Marius Strand Ødven

Lidar-Based SLAM for Autonomous Ferry

Master's thesis in Cybernetic and Robotics
Supervisor: Edmund Førland Brekke
January 2019

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

Abstract

This thesis gives a survey of existing lidar-based simultaneous localization and mapping (SLAM) which might suit an autonomous ferry, describes a data-collecting experiment using a half-scale prototype ferry, and presents and analyzes SLAM results. Three SLAM algorithms available as open-source through the Robot Operating System (ROS) community, namely Hector-SLAM, LOAM and BLAM are run on the collected lidar-data.

The data was acquired using a 3D lidar, a GNSS-receiver and an Inertial Navigation System serving as IMU. The experiment was set up such that the recorded data could be replayed to simulate various scenarios assumed likely for an operational autonomous ferry for passengers.

The output maps and pose estimates of the algorithms are run on different data-sets and are evaluated compared to ground truth data. The results are evaluated, and neither SLAM algorithm is deemed adequately suited to serve as a complete solution for an autonomous ferry. Ideas of what works and what does not are discussed, however, and thus the thesis represents a foundation for which others can build towards a SLAM solution for said purpose.

Sammendrag

Denne masteroppgaven gir en oversikt over eksisterende lidarbaserte SLAM-metoder som kan passe til en autonom ferge, beskriver et datainnsamlingseksperiment ved hjelp av en halvskala prototype-ferge og presenterer og analyserer SLAM-resultater. Tre SLAM-algoritmer tilgjengelige som åpen kilde via ROS-community, er kjørt på de innsamlede lidar-dataene.

Data brukt i oppgaven ble logget ved bruk av en 3D-LIDAR, en GNSS-mottaker og et treghetsnavigasjonssystem som fungerende IMU. Eksperimentet ble satt opp slik at de registrerte dataene kunne bli kjørt på nytt for å simulere ulike scenarier antatt sannsynlig for en operativ autonom passasjerferge.

Resulterende kart og posisjonsestimatene av algoritmene er kjørt på forskjellige datasett og evalueres i forhold til ground truth data. Resultatene er evaluert, i tillegg til en vurdering av hvorvidt SLAM-algoritmen anses tilstrekkelig egnet til å fungere som en løsning for en autonom ferge. Ideer om hva som fungerer og hva som ikke fungerer er diskutert diskutert, og dermed representerer avhandlingen et fundament som andre kan bygge på mot en SLAM-løsning for nevnte formål.

Preface

Whether at home, at the workplace or in the streets – mobile robots are becoming increasingly popular. Through transportation via Uber's self-driving cars and keeping it tidy via a robotic vacuum cleaner, humans are delegating monotonous tasks to robots, and making time to do as they desire. Industrial operations can be made both safer and more effective through automation. A snake robot discovering a faulty oil-pipe while traversing its insides before it bursts, for example, is both. It is more effective to fix the pipe before it bursts due to costs coming with the clean-up, and it is safer to dive to the bottom of the sea and fix the pipe in a controlled manner.

Unless possessing with the necessary abilities, a mobile robot can be dangerous as well. One of these essential tools is the ability to learn and know its environment. Being able to create a map and localizing itself within the map, gives the robot the means to plan paths and traverse them – all abilities required to achieve true autonomy. The field studying the two former is known as simultaneous localization and mapping (SLAM).

Say an autonomous passenger ferry uses high-precision GNSS-services as its navigation system. What can it do if for some reason the service should become unavailable? Drift into the rocks? It needs a backup navigation system, which is a role SLAM is more than capable to fill – either to make sure the vessel stays put in the middle of the fjord, or to plan a route to the dock, traversing it, and safely letting the passengers off. SLAM does not need to be a system staying idle, waiting for a disaster to happen. With a solution in place, its pose estimate can be fused with other pose estimates e.g. the high-precision GNSS estimate and an IMU. After all, the more information, the better.

As NTNU, among others, are working towards installing an autonomous ferry in Trondheim, a SLAM system is required. This gave rise to a series of different projects and theses aiming to test concept or develop new algorithms, both with various sensors. The author was to work with the 3D lidar, and with it collect data and identify and run possible suiting algorithms on them.

The work has been hard, but at the same time rewarding. Countless hours have been spent on tedious reading and debugging little documented code, as well as debugging in ROS. More importantly, the work has been rewarding allowing for growth, getting the hang of ROS and becoming a better and more experienced programmer.

Although no complete solution to the ferry's need for a SLAM capabilities has been found, some foundation has been laid, and some ideas going forward are presented to whomever will continue to look for the perfect lidar-based SLAM solution for an autonomous ferry.

The most pleasing aspect of the work, however, was sailing the half-scale prototype MilliAmpere, and logging the data. This is not only because it to the author comes natural with field work, but also due to the interest of so many people who stopped to watch or ask questions. It was simply enjoyable to be part of the project that is the Autoferry, and the author wishes it all the best going forward.

Provided Equipment and Information

The thesis subject was provided by supervisor Edmund Brekke for the pre-project and was worked on further in the scope of this thesis. At that time, the Velodyne VLP-16 was already acquired by the Autosea project for SLAM purposes, and the INS and GNSS-receiver was already installed on MilliAmpere. In other words, the author was provided with sensors and the experimental platform MilliAmpere. Also, a copy of the MilliAmpere workspace folder containing custom drivers for the IMU and GNSS-receiver was provided.

