continues to all the leaves. As this is computationally expensive, the updates for iSAM2 only affects the top of the Bayes tree. These changes can propagate further down the sub-trees if needed, for example during large loop closures. The summarized algorithm for the partial state updates can be seen in Algorithm 8. This shows how iSAM2 starts by updating the top of the Bayes tree and then updates all cliques until the variables does not change more than a small threshold.

Algorithm 8 Partial state update: Solving the Bayes tree in the nonlinear case returns an update Δ to the current linearization point Θ

In: Bayes tree $\mathcal{T}$
Out: update Δ
Starting from the root clique $C_r = F_r$:

1) For current clique $C_k = F_k : S_k$
   compute update Δ_k of frontal variables F_k from the 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

3.3.4 Fluid Relinearization

Fluid relinearization is done so that the linearization point only changes when needed, as relinearization is a computationally heavy task. This is done by keeping track of the va-

lidity of the linearization point for each variable. When relinearization is done in iSAM2, all relevant information will be removed from the Bayes tree and replaced by relinearizing the nonlinear factors. All cliques that are relinearized also needs to take into account the marginal factors that are passed up from their sub-trees. These marginal factors will be stored in the cache so the process does not have to start from scratch each time. The summarized fluid relinearization algorithm can be seen in Algorithm 9.

Algorithm 9 Fluid relinearization: The linearization points of select variables are updated based on the current delta Δ

In: linearization point Θ, delta Δ
Out: updated linearization point Θ, marked cliques M

1) Mark variables in Δ above threshold β : $J = \{\Delta_j \in \Delta | \Delta_j \geq \beta\}$.
2) Update linearization point for marked variables: $\Theta_j = \Theta_j \oplus \Delta_j$
3) Mark all cliques M that involve marked variables Θ_j and all their ancestors

3.3.5 Complexity

Due to the quadratic convergence properties of Gauss-Newton iterations near the minimum, iSAM2 usually require a fairly small number of iterations to converge. For exploration tasks with a constant number of constraints per pose, the complexity is constant $O(1)$, as only the top of the tree is affected. Loop closures are, on the other hand, more computationally expensive, as they need to recompute the earlier poses and landmarks alongside the current one and will therefore update more states in the Bayes tree. This gives a general bound of $O(n^3)$, where n are the number of variables, however as this is the upper bound, it rarely involves all poses and landmarks, meaning the true complexity is usually lower. Batch matrix factorization and backsubstitution can be performed in $O(n^{1.5})$ under certain assumptions that hold for most SLAM problems [100].

Chapter 4

Keypoint extractors and descriptors

4.1 Keypoint extractors

During the specialization project [3] and conference paper for the 2020 Fusion conference [50], several different keypoint extractors where compared in order to find which one was best suited to perform laser odometry on milliAmpere. The conclusion was that ISS was the superior keypoint extractor for the dataset. The other keypoint extractors that were considered were 3D SIFT, Harris Keypoint 3D and NARF.

4.1.1 Intrinsic Shape Signatures

Intrinsic Shape Signatures (ISS) is a keypoint extractor specifically made for 3D point clouds [42]. It is desirable to compute keypoints that are independent of the view and to do this, ISS uses the intrinsic reference frame, which is view independent. The intrinsic reference frame uses eigen analysis of the point clouds to determine the frame. For this computation, a weight is needed for each point to compensate for uneven sampling so that points from a sparsely sampled region contribute more than points from a densely sampled region. This weight is calculated as the inverse of the number of points within a spherical neighborhood.

After this a weighted covariance scatter matrix is computed using all points within a set frame radius. The eigenvalues and eigenvectors of this covariance matrix are then computed. The last step to get the intrinsic reference frame is to cross multiply the eigenvectors belonging to the two largest eigenvalues. The eigenvector belonging to the largest eigenvalue will then be the x-axis, the eigenvector to the second largest eigenvector will be the y-axis and the vector resulting from the cross multiplication will be the z-axis.

To determine the keypoints, the eigenvalues are once again used, and more specifically, the magnitude of the smallest eigenvalue, and the ratio between the smallest eigenvalue and the two larger. Setting a lower threshold for the smallest eigenvalue is similar to what is done in Harris corner detector to detect keypoints which has large variations in all directions. However, there does exists places where the eigenvalues have large variations

in all directions, but they are all similarly sized, and in order to reject these the ratio between the eigenvalues will be used.

The original paper introducing ISS also presents a descriptor, this is however not been implemented in PCL and will therefore not be explored any further.

4.1.2 3D Scale Invariant Feature Transform

3D Scale Invariant Feature Transform (SIFT) [101] is an extension of the Scale Invariant Feature Transform (SIFT) proposed by Lowe [45]. SIFT uses a cascade filtering approach in order to extract keypoints. The first stage of this filtering approach is to identify locations that are repeatable under different views. To do this a scale space is built by performing convolution on the point cloud with Gaussian kernels:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y). \tag{4.1}$$

To be able to detect stable keypoints in the scale-space, the Difference-of-Gaussian (DoG) is used by subtracting adjacent smoothed point clouds:

$$D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma). \tag{4.2}$$

With the help of DoG and the scale-space, one can extract stable features by comparing each point to the eight neighbors it has in the current scale and all nine neighbors from the scale above and below. If the chosen point is either larger or smaller than all 26 of its neighbors, it is selected as a keypoint. The last step of 3D-SIFT is to reject all keypoints where the curvature is low in order to obtain stable keypoints.

4.1.3 Harris Keypoint 3D

The Harris Keypoint 3D detector [44] is an extension of the Harris corner detector [102]. The Harris corner detector measures the local changes of pictures to determine corners using the local autocorrelation function, which is similar to the image gradients. The Harris keypoint 3D replaces the image gradients by surface normals. Using the surface normals, a covariance matrix, *Cov*, is calculated around each point. The keypoint response for each point is then defined by

$$r(x, y, z) = \det(Cov(x, y, z)) - k(trace(Cov)))^2, \tag{4.3}$$

where k is a positive parameter. In order to not have too many keypoints in a small neighborhood, a non-maximal suppression process and a thresholding process is performed to extract the final keypoints.

4.1.4 Normal Aligned Radial Feature

Normal Aligned Radial Feature (NARF) [43] transforms the point clouds into range images, and extracts the keypoints with the help of these 2D images. The keypoints are selected with two goals in mind. Firstly, the points should be selected from a stable surface with sufficient changes in the neighborhood, and secondly, they should be making use of the object borders so that outer shapes of objects from a certain perspective are chosen.

The NARF keypoint extractor has four steps. It starts by looking at the neighborhood of every point and calculates a score based on surface changes and directions in order to incorporate the border information. Then an interest value will be calculated on the basis of how much the dominant directions change within the neighborhood, and how much the surface itself changes. Smoothing will then be performed on the interest values, and lastly non-maximum suppression will be performed to obtain the final keypoints.

4.2 Keypoint descriptors

The specialization project [3] and conference paper for the 2020 Fusion conference [50] also compared several keypoint descriptors in order to find the best suited to perform laser odometry on milliAmpere, when using the ISS keypoint extractor. The conclusion was that SHOT was the superior keypoint extractor for the dataset. The other keypoint descriptors that were considered were PFH, FPFH, 3D SC, USC and NARF.

4.2.1 Signatures of Histograms of OrienTations

Signatures of Histograms of OrienTations (SHOT) [12] is inspired by the two dimensional descriptor used by SIFT. Three dimensional descriptors can be divided into two main categories, namely Signatures and Histograms. SHOT combines both these methods for an effective and robust descriptor, and it also creates a local reference frame that is both unique and unambiguous.

To compute a unique and unambiguous reference frame, SHOT builds on both 3D Shape Context (3D SC), which has an unambiguous local reference frame, and the descriptor introduced in ISS, which has a unique local reference frame. To calculate such a reference frame, a variant of total least squares estimation based on the normal direction is used. SHOT, like several of the other descriptors, uses Principal Component Analysis (PCA) to estimate the normals. However, SHOT uses a modified covariance matrix to increase the repeatability and improve the robustness, resulting in the following covariance matrix:

$$\mathbf{M} = \frac{1}{\sum_{i:d_i \leq R}(R - d_i)} \sum_{i:d_i \leq R} (R - d_i)(p_i - p)(p_i - p)^T. \tag{4.4}$$

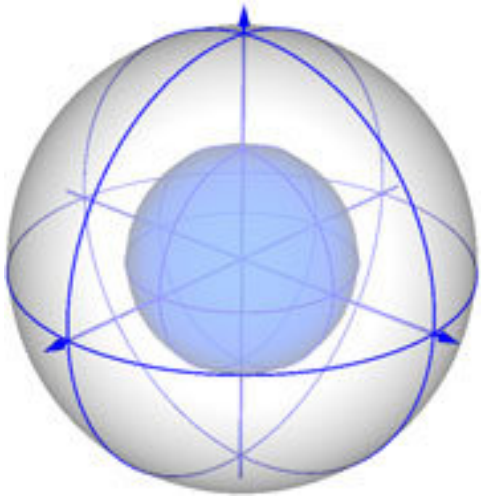
In this equation, R represents a spherical support radius, p_i represents the nearest neighbors, p represents the keypoint and $d_i = ||p_i - p||_2$.

The eigenvectors retrieved by PCA is a good starting point for a repeatable local reference frame, but it will have a sign disambiguation. To solve the disambugiation, the x-axis will be defined as:

$$\begin{aligned} S_x^+ &= \{i : d_i \leq R \wedge (p_i - p) \cdot x^+ \geq 0\}, \\ S_x^- &= \{i : d_i \leq R \wedge (p_i - p) \cdot x^- \geq 0\}, \end{aligned} \tag{4.5}$$

$$x = \begin{cases} x^+, & |S_x^+| \geq |S_x^-| \\ x^-, & \text{otherwise.} \end{cases}. \tag{4.6}$$

Figure 4.1 Signature structure for SHOT. Image taken from [12].



The x^+ is chosen as the eigenvector with the largest eigenvalue, while x^- is chosen as the opposite vector. This is also done with the z-axis, where the eigenvector with the lowest eigenvalue is chosen instead. The y-axis is obtained as $z \times x$, giving us the unique and unambiguous local reference frame.

The next step is to build the descriptor, which is based on intensity gradients. Even though gradients can be noisy, it is seen as very effective because of the high descriptive power it holds, as is proven by the two dimensional SIFT descriptor. To reduce the influence of the noise and thereby improving the robustness, SHOT uses local histograms of the intensity gradients.

The descriptor encodes histograms of the gradients, which are more representative of the local structure than plain three dimensional coordinates. Because of the local reference frame, geometric information about the locations of the points are included, mimicking a signature. This is done by computing a set of local histograms, and the descriptor therefore lays at the intersection between Signatures and Histograms.

The structure of the signature is chosen by an isotropic spherical grid, partitioned along the radial, azimuth and elevation axes, as can be seen in Figure 4.1. The original paper proposes 32-bin histograms, resulting from eight azimuth divisions and two elevation and radial divisions (Figure 4.1 only shows four azimuth divisions for clarity). The local histograms are then built up by accumulating points into bins based on a function of the angle between the normal at each point within the part of the grid and the normal at the feature point. To achieve robustness of variations of the point density, the descriptor is normalized to sum up to 1.

SHOT is implemented in PCL using a descriptor of length 352, meaning that it has more divisions than what is originally proposed in the paper. SHOT also has two extensions in PCL, one extension that utilizes several processors, based on the OpenMP interface, and one that includes color information [103].

4.2.2 Point Feature Histogram

Point Feature Histograms (PFH) [46], [104] uses a multi-dimensional histogram of the mean surface curvature to uniquely describe each keypoint that has been selected. In order to compute the mean surface curvature at a point, all the neighbors within a given radius need to be found. After this, the normals of all the selected points needs to be found, this is done by approximating them using PCA. Once all the normals are obtained, the existing viewpoint information is used to re-orient all the points consistently.

A Darboux frame [105] is defined using the neighborhood around a point, this frame is defined as:

$$u = n_s, v = (p_t - p_s) \times u, w = u \times v, \tag{4.7}$$

where n_s is the source normal, p_t is the target point and p_s is the source point.

Three features are calculated and categorized in a histogram, where each bin contains the percentage of the points with the feature-values within defined intervals. Each of the features uses five bins each, giving a 125-bin histogram as the descriptor.

4.2.3 Fast Point Feature Histogram

Fast Point Feature Histograms (FPFH) [47] is an extension of PFH to reduce the computational complexity and memory usage. This allows FPFH to be used for real-time applications, as PFH can be a major bottleneck in the registration framework. FPFH manages to reduce the computational complexity from $O(nk^2)$ to $O(nk)$, where n are the number of points, and k the number of neighbors within the search radius.

The first step is to compute the histogram of the three angles between a point p and its k-nearest neighbors. This produces the Simplified Point Feature Histogram (SPFH). In order to keep the descriptive power of PFH, the SPFH is calculated for all of the points neighbors. The resulting equation for calculating the FPFH is then:

$$FPFH(p) = SPFH(p) + \frac{1}{k} \sum_{i=1}^{k} \frac{1}{w_k} \cdot SPFH(p_k), \tag{4.8}$$

where w_k is the distance between the point and its kth neighbor.

The three angles are then binned into three 11-bin histograms that concatenate into a single 33-bin FPFH descriptor.

4.2.4 3D Shape Context

3D Shape Context (3D SC) descriptor [48] is an extension of the 2D Shape Context (2D SC) descriptor [106] originally used for images. 3D SC uses a sphere centered around the chosen keypoint, where the north pole of the sphere is oriented with the surface normal estimate of the point. The sphere is divided into bins, these bins are equally spaced in the azimuth and elevation dimensions, and logarithmically spaced along the radial dimension. Each bin corresponds to one element in the feature vector.

Each bin is then normalized with respect to the volume of the bin and the local point density. This normalization compensates for the large variation in bin sizes and point

densities. 3D SC does however not have a unique reference frame, creating the possibility for several descriptors for a single keypoint. This is due to a degree of freedom in the azimuth direction.

4.2.5 Unique Shape Context

The Unique Shape Context (USC) [49] is an extension of the 3D SC to deal with the problem of having multiple descriptors for the same point. To deal with this problem a local reference frame that is both unique and unambiguous is defined. This local reference frame is the same as the one introduced in the SHOT descriptor. In addition it will also decrease the memory usage, as each keypoint only has one descriptor.

4.2.6 Normal Aligned Radial Feature

The Normal Aligned Radial Feature (NARF) keypoints extractor introduced in Section 4.1.4 has its own descriptor [43]. This descriptor, is as SHOT, heavily influenced by SIFT [45]. It captures the existence of occupied and free space, making it robust to noise and making it possible to extract a unique local coordinate frame at each keypoint.

Normal aligned range value paths are built around each keypoint. This batch is built by using the unique coordinate system extracted from the keypoint extractor. The resulting x- and y-coordinates will define cells of the descriptor, where the value of the cells are based on the z-values of the points within the cell. The size of this image patch needs to be small enough to not surpass the resolution of the scan, while also high enough to keep enough descriptive power. The image patch is then smoothed using Gaussian blur, as is done in several of the keypoint extractors and descriptors.

The descriptor is then built from 36 different beams projected into a star-shaped pattern. These beams will calculate the change over the underlying image patch, weighted by the distance from the center, making the closer points more influential. To make the descriptor invariant to rotations around the normal, the histogram bin with the maximum value is chosen as the dominant orientation.

Chapter 5

Platform

This chapter describes the software that is used to develop the SLAM system, as well as an introduction to milliAmpere and the sensors used. It also shows the environment where the dataset used in this thesis is gathered and where it will operate.

5.1 Software

5.1.1 Robot Operating System

Robot Operating System (ROS) is a distributed and modular open-source platform for robotics software development created in 2007. ROS is made to simplify communication between different parts of robotic systems as a standardization of protocols. One of the benefits of using ROS is the possibility to use rosbags. Rosbags are a way to store data along with time, to be able to replay it by publishing the recorded data in the same sequence and after the same elapsed time as it was recorded.

5.1.2 Rviz

Rviz is a package in ROS which is used for visualizing data in the ROS framework. It can through standard ROS messages visualize point clouds, or other standard messages sent over ROS.

5.1.3 Point Cloud Library

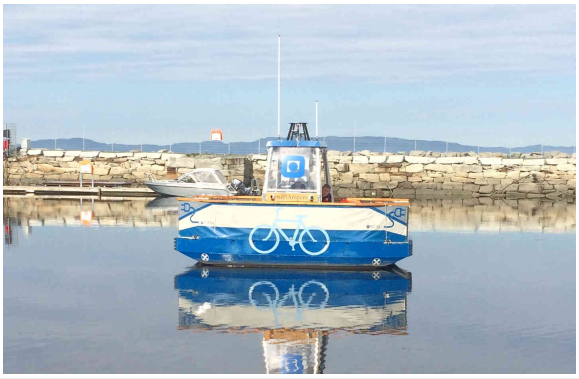
Point Cloud Library (PCL) is an open-source library for efficient point cloud handling. It contains algorithms and processing tasks and contains all of the keypoint extractors and descriptors discussed in this thesis. It is a templated C++ library, and it has ROS integration making it possible to translate between ROS data types and PCL data types.

5.1.4 Georgia Tech Smoothing and Mapping

GTSAM is introduced in Section 2.5.1. In this thesis it is used to implement the SLAM back-end, utilizing the iSAM2 method introduced in Section 3.3.

5.2 The autonomous ferry prototype milliAmpere

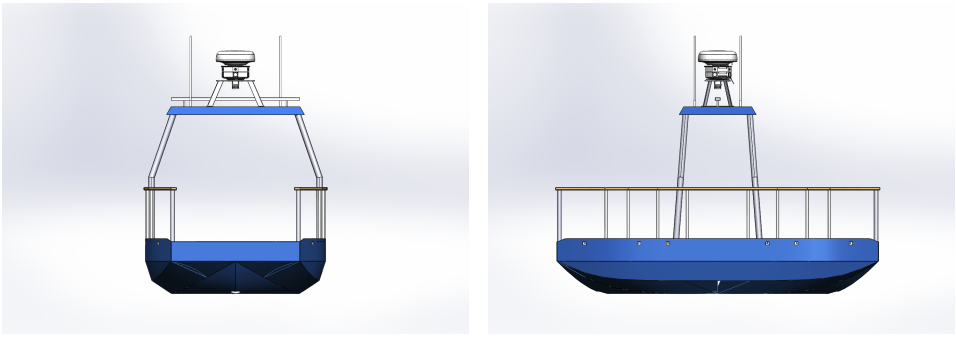
Figure 5.1 Image of milliAmpere being tested. On-board is Brage Sæther and Emil Hjelseth Thyri. Image taken by Nicholas Dalhaug.



milliAmpere is a prototype of the autonomous ferry being designed. Once ready, the ferry is intended to go between Ravnkloa and Brattøra in Trondheim, crossing a 110m wide passage bringing passengers over the river. There are docks on both sides of the passage, the northern dock is fixed while the southern dock is floating.

milliAmpere is approximately 3m tall, which is half the size of what the autonomous ferry will be, and most of the sensors are placed on top of the vessel. There are openings on both sides of the vessel so that pedestrians and cyclists can board and deboard.

Figure 5.2 Illustration of the ferry milliAmpere. Illustration created by Egil Eide.



(a) Front **(b)** Side

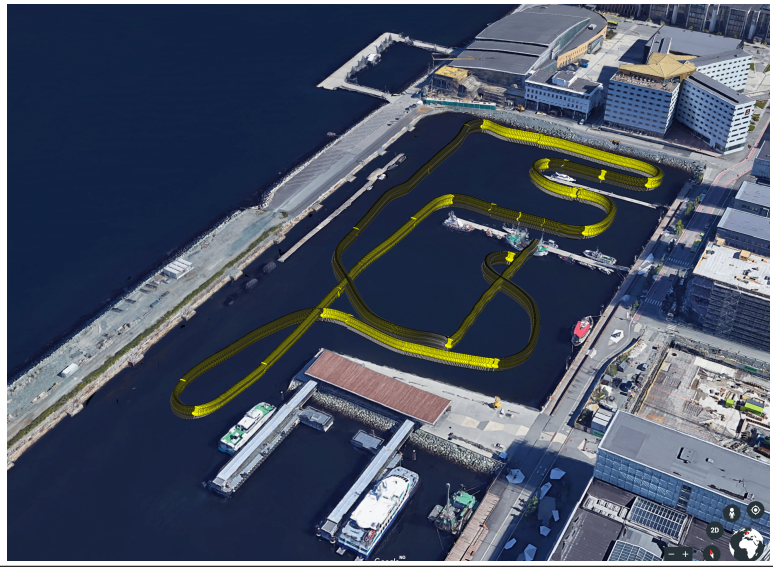
5.3 Logging Area

The logging of data was done in a previous master's thesis at NTNU [1]. The data was logged in Brattørabassenget, area A of Figure 5.3, while the finalized boat is proposed to operate in area B. In total 6305 point clouds were recorded over 636 seconds with milliAmpere travelling a trajectory of approximately 1060 meters.