Acknowledgements

I would like to thank my supervisor, Associate Professor Edmund Brekke, who offered insight and project guiding during the thesis work. Thanks also to Associate Professor Egil Eide, for facilitating experimental work on the ferry. My gratitude also goes to my fellow student Brage Sæther, for taking me sailing and logging data. Thanks to Rune Nordmo and Nicholas Dalhaug for interesting discussions around SLAM algorithms and sharing experiences.

Problem Description

For autonomous vehicles to be of practical utility they must be able to leave and approach their docking stations on their own. For this to be done in a safe manner, accurate navigation is necessary. While RTK-GPS often can give sufficient accuracy, it is not possible to depend on this system alone since GPS is not always available. The vehicle must therefore be equipped with other sensors, such as lidar, cameras and/or radio systems. This project will explore navigation methods based on simultaneous localization and mapping (SLAM) for the docking operation of an autonomous ferry that is planned to operate between Ravnkloa and Brattøra. The project includes the following tasks.

1. Conduct a literature survey on SLAM methods for lidar and SLAM for autonomous surface vehicles.

2. Record lidar, IMU and GNSS/RTK data from the ferry in the harbor environment, with a particular focus on docking experiments.

3. Implement SLAM on the recorded lidar data. Investigate the performance that results.

4. Simulate docking based on lidar data.

5. Write report.

Contents

Chapter 1

Introduction

This chapter provides the background and motivation for the problem description on which this work is based. It also swiftly introduces the SLAM problem, outlines related projects at NTNU, and presents the outline of the report.

1.1 Background

For more than a 100 years, citizens of Trondheim have had the need for a "shortcut" over the canal separating Ravnkloa and Fosenkaia on the north side of the city centre. Up until 1965 and seasonally since 1997 this was/ is solved by Fløtmann, a rowboat manned by certified "Fløtmenn".

When the councilman in 2012 opted to upgrade the historical area of Ravnkloa, a lifting bridge for cyclists and pedestrians was considered, and estimated to cost 42 million NOK. Such a bridge would, however, become an obstacle for the surface traffic on the river - as the current would drag vessels departing from the docks towards the bridge in addition to the canal being quite narrow (ca. 95 metres wide). Neither did it make sense to build such an expensive bridge only 200 metres from the next one. Thus, the idea of an autonomous ferry to cross the canal arose.

The Autoferry project at NTNU is responsible for acquiring and developing the technology to make the autonomous ferry a reality. With the main hypothesis of the project being that ferries can operate safely alongside other vessels in confined an congested environments, its verification requires a broad multi-disciplinary approach. Thus, the involved researchers have competence in the fields of control systems, autonomous systems, sensor fusion, robotic vision, instrumentation systems, communication systems, artificial intelligence, cyber security, risk management, power systems and human factors.

There are currently several unmanned cable driven ferries in Norway, and the project aims to replace these by removing the cable and enable them to operate autonomously, and thus open up new possibilities for both urban and coastal transport. As many cities historically are built adjacent rivers, and a large part of Norwegians live by the coast, the markets for such autonomous ferries are potentially vast both nationally and internationally due to high crew cost. No proper market analysis has been conducted, but several Norwegian counties have shown their interest in installing such a vessel.

The ferry on the canal is planned to operate much like a horizontal elevator, where it will arrive from the other side of the canal if a designated button is pushed on the dock. It will carry 12 passengers along with their bicycles and/or wheelchairs, using approximately 1 minute to cross the canal. The vessel will use electrical drive drawing power from a battery which will be charged by induction at the dock. It will primarily navigate using high precision GNSS, and will for safety reasons be equipped with an anti-collision system and at least one back-up navigation systems in case the primary should fail.

Figure 1.1: Author and co-student doing tests on MilliAmpere some time during spring 2018.

1.2 Motivation

A failure in the primary navigation system could occur if i.e. the GNSS services should become unavailable or unreliable. This can happen both because of a fault in the on-board GNSS-receiver, the service itself being faulty, and a fault in the base station in the case of RTK being implemented. Reasons for the faults could be hardware errors, jamming/spoofing, solar activity or good old human errors.

There was for example an instance related to retiring an old satellite in January 2016, where multiple other satellites for unknown reasons suddenly broadcast a UTC-signal off by 13 microseconds for several hours [38]. Though this error had no direct impact on positioning and navigation, it indirectly affected systems relying on GPS-time for synchronization between hardware modules, and GPS-receivers programmed to reject broadcasts with a time-step larger then said error.

Should the GNSS-service for any reason be deemed unfit or impossible to use, the autonomous ferry would be left to navigate using an inertial measurement unit (IMU) only. This is problematic because IMU-measurements are noisy, and their errors are integrated when computing the states. Thus, the solution will drift, and navigation system is not satisfying for high precision operations. Hence a back-up is needed, and it could well be based on SLAM, for which reasons given in the next section.

1.3 SLAM

1.3.1 The SLAM Problem

Simultaneous localization and mapping, mostly known by its acronym *SLAM*, refers to the problem of creating and updating a map of an area, while simultaneously determining the location of the device at work within said map. Finding a solution to this problem is considered necessary for a mobile robot to be truly autonomous. It has been formulated and solved as a theoretical one in a number of different forms, and has been implemented in a number of different domains - including indoor robots and outdoors-, underwater- and airborne systems. Substantial issues remain, however, in realizing a general SLAM solution and in building and using perceptually rich maps as part of a SLAM algorithm.

Consider a mobile robot moving through a static, unknown environment. If the pose of the robot is known with certainty using for example GPS, we have a straightforward mapping problem. Vice versa, if the robot with certainty knows the positions of objects in its environment, we have a localization problem. In such situations, *Monte Carlo Localization* could be a fitting algorithm for generating position estimates [6]. However, when both the pose and the map are unknown, motion noise corrupts the pose estimates, which in turn corrupts perceived locations of objects in the world (landmarks) along with sensor measurement noise. In other words; error in the robot's pose correlates errors in the map, and thus map and robot pose must be estimated simultaneously.

SLAM has many applications in modern robotics. As mentioned, the technology can be utilized in situations where other forms of position estimation techniques are unavailable or inaccurate, such as operations indoor, sub sea underground or on the surface of other planets. SLAM can also be used for the purpose of providing maps itself. Examples are during urban search and rescue (USAR) situations [13] and in- and outdoor environment surveying [37]. Mobile robots utilizing SLAM are in fact becoming a common household article, as autonomous vacuum cleaners such as iRobots Roomba [46].

1.3.2 Why SLAM for autonomous ferry

Because the SLAM problem does not necessarily require GNSS-service to solve, a SLAM method can be a suitable back-up navigation-system for the autonomous ferry. If a solution with satisfying accuracy can be found, it would provide the ferry with at least the means to stay safe until help/maintenance arrives, though preferably be able to navigate towards its docking station on its own.

Additionally and in particular, SLAM could prove valuable in docking operations. It is in general a likely scenario that an autonomous ferry will use floating docks for battery recharging, passenger/cargo loading- and unloading. As the position of many floating docks change with the tide, relying on a saved GNSS-reference might not be a viable solution. SLAM could be a better choice in this scenario, though it would depend on the docking position being recognizable and a method for maneuvering towards it.

Another reason why an autonomous ferry would benefit from a SLAM solution is the possibility of combining it with an inertial navigation system to achieve more accurate estimates than either can do on their own. This can for instance be done with an Extended Kalman Filter (EKF) [30].

1.4 Previous Related Work by NTNU Students

Numerous projects and master theses involving SLAM have been conducted at NTNU in recent years. SLAM software has been implemented on LEGO-robots [31], and attempts have been made on making several LEGO-robots cooperate in performing SLAM algorithms indoors [7], [28]. Mapping based on extracting line segments from IR-sensor data and navigation based on a wall-following algorithm has been implemented on an NTX-robot as well.

[40] and then [41] implemented the Hector SLAM algorithm [38] successfully on an indoors unmanned ground vehicle (UGV) and an indoors unmanned surface vehicle (USV). [41] served as a precursor for this thesis and its pre-project where the same algorithm has been implemented. [41] performed Hector-SLAM on small scale in a controlled and appropriately self-structured environment, and was able to explore unknown areas using the created map. This was in [39] taken to some extent a step further, where the SLAM algorithm was implemented on on real-life data in a harbour environment.

[42] and [43] developed a Dynamic Positioning control system (DP) for a scale model of DNV GLs concept ship ReVolt with the purpose of developing the technology for an autonomous ferry. Through that work, a survey of appropriate GPS receivers and Inertial Measurement Units (IMU) were conducted, and the sensors were acquired. The same sensors were used in the data-collecting experiment described in Chapter 4, along

with the driver software installed in [42].

1.5 Outline

The thesis is organized as follows: Chapter 2 introduces SLAM, basic concepts and challenges therein. Chapter 3 presents various lidar-based open-source SLAM algorithms. Chapter 4 describes hardware used and the data-logging procedure, while Chapter 5 gives short descriptions of software used in the work.

Chapter 2

Simultaneous Localization and Mapping

While a short introduction to the SLAM problem was given in the previous chapter, we will now explore some of the aspects within the SLAM problem at a deeper level. Surveys of possible map representations and sensors are given, followed by descriptions of techniques to help solve challenges in SLAM.

2.1 Map Representations

Several map representations are recognized as suited for SLAM purposes, and literature broadly divides them into *metric* and *topological* map representations [9].

2.1.1 Metric map representations

Metric maps capture the geometric properties of the environment, and among them *occupancy grid maps* and *landmark-based maps* are considered.

Occupancy Grid Maps

Occupancy grid maps are discrete cell maps which may contain 2D, 2.5D or 3D information. They are advantageously very visually comprehensable to human eyes, as they resemble actual regular maps, or floor-plans in particular (in 2D especially).

The grid cells can be thought of as pixels in an image. Each cell has a predefined location corresponding to a real-world location and a predefined size, where the latter corresponds to the resolution of the grid map. Within the cells we find information regarding the probability that the corresponding real-world location contains free or occupied space. The information can i.e. be expressed as an estimated *log-odds*, binary variable [13], or a tree structure([45], [53], [46]) determining whether the cell is occupied or not.

In addition to grid maps being easy to interpret by humans, the map representation is advantageous when performing automatic path planning. A third advantage is the fact that grid maps can be created for practically any resolution, allowing for maps with the level of detail appropriate to the application and its requirements.

A possible disadvantage of occupancy grid maps are the large amount of memory they can require. Operating a large number of sizable maps with high accuracy can require significant amount of memory, even in modern computers. As mobile robots often have limited size and therefore smaller computers, this might require some trade-off.

Landmark-based Maps

Landmark-based maps identify and maintain the location of certain features in the environment. These may be corners, line segments or points - and are called *landmarks*.