Figure 5.3 Adapted from Gule Sider Kart and slightly modified with Paint, the map shows the area of Brattøra, in the centre of Trondheim. A depicts the area of data logging, while B depicts the area where the finalized auto-ferry is proposed to operate. Image and caption taken from [1].

It was desired to make a large dataset, with several different operating conditions in order to test how well the method works in different environments. Starting and stopping at the same location and crossing paths was also important to allow for loop-closures. The full trajectory and the environment can be seen in Figure 5.4. Seen in the bottom left of this trajectory is a corridor-like environment, which will be similar to the true operating environment. In this trajectory, the lidar will only be able to reflect signals to the sides of the boat, like a corridor, and not for the full 360 degrees. This is an environment that is usually hard for odometry and SLAM, as the surrounding does not change much from one point cloud to the next and that there are no corners. Another difficulty is that some of the docks seen in the figures are not seen well by the lidar, because they are low and are therefore most of the times outside the vertical field of view of the lidar.

Figure 5.4 Screen-shot of the ferry's trajectory from which the data is logged. The almost straight line from the middle floating dock is the return, while the curved one is from shipping out. Longitude and Latitude was from a rosbag in MATLAB, converted to a .KML-file and uploaded in GOOGLE earth. Note that at the of the logging procedure, the boat count was higher, and the construction site on the right had become structure much visible to the LIDAR. Image and caption taken from [1].



5.4 Sensors

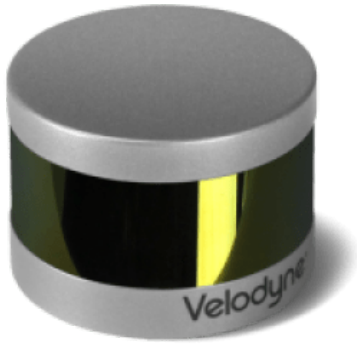
5.4.1 Lidar

The lidar used in this project is the Velodyne VLP-16, also known as the Velodyne LiDAR Puck. This is a 3D lidar with 16 channels and a measurement range of 100m, with a range accuracy up to ± 3 cm. It has a vertical field of view of 30° from $+15.0^\circ$ to -15.0°, and a horizontal field of view is 360°. The vertical angular resolution is 2.0° and the horizontal angular resolution is $0.1^\circ - 0.4^\circ$. It has a rotation rate of 5 Hz - 20 Hz, and the rotation rate was set to 10 Hz during the data collection, giving a horizontal angular resolution of 0.2°.

5.4.2 Global Navigation Satellite System

The GNSS receiver used in milliAmpere is the Hemisphere Vector VS330. This receiver supports Real Time Kinematics (RTK)-GNSS, and it communicates through a Satel Radio Link mounted under the roof. It has two antennas and an internal gyro stabilizer, making it able to measure the heading at an accuracy of 0.05 degrees. The horizontal accuracy is ± 0.30 metres without the RTK and ± 0.01 meter with RTK, with an update rate specified as 20 Hz.

Figure 5.5 A picture of the lidar used in this project. Image taken from [13].



5.4.3 Inertial Measurement Unit

The Inertial Measurement Unit (IMU) used on milliAmpere is Xsens MTI-G-710, it is an inertial navigation system with a GNSS-receiver, magnetometer and a barometer integrated with the IMU. It gives acceleration and angular velocities in the body frame, with an update rate of 100Hz.

Chapter 6

SLAM system

The main contribution of this thesis is the development of a feature-based lidar SLAM system, which can operate on ASVs in harbour environments. This is possibly the first feature-based lidar SLAM system developed to operate in harbour environments, and might also be the first to utilize keypoints and descriptors in order to solve the difficult loop closure problem. The pipiline for this SLAM system can be seen in Figure 6.2 and the system will be presented in detail in this chapter.

The developed SLAM system consists of a front-end and a back-end, as typical SLAM systems are, which can be seen in Figure 6.1. The front-end is further split into feature extraction and data association, and the data association is again split into short-term and long-term. The short-term data association can be considered to be odometry, and the long-term can be considered to be loop closures. For the SLAM system developed in this thesis, ISS will be used to extract the keypoints, and SHOT will be used as the descriptor in order to perform the data association. The same keypoints will be used for tracking, mapping, relocalization, and loop closing, this idea comes from ORB-SLAM [7]. For the back-end MAP estimation, the iSAM2 framework is used. Using iSAM2 makes it possible to fuse in other sensors to the factor-graph, and the system developed supports both GNSS and IMU data.

6.1 Pre-processing the data

Before the keypoint are extracted, the data should be pre-processed in order to make the extraction more computationally efficient. Pre-processing the data will remove thousands of points from the original point clouds, meaning the keypoint extractors has to look through fewer points. The pre-processing done in this thesis is the same as the one used in the specialization project [3] and conference paper for the 2020 Fusion conference [50]. A similar pre-processing was also done in the previous projects, [1] and [2], however, these only included conditional removal. The data before and after pre-processing can be seen in Figure 6.3.

Figure 6.1 Front-end and back-end in a typical SLAM system. Image taken from [6].

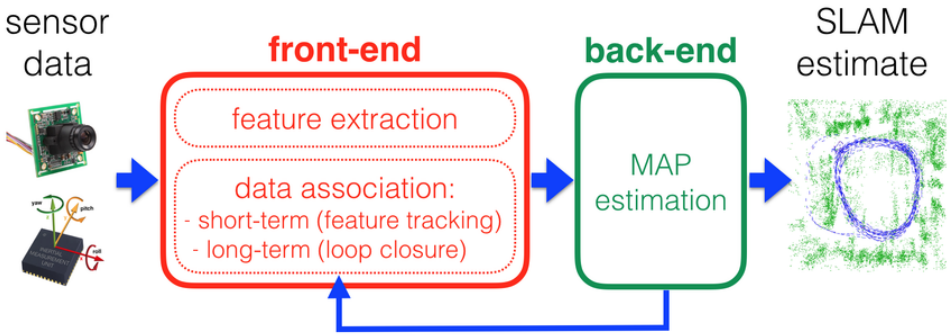


Figure 6.2 SLAM system pipeline.

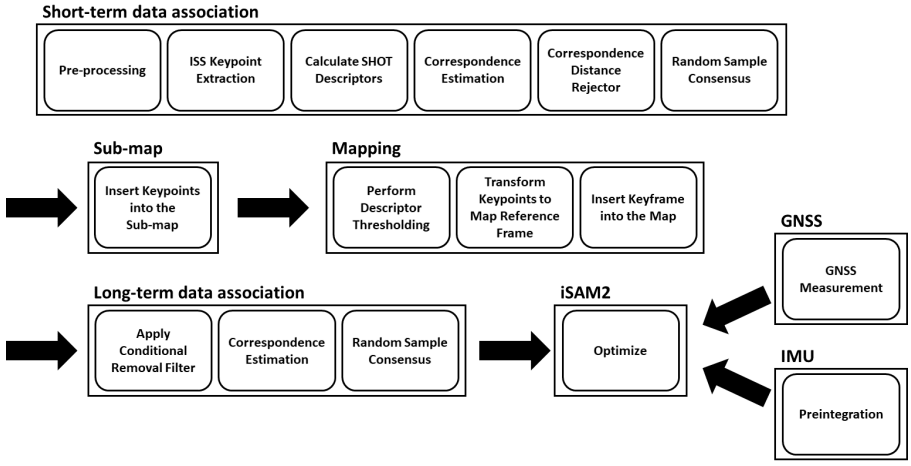
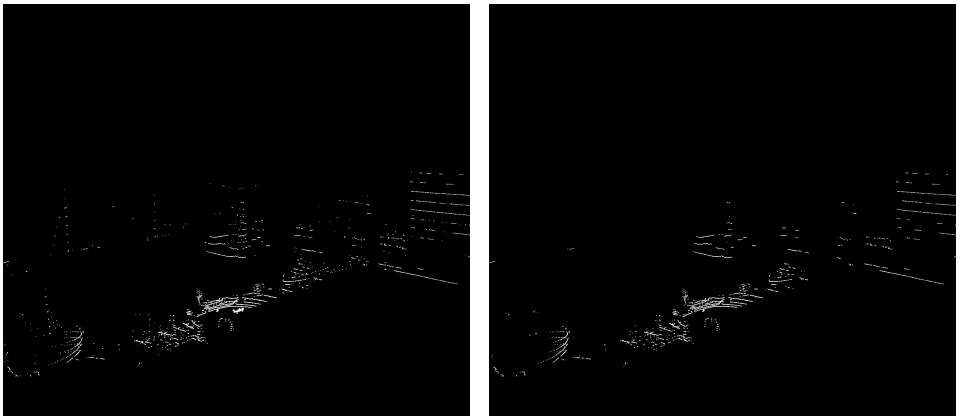


Figure 6.3 Lidar data before and after filtering.

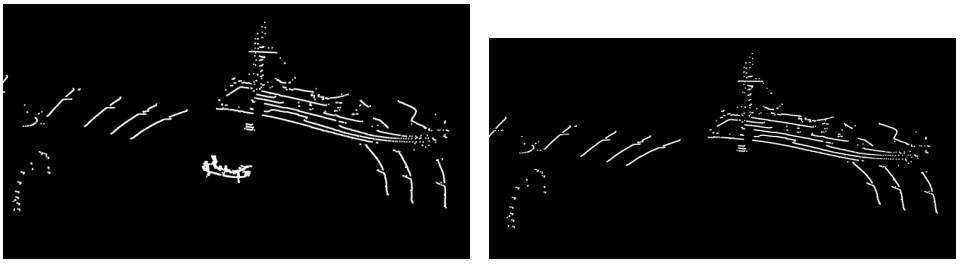


(a) Before filtering (7713 points) **(b)** After filtering (3227 points)

6.1.1 Conditional removal

When visualizing the original dataset, there are points close to the lidar corresponding to milliAmpere itself. Tracking the points on milliAmpere will cause problems, as those points will be dynamic. Recall that SLAM systems often struggle in dynamic environments, as discussed in Section 2.4. Using a conditional removal filter from PCL, all the points within a certain distance to the lidar will be removed as can be seen in Figure 6.4. It can clearly be seen that this operation removes the points created by reflecting milliAmpere from the lidar data, while maintaining the rest of the data.

Figure 6.4 Lidar data before and after the conditional removal filter is applied.



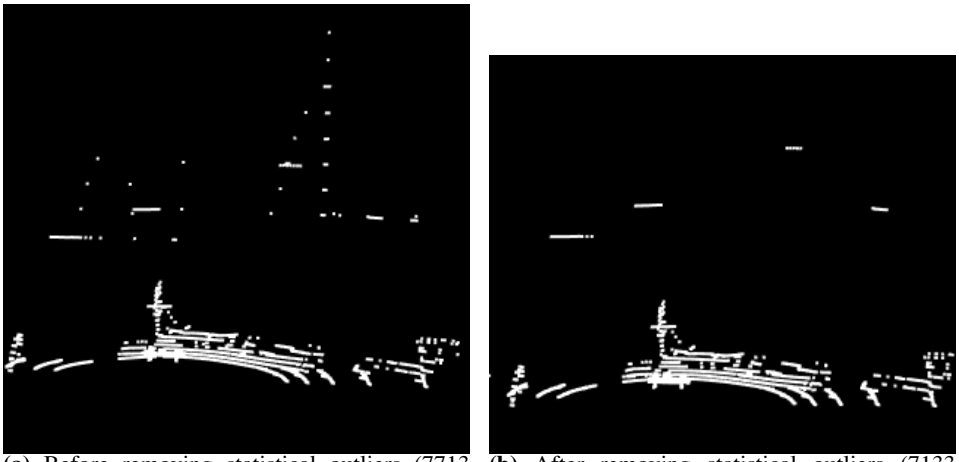
(a) Before conditional removal (7713 points) **(b)** After conditional removal (5624 points)

6.1.2 Removing statistical outliers

To further reduce the computational complexity, statistical outliers are removed. A statistical outlier is in this context understood as a point which does not have enough neighbors within a given radius. These points often occur from reflections on the water, but they can

also occur for objects far away due to the angular resolution of a lidar. Points with few neighbors will not be able to provide good keypoints or descriptors, as they often use the normal estimation uses the points in the neighborhood. A statistical outlier removal filter from PCL is implemented, and the results can be seen in Figure 6.5. In the background of Figure 6.5a there are several points, which looks like they are from a sailboat. These same points are removed in Figure 6.5b as they are do not have enough neighbors close by. The statistical outlier removal filter from PCL applied in this system requires each point to have at least 3 neighboring points within 1 meter to not be filtered away.

Figure 6.5 Lidar data before and after removing statistical outliers.



(a) Before removing statistical outliers (7713 points)

(b) After removing statistical outliers (7133 points)

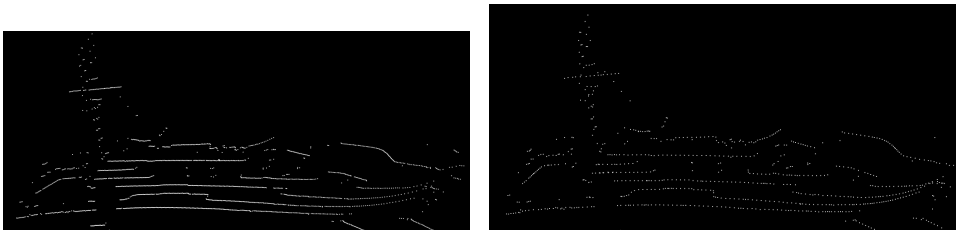
6.1.3 Voxel grid

Finally, a voxel grid is used to downsample the point clouds using a voxelized grid approach. This is done by implementing the voxel grid filter in PCL. These grids can be seen as tiny three dimensional boxes in space, where all the points within the box are downsampled to its centroid. Doing this greatly reduces the total number of points in the point clouds as can be seen in Figure 6.6. For this system, leaf sizes of 0.1 meters in all directions are used.

6.2 Feature extraction and short-term data association

The feature extraction and short-term data association used in this thesis is, like the preprocessing, similar to what is done in the specialization project [3] and the conference paper for the 2020 Fusion conference [50]. This pipeline was inspired by [107], and odometry pipeline can be seen in the top row of Figure 6.2.

Figure 6.6 Lidar data before and after applying the voxel grid filter.



(a) Before applying the voxel grid (7713 points) **(b)** After applying the voxel grid (3968 points)

After the data have been pre-processed, the ISS keypoint extractor is used to extract repeatable and stable keypoints. To best match keypoints between consecutive point clouds, the SHOT descriptor is calculated at each keypoint, and a correspondence estimation function in PCL is used to find correspondences between consecutive point clouds. As these correspondences are prone to false positives, two correspondence rejectors have been included in this pipeline, firstly a distance rejector, which removes matches over a threshold distance. As the odometry algorithm is performed at a very high rate, it can safely be assumed that the correct correspondences have not moved a large distance. Secondly, RANSAC is implemented to get a consistent transformation between consecutive point clouds, using the correspondence rejector sample consensus function available in PCL. This function removes false positives, and will calculate a motion estimation based on the resulting inliers. RANSAC algorithms are often difficult to be tuned, however, this built-in function only allows us to change two variables, the number of iterations, and the inlier threshold. The inlier threshold was set to 0.2 meters, due to resolution provided by the Velodyne VLP-16, and that it proved good results while testing. The maximum number of iterations were set to 800, as this found the correspondences when possible, while not being too computationally complex.

At a few occasions, this feature-based matching does not manage to find correspondences between consecutive point clouds. In order for the trajectory to not diverge, a backup is needed, and for this system it was decided to use ICP. This is a scan-matching algorithm, and it might increase the run-time of the algorithm, however it is rarely needed, as the feature-based matching is used for the vast majority of the point clouds.

The resulting motion estimate from RANSAC or ICP are provided in the form of a transformation matrix, $\mathbf{T}$. This transformation represents the movement of milliAmpere between the consecutive point clouds. The transformation matrix consists of a rotation matrix, $\mathbf{R}$, and a translation vector, $\mathbf{t}$. In order to calculate the full trajectory of the system, these matrices can be concatenated over time as presented in Section 2.9.

6.3 Mapping

SLAM is not only about estimating the correct poses and trajectories, but it is also about creating a map of the surroundings. Recall that three different maps were introduced in Section 2.4, the topological map, the feature-based metric map and the dense metric

map. As a lidar is used, all the metric information is available, and it will therefore be advantageous to use a metric map and not the topological map. The map creates by this SLAM system could be used for path planning, collision avoidance, or as a visualization tool to understand the environment. Neither of these applications require a dense map, as dense maps are more used for reconstruction. A dense map could be advantageous for using the SLAM system for docking, as it could provide more accurate information around the docking platform, however this would require more computational power and a large memory. Keypoints are also already extracted, making the feature-based metric map a good choice.

As a feature-based metric map has been chosen, it needs to be decided what keypoints to be used and how often new keypoints should be added to the map. If all keypoints are added to the map, it will grow very large and quickly be infeasible to find loop closures for any SLAM system, even while using iSAM2.

It was therefore decided like they do in LOAM [22], ORB-SLAM [7], and several other state-of-the-art SLAM systems, to perform the mapping algorithm at a lower frequency than the odometry algorithm. New keyframes [108] are added to the map every time milliAmpere has moved more than 3 meters, or at a frequency 1/60th of the odometry algorithm if it has moved less than 3 meters. This distance is calculated by the odometry algorithm presented in Section 6.2. This results in a map that is feasible for the iSAM2 back-end while also spatially distributing the points by not adding keyframes often if the ASV is not moving, or slowly moving. In addition to reducing the frequency of which the keypoints are added, the quality of keypoints also needs to be considered, as stable and repeatable keypoints will give the best results when it comes to loop closure. In order to do this, only the keypoints that are matched using the RANSAC function introduced earlier are considered. To make sure that only the most descriptive keypoints are kept, a hard threshold on the descriptor similarities between the consecutive scans are also applied, to make sure the descriptive power is kept in the map. The descriptive power is important as this map will be used for long-term data association. After this is performed, the keypoints are transformed into the map reference frame, using the pose estimate from the odometry. In addition to storing a map of the spatial information of the keypoints, a map with the SHOT descriptor values is also stored, meaning a reduction in map size greatly reduces the memory required by the system.

Figure 6.7 shows a map created using all keypoints matched by RANSAC at the same frequency as the odometry algorithm, and using GNSS to estimate the pose. This map is very descriptive, and there are many points overlapping, creating places with a high number of keypoints, especially at some walls. Figure 6.8 shows the map where keypoints are only added at the keyframes. It is easy to see that while this reduced the keypoints by more than 90%, from almost 60000 points to below 4000 points, it could still be used for for all of the applications introduced above. Figure 6.9 shows the map created when descriptor thresholding is applied as well, and it can again be seen that most the information is still there even though the map now only contains 1500 points. This reduction in will greatly reduce the memory usage of the system, and also make it easier for the SLAM system to perform loop closures more efficiently using the same map.

Figure 6.7 All keypoints matched using RANSAC (57539 points).

Figure 6.8 Keypoints from the keyframes using RANSAC (3457 points).

Figure 6.9 Keypoints from the keyframes using RANSAC and descriptor thresholding (1511 points).

6.4 Long-term data association

Recall that long-term data association is often called loop closures and is what separates a SLAM system from an odometry system. Loop closures will allow the system to recognize a place it has been to before and thereby adjust its states using this information.

Several different state-of-the-art data association methods were introduced in Section 2.8. RANSAC was chosen as the method implemented in this system as it is available open source through PCL, and can therefore easily be connected to the rest of the system. This was also used and proven successful for the odometry pipeline introduced earlier. The long-term data association will be based on several of the same principles as the odometry pipeline. However, a major difference is the computation required, as the odometry only compares one scan of keypoints to the next, while the long-term needs to compare the recent scans to the entire map. The parameters of the RANSAC algorithm are also changed, as false positives for loop closures will degrade the quality of the entire system. The inlier threshold is therefore reduced to 0.1 meters, and the maximum number of iterations were slightly increased to 1000 to make sure that enough combinations are explored now that the maps are larger.

As the odometry of the system is quite accurate in the xy-plane, the loop closure candidates will be the keypoints in the map that are spatially close to the current odometry pose estimate. This is done in several state-of-the-art SLAM system, among them Large-Scale Direct Monocular SLAM (LSD-SLAM) [10]. In order to do this, and thereby greatly reducing the computational complexity, a conditional removal filter of 80x80 meters is ap-

plied with the center in the odometry pose estimate. This distance is chosen as the furthest trustworthy keypoints are around 30 meters away in each direction, leaving a 10 meter error on the odometry, so that it is possible to find loop closures even if the odometry has a quite large error.

Recall that there are three ways to perform loop closures, image-to-map, image-to-image, and map-to-map. As a feature-based metric map is already created, either the image-to-map or map-to-map could be used to utilize this map. As the loop closure algorithm is quite computationally expensive, it should be, as the mapping algorithm, run at a lower frequency than the odometry algorithm. It was decided to perform the loop closure algorithm at a frequency 12 times lower than the odometry algorithm. In order to not lose the information between each time the algorithm is run, the map-to-map method was chosen. This will also provide more keypoints to be compared, increasing the chances of finding a correct loop closure, while at the same time reducing the probability of false positives. A sub-map of the recent scans therefore needs to be created in order for this method to work. This sub-map is created by the keypoints matched using the RANSAC function at every 4th iteration of the odometry algorithm. Similarly to the full map, both the spatial information and the SHOT descriptor values are stored in the memory. Resulting in a sub-map from three different scans, that needs to be matched to the full map.