A great advantage with landmark-based representation versus grid maps, is the compactness of the model. As only feature points isolated from the structure of the environment are considered as landmarks, memory resources and computational costs are minimized [9]. This is clear if one interprets grid map versions to treat every occupied cell as a landmark. Because of this, landmark-based maps are well suited for managing more than one version of the map at the time, allowing for multi-hypothesis tracking [6].

On the other hand, due to the foregoing, the map representation is not suited for automatic obstacle avoiding and path planning. This is because, contrary to that in occupancy grid maps, the lack of a landmark does not equal a free, unoccupied space.

What makes SLAM using landmark-based maps difficult, is the problem of *data association*. Data association concerns recognizing observed landmarks correctly - that is, either the correct previously observed landmark, a new landmark or not a landmark at all. Errors on the data association can lead to catastrophic consequences in that the SLAM method fails. When choosing landmarks, [44] gives a shortlist for which candidates to appoint as landmark:

- Landmarks should be easily re-observable, meaning they should be detectable from different positions and different angles.
- Individual landmarks should be unique enough so that they are distinguishable from

each other.

- Landmarks should be plentiful in the environment, so that the robot is in no risk to spend extended time without any visible, and thus may get lost.

- Landmarks should be stationary. If not, the robots relative location to the landmark is useless.

2.1.2 Topological Map Representations

Topological maps represent the environment as a list of significant places that are connected by arches - in other words, a *graph*. Such a representation of the world simplifies the the problem of mapping large extensions [9], but the lack of metric information makes it unfit for guiding an autonomous vehicle. Both these and landmark-based map representations are, however, excellent if the main purpose is accurate localization rather than mapping or both.

2.2 Possible Sensors

A wide range of sensors can be used for SLAM purposes, and different sensors are used in different contexts. A wheeled indoors robot would not use the same means as an unmanned aerial vehicle (UAV) or an autonomous underwater vehicle (AUV). As a USV with lidar, IMU and GNSS is used in the scope of this thesis, this context is given the most focus, although some other relevant possibilities are presented as well - in consideration with the application at hand.

Sensors broadly fit into two main categories: exteroceptive sensors and interoceptive sensor. The former looks "outside" the robot generate absolute position measurements, while the latter looks "inside" the robot to measure relative position.

2.2.1 Exteroceptive Sensors

Exteroceptive sensors look "outside" the robot to generate absolute position measurements.

Range finders

Range finders tell the distance to the nearest object in the nearest direction or sector. Examples are sonar, radar and lidar based systems.

Not taking cameras into account, sonars, or ultrasonic sensors are generally the cheapest source of spacial sensing for mobile robots [47]. They are particularly for underwater purposes, though simply for the reason that other sensors such as cameras or lidar/radar struggle there. The low frequency sound waves of sonars minimizes absorption, which makes the resolution better in sub sea applications. Ultrasonic sensors are also popular in indoors robotics, were distances are relatively small. Ultrasonic sensors are compatible with most surfaces, but the low spatial resolution and sensing range combined with slow response and in particular sensitivity to environmental factors, raises questions whether they would suit a SLAM method for an autonomous ferry.

The RADAR system and the LIDAR system work much in the same way, with the only difference being the use of radio waves in the former, versus light in the latter. They emit rapid radio or laser signals, the signals bounce off obstacles and returns. By calculating the time of flight (ToF), the distance to the objects can be estimated with high accuracy.

Both LIDAR and RADAR systems are compatible with autonomous vehicles. The fact that the autonomous Tesla car relies on RADAR as the primary sensor, and Google, Uber and Toyotas versions rely on their light emitting rival, substantiates that statement [3]. Some differences in service delivered and price do, however, exists.

First of all, RADAR systems are relatively less expensive than LIDAR systems, although the difference was much larger a few years ago. The former also works equally well in all weather conditions, whereas the performance of LIDARs starts to dwindle in snow, fog, rain and dusty weather conditions. As RADAR systems can utilize the Doppler frequency shift to accurately determine relative traffic speed, it would seem that this would be the better choice. However, what might make LIDAR sensors a more popular for SLAM purposes, is the fact that high-end ones can identify the details of a few centimeters at more than 100 metres [3]. This is advantageous for landmark extraction in particular.

LIDAR sensors come with a range of different specifications. Price will obviously be an important question, as will the dimension of the sensor workspace (2D or 2.5D). Other important metrics when choosing a LIDAR system are horizontal and vertical Field of View (FoV), horizontal and vertical resolution and the time it takes for the LIDAR to complete one revolution. One LIDAR revolution is hereby called *scan*.

Vision Sensors

These sensors mainly come in three types: monocular cameras, stereo cameras and RGB-D cameras. Contrary to LIDARs, the sensory data of two former do not contain range data. The information is however achievable at the price of some computational cost. Although RGB-D directly compute the depth in the images, their sensitivity to illumination makes them unfit to receive direct sunlight. As the dark of night would likely cause important information to be imperceptible to the cameras, it is unclear how well suited these are as the main exteroceptive sensor for a SLAM system on an autonomous ferry. Infrared cameras would definitely be needed.

Vision sensors are however advantageous over range finders in the rich visual information available in the images. This allows for sparse key-point extraction using appropriate feature extraction methods. These key-points are more distinctive than geometric structures such as corners and edges [47], making them less susceptible for data association errors.

Working with images also allows for treating landmarks as a classification problem using deep learning. A set of landmarks of known geometric characteristics and sizes could be used to train networks such as [29], versions of R-CNN or SSD [2] to classify landmarks, and then compare image with the real sizes. Furthermore, it has been shown that deep neural networks can be trained to find the relative pose between two images, directly from the image pair [16].

GNSS

Although data from GNSS-services really relates better to the upcoming proprioceptive sensors, they per definition belong here. Both GNSS position estimates, GNSS compass heading estimates and GNSS time can be relevant information for a SLAM solution.

2.2.2 Proprioceptive Sensors

Proprioceptive sensors measure values internal to the system to generate relative position. Wheel odometry in the form of encoders or hall sensors are popular with wheeled robots, as they give a pretty accurate estimate on distance travelled. As these sensors measure wheels spinning, they are not possible options for an autonomous ferry.

IMUs however, are, and can be used to get an odometric estimate. They consist of a gyroscope and an accelerometer measuring angular velocity and linear acceleration, respectively. Integrating these data give the required angles, velocities and position estimates. However, as mentioned in chapter 1.3, the estimates are error prone. For this reason, a solution based on *dead reckoning* will drift, and the estimate are best to give

starting points for scan matching algorithms, or fused with other sensor data i.e. GPS pose estimate or SLAM pose estimate.

Note
SLAM has been implemented using several other means of sensing as well. [18] presents a SLAM solution using tactile sensors. That is, the robot sensed the environment through touch only. Other examples are using range measurements from radio beacons [33], and measuring strength of WIFI-signals from access points in the vicinity [10].

2.3 Feature and Scan Matching

As mentioned on visual sensors, matching features in two successive images can give e good estimation of the relative pose between them. The features can be extracted using methods like the *Harris Corner Detector*, *Scale Invariant Feature Transform* (SIFT) or *Speeded Up Robust Features* (SURF) [2].

Likewise, examples of feature extracting algorithms for laser scanners like LIDARs are *Spike Landmark extraction* and *Random Sampling Consensus* (RANSAC) [44]. Spikes are extreme value differences in the distance to a location measured from different angles. This indicates a geometric change between the angles, and the location can be interpreted as a landmark. RANSAC attempts to extract straight lines from a laser scan, and interpret them as landmark. The landmarks are found by taking a random sample of the laser readings and do a least squares approximation to find the best fit line running between them.

2.3.1 Scan Matching

Scan matching is a concept frequently used in SLAM algorithms([69], [50]), and some solely rely on it [13]. The concept is combining range measurements from one scan to the next. The result can be used to estimate the transformation between them. Is the matching done with an existing map, new scans are implicitly matched with all preceding scans [13].

There exist numerous ways to perform scan matching, with *Iterative Closest Point* (ICP), *Polar Scan Matching* (PSM) and *Normal Distribution Transform* (NDT) seemingly being of the most popular ones. A couple more will be presented in Chapter 3.

ICP is originally a general approach for registering 3D point clouds [13], an apprach with which work on scan matching based localization started in 1992 [51]. Variations of the method are many, but the basic idea is to iteratively refine the relative pose of two overlapping scans by minimizing the sum of squared distances between the corre-

sponding points in the two scans. By thresholding distances between matched scans and their change of orientation, the method seeks to eliminate false matches. Using Nearest Neighbor methods, ICP's search for point correspondences is, however, expensive [13].

PSM exploits the natural polar form of the scans, making search for corresponding points less computationally demanding [14]. However the method prepossesses both the current and the reference scan in order to remove objects likely to move. Still, [14] shows that the PSM approach can be faster than ICP, and shows its application to the SLAM problem through a Kalman Filter implementation.

NDT models the distribution of all reconstructed 2D-points of one scan by a collection of local normal distribution [1]. In order to estimate a geometric transform relating to the new scan, a measure is defined by mapping all points according to the transform. The corresponding local normal distribution are then evaluated, and the sum of them are maximized through a Newton's method optimization. The main idea with the transform is that by matching piece-wise continuous and differentiable probability densities, no explicit correspondences have to be established. NDT is successfully applied to the SLAM problem in mid-size indoor environments by [1]. To the author's knowledge, a question still remains if outdoor environments can be modelled good enough by local normal distributions, as well.

2.4 EKF-SLAM

Kalman Filters have numerous applications in technology, and is commonly implemented in navigation and control for vehicles, but also in fields such as signal processing and econometrics. Concisely summarized, the Kalman Filter is an algorithm using a series of measurements containing statistical noise and other inaccuracies over time, and produces estimates of unknown variables by estimating an assumed Gaussian-linear joint probability distribution over them for each time frame.

The use of an extended Kalman filter to solve the SLAM problem (EKF-SLAM) was first introduced in an article from 1986 [21]. This approach models the motion and observations as state space models with additive gaussian noise, uses a high dimensional EKF to provide a posterior over a map and the robot pose [34]. A detailed description of EKF-Slam can be found in [8].

As EKFs linearizes non-linear models about the most likely states, convergence and consistency can only be guaranteed in the linear case. This approximation is however good if the true models are approximately linear and if the time step is sufficiently small [34].

In two dimensional space, the map is typically implemented as a set of landmarks, with a representation of two states in the EKF for each landmark. Adding the three states representing the robot pose, we have a $2N+3$ dimensional state-space for N landmarks. Thus,

as the map grows large, the complexity of the EKF representation grows quadratically both in memory and computational time. This is because the joint covariance matrix must be updated at every observation in order to maintain the correlations between all state variables.

Another challenge with the EKF scheme, is related to data association and was discussed in section 2.6.