The loop closure in then performed using the same pipeline as in the odometry introduced in Section 6.2, however the transformation matrix from RANSAC is not utilized. Instead all the keypoints are given a unique ID that is sent to the iSAM2 back-end alongside the coordinates. When a loop closure is detected by the front-end, the ID of the keypoint from the full map is sent to the back-end instead of the ID of keypoint from the sub-map. iSAM2 will in the back-end use this information to perform the loop closure optimization.

6.5 iSAM2 SLAM back-end

iSAM2 was introduced in Section 3.3 and is used for the back-end of the SLAM system created in this thesis. In the system created, there will be two different variable nodes in the graph, before the IMU is fused into the system. These are the pose nodes, containing the translation and rotation of milliAmpere, and the landmark nodes, containing the location of the keypoints.

The pose nodes gets its information from the front-end. A built-in factor node in GTSAM called BetweenFactor is then used to connect the pose nodes between consecutive pose nodes. This factor node utilizes the information of the transformation between two poses and uses this alongside a noise model to estimate the pose. A small multivariate Gaussian noise is assumed between the poses. In addition to the BetweenFactors, an initial estimate is also required for all pose nodes, which is set to the previous estimate from iSAM2 concatenated with the latest estimate sent from the front-end odometry. If only pose nodes are added into the iSAM2 framework, it will be the same as performing odometry, as there is no way for the system to improve the system, as no other information is added.

The second type of node added into the iSAM2 back-end are the landmark nodes, which also receives the information from the front-end. To include the landmark nodes

into the factor-graph, another built-in factor node in GTSAM called BearingRangeFactor is used. This factor node will connect one pose node to one landmark node based on the bearing and range of where the landmark is seen. In order to use this, the bearing and the range needs to be calculated. The range is calculated as the distance between the lidar and the keypoint, while the bearing uses the representation of a 3D point on a unit sphere to show the direction. This gives a unique bearing and range for each keypoint. Just like for the BetweenFactor, a small multivariate Gaussian noise is assumed for the BearingRangeFactor between the poses and landmarks.

Adding uncertainty for transformations matrices is not trivial and needs to be done using Lie Theory, which was introduced in Section 2.10. This will make sure that uncertainty and perturbations added to a transformation matrix does not remove it from the special Euclidean group in 3D, $SE(3)$.

After all the pose nodes and landmark nodes are added to the graph the optimizer needs to solve the nonlinear problem. In order to do this a nonlinear solver is needed, the available nonlinear solvers in GTSAM are introduced in Section 2.6. Levenberg-Marquardt is not available for iSAM2 in GTSAM, and the choice will therefore be between Gauss-Newton and Powell's Dogleg. Gauss-Newton converges faster [109], but Powell is more robust to highly nonlinear systems. As it is believed that this system is not highly nonlinear, Gauss-Newton is chosen due to the faster convergence. This optimization is then run 50 times each time iSAM2 receives new pose data from the front-end to make sure that the states are updated adequately.

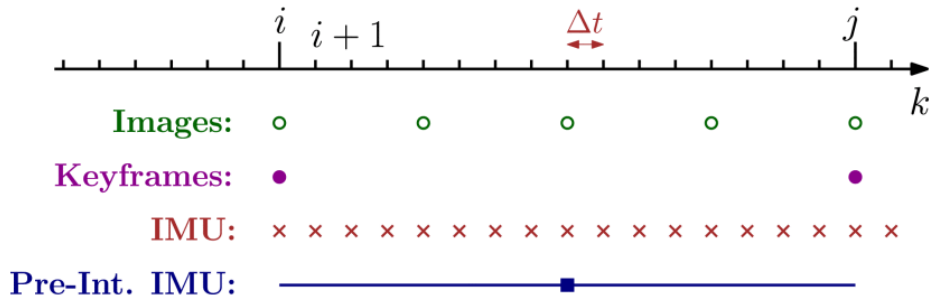
6.6 Sensor fusion IMU

When fusing in the IMU, two new variable nodes are added to the graph, the sensor bias and the velocity. When using the IMU, the initial estimates for pose and velocity will be calculated by concatenating data from the IMU with the previous pose and velocity estimates. The initial value for the sensor bias node will always be equal to the previous estimate, as the sensor bias is assumed to be constant. The IMU runs at a much higher frequency that the rest of the system, as it runs on 100 Hz. If one were to add a new pose, velocity and sensor bias node at each measurement, it would make the iSAM2 factor-graph grow very large quickly. Preintegration, as introduced in Section 2.11, is therefore used to combine many IMU measurements into a single node as seen in Figure 6.10. This preintegration is implemented in GTSAM through the built-in factor node CombinedImuFactor. This node creates factors between the pose node, velocity node, and the sensor bias node through the available information.

6.7 Fusion with GNSS

Fusing in the GNSS is less complicated than the IMU and does not require the addition of new variable nodes. This only requires a PriorFactor node, which is a built-in factor in GTSAM, this provides a prior to the pose variable nodes. GTSAM does also have a built-in GPS node, however, this only supports translation and not rotation, and using a PriorFactor node is therefore preferred as our RTK-GNSS system contains accurate information on

Figure 6.10 Different rates for IMU and lidar. These are not the exact rates used in this system, however it represents how keyframes and preintegration is used in the system. Image taken from [14].



both translation and rotation. The noise added to the PriorFactor will be a very small multivariate Gaussian noise as the RTK-GNSS is accurate down to a few centimeters.

Chapter 7

Results

7.1 Odometry

The odometry pipeline was introduced in Section 6.2 and the motion estimation is performed by RANSAC. As RANSAC is a non-deterministic algorithm, the resulting trajectory will be slightly different each time the system runs the algorithm. In order to produce the most representative results, the algorithm is run 10 times for each scenario and the results will be the average over these results. In addition to look at the RSE error, the final error will also be computed. This final error will represent how far away the algorithm is from the ground truth at the last iteration, and could be a good indication of whether or not it is possible to use it for docking.

The errors after running the system for 10 runs and averaging over the results can be seen in Table 7.1. These results show that the system is slightly less accurate than a regular GNSS receiver on average. The RSE of a regular GNSS receiver is 4 meters, which is barely lower than what the odometry does on average, with 4.6 meters for the 2D RSE. For the odometry run-through with the lowest error, the 2D RSE is only 2.9 meters, and the final 2D RSE error is only 0.5 meters. These results are however not guaranteed, and is affected by how well the ISS keypoints are extracted and matched with the RANSAC algorithm.

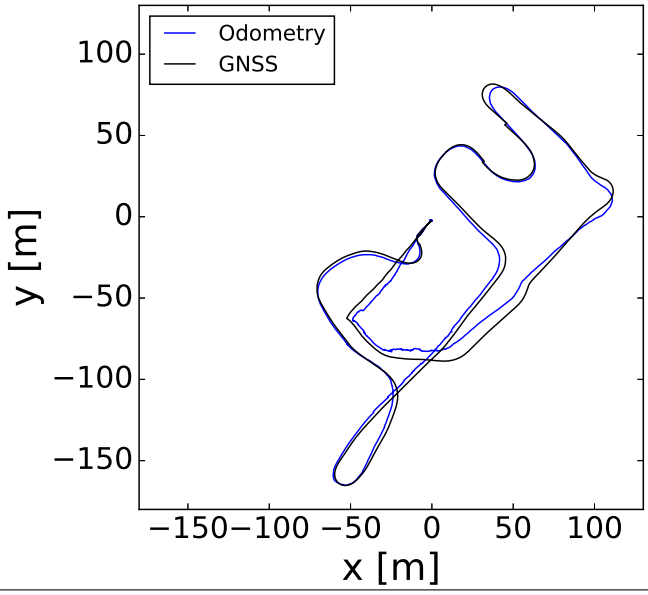
The trajectory closest to the ground truth created using the odometry algorithm is seen

	Lowest	Lowest final	Average	Average final
x error	2.2	0.4	2.9	3.3
y error	1.5	0.3	3.1	4.5
z error	22.3	35.5	27.6	53.8
2D RSE	2.9	0.5	4.6	5.8
3D RSE	22.8	35.5	28.2	54.2

Table 7.1: Odometry errors for the best run, and the average over 10 runs

in Figure 7.1, with the errors seen in Figure 7.1. From this it can be seen that the largest 2D RSE is less than 10 meters, which is the 95% confidence interval for a regular GNSS receiver, however most of the time it is below 4, which is what the RSE of a regular GNSS receiver. It can also be seen how the error in the z-direction grows much larger than the error in x- and y-direction. The total 2D RSE is less than 0.3% of the distance travelled, as the trajectory is approximately 1060 meters, and the max 2D RSE is less than 1% of the travelled distance.

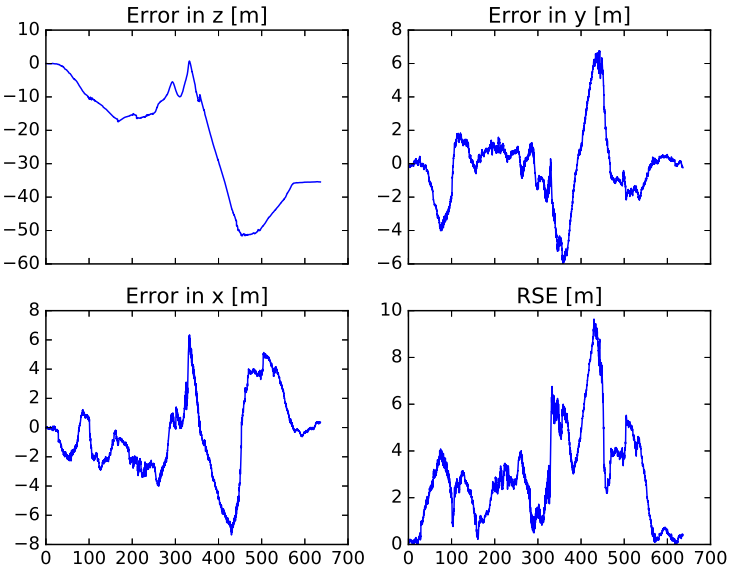
Figure 7.1 Trajectory from laser odometry.



It is obvious from the table that this odometry does not perform well for the z-direction and therefore also the 3D RSE. The error in z-direction grows very quickly and is highly inaccurate every time the odometry algorithm is run. This is often a problem when using lidars and especially ones with only 16 channels. This error is mostly due to the vertical distance between the channels. When the ASV is experiencing pitch and roll, the channels will reflect the new point either slightly above or below the previous points, this is especially true for corridor environments, and will consequently create an error in pitch, which will result in errors in the z-direction. This is especially obious between 350 and 420 seconds in Figure 7.2, where the error in z-direction is growing rapidly. For this time interval, the ASV is driving straight in a corridor-like environment, and only sees a dock on the right side. This will likely not affect the result in the xy-plane by much, however a small error in pitch will result in a large error in the z-direction over time. It will in the following sections be shown how fusing in both the IMU and the GNSS will improve the error in z-direction.

When it comes to using the odometry for docking, the final error needs to be investi-

Figure 7.2 Error in x-, y- and z-direction for the laser odometry and RSE for the xy-plane over time.

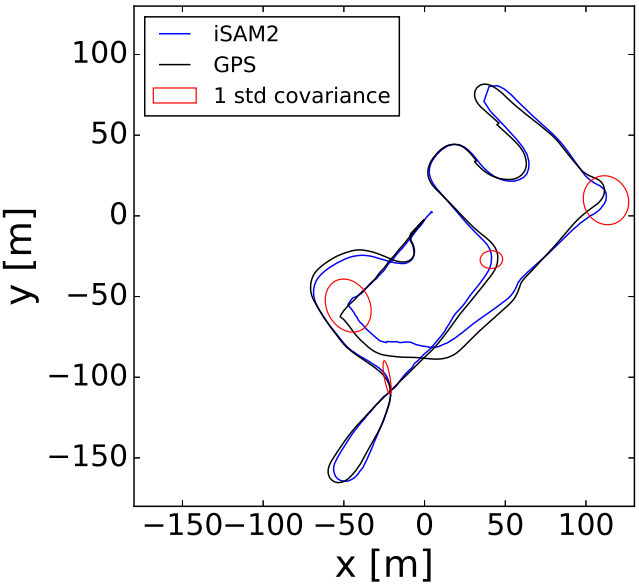


gated. For the best run, the 2D RSE is only 0.5 meters, which would likely be good enough to support a docking system, however this error is much larger on average. The average final 2D RSE is 5.8 meters. If the error in the z-direction is included as well it can quickly be seen that the 3D RSE will be way too large to even be considered for docking. That the odometry algorithm is not accurate enough to perform docking was expected, as all odometry algorithms drifts over time and it is expected that the drift over 636 seconds and 1060 meters is too large to be trusted.

The uncertainty when using only odometry is quite large as seen in Figure 7.3, where the ellipses represents one standard deviations. It can be seen that this uncertainty grows quite fast, and this is because the uncertainty is concatenated over time. There is also nothing that corrects this uncertainty, and it will therefore keep growing for the entire trajectory, and will be extremely large by the end of the dataset. This will make it so that there is no guarantee that the pose estimates are accurate, or even close to the real pose. It can be seen that the ground truth is within the covariance ellipse for each of the four randomly chosen points. This uncertainty will be reduced when fusing in GNSS, IMU and utilize loop closures.

The resulting map created using the odometry for pose estimation, reduced down to two dimensions, can be seen in Figure 7.4. It can quickly be seen that this map is quite accurate compared to the map created using the GNSS measurements as seen in Figure 6.9, and it is likely that this map can be used for both collision avoidance from the docks, path planning or other simple tasks that require a map. It is also a very useful when developing

Figure 7.3 Uncertainty for the odometry. Each red covariance ellipse represent one standard deviation.



the system to be able to visualize the keypoints that the system uses.

7.2 Odometry + GNSS

Most of the time, milliAmpere will have access to GNSS measurements, meaning that they should be used when available. Three different scenarios that could happen when milliAmpere is operating will be presented. The first scenario is if GNSS is available throughout the entire trajectory, this will be the case most of the time, as the GNSS signals are rarely jammed. The second scenario is if the GNSS is available for the first half of the trajectory and then lost, this could happen if the GNSS signals are jammed, or malfunctions. The final scenario is spurious GNSS measurements, this could be due to problems with the receiver, jamming, or the program not receiving the measurements correctly, and it will give an understanding as to how often the system needs the GNSS measurements to still be close to the ground truth.

For the first scenario, it is expected that the resulting trajectory is very close to the ground truth as GNSS is used for all measurements. This can be seen in Figure 7.5. The errors can be seen in Table 7.2, and they are as expected low. This shows that if GNSS is available, the estimates from this system will be close to the measurements from the RTK-GNSS.

For the second scenario, the GNSS measurements are lost halfway through the trajectory. When this happens, the odometry will take over from the location where the GNSS

Figure 7.4 2D map created using the state estimates from the odometry.

	Average	Final
x error	0.4	0.1
y error	0.3	0.2
z error	0.1	0.1
2D RSE	0.6	0.2
3D RSE	0.6	0.3

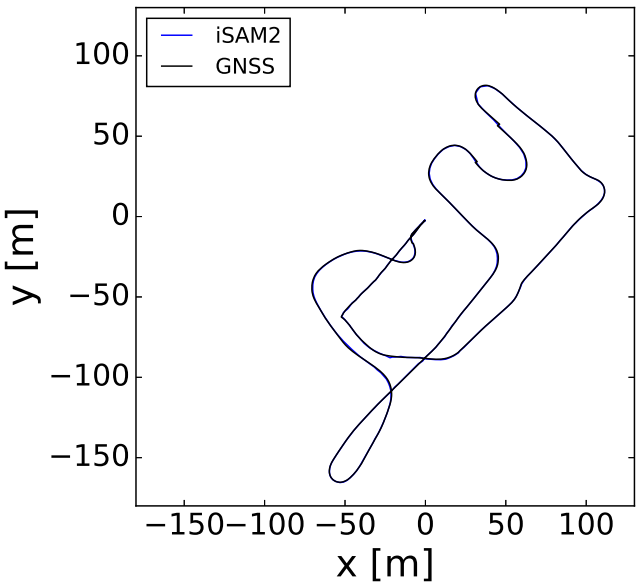
Table 7.2: Errors when GNSS is available

signal is lost, and the resulting trajectory can be seen in Figure 7.6.

The average errors after running this algorithm for 10 runs can be seen in Table 7.3. The first column are the errors for the first half, when the GNSS measurements are available, and as expected, they are very low. The second column shows the errors for the second half, where the odometry is used to estimate the poses. These grow quickly, as seen from Figure 7.7, and especially in the z-direction. Looking at the third column, showing the final error, it can be seen that using only the odometry is not accurate enough to help out a docking system, even if GNSS measurements are available for the first half of the trajectory.

For the last scenario, spurious GNSS measurements are received around every 40 seconds. The trajectory resulting from this can be seen in Figure 7.8, where the red crosses marks the locations where the GNSS signals are received. This shows how iSAM2 manages to use the odometry to fill in the missing information between the GNSS measurements with a small error, and proves that if the GNSS signals are lost for a short period of time, the odomety system will be able to act as a backup while waiting to receive the next measurement. It can however be seen from Figure 7.9 that there are some quite large

Figure 7.5 Trajectory when GNSS is available for the entire trajectory.



	First half	Second half	Final
x error	0.2	3.8	3.2
y error	0.1	2.2	1.3
z error	0.4	22.2	30.4
2D RSE	0.2	4.4	3.6
3D RSE	0.6	22.8	30.6

Table 7.3: Errors when GNSS is available for the first half of the trajectory, averaged over 10 runs

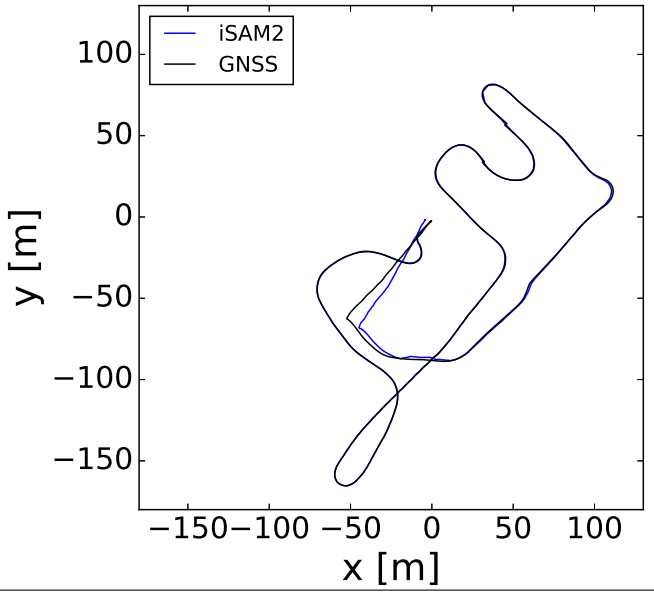
errors between the measurements, where the error in the z-direction is close to 10 meters and the 2D RSE is close to 6 meters. This shows that the system still have some flaws before IMU and loop closures are implemented.

Table 7.4 shows the errors when receiving spurious GNSS measurements. The average errors are small, and shows how well iSAM2 can extrapolate information when several measurements are available. These results also goes to show that milliAmpere does not need to receive the GNSS measurements often in order for this system to be accurate.

7.3 Odometry + IMU

In order to make the system as accurate as possible, all available sensors giving valuable information should be used. The IMU has therefore been fused into iSAM2 using prein-

Figure 7.6 Trajectory when GNSS is available for the first half.



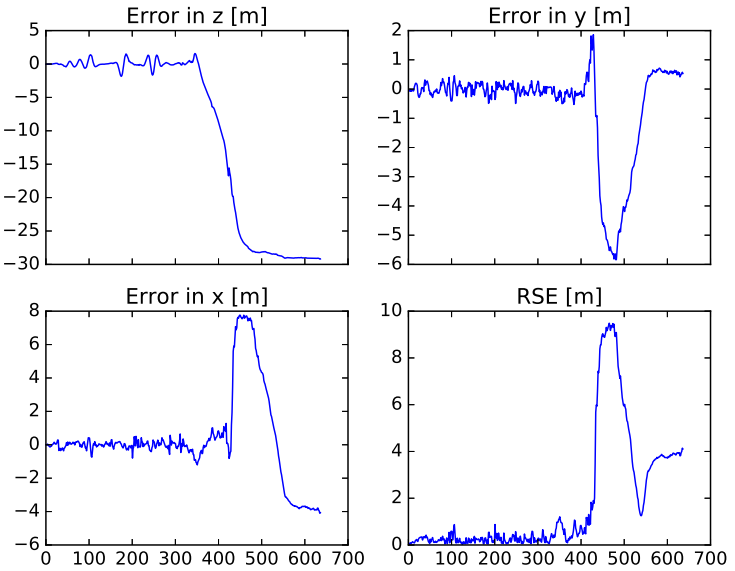
	Average	Final
x error	0.6	0.4
y error	0.6	0.2
z error	1.3	0.2
2D RSE	0.9	0.4
3D RSE	1.8	0.5

Table 7.4: Errors when GNSS is available every 40 seconds, averaged over 10 runs

tegration, and the resulting trajectory can be seen in Figure 7.10. The errors in x-,y-, and z-direction, as well as the 2D RSE can be seen in Figure 7.11, and the average errors over 10 runs can be seen in Table 7.5.

Comparing these errors to the odometry errors without IMU, seen in Figure 7.2, it can quickly be seen how the error in z-direction is decreasing at a more constant pace, suggesting a more consistent error in the pitch direction. Before the IMU was fused into the system, the error in z-direction increased very rapidly over a short period of time and behaved very unpredictable. After the IMU was fused into the system, the error in z-direction increased slowly over the entire trajectory. Fusing in IMU does not, however, improve the final error of the z-direction by much. This is mainly due to the fact that that the sensor bias will be optimized based on the available information, and since there are no loop closures or GNSS measurements available, this bias will drift alongside the system. This does however show great promise for when more information is added to the system.

Figure 7.7 Errors over time when GNSS is available for the first half of the trajectory.



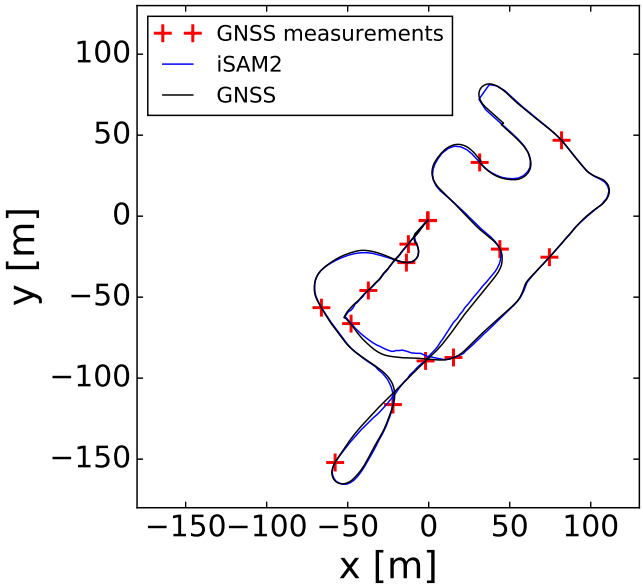
	Average	Final
x error	3.0	3.6
y error	1.8	1.4
z error	26.2	49.1
2D RSE	3.8	3.9
3D RSE	26.7	49.3

Table 7.5: Errors when IMU is fused with the odometry

Comparing Table 7.5 to Table 7.1, it can be seen that both the 2D and the 3D RSE are reduced by fusing the IMU into the system. The 2D RSE is now on average lower than for a normal GNSS receiver is specified to be. The same is true for the final error, as the average final 2D RSE is reduced to 3.9 meters. The final 3D RSE is however still very large, and the system is still not accurate enough to support any docking system.

After the IMU is fused in, the uncertainty of the system is decreased. This can be seen in Figure 7.12, where the covariance ellipses for one standard deviation can be seen for the trajectory. Comparing this to before the IMU was fused into the system, as seen in Figure 7.3, it is clear that the uncertainty is smaller, and especially late in the trajectory. Again, it can be seen that the ground truth is within the covariance matrix for all of the points. The uncertainty also at one point starts to get smaller, even when no loop closure is utilized. This might be to the fact that iSAM2 utilizes the landmarks to try to optimize for the location of walls, or it could be due to the fact that the IMU provides more accurate

Figure 7.8 Trajectory when GNSS is available on the red crosses (every 40 seconds).



readings at these locations. It is however unexpected, as it was expected that it would grow for the entire trajectory, as it does for odometry, however, at a slower pace.

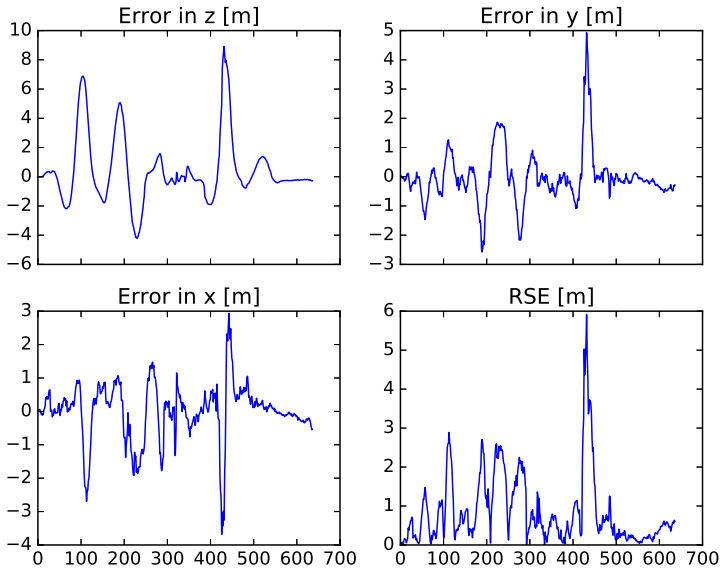
7.4 Odometry + IMU + GNSS

The next step for the system is to fuse in both IMU and the GNSS measurements at the same time. Two of the same scenarios introduced in Section 7.2 will be used, as the scenario where GNSS measurements are available all the time is not as interesting, as data from the IMU not will change that result and it will remain close to ground truth.

For the first scenario, the resulting trajectory can be seen in Figure 7.13 and the errors can be seen in Figure 7.14. The errors are obviously very low for the first half of the trajectory, as GNSS measurements are available. For the second part of the trajectory, the 2D RSE is close to the same as before the IMU was fused into the system,. The error in z-direction, on the other hand, has been greatly improved, and is reduced to almost a third of what it was before the IMU was fused into the system.

When running this system for 10 runs and averaging the errors, we get the results that can be seen in Table 7.6. Comparing this to Table 7.3, which is before the IMU is fused into the system, shows that the 2D RSE does not really improve much, however, both the average and final for the error in z-direction and the 3D RSE are reduced by approximately two thirds. This shows how the sensor bias of the IMU is optimized within the iSAM2 framework, and how this optimization improved the results.

Figure 7.9 Errors over time when GNSS is available every 40 seconds.



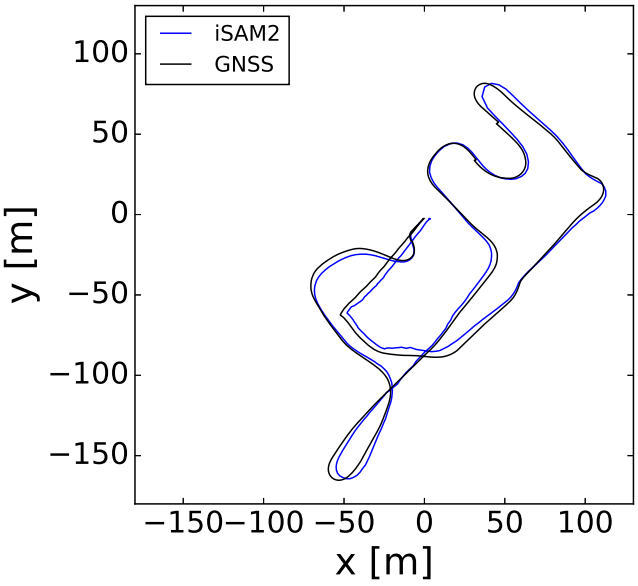
The resulting trajectory for the second scenario, can be seen in Figure 7.8. As for the first scenario, the errors in x- and y-directions are about the same as before the IMU is fused into the system, however the error in the z-direction is again greatly improved. The peaks in z-directional errors before IMU was fused into the systems were about 10 meters, and this has been reduced to only 2 meters, as seen in Figure 7.16. The 2D RSE is also small for large parts of the trajectory, however it has one spike after around 420 seconds between two GNSS measurements of almost 10 meters.

In Table 7.7 the average errors over 10 runs can be seen. This shows how the error in the z-direction is greatly improved while the errors in both the x- and y-direction actually increases by a small amount. The final errors are still very small, and could potentially be used for a docking system, especially if a GNSS measurement is received close to the

	First half	Second half	Final
x error	0.5	3.4	2.1
y error	0.4	2.2	2.9
z error	0.1	5.1	10.0
2D RSE	0.7	4.5	3.7
3D RSE	0.7	7.5	10.8

Table 7.6: Errors when GNSS is available for the first half of the trajectory and IMU is fused into the system, averaged over 10 runs.

Figure 7.10 Trajectory when IMU is fused with the odometry.



dock.

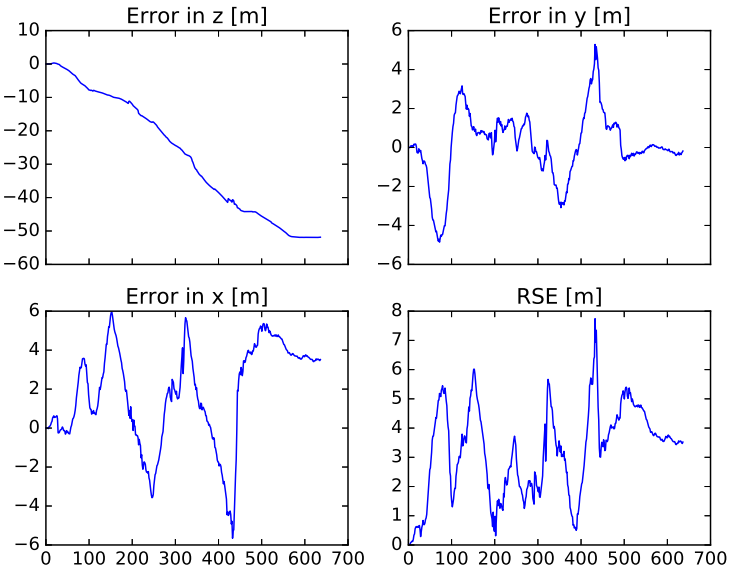
7.5 Odometry + Loop closure

Recall that in order to create a full SLAM system, loop closure is needed, as this is what separates a SLAM system from odometry. Loop closures are often the most difficult part when developing a SLAM system, as the place recognition needs to be accurate and has to take perceptual aliasing into account. When using lidars, and especially a 16-channel lidar without color information, it is often hard to recognize places, as it is unlikely for the lidar rays to hit the exact same location several times in a run, and that there is a limited amount of information around each keypoint for the descriptor to utilize.

	Average	Final
x error	1.0	0.2
y error	0.8	0.3
z error	0.2	0.1
2D RSE	1.4	0.4
3D RSE	1.4	0.4

Table 7.7: Errors when GNSS is available every 40 seconds and IMU is fused into the system, averaged over 10 runs

Figure 7.11 Errors over time when IMU is fused with the odometry.

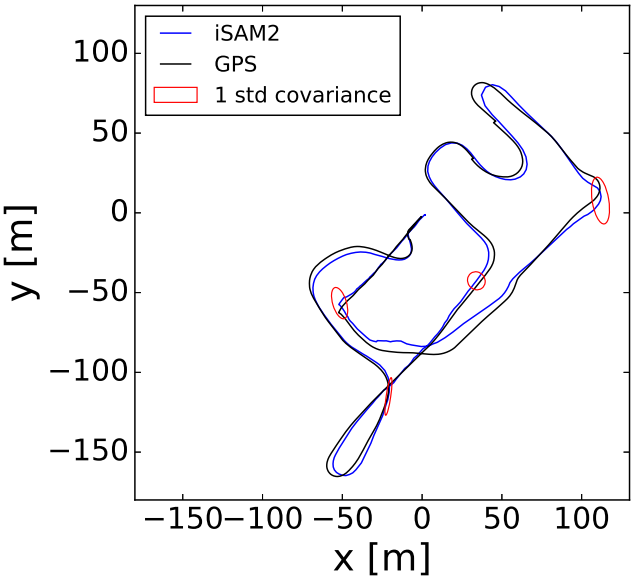


The resulting trajectory when loop closure is implemented can be seen in Figure 7.17, with the errors seen in Figure 7.18. For this run-through, long-term loop closure were found at two separate locations. The first location is at the start and end, in the docking area, this is the easiest place to achieve successful loop closure, as milliAmpere spends more time there and the points are seen at the same place as previous in the trajectory. Therefore, the descriptors does not need to be as scale- and rotation-invariant as if the keypoints were seen at different scales.

The second place a successful loop closure is achieved for this trajectory is at the beginning of the straight trajectory for when milliAmpere is on its way to the docking platform. At this location, the keypoints are seen at a larger distance than previous in the trajectory. This makes the loop closure much harder, and while this was done successfully several times, it was not achieved consistently for the system. There are also some other locations where loop closures should be possible if the descriptors values were to be good enough, however this was not achieved.

In Figure 7.19 it can be seen how the system manages to close a loop at the start and end point of the trajectory. The white points are the map created using the odometry for the pose estimates. The red points and the green points are successful loop closures matched using the algorithm implemented in the system. This shows how the algorithm manages to close the loop, even when the map is created by the odometry estimates, which are not perfect. A similar figure could be made for the second location where successful loop closure was managed, however it would be for less points, as milliAmpere does not spend much time at that location.

Figure 7.12 Uncertainty for the odometry fused with the IMU. Each red covariance ellipse represents one standard deviation.

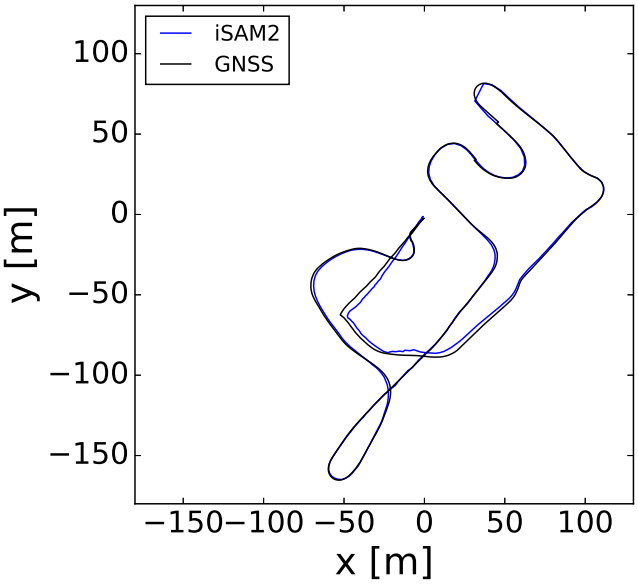


Looking at the errors in Figure 7.18 and the average errors over 10 runs in Table 7.8, it can be seen that the overall average error does not improve much. This is due to the fact that the loop was only achieved at a few locations, and that the system sometimes converged to a local minima, where the yaw angle was not correct, leading to some of the runs having a larger error. The final error is however very low, and it starts getting promising to use this system as an aid to a docking system. Looking at the 2D RSE in Figure 7.18 it can be seen that this error is less than 1 meter for the final 100 seconds of the trajectory. The average final error in x- and y-direction are less than half a meter, however the z-direction is still quite large at more than 2.5 meters. This is because the uncertainty in z-direction grows quickly for the odometry, and this problem will be reduced when the IMU is fused in alongside the loop closures.

	Average	Final
x error	2.2	0.3
y error	3.6	0.4
z error	13.0	2.6
2D RSE	4.6	0.5
3D RSE	14.1	2.7

Table 7.8: Errors when loop closure is implemented with the odometry

Figure 7.13 Trajectory when GNSS is available for the first half and IMU is fused into the system.

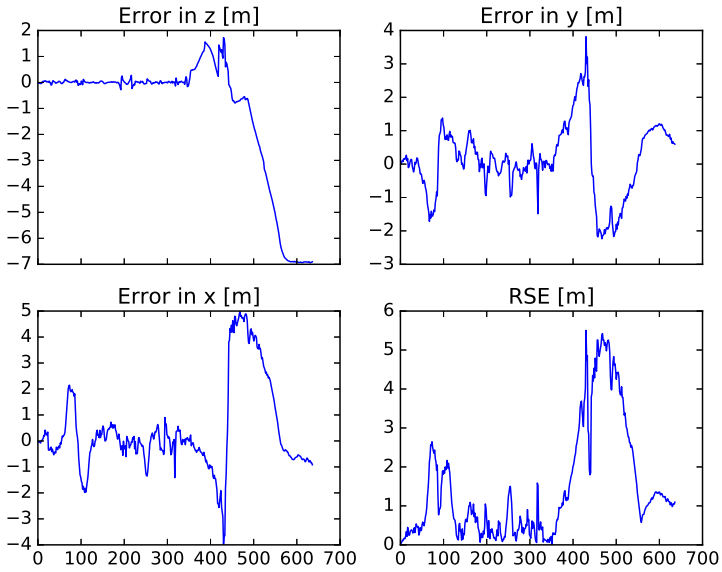


This system managed to achieve successful loop closure in the docking area for all the 10 runs, showing that the system is consistent when the keypoints are seen at the same scale as earlier in the map. This shows great promise for an ASV that is moving between two docks. As earlier mentioned, it was much harder to close the loop at the second location, and it is believed that this is due to the fact that fewer points are located inside the radius of the keypoint descriptors, and thereby causing them to be less descriptive. It could be possible to tune this in the pre-processing that is introduced in Section 6.1, but this often had a negative effect on the odometry. Due to this, and time constraints, it was believed that having quality odometry, and a loop closure that is consistent, even if it only achieved consistently in one location, shows how well this system is behaving. It is also believed that this loop closure could be done much more accurate by using a 64-channel lidar, as it would create a denser representation of the map in the z-direction, and by using a lidar with color information, as both of these upgrades would allow more information to the descriptors, providing a greater descriptive power.

7.6 Odometry + Loop closure + GNSS

The SLAM system featuring the loop closures will then fuse the GNSS measurements into the system. The only scenario that will be tested when loop closure is included is when the GNSS signal is lost halfway through the trajectory. Using spurious GNSS measurements

Figure 7.14 Errors over time when GNSS is available for the first half of the trajectory and IMU is fused into the system.



would not show how the loop closure is contributing to the system, as the measurements received by the RTK-GNSS are more accurate than the loop closures. The resulting trajectory can be seen in Figure 7.20 and the errors for this trajectory can be seen in Figure 7.21.

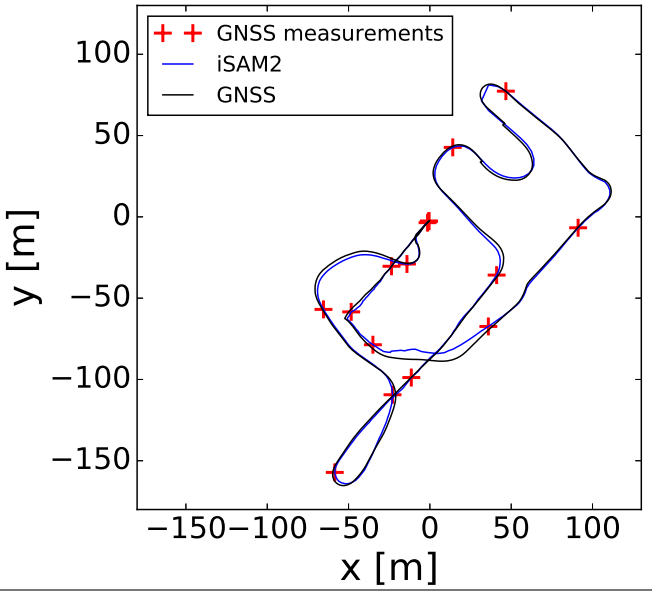
Looking at the average errors over 10 runs, that can be seen in Table 7.9, it is clear that the results are greatly improved compared to when the loop closure was not a part of the system. The error for the first half is still very low, as the GNSS measurements are available, and the errors in x- and y-direction for the second half of the trajectory are almost halved. The errors in the z-direction are still quite large, and peaks at almost 14 meters, causing a large 3D RSE.

The final errors are again very close to the ground truth, as would be expected when successful loop closure is performed. As the GNSS data is available, the system does not converge to a local minima anymore, which caused an error in the yaw angle in the previous section. This shows how well iSAM2 manages to optimize the system once more information is available, and shows how important it is in this system to use all the available information.

7.7 Odometry + IMU + Loop closure

The next step for the SLAM system is then to include both loop closures and fuse in the IMU at the same time. The resulting trajectory can be seen in Figure 7.22, while the errors

Figure 7.15 Trajectory when GNSS is available on the red crosses (every 40 seconds) and IMU is fused into the system.