2.5 Particle Filters and Rao-Blackwellization

Like Kalman filters, particle filters represent algorithms for estimating state probability distributions. The representation of these probability distributions, however, differ. While Kalman filters represent them using multivariate Gaussians, Particle filters use *particles*. These particles can be thought of as samples of an unknown posterior distribution [23].

In its simplest form, the particle filter is a two-step algorithm. As a measurement is made, the current set of particles are given an *importance weight* based on how well they coincide with said measurement. This way, areas of high probability contain more particles than areas of low probability. This is achieved through the following *resampling* step, where a new set of particles is arbitrarily chosen. The new particles are chosen with a probability proportional to their assigned importance weight. A more in-depth description of the filter is given in (probabilistic robotics).

Using enough particles, the particle filter can approximate arbitrarily complex and multimodal probability distributions. The *Monte Carlo localization* approach (MCL) takes advantage of this, and is a fast, accurate and memory-intensive global localization method in which the environment is known [19].

In their native form, particle filters scale badly in the number of dimensions, as the number of particles needed grows exponentially in the number of dimensions of the estimation. Further, as a solution to the SLAM problem, a particle filter would consider each particle as a position and a map linked to it, leading to sampling over all possible maps and positions.

However, an important observation introduced by [26]: the fact that because of the conditional dependencies of the SLAM problem, correlations between elements of the map only arise through pose uncertainty. This makes it possible to estimate landmarks independently, and a particle filter can factorize the SLAM posterior $P(\boldsymbol{x}_{0:t}, m|\boldsymbol{z}_{0:t})$ into

$$p\boldsymbol{x}_{0:t}, m|\boldsymbol{z}_{0:t}) = p(\boldsymbol{x}_{0:t}|\boldsymbol{z}_{0:t}) \cdot p(m|\boldsymbol{x}_{0:t}, \boldsymbol{z}_{0:t}) \tag{2.1}$$

where $\boldsymbol{x}$ denotes the set robot poses, or the trajectory if you will, m denotes the map and $\boldsymbol{z}$ denotes the set of observations made. The first factor can be thought of as the path posterior, and the second, the map posterior.

Introduced in [32] in 2002, *FastSLAM* utilizes this factorization. FastSLAM calls it *Rao-Blackwellization*, because it is an instance of the Rao-Blackwellized particle filter [4]. Contrary to EKF-SLAM with its one large covariance matrix, FastSLAM keeps an extended Kalman filter of each landmark, in each particle. At the same time, the scheme can boast of logarithmic complexity. This was a significant upgrade at the time of introduction, as most attempts to solve the SLAM problem up until then was EKF based, with its quadratic complexity [34].

One other advantage with FastSLAM, is the natural multi-hypothesis data association. The data associations are done per particle, and if things are working as intended, particles with the correct information get higher importance weight, while others tend to die in the filter.

FastSLAM has shown capabilities of creating an accurate map, and an example run on the popular Victoria Park data-set can be seen in [34]. Although the algorithm originally is landmark-based, it can be modified to use grid-maps, e.g. [15] and [45].

2.6 Loop-Closure

Loop-closure detection is crucial for enhancing the robustness of both topological and metrical SLAM algorithms [25]. The loop closure problem consists of detecting when the robot has returned to a previously visited location after having discovered new terrain, and recognizing which part of the constructed map it represents. Of all sub-problems in SLAM, this is considered a harder one [15].

Loop closure can be handled explicitly by correcting backwards in time, as in ([53],[50],[15]). Some SLAM algorithms deal with the problem implicitly through built-in mechanisms such as ([48], [52], [5], [6]). Other algorithms do not take the problem into account, and relies on them being accurate enough to avoid it. Examples of such algorithms are ([13], [27], [49], [11]).

When a SLAM algorithm fails to handle loop closure, the same physical location is represented at two or more different locations in the map. This leads to map inconsistencies which can e.g. be seen as double walls. Such map inconsistencies cause problems for path planning, although the robot pose estimate might still be sufficient.

2.7 Graph-SLAM

A key disadvantage of filter techniques such as those discussed in Section 4 and Section 5, is that data is discarded after processing. This removes the possibility of revisiting relevant

data while building the map, in case corrections are in order. GraphSLAM, introduced by [48] seeks to avoid this issue by exploiting that the posterior of the full SLAM problem naturally forms a sparse graph. The graph leads to a sum of nonlinear constraints, which when optimized yields a maximum likelihood map with a set of corresponding robot poses.

The edges in the graph can be thought of as motion edges and measurement edges, where the former links to poses and the latter a pose to an observed map feature. The edges all correspond to a nonlinear constraint, named by [48] as *information constraints*. These information constraints sum up to a nonlinear least squares problem. The GraphSLAM scheme linearizes this set of nonlinear constraints, resulting in a sparse information matrix and an information vector. The matrix is then reduced using variable elimination techniques, which lowers the dimensionality of the optimization problem, defined only over robot poses. The path posterior map is then recovered using standard inference techniques.

Chapter 3

Explored Open-Source SLAM Algorithms

In this chapter, summaries of the SLAM algorithms performed on the recorded data is given. Following that, short introductions of algorithms that where investigated, but for various reasons discarded.

3.1 Hector SLAM

Kohlbrecher, von Stryk, Meyer and Klingauf presents a SLAM approach in [13], developed for and UGV built to compete in the 2011 urban search and rescue (USAR) competition Robocup. As USAR typically include roll and pitch motion along with a structured environment, the paper presents a use case where inertial measurements are combined with LIDAR measurements for estimating full 3D motion.