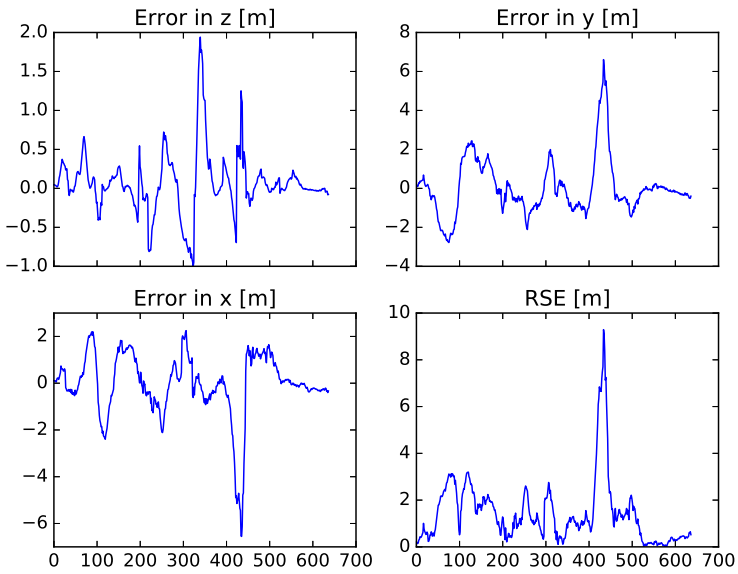
for this trajectory can be seen in Figure 7.23. From visual inspection, it can quickly be seen that the SLAM system provides a good pose estimate, and that this pose estimate is never far off the ground truth.

In Table 7.10 the average errors from 10 runs can be seen. In addition to having the average errors and the final errors, the largest errors in all direction are also included to show how consistent this system is. Over the 10 runs, the worst error was barely larger than the average error, and the final error is still very small, as loop closure is utilized. Again, the system managed to close the loop for all 10 runs, and with the additional information provided by the IMU, iSAM2 manages to provide very accurate estimates for the poses. This shows that in addition to manage almost half the error of a regular GNSS receiver

	First half	Second half	Final
x error	0.4	2.0	0.4
y error	0.4	1.1	0.7
z error	0.1	9.0	1.9
2D RSE	0.6	2.5	0.8
3D RSE	0.6	9.4	2.1

Table 7.9: Errors when loop closure is implemented with the odometry and GNSS measurements are received for the first half of the trajectory

Figure 7.16 Errors over time when GNSS is available every 40 seconds and IMU is fused into the system.



in the two dimensional plane, the system is now also more accurate than a regular GNSS receiver in the z-direction, as a regular GNSS receiver has an RSE in the z-direction of about 6 meters. The 3D RSE is now only 3.4 meters, which is far more accurate than the odometry system managed in 2D, even when IMU was fused into the system.

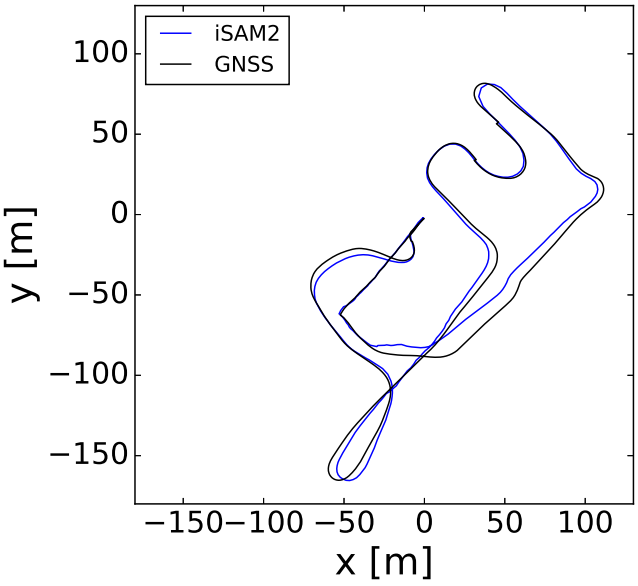
This system could now potentially be used to support a docking algorithm. Having a small 3D RSE of only 1.5 meters near the docking area goes to prove that the system is accurate enough to at least be able to assist a docking algorithm with its pose estimate. In addition to providing accurate pose estimates for a docking algorithm, the map created by the system, as seen in Figure 7.24, could be used as well.

When comparing the map created by the SLAM system in Figure 7.24 to the map

	Average	Worst	Final
x error	2.0	2.1	0.8
y error	1.3	1.6	0.4
z error	1.7	2.3	1.2
2D RSE	2.6	2.8	0.9
3D RSE	3.4	3.7	1.5

Table 7.10: Errors when loop closure is implemented with the odometry and IMU is fused into the system

Figure 7.17 Trajectory when loop closure is implemented with the odometry.



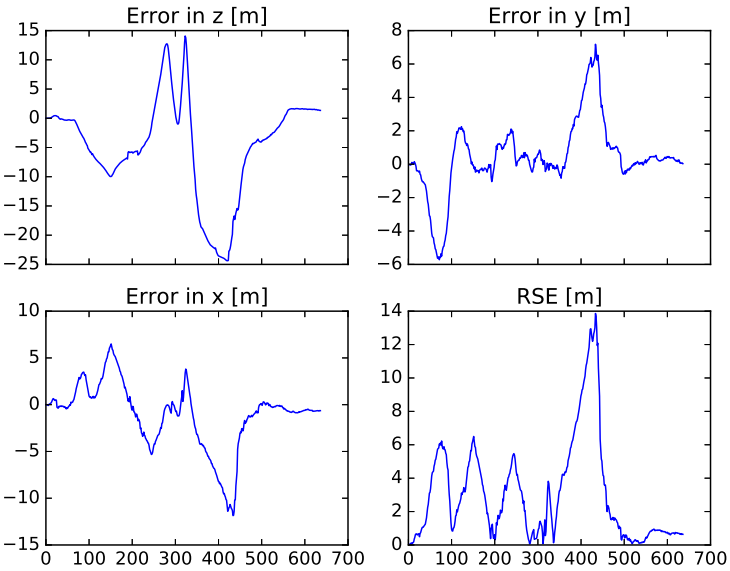
created by the odometry, as seen in Figure 7.4, it can quickly be seen how it is improved. First of all, it can be seen that the map created around the dock is much more accurate, as this area for the odometry has drifted by quite a bit. The map created by the odometry also creates three different walls on the left side of the map, due to the drift, and this is fixed by the SLAM system, where it can clearly be seen that there are only two walls. Overall, this map is very similar to the ground truth, as seen in Figure 6.9, which is expected when the errors are so small.

The uncertainty of the SLAM system can be seen in Figure 7.25, where the red circles are the covariance ellipses for one standard deviation. For the odometry, it was shown how the uncertainty grew over the full trajectory, which is expected when it is not corrected by loop closures. This uncertainty grew so big that the system could not guarantee anything, and could therefore not be trusted as a back-up system. The SLAM system, on the other hand, has a much smaller uncertainty, and the information it provides is therefore much more reliable. This small uncertainty also seem correct, as the ground truth is within one standard deviation of the trajectory for all the four randomly chosen covariance ellipses on the trajectory.

7.8 Odometry + IMU + Loop closure + GNSS

Now, the full SLAM system will be implemented, utilizing loop closures and fusing in both the IMU and GNSS measurements. The GNSS measurements will be available for

Figure 7.18 Errors over time when loop closure is implemented with the odometry



the first half of the trajectory, before they are lost as is done in the previous sections. The resulting trajectory can be seen in Figure 7.26, and the corresponding errors can be seen in Figure 7.27.

Table 7.11 shows the errors averaged over 10 runs. Again it can be seen that the system is accurate, both for the first half, second half and for the final error. This shows the potential of how accurate the system developed in this thesis can be, and how well it could act as a back-up system if the GNSS signal is lost. This also goes to show, as before the GNSS was fused in, that the system could be used as an an aid for a docking system. The average 3D RSE for the second half of the trajectory is only 2.5 meters, proving that the system is never far off the ground truth.

	First half	Second half	Final
x error	0.4	1.8	0.8
y error	0.4	1.1	0.4
z error	0.1	0.8	0.8
2D RSE	0.6	2.3	0.9
3D RSE	0.6	2.5	1.2

Table 7.11: Errors when loop closure is implemented with the odometry, fused with IMU, and GNSS measurements are received for the first half of the trajectory

Figure 7.19 Example of loop closure performed by the system.

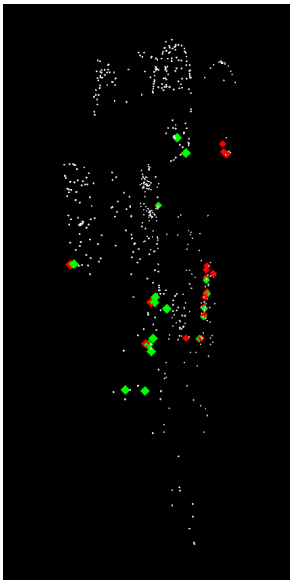


Figure 7.20 Trajectory when loop closure is implemented with the odometry and GNSS measurements are received for the first half.

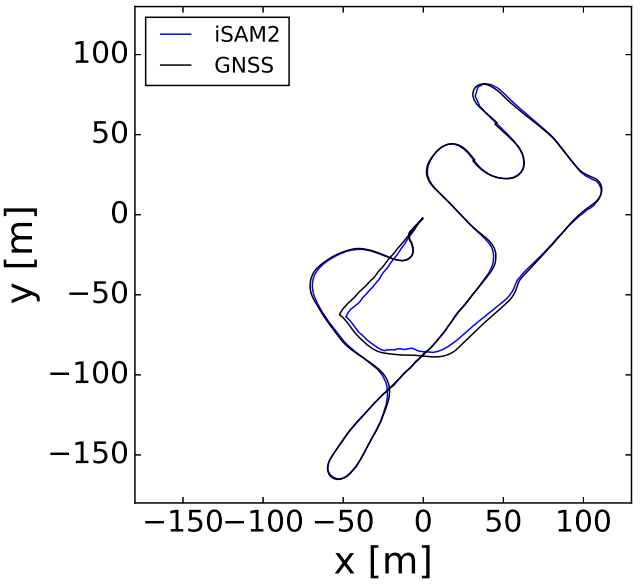


Figure 7.21 Errors over time when loop closure is implemented with the odometry and GNSS measurements are received for the first half of the trajectory.

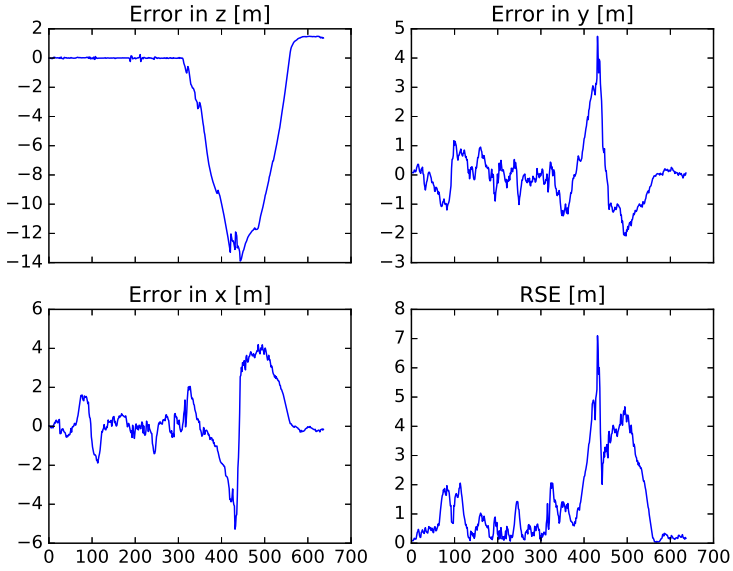


Figure 7.22 Trajectory when loop closure is implemented with the odometry and IMU is fused into the system.

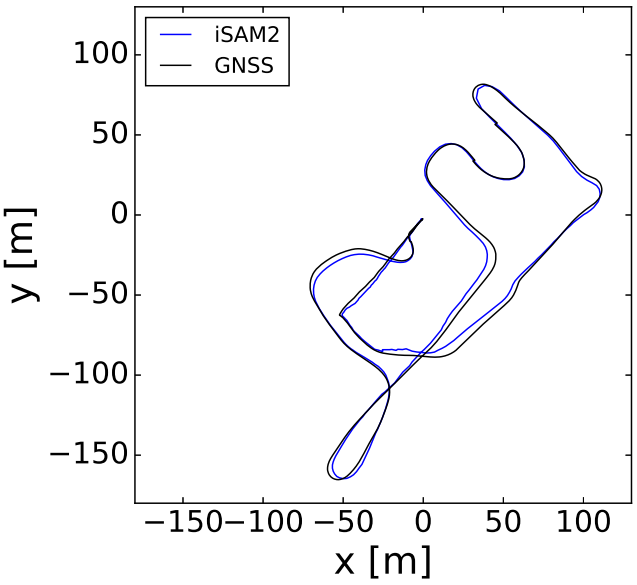


Figure 7.23 Errors over time when loop closure is implemented with the odometry and IMU is fused into the system.

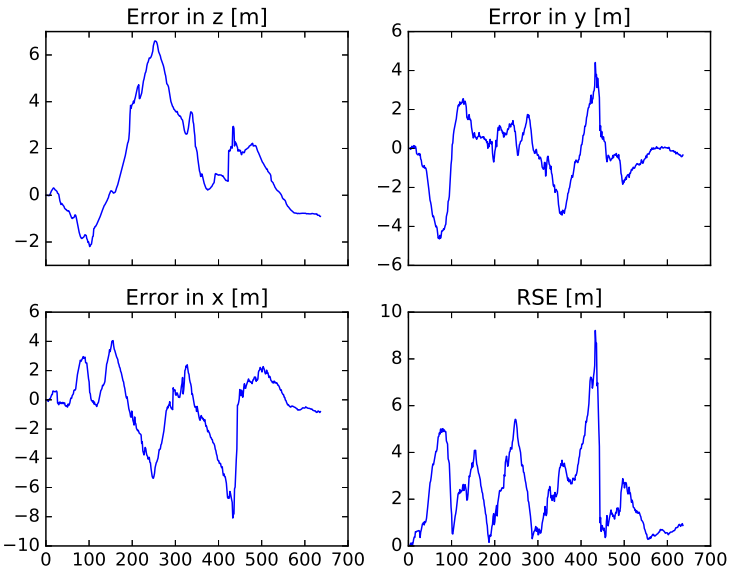


Figure 7.24 Map created using the SLAM system.

Figure 7.25 Uncertainty for the SLAM system when IMU fused into the system, and loop closures are utilized.

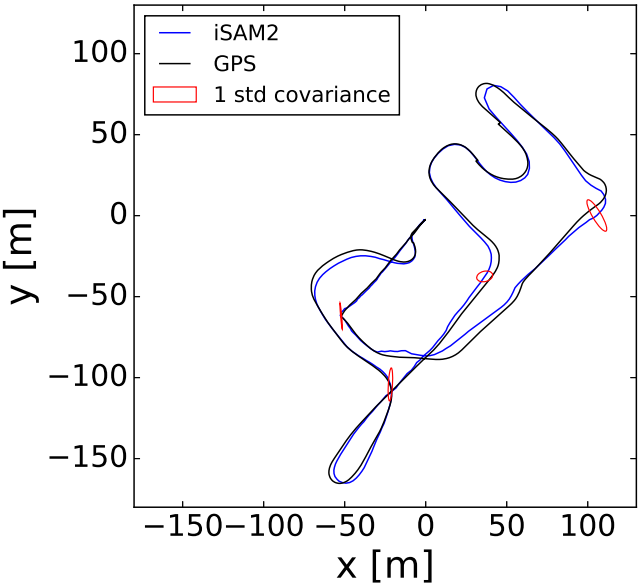


Figure 7.26 Trajectory when loop closure is implemented with the odometry, fused with IMU, and GNSS measurements are received for the first half.

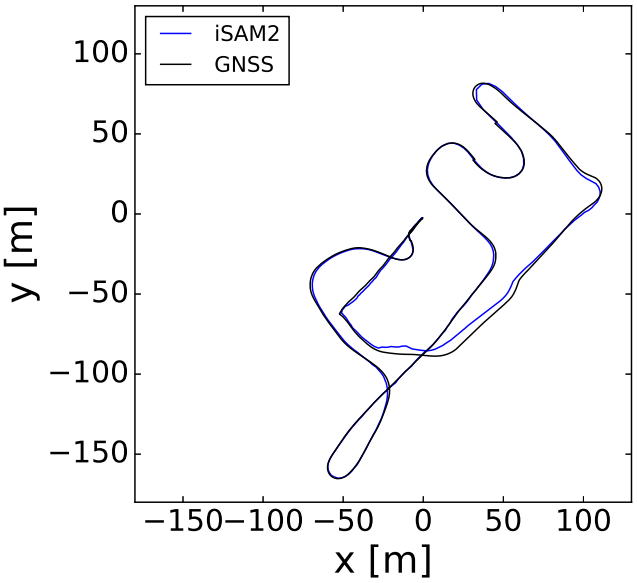
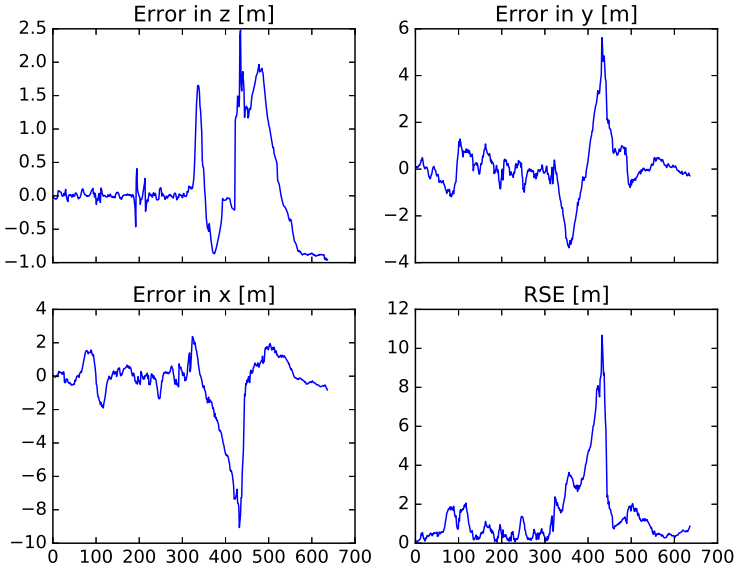


Figure 7.27 Errors over time when loop closure is implemented with the odometry, fused with IMU, and GNSS measurements are received for the first half of the trajectory.



Chapter 8

Conclusion and future work

In this thesis, a successful feature-based lidar SLAM system has been developed. It has proved to be both consistent and more accurate than a regular GNSS receiver, both for the two dimensional error in the x- and y-directions, and for the z-directional error. Comparing these results to what is achieved by Hector-SLAM, LOAM and BLAM [1], shows how this system is a clear improvement. It is also a clear improvement to localization using UKF and ICP [2], and could now potentially be used as a back-up pose estimator for milliAmpere. In addition, it is believed that the results can be further improved by tuning the parameters available in both the front-end and back-end of the SLAM system.

This system might also be one of the first 3D lidar SLAM systems which utilizes descriptors and keypoints for loop closures. Both LeGO-LOAM, introduced in Section 3.1.3, and Google Cartographer, introduced in Section 3.1.4, utilizes scan-matching to perform the loop closures, and the author of this thesis has yet to find any other state-of-the-art lidar SLAM systems that closes loops by using keypoints and descriptors. This could be due to several reasons, among them how hard it is to make the system scale-invariant and that the location of the keypoints might differ between the scans as the rays do not hit the exact same locations each time.

The loop closure is not perfect yet, and there are several ways to improve this. As mentioned earlier, it would most likely be improved by upgrading the lidar, as it could provide color information for the descriptors, and could also provide a denser point cloud with more information, however this could be very expensive, as a high-grade lidar can cost close to 1 MNOK. Another option could be to use the odometry information from the lidar and combine it with loop closure performed by a camera. Place recognition for cameras is a mature research field [110], and it could therefore benefit the system greatly to use cameras for loop closure [111], [112]. It should be possible to use the same factor graph framework with iSAM2 that is created in this thesis to implement this. This shows one of the many strengths of using the iSAM2 framework to perform SLAM. It should also be possible to utilize some of the other factor graph frameworks introduced in Section 2.5.

There are also several things that can be added to this system. As of now, the back-end does not send the information back to the front-end, due to a bug in C++ for Ubuntu

16.04 where it was not possible to include PCL and GTSAM in the same file. Poses and landmarks in the front-end are therefore not updated as more information is fused into the system, or when loop closures are performed. Adding this to the system should further improve the accuracy and consistency of the loop closures.

Another thing that can be added to the system is an adaptive optimization for iSAM2, by changing between Gauss-Newton and Powell's Dogleg method based on a convexity test. If the system is convex, Gauss-Newton could be used, and if not, Powell's Dogleg could would be used. This would make it so that the system would not diverge if the problem was to lose the convexity property, and should make the system more reliable.

An option to creating a full SLAM system would be to only use localization in a known map. SLAM is often used in unknown environments, while milliAmpere will be operating in a known environment. It should be possible to create a map of the surroundings before it is operating and then localize milliAmpere in that stored map. This has been successfully done for autonomous cars [113], [114], and should be possible to extend to autonomous ferries.

This system by itself should also not be trusted in order to perform docking, as it can have errors or diverge, however it could be used as an aid for a visual-based docking system, and should be able to assist with both direction and distances for when a visual-based docking system should take over. There are several ways to do this, for example by creating some known markers on the dock, this has been done for underwater vehicles [115], [116].