The open-source implementation of the SLAM algorithm [24], however, solely depends on LIDAR measurements. Further, because a motivation for the approach was to be compatible with low-weight computational units, the implementation assumes accurate, high-resolution and high-frequency range measurements. Although the algorithm supports full 3D motion estimation, it performs 2D mapping. It returns a grid map and a pose for every iteration, and saves the trajectory of the robot. The method has been named *Hector-SLAM*, and will be referred to by that name for the rest of this thesis.

The underlying principle of Hector-SLAM is scan matching. The scan matching procedure used is purpose-designed, based on ideas in image registration for computer vision originally presented in (an iterative image registration matching for stereo vision), namely regarding image matching for stereo vision. The algorithm initializes by writing the first received scan to the map, and subsequent scans are matched with the map by seeking to

find the rigid transformation of robot pose that minimizes

$$\boldsymbol{\xi}^* = \arg\min_{\xi} \sum_{i=1}^{n} [1 - M(\boldsymbol{S}_i(\boldsymbol{\xi}))]^2 \tag{3.1}$$

where $\boldsymbol{S}_i(\boldsymbol{\xi})$ are the world coordinates of scan endpoint $\boldsymbol{s}_i = (s_{i,x}, s_{i,y})^T$:

$$\boldsymbol{S}_i(\boldsymbol{\xi}) = \begin{bmatrix} cos(\psi) & -sin(\psi) \\ sin(\psi) & cos(\psi) \end{bmatrix} \begin{bmatrix} s_{i,x} \\ s_{i,y} \end{bmatrix} + \begin{bmatrix} p_x \\ p_y \end{bmatrix}. \tag{3.2}$$

$M(\boldsymbol{S}_i(\boldsymbol{\xi}))$ returns the occupancy value of the input world coordinate. With a starting estimate ξ given by the navigation filter, the algorithm wants to estimate a $\Delta\xi$ which optimizes the error measure according to

$$\sum_{i=1}^{n} [1 - M(\boldsymbol{S}_i(\boldsymbol{\xi} + \Delta\xi))]^2 \to 0. \tag{3.3}$$

After performing a first ordered Taylor expansion of $M(\boldsymbol{S}_i(\boldsymbol{\xi} + \Delta\boldsymbol{\xi}))$ and minimizing by setting the partial derivative with respect to $\Delta\boldsymbol{\xi}$ to zero, solving for $\Delta\boldsymbol{\xi}$ gives the Gauss-Newton equation for the minimization problem:

$$\Delta\boldsymbol{\xi} = \boldsymbol{H}^{-1} \sum_{i=1}^{n} \left[\nabla M(\boldsymbol{S}_i(\boldsymbol{\xi})) \frac{\partial \boldsymbol{S}_i(\boldsymbol{\xi}}{\partial \boldsymbol{\xi}}\right]^T [1 - M(\boldsymbol{S}_i(\boldsymbol{\xi}] \tag{3.4}$$

where the Hessian matrix denotes:

$$\boldsymbol{H} = \left[\nabla M(\boldsymbol{S}_i(\boldsymbol{\xi})) \frac{\partial \boldsymbol{S}_i(\boldsymbol{\xi})}{\partial \boldsymbol{\xi}}\right]^T \left[\nabla M(\boldsymbol{S}_i(\boldsymbol{\xi})) \frac{\partial \boldsymbol{S}_i(\boldsymbol{\xi})}{\partial \boldsymbol{\xi}}\right]. \tag{3.5}$$

Equations 3.4 and 3.5 both rely on the gradient of the map at some coordinate P_m: $\nabla M(P_m)$. The occupancy value $M(P_m)$ is approximated using bi-linear filtering of the four closest integers representing the midpoints of the map cells of P_m. The interpolation scheme yields:

$$\begin{aligned} M(P_m) \approx \frac{y - y_0}{y_1 - y_0} \left(\frac{x - x_0}{x_1 - x_0} M(P_{11}) + \frac{x_1 - x}{x_1 - x_0} M(P_{01}) \right) \\ + \frac{y_1 - y}{y_1 - y_0} \left(\frac{x - x_0}{x_1 - x_0} M(P_{10}) + \frac{x_1 - x}{x_1 - x_0} M(P_{00}) \right) \end{aligned}. \tag{3.6}$$

The gradient is then approximated by

$$\begin{aligned} \frac{\partial M}{\partial x}(P_m) &\approx \frac{y - y_0}{y_1 - y_0}(M(P_{11}) - M(P_{01})) \\ &\quad + \frac{y_1 - y}{y_1 - y_0}(M(P_{10}) - M(P_{00})) \\ \frac{\partial M}{\partial y}(P_m) &\approx \frac{x - x_0}{x_1 - x_0}(M(P_{11}) - M(P_{01})) \\ &\quad + \frac{x_1 - x}{x_1 - x_0}(M(P_{10}) - M(P_{00})) \end{aligned}. \tag{3.7}$$