Due to the Covid-19 situation and the time constraints caused by this, it was unfortunately not possible to record more data to further test the consistency of the system for a new dataset. Due to the same reason it was not possible to test the system in real time on milliAmpere. Testing on several datasets would show how the results are affected by tuning and optimization of the current dataset, and how well parameters and methods would work for similar scenarios. It would also be interesting to record data in the area where the finalized auto-ferry is proposed to operate, as this would give a good indication as to how good this system is for that location.

Bibliography

[1] M. S. Ødven, "Lidar-based slam for autonomous ferry," msc thesis, Norwegian University of Science and Technology, Jan 2019.

[2] N. Dalhaug, "Lidar-based localization for autonomous ferry," msc thesis, Norwegian University of Science and Technology, June 2019.

[3] E. Skjellaug, "Feature-based laser odometry for autonomous ferry," specialization thesis, Norwegian University of Science and Technology, Dec. 2019.

[4] E. Brekke, "The multivariate gaussian," in *Fundamentals of Sensor Fusion*, ch. 3, pp. 37–46, 2020.

[5] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. Leonard, and F. Dellaert, "isam2: Incremental smoothing and mapping with fluid relinearization and incremental variable reordering," in *proceedings of IEEE International Conference on Robotics and Automation, Shanghai, China*, pp. 3281–3288, May 2011.

[6] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age," *IEEE Transactions on Robotics*, vol. 32, pp. 1309–1332, Dec 2016.

[7] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "Orb-slam: A versatile and accurate monocular slam system," *IEEE Transactions on Robotics*, vol. 31, pp. 1147–1163, Oct 2015.

[8] R. A. Newcombe, S. J. Lovegrove, and A. J. Davison, "Dtam: Dense tracking and mapping in real-time," in *proceedings of International Conference on Computer Vision, Barcelona, Spain*, pp. 2320–2327, Nov 2011.

[9] K. Madsen, H. Nielsen, and O. Tingleff, "Methods for non-linear least squares problems (2nd ed.)," *Technical University of Denmark*, p. 60, Jan 2004.

[10] J. Engel, T. Schöps, and D. Cremers, "Lsd-slam: Large-scale direct monocular slam," in *proceedings of European Conference on Computer Vision, Zürich, Switzerland*, pp. 834–849, Sep 2014.

[11] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. J. Leonard, and F. Dellaert, "isam2: Incremental smoothing and mapping using the bayes tree," *The International Journal of Robotics Research*, vol. 31, pp. 217–236, Feb 2012.

[12] F. Tombari, S. Salti, and L. Di Stefano, "Unique signatures of histograms for local surface description," in *proceedings of European Conference on Computer Vision, Crete, Greece*, pp. 356–369, Sep 2010.

[13] Velodyne LiDAR, Inc, *Velodyne LiDAR Puck Datasheet*, 2018. 63-9229 Rev-H.

[14] C. Forster, L. Carlone, F. Dellaert, and D. Scaramuzza, "On-manifold preintegration for real-time visual–inertial odometry," *IEEE Transactions on Robotics*, vol. 33, pp. 1–21, Aug 2017.

[15] J. E. Manley, "Unmanned surface vehicles, 15 years of development," in *OCEANS, Quebec City, QC, Canada*, pp. 1–4, Sep. 2008.

[16] Z. Liu, Y. Zhang, X. Yu, and C. Yuan, "Unmanned surface vehicles: An overview of developments and challenges," *Annual Reviews in Control*, vol. 41, pp. 71 – 93, 2016.

[17] A. Golden and R. Weisbrod, "Trends, causal analysis, and recommendations from 14 years of ferry accidents," *Journal of Public Transportation*, vol. 19, pp. 17–27, Mar 2016.

[18] K. L. Nilsen, "Førerløs framtid for fløtmann-båten," *Trondheim24*, June 10th 2017.

[19] P. J. Besl and N. D. McKay, "A method for registration of 3-d shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, pp. 239–256, Feb 1992.

[20] M. Magnusson, *The Three-Dimensional Normal-Distributions Transform – an Efficient Representation for Registration, Surface Analysis, and Loop Detection*. PhD thesis, Örebro University, Dec 2009.

[21] S. Kohlbrecher, O. von Stryk, J. Meyer, and U. Klingauf, "A flexible and scalable slam system with full 3d motion estimation," in *proceedings of IEEE International Symposium on Safety, Security, and Rescue Robotics, Kyoto, Japan*, pp. 155–160, Nov 2011.

[22] J. Zhang and S. Singh, "Loam: Lidar odometry and mapping in real-time," in *proceedings of Robotics: Science and Systems, Berkeley, CA, USA*, pp. 109–111, July 2014.

[23] T. Shan and B. Englot, "Lego-loam: Lightweight and ground-optimized lidar odometry and mapping on variable terrain," in *proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems, Madrid, Spain*, pp. 4758–4765, Oct 2018.

[24] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-time loop closure in 2d lidar slam," in *proceedings of IEEE International Conference on Robotics and Automation, Stockholm, Sweden*, pp. 1271–1278, May 2016.

[25] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *International Journal of Robotics Research (IJRR)*, 2013.

[26] F. R. Kschischang, B. J. Frey, and H. . Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Transactions on Information Theory*, vol. 47, pp. 498–519, Feb 2001.

[27] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "G2o: A general framework for graph optimization," in *proceedings of IEEE International Conference on Robotics and Automation, Shanghai, China*, pp. 3607–3613, May 2011.

[28] S. Lange, N. Sünderhauf, and P. Protzel, "Incremental smoothing vs. filtering for sensor fusion on an indoor uav," in *of proceedings of IEEE International Conference on Robotics and Automation, Karlsruhe, Germany*, pp. 1773–1778, May 2013.

[29] J. Neira and J. D. Tardós, "Data association in stochastic mapping using the joint compatibility test," *IEEE Transactions on Robotics and Automation*, vol. 17, pp. 890–897, Dec 2001.

[30] D. Bertsekas, "A new algorithm for the assignment problem," *Mathematical Programming*, vol. 21, pp. 152–171, Dec 1981.

[31] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval Research Logistics Quarterly*, vol. 2, pp. 83–97, Mar 1955.

[32] M. Montemerlo and S. Thrun, "Simultaneous localization and mapping with unknown data association using fastslam," in *proceedings of IEEE International Conference on Robotics and Automation, Taipei, Taiwan*, pp. 1985–1991, Sept 2003.

[33] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, pp. 381–395, June 1981.

[34] E. Baser, V. Balasubramanian, P. Bhattacharyya, and K. Czarnecki, "Fantrack: 3d multi-object tracking with feature association network," in *proceedings of IEEE Intelligent Vehicles Symposium, Paris, France*, pp. 1426–1433, June 2019.

[35] P. Ondruska and I. Posner, "Deep tracking: Seeing beyond seeing using recurrent neural networks," in *proceedings of AAAI Conference on Artificial Intelligence, Phoenix, AZ, USA*, pp. 3361–3367, Feb 2016.

[36] K. Yoon, D. Y. Kim, Y.-C. Yoon, and M. Jeon, "Data association for multi-object tracking via deep neural networks," *Sensors*, vol. 19, p. 559, Feb 2019.

[37] E. Nelson, "B(erkeley) l(ocalization) a(nd) m(apping)."

[38] S. Filipe and L. A. Alexandre, "A comparative evaluation of 3d keypoint detectors in a rgb-d object dataset," in *proceedings of International Conference on Computer Vision Theory and Applications, Lisbon, Portugal*, pp. 476–483, Jan 2014.

[39] F. Tombari, S. Salti, and L. Di Stefano, "A performance evaluation of 3d keypoint detectors," in *proceedings of International Conference on 3D Imaging, Modeling, Processing, Visualization and Transmission, Hangzhou, China*, pp. 236–243, May 2011.

[40] L. A. Alexandre, "3d descriptors for object and category recognition: a comparative evaluation," in *Workshop on Color-Depth Camera Fusion in Robotics at the IEEE/RSJ International Conference on Intelligent Robots and Systems, Vilamoura, Portugal*, Oct 2012.

[41] X.-F. Han, J. S. Jin, J. Xie, M.-J. Wang, and W. Jiang, "A comprehensive review of 3d point cloud descriptors," *ArXiv*, vol. abs/1802.02297, Feb 2018.

[42] Y. Zhong, "Intrinsic shape signatures: A shape descriptor for 3d object recognition," in *proceedings of International Conference on Computer Vision Workshops, Kyoto, Japan*, pp. 689–696, Sep. 2009.

[43] B. Steder, R. B. Rusu, K. Konolige, and W. Burgard, "Point feature extraction on 3d range scans taking into account object boundaries," in *proceedings of IEEE International Conference on Robotics and Automation, Shanghai, China*, pp. 2601–2608, May 2011.

[44] I. Sipiran and B. Bustos, "Harris 3d: a robust extension of the harris operator for interest point detection on 3d meshes," *The Visual Computer*, vol. 27, p. 963, July 2011.

[45] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, pp. 91–110, Nov 2004.

[46] R. B. Rusu, N. Blodow, Z. C. Marton, and M. Beetz, "Aligning point cloud views using persistent feature histograms," in *proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems, Nice, France*, pp. 3384–3391, Sep. 2008.

[47] R. B. Rusu, N. Blodow, and M. Beetz, "Fast point feature histograms (fpfh) for 3d registration," in *proceedings of IEEE International Conference on Robotics and Automation, Kobe, Japan*, pp. 3212–3217, May 2009.

[48] A. Frome, D. Huber, R. Kolluri, T. Bülow, and J. Malik, "proceedings of recognizing objects in range data using regional point descriptors," in *European Conference on Computer Vision, Prague, Czech Republic*, pp. 1–14, May 2004.

[49] F. Tombari, S. Salti, and L. Di Stefano, "Unique shape context for 3d data description," in *proceedings of ACM Workshop on 3-D Object Retrieval, Firenze, Italy*, Oct 2010.

[50] E. Skjellaug, E. Brekke, and A. Stahl, "Feature-based laser odometry for autonomous surface vehicles utilizing the point cloud library," in *proceedings of Fusion, virtual conference*, July 2020.

[51] M. O. A. Aqel, M. H. Marhaban, M. I. Saripan, and N. B. Ismail, "Review of visual odometry: types, approaches, challenges, and applications," *SpringerPlus*, vol. 5, pp. 1897–1924, Dec 2016.

[52] D. M. Helmick, Yang Cheng, D. S. Clouse, L. H. Matthies, and S. I. Roumeliotis, "Path following using visual odometry for a mars rover in high-slip environments," in *proceedings of IEEE Aerospace Conference, Big Sky, MT, USA*, pp. 772–789, Mar 2004.

[53] D. Fernandez and A. Price, "Visual odometry for an outdoor mobile robot," in *proceedings of IEEE Conference on Robotics, Automation and Mechatronics, Singapore, Singapore*, vol. 2, pp. 816–821, Dec 2004.

[54] P. Biber and W. Straßer, "The normal distributions transform: A new approach to laser scan matching," in *proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems, Las Vegas, NV, USA*, vol. 3, pp. 2743 – 2748 vol.3, Nov 2003.

[55] P. C. Mahalanobis, "On the generalised distance in statistics," in *proceedings of the National Institute of Sciences of India*, vol. 2, pp. 49–55, 1936.

[56] F. Dellaert and M. Kaess, "Factor graphs for robot perception," *Foundations and Trends in Robotics*, vol. 6, pp. 1–139, Aug 2017.

[57] B. Williams, M. Cummins, J. Neira, P. Newman, I. Reid, and J. D. Tardós, "A comparison of loop closing techniques in monocular slam," *Robotics and Autonomous Systems*, vol. 57, pp. 1188–1197, Dec 2009.

[58] P. Lajoie, S. Hu, G. Beltrame, and L. Carlone, "Modeling perceptual aliasing in slam via discrete–continuous graphical models," *IEEE Robotics and Automation Letters*, vol. 4, pp. 1232–1239, Jan 2019.

[59] M. Cummins and P. Newman, "Fab-map: Probabilistic localization and mapping in the space of appearance," *The International Journal of Robotics Research*, vol. 27, pp. 647–665, June 2008.

[60] S. Saeedi, M. S. M. Trentini, and H. Li, "Multiple-robot simultaneous localization and mapping: A review," *Journal of Field Robotics*, vol. 33, pp. 3–46, Sept. 2016.

[61] C. Forster, S. Lynen, L. Kneip, and D. Scaramuzza, "Collaborative monocular slam with multiple micro aerial vehicles," in *proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems, Tokyo, Japan*, pp. 3962–3970, Nov 2013.

[62] G. Loianno, Y. Mulgaonkar, C. Brunner, D. Ahuja, A. Ramanandan, M. Chari, S. Diaz, and V. Kumar, "A swarm of flying smartphones," in *proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems, Daejeon, South Korea*, pp. 1681–1688, Oct 2016.

[63] T. Cieslewski, S. Choudhary, and D. Scaramuzza, "Data-efficient decentralized visual slam," in *proceedings of IEEE International Conference on Robotics and Automation, Brisbane, QLD, Australia*, pp. 2466–2473, May 2018.

[64] C. Bibby and I. Reid, "Simultaneous localisation and mapping in dynamic environments (slamide) with reversible data association," in *proceedings of Robotics: Science and Systems Conference, Atlanta, GA, USA*, pp. 105–112, June 2007.

[65] C. C. Wang, C. Thorpe, S. Thrun, M. Herbert, and H. F. Durrant-Whyte, "Simultaneous localization, mapping and moving object tracking," *International Journal of Robotics Research*, vol. 26, pp. 889–916, Sept 2007.

[66] C. S. Lee, D. E. Clark, and J. Salvi, "Slam with dynamic targets via single-cluster phd filtering," *IEEE Journal of Selected Topics in Signal Processing*, vol. 7, pp. 543–552, Mar 2013.

[67] B. Bescos, J. M. Fácil, J. Civera, and J. Neira, "Dynaslam: Tracking, mapping, and inpainting in dynamic scenes," *IEEE Robotics and Automation Letters*, vol. 3, pp. 4076–4083, July 2018.

[68] K. Konolige, G. Grisetti, R. Kümmerle, W. Burgard, B. Limketkai, and R. Vincent, "Efficient sparse pose adjustment for 2d mapping," in *proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems, Taipei, Taiwan*, pp. 22–29, Oct 2010.

[69] G. Grisetti, R. Kümmerle, C. Stachniss, U. Frese, and C. Hertzberg, "Hierarchical optimization on manifolds for online 2d and 3d mapping, anchorage, ak, usa," in *2010 IEEE International Conference on Robotics and Automation*, pp. 273–278, May 2010.

[70] S. Agarwal, K. Mierle, and Others, "Ceres solver." `http://ceres-solver.org`.

[71] F. Dellaert, "Factor graphs and gtsam: A hands-on introduction," Tech. Rep. GT-RIM-CP&R-2012-002, Georgia Institute of Technology, Sept 2012.

[72] M. Kaess, V. Ila, R. Roberts, and F. Dellaert, "The bayes tree: An algorithmic foundation for probabilistic robot mapping," in *proceedings in International Workshop on the Algorithmic Foundations of Robotics, Singapore, Singapore*, pp. 157–173, Jan 2010.

[73] J. Engel, J. Sturm, and D. Cremers, "Semi-dense visual odometry for a monocular camera," in *proceedings of IEEE International Conference on Computer Vision*, pp. 1449–1456, Dec 2013.

[74] L. Polok, V. Ila, M. Solony, P. Smrz, and P. Zemcik, "Incremental block cholesky factorization for nonlinear least squares in robotics," in *proceedings of Robotics: Science and Systems Confernce, Berlin, Germany*, pp. 328–336, June 2013.

[75] K. Levenberg, "A method for the solution of certain nonlinear problems in least squares," *Quarterly of Applied Mathematics*, vol. 2, pp. 164–168, July 1944.

[76] M. Powell, "A new algorithm for unconstrained optimization," in *proceedings of Nonlinear Programming, Madison, WI, USA*, pp. 31–65, May 1970.

[77] R. S. Dembo, S. C. Eisenstat, and T. Steihaug, "Inexact newton methods," *SIAM Journal on Numerical Analysis*, vol. 19, pp. 400–408, Mar 1982.

[78] M. L. A. Lourakis and A. A. Argyros, "Is levenberg-marquardt the most efficient optimization algorithm for implementing bundle adjustment?, beijing, china," in *proceedings of Tenth IEEE International Conference on Computer Vision*, vol. 1, pp. 1526–1531, Oct 2005.

[79] J. Kocić, N. Jovičić, and V. Drndarević, "Sensors and sensor fusion in autonomous vehicles," in *proceedings of Telecommunications Forum, Belgrade, Serbia*, pp. 420–425, Nov 2018.

[80] X. Meng, H. Wang, and B. Liu, "A robust vehicle localization approach based on gnss/imu/dmi/lidar sensor fusion for autonomous vehicles," *Sensors*, vol. 17, p. 2193, Sept 2017.

[81] V. Ilci and C. Toth, "High definition 3d map creation using gnss/imu/lidar sensor integration to support autonomous vehicle navigation," *Sensors*, vol. 20, p. 899, Feb 2020.

[82] H. Ye, Y. Chen, and M. Liu, "Tightly coupled 3d lidar inertial odometry and mapping," in *proceedings of International Conference on Robotics and Automation, Montreal, QC, Canada*, pp. 3144–3150, May 2019.

[83] W. E. L. Grimson, *Object Recognition by Computer: The Role of Geometric Constraints*. MIT press, 1990.

[84] D. A. Castañon, "New assignment algorithms for data association," in *proceedings of SPIE Signal and Data Processing of Small Targets, Orlando, FL, USA*, pp. 313–323, Aug 1992.

[85] M. Montemerlo, *FastSLAM: A factored solution to the simultaneous localization and mapping problem with unknown data association*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, June 2003.

[86] A. Doucet, N. de Freitas, K. Murphy, and S. Russell, "Rao-blackwellised particle filtering for dynamic bayesian networks," in *proceedings of Conference on Uncertainty in Artificial Intelligence, Stanford, CA, USA*, pp. 176–183, July 2000.

[87] Y. LeCun, P. Haffner, L. Bottou, and Y. Bengio, "Object recognition with gradient-based learning," *Shape, Contour and Grouping in Computer Vision*, vol. 1681, pp. 319–345, 1999.

[88] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations* (D. E. Rumelhart and J. L. McClelland, eds.), vol. 1, ch. 8, pp. 318–362, MIT Press, 1985.

[89] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, pp. 1735–1780, Nov 1997.

[90] J. Solà, J. Deray, and D. Atchuthan, "A micro lie theory for state estimation in robotics," *Institut de Robotica i Informàtica Industrial, Barcelona, Tech. Rep. IRI-TR-18-01*, Dec 2018.

[91] T. Lupton and S. Sukkarieh, "Visual-inertial-aided navigation for high-dynamic motion in built environments without initial conditions," *IEEE Transactions on Robotics*, vol. 28, pp. 61–76, Nov 2012.

[92] K. Konolige, G. Grisetti, R. Kümmerle, B. Limketkai, and R. Vincent, "Efficient sparse pose adjustment for 2d mapping," in *proceedings of IEEE/RSJ International Conference on Itelligent Robots and Systems, Taipei, Taiwan*, pp. 22–29, Oct 2010.

[93] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *proceedings of International Conference on Computer Vision, Barcelona, Spain*, pp. 2564–2571, Nov 2011.

[94] R. Mur-Artal and J. D. Tardós, "Fast relocalisation and loop closing in keyframe-based slam," in *proceedings of IEEE International Conference on Robotics and Automation, Hong Kong, China*, pp. 846–853, June 2014.

[95] M. Kaess, A. Ranganathan, and F. Dellaert, "isam: Incremental smoothing and mapping," *IEEE Transactions on Robotics*, vol. 24, pp. 1365–1378, Nov 2008.

[96] R. Cowell, A. Dawid, S. Lauritzen, and D. Spiegelhalter, *Probabilistic Networks and Expert Systems*. Springer, June 1999.

[97] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1988.

[98] S. Arnborg, D. G. Corneil, and A. Proskurowski, "Complexity of finding embeddings in a k-tree," *SIAM Journal Algebraic Discrete Methods*, vol. 8, p. 277–284, Apr. 1987.

[99] T. A. Davis, J. R. Gilbert, S. I. Larimore, and E. G. Ng, "A column approximate minimum degree ordering algorithm," *ACM Transactions on Mathematical Software*, vol. 30, p. 353–376, Sept. 2004.

[100] P. Krauthausen, A. Kipp, and F. Dellaert, "Exploiting locality in slam by nested dissection," in *proceedings of Robotics: Science and Systems, Philadelphia, USA*, p. 10, August 2006.

[101] R. Hänsch, T. Weber, and O. Hellwich, "Comparison of 3d interest point detectors and descriptors for point cloud fusion," in *proceedings of ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Zürich, Switzerland*, vol. 2, Aug 2014.

[102] C. Harris and M. Stephens, "A combined corner and edge detector," in *proceedings of Alvey Vision Conference, Manchester, UK*, pp. 147–151, Sept 1988.

[103] F. Tombari, S. Salti, and L. Di Stefano, "A combined texture-shape descriptor for enhanced 3d feature matching," in *proceedings of IEEE International Conference on Image Processing, Brussels, Belgium*, pp. 809–812, Sept 2011.

[104] R. B. Rusu, Z. C. Marton, N. Blodow, and M. Beetz, "Learning informative point classes for the acquisition of object model maps," in *proceedings of International Conference on Control, Automation, Robotics and Vision, Hanoi, Vietnam*, pp. 643–650, Dec 2008.

[105] M. P. do Carmo, *Differential Geometry of Curves and Surfaces*. Prentice-Hall, Inc. Englewood Cliffs, New Jersey, 1976.

[106] S. Belongie, J. Malik, and J. Puzicha, "Shape context: A new descriptor for shape matching and object recognition," in *proceedings of Advances in Neural Information and Processing Systems, Denver, CO, USA*, vol. 2, pp. 831–837, July 2000.

[107] D. Holz, A. E. Ichim, F. Tombari, R. B. Rusu, and S. Behnke, "Registration with the point cloud library: A modular framework for aligning in 3-d," *IEEE Robotics Automation Magazine*, vol. 22, pp. 110–124, Dec 2015.

[108] S. Leutenegger, P. Furgale, V. Rabaud, M. Chli, K. Konolige, and R. Siegwart, "Keyframe-based visual-inertial slam using nonlinear optimization," in *Proceedings of Robotics: Science and Systems*, (Berlin, Germany), June 2013.

[109] P. Lv, J. Guo, Q. Sha, J. Jiang, X. Mu, T. Yan, and B. He, "The comparison of gauss-newton and dog-leg in isam for auv," in *proceedings of IEEE Underwater Technology, Kaohsiung, Taiwan*, pp. 1–4, Apr 2019.

[110] S. Lowry, N. Sünderhauf, P. Newman, J. J. Leonard, D. Cox, P. Corke, and M. J. Milford, "Visual place recognition: A survey," *IEEE Transactions on Robotics*, vol. 32, pp. 1–19, Nov 2016.

[111] Y. Shin, Y. S. Park, and A. Kim, "Direct visual slam using sparse depth for camera-lidar system," in *proceedings of IEEE International Conference on Robotics and Automation, Brisbane, QLD, Australia*, pp. 5144–5151, May 2018.

[112] C. Debeunne and D. Vivet, "A review of visual-lidar fusion based simultaneous localization and mapping," *Sensors*, vol. 20, p. 2068, Apr 2020.

[113] J. Levinson, M. Montemerlo, and S. Thrun, "Map-based precision vehicle localization in urban environments," in *in proceedings of Robotics: Science and Systems, Atlanta, GA, USA*, pp. 121–128, June 2007.

[114] J. Levinson and S. Thrun, "Robust vehicle localization in urban environments using probabilistic maps," in *proceedings of IEEE International Conference on Robotics and Automation, Anchorage, AK, USA*, pp. 4372–4378, May 2010.

[115] M. Dunbabin, B. Lang, and B. Wood, "Vision-based docking using an autonomous surface vehicle," in *proceedings of IEEE International Conference on Robotics and Automation, Pasadena, CA, USA*, pp. 26–32, May 2008.

[116] M. Myint, K. Yonemori, K. Lwin, A. Yanou, and M. Minami, "Dual-eyes vision-based docking system for autonomous underwater vehicle: an approach and experiments," *Journal of Intelligent and Robotic Systems*, vol. 92, pp. 159–186, Sept 2018.

Appendix A

Feature-Based Laser Odometry for Autonomous Surface Vehicles utilizing the Point Cloud Library

Feature-Based Laser Odometry for Autonomous Surface Vehicles utilizing the Point Cloud Library*

Even Skjellaug, Edmund F. Brekke and Annette Stahl
Department of Engineering Cybernetics
Norwegian University of Science and Technology (NTNU)
Trondheim, Norway
evenskj@stud.ntnu.no, edmund.brekke@ntnu.no, annette.stahl@ntnu.no

***Abstract*—This paper proposes a pipeline for feature-based laser odometry for autonomous surface vehicles (ASVs) operating in urban environments. In particular, we investigate the suitability of several keypoint extractors and keypoint descriptors available through the Point Cloud Library (PCL). The complete odometry system, using the different extractors and descriptors, is implemented using the iSAM2 framework, and validated on real lidar data recorded onboard the autonomous ferry prototype MilliAmpere. The results demonstrate that accuracy similar to a standard Global Navigation Satellite System (GNSS) receiver can be achieved after 10 minutes even without loop closure. This study can be used as a starting point for future research on Simultaneous Localization And Mapping (SLAM) for ASVs.**

I. Introduction

When operating an autonomous surface vehicle (ASV) it is important to know the location of the vehicle at all times in order to avoid dangerous situations. This is usually done by a Global Navigation Satellite System (GNSS). A GNSS is accurate, but it can be jammed intentionally or unintentionally, and an ASV is therefore in need of a backup localization system. This can be based on exteroceptive sensors such as cameras, radar or lidar (i.e., a laser scanner). Key strengths of lidar as a localization sensor is the fact that it provides both range, bearing and elevation, and its precise accuracy. Different possible localization techniques include sensor-based odometry, localization relative to a known map, or simultaneous localization and mapping (SLAM).

There has been extensive research on laser odometry and SLAM using lidars. Some of the state-of-the-art methods are Google Cartographer [1], Lidar Odometry and Mapping (LOAM) [2], Lightweight and Ground-Optimized Lidar Odometry and Mapping (LeGO-LOAM) [3] and Hector-SLAM [4]. These methods focus mainly on ground vehicles, and among these papers, only [4] reports an implementation on a surface vehicle. However, this surface vehicle was not operating in a harbour environment, but rather a lake with dense vegetation.

Laser odometry and SLAM can be performed either direct or indirect. The direct methods usually utilize scan-matching, while the indirect methods usually are feature-based. State-of-the-art scan-matching algorithms are Iterative Closest Point (ICP) [5] and Normally Distributed Transform (NDT) [6]. Scan-matching takes two full point clouds and tries to calculate the transformation between them by matching each individual point from one point cloud to the next. Feature-based methods extract keypoints from the full point cloud, in order to make a sparse cloud. Feature-based laser odometry has the potential to be both faster and more accurate than direct laser odometry, and it also has the potential of being extended into a feature-based SLAM system. However, this requires both stable and repetitive keypoints. The state-of-the-art lidar methods mentioned above include both indirect and direct methods, as LOAM and LeGO-LOAM are feature-based, and Hector-SLAM and Google Cartographer use scan-matching.

One of the major challenges in performing laser odometry and SLAM in harbour environments is that a lot of the information is lost due to the fact that the lidar does not reflect water as well as solid ground surfaces. As the area is not contained, information is not only lost due to lacking reflection of the water, but a lot of the rays will not be reflected as they do not hit any object. This results in a sparser point cloud than what is usually obtained by ground vehicles, or vehicles in contained environments.

Another major challenge is data association. Some of the state-of-the-art methods for data association are Joint Compatibility Branch and Bound [7], the Auction algorithm [8], the Hungarian algorithm [9], FastSLAM [10], and Random Sample Consensus (RANSAC) [11]. In the last couple of years there has also been research on how to perform the data association using deep learning [12], [13], [14]. In this paper RANSAC will be used to perform the data association, as it is already implemented in Point Cloud Library (PCL).

The work presented in this paper builds on two recent master theses [15], [16] which investigated laser-based localization for the autonomous ferry prototype MilliAmpere, currently being developed by the Norwegian University of Science and Technology (NTNU). The first thesis implemented Hector-SLAM, LOAM and Berkeley Localization And Mapping (BLAM), which are all available open source through Robot Operating System (ROS). Hector-SLAM produced the best results on the recorded data. However the thesis concludes that neither

*This work was supported by the Research Council of Norway through the Centers of Excellence funding scheme, project number 223254, Centre for Autonomous Marine Operations and Systems (AMOS), and through the MAROFF funding scheme, project number 295033, Autonomous ships, intentions and situational awareness. The lidar data were recorded by Marius Ødven [15].

of the methods provided accurate enough results to be used as the primary pose estimator for MilliAmpere. The second thesis investigated localization by means of particle filters. The conclusion from this thesis was that even though one can get accurate localization, it is not computationally efficient enough to be able to run on MilliAmpere in real-time.

This paper will focus on feature-based laser odometry using keypoint extractors and descriptors from PCL in a marine harbour environment. In order to use keypoint extractors and descriptors for laser odometry, they need to be able to run real-time and provide stable and repeatable keypoints. The keypoint extractors implemented and discussed in this paper are Intrinsic Shape Signatures (ISS) [17], Normal Aligned Radial Features (NARF) [18], 3D-Scale Invariant Feature Transform (3D-SIFT) [19] and Harris Keypoint 3D [20], and the descriptors implemented are Point Feature Histograms (PFH) [21] [22], Fast Point Feature Histograms (FPFH) [24], Signatures of Histograms of OrienTations (SHOT) [25] and 3D Shape Context (3D SC) [26].

This paper has two main contributions. The first contribution of this paper is to propose a solution for feature-based laser odometry for ASVs. In contrast to the attempts with open source SLAM methods in [15], the approach taken in this paper uses an established factor graph library, GTSAM [27], to develop a solution which is more transparent, and which more easily can be adapted to SLAM or pure localization, depending on the operation needs for the ASV.

The second contribution of this paper is to compare keypoint extractors and descriptors in PCL for laser odometry. PCL is chosen as it is available open source and it is both flexible and easy to implement. Comparisons of these keypoint extractors and descriptors has been done before, but only in the context of object recognition [28], [29], [30]. These survey articles have only considered databases where each entry has been one specific categorized item, and not real-life surroundings. By comparing the keypoint extractors and descriptors, we will achieve a greater understanding as to how they work in real-life surroundings. Furthermore, as the system is intended to run in real time, we also investigate the run-time for the different extractors and descriptors.

The outline of this paper is as follows: Section II introduces the odometry pipeline. The keypoint extractors are introduced in Section III and the descriptors in Section IV. Section V shows the area the data was logged and introduces the sensors used, the resulting laser odometry is shown in Section VI. Lastly, the conclusion is given in Section VII.

II. Odometry pipeline

Inspired by [31], Fig. 1 shows the pipeline of how the odometry is calculated. The data are first pre-processed in order to reduce the computational complexity. This is done by first removing the points that the lidar can see on the ASV using a conditional removal filter, as these points will follow the ASV and therefore will not be static. Then statistical outliers are removed; these are points that do not have enough neighbors within a given radius. In the end, a voxel grid is applied in order to downsample the point cloud.

After the data have been pre-processed, a keypoint extractor is used to extract repeatable and stable keypoints. To best match keypoints between consecutive point clouds, descriptors are calculated at each keypoint, and a correspondence estimation function in PCL is used to match these. As these correspondences are prone to false positives, two correspondence rejectors have been included in this pipeline, firstly a distance rejector, which removes matches over a threshold distance. Secondly, RANSAC is implemented to get a consistent transformation between consecutive point clouds. The distance rejector is mostly included to reduce the correspondences and thereby the iterations needed to be run by RANSAC. In order to calculate the motion estimation, ICP is used as the last part of the odometry pipeline. After calculating the transformation through RANSAC, it is used as the initial guess of the ICP in order to further improve the estimate without increasing the computational complexity by much. Sometimes this pipeline will not find any matches between the keypoints and in order for the trajectory to not diverge a backup is needed. This backup will be performed by using ICP, which might increase the run-time of the pipeline. This is however different to what is being used in [4], as this method takes in the entire point clouds, while Hector-SLAM represents the world as an occupancy grid to do the scan-matching.

The motion estimation comes from the ICP, as the result of the computations is a transformation matrix, T. This transformation matrix represents the movement of the ASV between the consecutive point clouds. It consists of a rotation matrix, R, and a translation, t, as seen in the following equation:

$$\mathrm{T} = \begin{bmatrix} \mathrm{R} & \mathrm{t} \\ 0^T & 1 \end{bmatrix}. \tag{1}$$

In order to get the full trajectory, the transformation matrices are concatenated as in the following equation:

$$\mathrm{T}_a\mathrm{T}_b = \begin{bmatrix} \mathrm{R}_a\mathrm{R}_b & \mathrm{R}_a\mathrm{t}_b + \mathrm{t}_a \\ 0^T & 1 \end{bmatrix}. \tag{2}$$

This full trajectory is then implemented using the iSAM2 [32] framework from the open source GTSAM C++ library. Doing this creates a factor graph of the poses and the keypoints at each pose, which will be optimized by iSAM2.

III. Keypoint extractors

ISS, NARF, 3D-SIFT and Harris Keypoint 3D were chosen based on their availability in PCL and their performance in the object recognition survey article [30].

A. Intrinsic Shape Signatures

ISS uses the intrinsic reference frame in order to compute keypoints that are independent of the view. A covariance matrix is calculated using all points within a set frame radius. This covariance matrix is used to calculate the eigenvalues and eigenvectors. The intrinsic reference frame is then calculated using the eigenvalues and the eigenvectors. The eigenvector

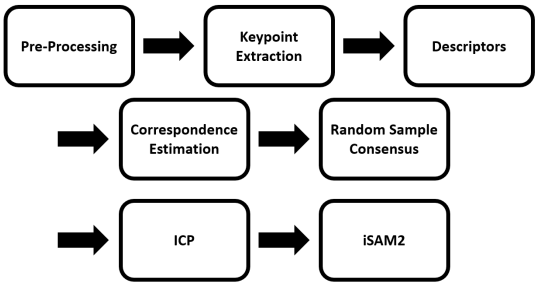


Fig. 1. Odometry pipeline

belonging to the largest eigenvalue will then be the x-axis, the eigenvector to the second largest eigenvector will be the y-axis and the vector resulting from the cross multiplication will be the z-axis.

To determine the keypoints, the eigenvalues of the covariance matrix is used. ISS uses the magnitude of the smallest eigenvalue as well as the ratio between two successive eigenvalues. Setting a lower threshold on the smallest eigenvalue is done in order to only include points with large variations along each principal direction, while the ratio of the eigenvalues is used in order to reject points where the spread is similarly large along all directions.

B. Normal Aligned Radial Features

NARF transforms the point clouds into range images and extracts the keypoints from these 2D images. The keypoints are selected with two goals in mind. Firstly, the points should be selected from a stable surface with sufficient changes in the neighborhood, and secondly, they should be making use of the object borders so that outer shapes of objects from a certain perspective are chosen.

The NARF keypoint extractor has four steps. It starts by looking at the neighborhood of every point and calculates a score based on surface changes and directions in order to incorporate the border information. Then an interest value will be calculated on the basis of how much the dominant directions change within the neighborhood and how much the surface itself changes. Smoothing will then be performed on the interest values and lastly non-maximum suppression will be performed to find the keypoints.

C. 3D Scale Invariant Feature Transform

3D-SIFT is an extension of the Scale Invariant Feature Transform (SIFT) proposed by Lowe [33]. SIFT uses a cascade filtering approach in order to extract keypoints. The first stage of this filtering approach is to identify locations that are repeatable under different views. To do this a scale space is built by performing convolution on the point cloud with Gaussian kernels:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y). \tag{3}$$

To be able to detect stable keypoints in the scale-space, the Difference-of-Gaussian (DoG) is used by subtracting adjacent smoothed point clouds:

$$D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma), \tag{4}$$

where k is a constant integer.

With the help of DoG and the scale-space one can extract stable features by comparing each point to the eight neighbors it has in the current scale and all nine neighbors from the scale above and below. If the chosen point is either larger or smaller than all 26 of its neighbors, it is selected as a keypoint. The last step of 3D-SIFT is to reject all keypoints where the curvature is low in order to get stable keypoints.

D. Harris Keypoint 3D

The Harris Keypoint 3D detector is an extension of the Harris corner detector [34]. The Harris corner detector measures the local changes of pictures to determine corners using the local autocorrelation function, which is similar to the image gradients. The Harris keypoint 3D replaces the image gradients by surface normals. Using the surface normals, a covariance matrix *Cov* is calculated around each point. The keypoint response for each point is then defined by

$$r(x, y, z) = det(Cov(x, y, z)) - k(trace(Cov)))^2, \tag{5}$$

where k is a positive parameter. In order to not have too many keypoints in a small neighborhood, a non-maximal suppression process and a thresholding process is performed in order to extract the final keypoints.

IV. Keypoint descriptors

Keypoint descriptors are implemented in order to improve the matching, and to reduce the computational complexity required from RANSAC. This is because descriptors will reduce the number of correspondences in the search compared to matching by distance. PFH, FPFH, SHOT and 3D SC were then chosen based on their availability in PCL and their performance in the object recognition survey articles [28], [29]. Unique Shape Context (USC) was also considered as it is an extension of 3D SC, however at the time of writing, it was not available in the latest version of PCL that was compatible with ROS.

A. Point Feature Histograms

PFH uses a multi-dimensional histogram of the mean surface curvature to uniquely describe each keypoint that has been selected. In order to compute the mean surface curvature at a point, all the neighbors within a given radius need to be found. After this, the normals of all the selected points need to be found, this is done by approximating them using Principal Component Analysis (PCA). Once all the normals are obtained, the existing viewpoint information is used to re-orient all the points consistently.

A Darboux frame [23] is defined using the neighborhood around a point, this frame is defined as:

$$u = n_s, v = (p_t - p_s) \times u, w = u \times v, \quad (6)$$

where n_s is the source normal, p_t is the target point and p_s is the source point. Using this, the following three features are calculated:

$$\begin{aligned} f_1 &= \langle v, n_t \rangle \\ f_2 &= \frac{\langle u, p_t - p_s \rangle}{||p_t - p_s||} \\ f_3 &= atan(\langle w, n_t \rangle, \langle u, n_t \rangle) \end{aligned} \quad (7)$$

These features are then categorized in a histogram, where each bin contains the percentage of the points with the feature-values within defined intervals. Each of the features uses five bins each, giving a 125-bin histogram as the descriptor.

B. Fast Point Feature Histograms

FPFH is an extension of PFH to reduce the computational complexity and memory usage. This allows FPFH to be used for real-time applications, as PFH can be a major bottleneck in the registration framework. FPFH manages to reduce the computational complexity from $O(nk^2)$ to $O(nk)$.

The first step is to compute the histogram of the three angles between a point p and its k-nearest neighbors. This produces the Simplified Point Feature Histogram (SPFH). In order to keep the descriptive power of PFH, the SPFH is calculated for all of the points neighbors. The resulting equation for calculating the FPFH is then:

$$FPFH(p) = SPFH(p) + \frac{1}{k} \sum_{i=1}^{k} \frac{1}{w_k} \cdot SPFH(p_k), \quad (8)$$

where w_k is the distance between the point and its kth neighbor.

The three angles are then binned into three 11-bin histograms that concatenate into a single 33-bin FPFH descriptor.

C. Signatures of Histograms of OrienTations

SHOT is based on a unique and unambiguous local reference frame. This reference frame is created using eigenvalue decomposition around a certain point. After the reference frame is created, a sphere centered around the point divides the neighborhood so that in each grid bin a weighted histogram of normals is obtained. SHOT uses nine values to encode the reference frame and 352 values for the descriptor, which comes from 32 divisions of the spherical grid and the use of 11 different shape bins. In the end the descriptor is normalized to sum up to 1 in order to achieve robustness of variations of the point density.

D. 3D Shape Context

3D SC descriptor is an extension of the 2D Shape Context (2D SC) descriptor originally used for images [35]. 3D SC uses a sphere centered around the chosen keypoint, where the north pole of the sphere is oriented with the surface normal

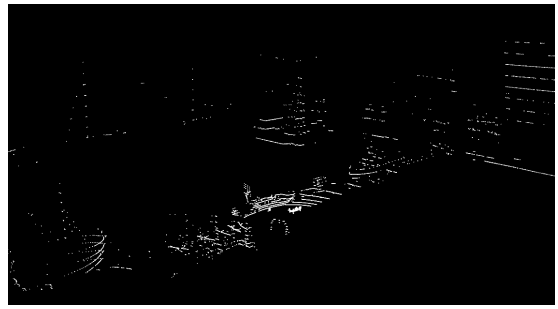


Fig. 2. Point cloud recorded by the ASV (MilliAmpere) before pre-processing the data (7713 points)

estimate of the point. The sphere is divided into bins, these bins are equally spaced in the azimuth and elevation dimensions, and logarithmically spaced along the radial dimension. Each bin corresponds to one element in the feature vector.

Each bin is then normalized with respect to the volume of the bin and the local point density. This normalization compensates for the large variation in bin sizes and point densities. 3D SC does however not have a unique reference frame, creating the possibility for several descriptors for a single keypoint. This is due to a degree of freedom in the azimuth direction.

V. Experiment

The data used in this paper were gathered in 2018 in Brattørabassenget, Trondheim, Norway. The data were logged in a harbour environment surrounded by docks and buildings using a Velodyne VLP-16 lidar. 636 seconds of lidar data were recorded with the lidar placement as seen in Fig. 5, resulting in 6305 point clouds. One of the point clouds that are recorded can be seen in Fig. 2, this point cloud is recorded close to the dock. For these point clouds, the average number of points is 5687, which is significantly less than the about 30000 points the Velodyne VLP-16 is capable of receiving, showing that around 80% of the signals sent out are not received. Fig. 4 shows the trajectory driven during the recordings in its true environment, this trajectory is approximately 1060 meters. Real Time Kinetic Global Navigation Satellite System (RTK-GNSS) data were also recorded and is considered the ground truth throughout this paper as it is highly accurate. Both the RTK-GNSS data and the lidar data were recorded using ROS, and can therefore be re-run in real-time. The computer used for the implementation of the odometry pipeline has an i7-8700@3.20GHz CPU.

When pre-processing the data in this paper, statistical outliers are removed if there are no more than 3 neighbors within a radius of 1 meter, and the voxel grid applied uses a leaf size of 0.1 meters. The resulting point cloud after pre-processing can be seen in Fig. 3. In order to make a fair comparison between the descriptors, the radius search is set to 5 meters for all the descriptors. The descriptors that use normals will

Fig. 3. Point cloud recorded by the ASV (MilliAmpere) after pre-processing the data (3227 points)

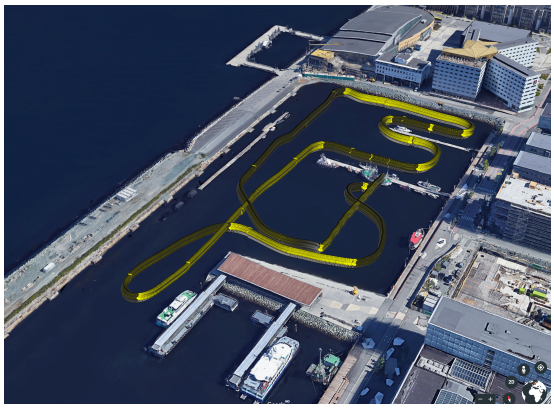


Fig. 4. Screen-shot of the ferry's trajectory from which the data is logged. Longitude and Latitude was uploaded in GOOGLE earth. Note that at the end of the logging procedure, the boat count was higher, and the construction site on the right had become more visible to the lidar. Image and caption taken from [15]

in addition use the same normals, which are estimated using a radius search equal to 3 meters. For the keypoint extractors, it is not as easy to make a fair comparison, as they all utilize different parameters. The parameters were therefore chosen similarly to the parameters presented by the code examples available in PCL, with only minor tuning.

VI. Results

In order to compare the keypoint extractors for laser odometry two main criteria will be used, run-time and the root mean square error (RMSE) of the odometry compared to the RTK-GNSS. In order to test the run-time, keypoints will be extracted and matched for all 6305 point clouds. The results can be seen in table I. This shows that all the keypoint extractors are fast enough to run real-time without descriptors, with ISS being the fastest. The table also show the average RMSE, where 3D-SIFT outperforms all the other extractors.

Fig. 6 shows the resulting trajectories from matching keypoints using RANSAC, and Fig. 7 shows the z-axis as the odometry is performed in 3D. From these images it can be seen

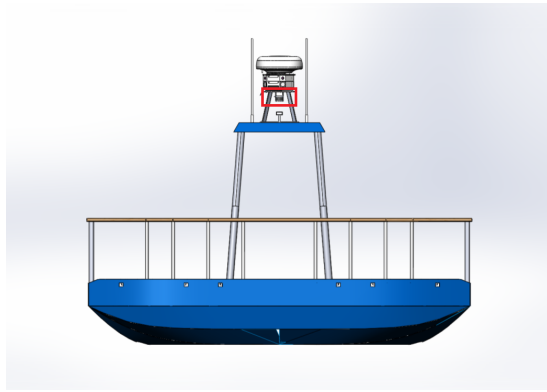


Fig. 5. Illustration of the ASV (MilliAmpere) used to record the data. The lidar is placed above the roof and is highlighted by the red square. Illustration created by Egil Eide.

TABLE I
Total runtime to extract keypoints and match them and the resulting average RMSE

Keypoint extractor	Time [s]	Average RMSE [m]
3D-SIFT	425	21.7
NARF	376	141
Harris Keypoint 3D	452	60.3
ISS	329	47.3

that the trajectories from NARF and Harris do not resemble the ground truth at all, and will therefore be excluded from further testing. Both ISS and 3D-SIFT do on the other hand resemble the ground truth, and 3D-SIFT performs especially well considering no descriptor is used.

As both ISS and 3D-SIFT perform well on the data, the descriptors are implemented for both scenarios. This should give a better understanding which keypoint extractor is the preferred one, and exposes the descriptors to different scenarios. The comparison of the descriptors is done by comparing the resulting trajectories to the RTK-GNSS, and by calculating the run-times to see whether they are suitable for real-time applications. The run-times of the descriptors using ISS and 3D-SIFT as keypoint extractors can be seen in table II. From this table it is clear that PFH is not suitable for real-time applications and it is therefore be excluded from further implementation.

TABLE II
Time to extract keypoints and descriptors (3.5 m radius) and to match them

Keypoint descriptor	ISS [s]	SIFT [s]
FPFH	548	542
SHOT	394	529
3D SC	577	664
PFH	1684	2068

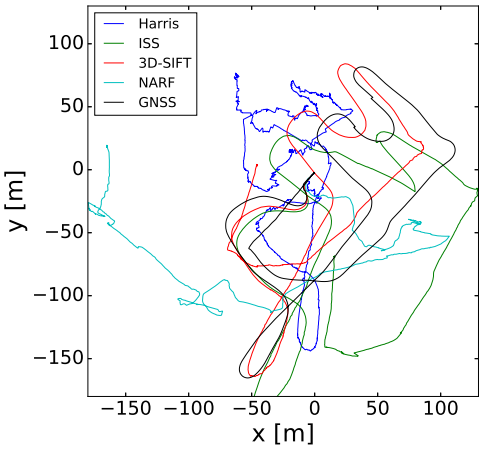


Fig. 6. Trajectory of different keypoint extractors

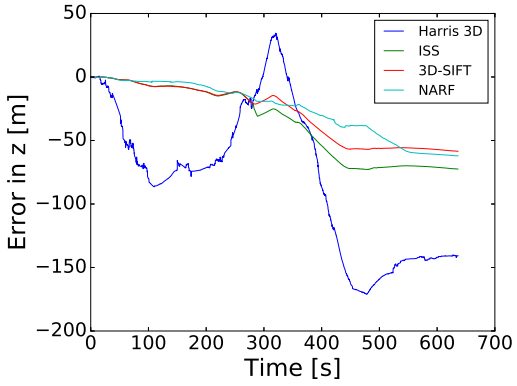


Fig. 7. Z-axis of different keypoint extractors

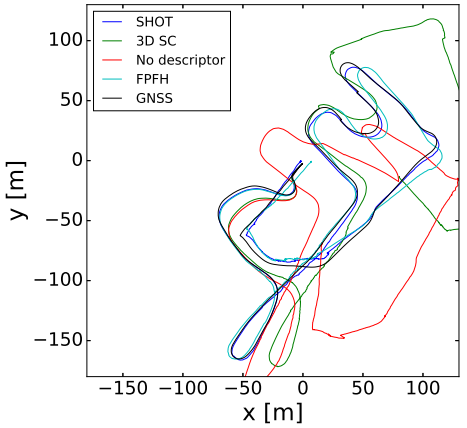


Fig. 8. Trajectory of ISS using different keypoint descriptors

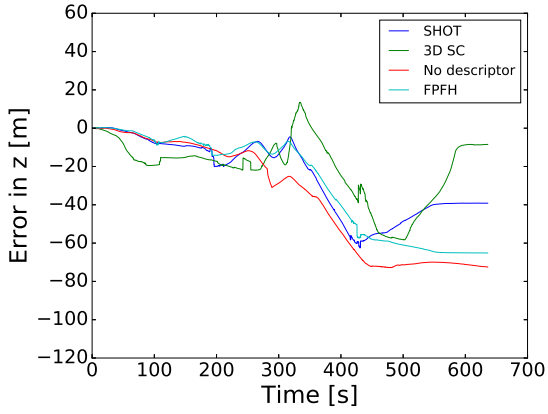


Fig. 9. Trajectory of ISS using different keypoint descriptors

The three other descriptors are implemented, and the resulting trajectories with the ISS keypoint extractor can be seen in Fig. 8. It is clearly shown that using a descriptor for the ISS greatly improves the RMSE of the laser odometry. SHOT is doing especially well compared to FPFH and 3D SC. As SHOT is both the fastest and the one giving the most accurate result, it will be the recommended descriptor when using the ISS keypoint extractor.

The biggest weakness when using 3D SC is that it fails while the ASV is turning, this causes the backup system to take over pose estimation using ICP. ICP however, does not match well when significant rotations are present, unless the number of iterations is high. This is the cause of the large errors that can be observed in the top right corner, and causes 3D SC to be an unreliable descriptor. FPFH on the other hand, is more reliable, and will rarely cause large errors in the system, but it is both slower and less accurate than SHOT. The accuracy of both FPFH and 3D SC could possibly be improved by tuning the parameters, but we do not see any reason why tuning their parameters should make either of them superior to SHOT.

Similar results are shown when using SIFT as seen in Fig. 10. 3D SC is still too unreliable to work as the descriptor, while both FPFH and SHOT seem like reliable options. SHOT is still faster than FPFH, however it is not much faster, as it is in the case when ISS is used.

Even though the laser odometry gives impressive results in the x-y-plane, it does have large errors in the z-direction as seen in Fig. 9 and Fig. 11. This is often a problem when using lidars and especially the ones with only 16 channels. This error is due to the vertical distance between the channels. When the ASV is experiencing pitch and roll, the channels

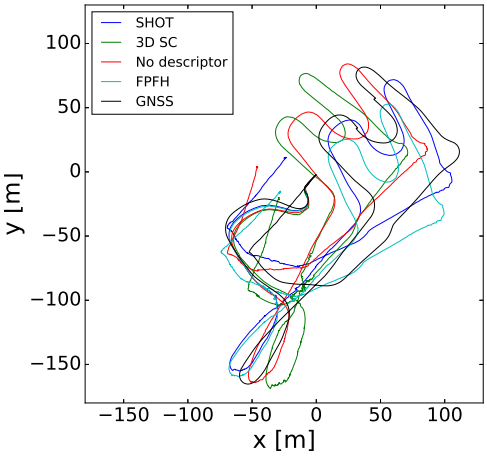


Fig. 10. Trajectory of SIFT using different keypoint descriptors

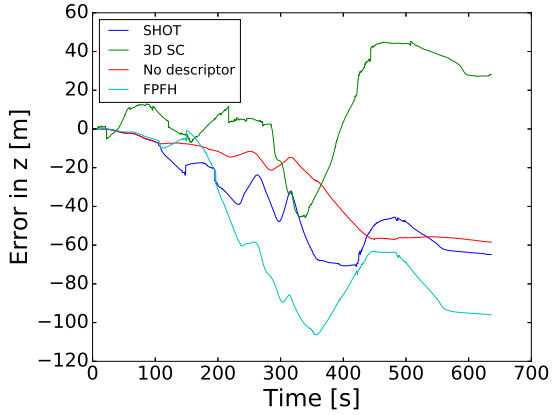


Fig. 11. Trajectory of SIFT using different keypoint descriptors

will reflect the new points either above or below the previous points, this is especially true for corridor environments, and will consequently create an error in the z-direction. This can be seen well between 300 and 400 seconds where the error in z-direction is growing quickly. For this time interval, the ASV is driving straight and only sees the dock on the left side of the vehicle. This will likely not affect the results in the x-y-plane by much, as a small error in pitch will result in a large error in the z-direction over time. It should however be possible to greatly reduce this error by fusing in IMU and/or GNSS or by extending the laser odometry to a SLAM system which includes loop closures.

Using SHOT and ISS, we manage to get an average RMSE of just 3 meters over the full 636 seconds of data in the

TABLE III
AVERAGE RMSE FOR MATCHING USING KEYPOINTS AND DESCRIPTORS

Keypoint descriptor	ISS [m]	SIFT [m]
FPFH	6.08	17.09
SHOT	3.09	15.26
3D SC	72.23	23.07

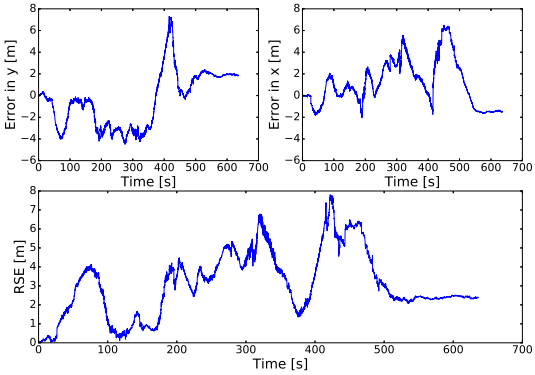


Fig. 12. Errors for laser odometry using ISS and SHOT

x-y-plane. As the RMSE of GPS is 4 meters, this goes to show that the feature-based laser odometry implemented in this paper is as accurate as the GPS is specified to be. The largest RMSE is 7.8 meters, which is within the GPS's 95% confidence interval, meaning that the laser odometry is within this confidence interval for the full 636 seconds. The RMSE over time and the errors in x- and y-direction can be seen in Fig. 12.

VII. CONCLUSION

It has been shown in this paper that feature-based laser odometry for an ASV can be performed using the keypoint extractors and descriptors provided in PCL and that the results will be within the specifications of a standard GNSS receiver.

ISS, NARF, Harris and 3D-SIFT were compared in order to see which keypoint extractors are the most suitable for laser odometry in a marine harbour environment. It is clearly seen from the results that neither Harris or NARF gave satisfying results. As for descriptors, SHOT, FPFH, 3D SC and PFH were explored in order to improve the odometry. PFH was, however, not able to run real-time. SHOT, FPFH and 3D SC were implemented and the results showed that 3D SC was not reliable enough to be used as the descriptor. FPFH did perform well, but SHOT was both faster and more accurate, making it the superior descriptor for the data provided from MilliAmpere.

Further work includes extending the laser odometry system into a feature-based SLAM system, and fusing in IMU and GNSS data. This should greatly reduce the error in z-direction while also further improving the results in the x-y-plane.

REFERENCES

[1] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-time loop closure in 2D lidar SLAM," in proceedings of IEEE International Conference on Robotics and Automation, Stockholm, Sweden. May 2016, pp. 1271-1278

[2] J. Zhang and S. Singh, "LOAM: Lidar odometry and mapping in real-time," in proceedings of Robotics: Science and Systems, Berkeley, USA. July 2014, pp. 109–111

[3] T. Shan and B. Englot, "LeGO-LOAM: Lightweight and ground-optimized lidar odometry and mapping on variable terrain," in proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems, Madrid, Spain. Oct 2018, pp. 4758-4765

[4] S. Kohlbrecher, O. Stryk, J. Meyer, and U. Klingauf, "A flexible and scalable SLAM system with full 3D motion estimation," in proceedings of IEEE International Symposium on Safety, Security, and Rescue Robotics, Kyoto, Japan. Nov 2011, pp. 155-160

[5] P. J. Besl and N. D. McKay, "A method for registration of 3-D shapes," in IEEE Transactions on Pattern Analysis and Machine Intelligence. Feb 1992, pp 239-256

[6] M. Magnusson, "The three-dimensional normal-distributions transform - an efficient representation for registration, surface analysis, and loop detection," PhD thesis. Örebro University. Dec 2009

[7] J. Neira and J. D. Tardós "Data Association in Stochastic Mapping Using the Joint Compatibility Test," in IEEE Transactions on Robotics and Automation. Dec 2001, pp. 890-897

[8] D. Bertsekas "A new algorithm for the assignment problem," Mathematical Programming. 1981, pp. 152–171

[9] H. W. Kuhn "The Hungarian method for the assignment problem," Naval Research Logistics Quarterly. 1955, pp. 83-97

[10] M. Montemerlo and S. Thrun "Simultaneous Localization and Mapping with Unknown Data Association Using FastSLAM," in proceedings of IEEE International Conference on Robotics and Automation, Taipei, Taiwan. Sept 2003, pp. 1985-1991

[11] M. A. Fischler and R. C. Bolles "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," in proceedings of Communications of the ACM, New York, USA. June 1981, pp. 381–395

[12] E. Baser, V. Balasubramanian, P. Bhattacharyya, and K. Czarnecki "FANTrack: 3D multi-object tracking with feature association network," in proceedings of IEEE Intelligent Vehicles Symposium, Paris, France. June 2019, pp. 1426-1433

[13] P. Ondruska and I. Posner "Deep tracking: seeing beyond seeing using recurrent neural networks," in proceedings of AAAI Conference on Artificial Intelligence, Phoenix, USA. Feb 2016, pp. 3361-3367

[14] K. Yoon, D. Y. Kim, Y. C. Yoon, and M. Jeon "Data association for multi-object tracking via deep neural networks," in Sensors. Jan 2019, pp. 559

[15] M. S. Ødven, "Lidar-based SLAM for autonomous ferry," MSc thesis. Norwegian University of Science and Technology. Jan 2019

[16] N. Dalhaug, "Lidar-based localization for autonomous ferry," MSc thesis. Norwegian University of Science and Technology. June 2019

[17] Y. Zhong, "Intrinsic shape signatures: a shape descriptor for 3D object recognition," in proceedings of IEEE International Conference on Computer Vision Workshops, Kyoto, Japan. Sept 2009, pp. 689-696

[18] B. Steder, R. B. Rusu, K. Konolige, and W. Burgard, "Point feature extraction on 3D range scans taking into account object boundaries," in proceedings of IEEE International Conference on Robotics and Automation, Shanghai, China. May 2011, pp. 2601-2608

[19] R. Hänsch, T. Weber, and O. Hellwich, "Comparison of 3D interest point detectors and descriptors for point cloud fusion," in proceedings of ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Zürich, Switzerland. Sept 2014, pp. 57–64

[20] I. Sipiran and B. Bustos, "Harris 3D: a robust extension of the Harris operator for interest point detection on 3D meshes," in The Visual Computer. July 2011, pp. 963–976

[21] R. B. Rusu, N. Blodow, Z. C. Marton, and M. Beetz, "Aligning point cloud views using persistent feature histograms," in proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems, Nice, France. Sept 2008, pp.3384-3391

[22] R. B. Rusu, N. Blodow, Z. C. Marton, and M. Beetz, "Learning informative point classes for the acqusition of object model maps," in proceedings of International Conference on Control, Automation, Robotics and Vision, Hanoi, Vietnam. Dec. 2008, pp. 643-650

[23] M. P. do Carmo "Differential Geometry of Curves and Surfaces," in Prentice-Hall, Inc., Englewood Cliffs, New Jersey. 1976

[24] R. B. Rusu, N. Blodow, and M. Beetz, "Fast point feature histograms (FPFH) for 3D registration," in proceedings of IEEE International Conference on Robotics and Automation, Kobe, Japan. May 2009, pp. 3212-3217

[25] F. Tombari, S. Salti, and L. Di Stefano, "Unique signatures of histograms for local surface description," in proceedings of European Conference on Computer Vision, Crete, Greece. Sept 2010, pp. 356–369

[26] A. Frome, D. Huber, R. Kolluri, T. Bülow, and Jitendra Malik, "Recognizing objects in range data using regional point descriptors," in proceedings of European Conference on Computer Vision, Prague, Czech Republic. May 2004, pp. 1-14

[27] F. Dellaert "Factor graphs and GTSAM: a hands-on introduction," Technical Report GT-RIM-CP&R-2012-002, Georgia Institue of Technology, Atlanta, USA. Sept 2012

[28] L. A. Alexandre, "3D descriptors for object and category recognition: a comparative evaluation," in proceedings of IEEE International Conference on Intelligent Robotic Systems, Vilamoura, Portugal. Oct 2012, pp. 1-6

[29] S. Filipe and L. A. Alexandre, "A comparative evaluation of 3D keypoint detectors in a RGB-D object dataset," in proceedings of International Conference on Computer Vision Theory and Applications, Lisbon, Portugal. Jan 2014, pp. 476-483

[30] F. Tombari, S. Salti, and L. Di Stefano, "Performance evaluation of 3D keypoint detectors," in proceedings of International Conference on 3D Imaging, Modeling, Processing, Visualization and Transmission, Hangzhou, China. May 2011, pp. 236-243

[31] D. Holz, A. E. Ichim, F. Tobari, R. B. Rusu, and S. Behnke, "Registration with the Point Cloud Library: A Modular Framework for Aligning in 3-D," in IEEE Robotics and Automation Magazine. Dec 2015, pp. 110-124

[32] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. Leonard, F. Dellaert, "iSAM2: Incremental smoothing and mapping with fluid relinearization and incremental variable reordering," in proceedings of IEEE International Conference on Robotics and Automation, Shanghai, China. May 2011, pp. 3281-3288

[33] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," in International Journal of Computer Vision. Nov. 2004, pp. 91-110

[34] C. Harris and M. Stephens, "A combined corner and edge detector," in proceedings of Alvery Vision Conference, Manchester, UK. Sept 1988, pp. 147–151

[35] S. Belongie, J. Malik, and J. Puzicha, "Shape context: a new descriptor for shape matching and object recognition," in proceedings of Neural Information Processing Systems, Denver, USA. July 2000, pp 831-837

Even Skjellaug

Feature-Based Lidar SLAM for Autonomous Surface Vehicles Operating in Urban Environments

NTNU
Norwegian University of
Science and Technology