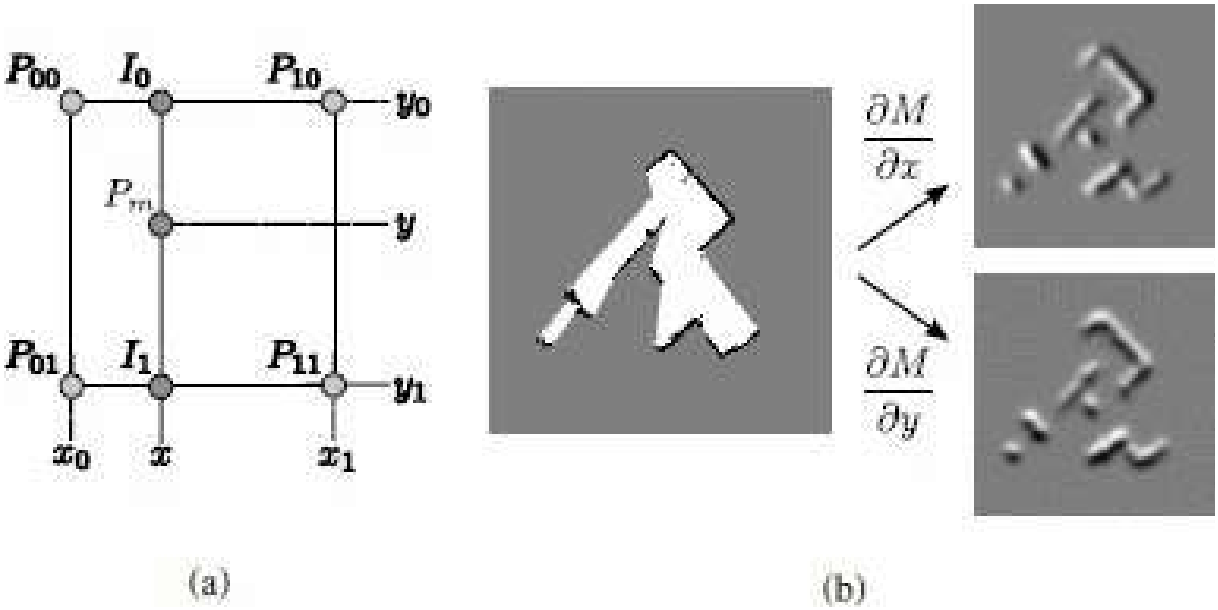


Figure 3.1: (a) Bi-linear filtering of the occupancy grid map. (b) Occupancy grid map and its spacial derivatives. Courtesy of [13]

Being a gradient decent-based approach, the scan-matching algorithm has a risk of getting stuck in a local minimum, which is a source of error. The method mitigates this through another approach inspired by computer vision - namely *image pyramids*. Maps are stored at different resolutions, as shown in Figure 3.2. The generation of different resolution is not done by down-sampling, but by producing scaled versions of the current scan. This is more computationally effective and ensures consistency across the layers. The scan matching procedure is performed on all levels of this image pyramid structure. Starting the matching at the coarsest layer, its resulting estimate is used as a starting estimate for the next. $\hat{\boldsymbol{\eta}}$ is assumed to be more precise as the matching works its way towards the final representation. Position data can be used to aid the algorithm in providing a starting

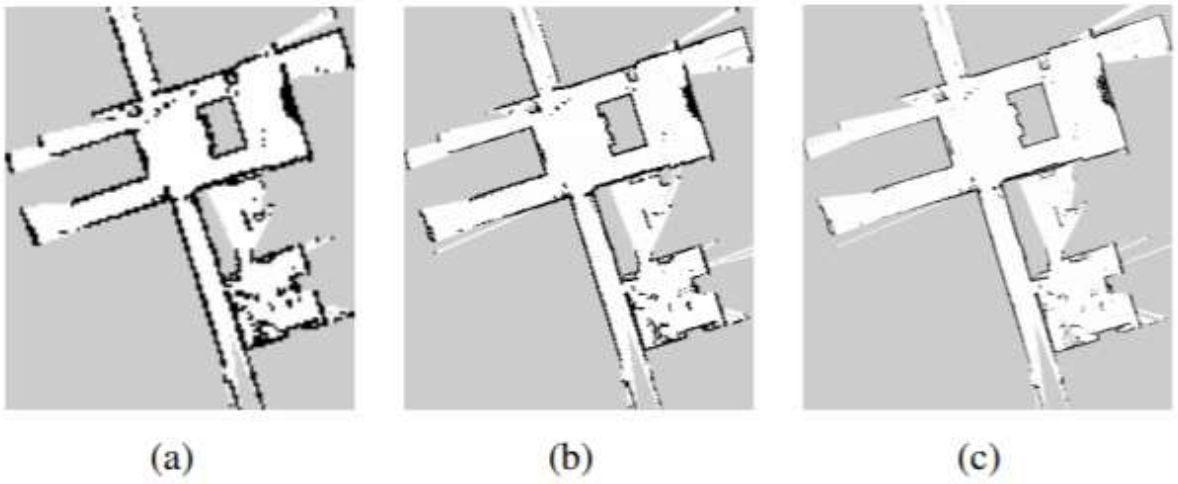


Figure 3.2: Image pyramid structure in Hector-SLAM, where the same map is generated and stored at multiple resolutions. Courtesy of [13]

estimate for the scan matching through a Kalman filter or other sensor fusion methods.

In turn, the odometry result of the scan matcher can be fused with the belief state through covariance intersection(using covariance intersection for SLAM), to maintain the full correlation structure between the robot and the map.

The Hector-SLAM does not possess mechanisms to handle loop-closure. Still, the algorithm has shown promise as an accurate and relative to others, quite simple approach to the SLAM. Promising results for indoor and outdoor UGV-mapping can be seen in (hector-paper), for indoor USV-mapping in a structured environment in (Ueland). Preliminary tests for USV-mapping in harbour environment was done in (project)

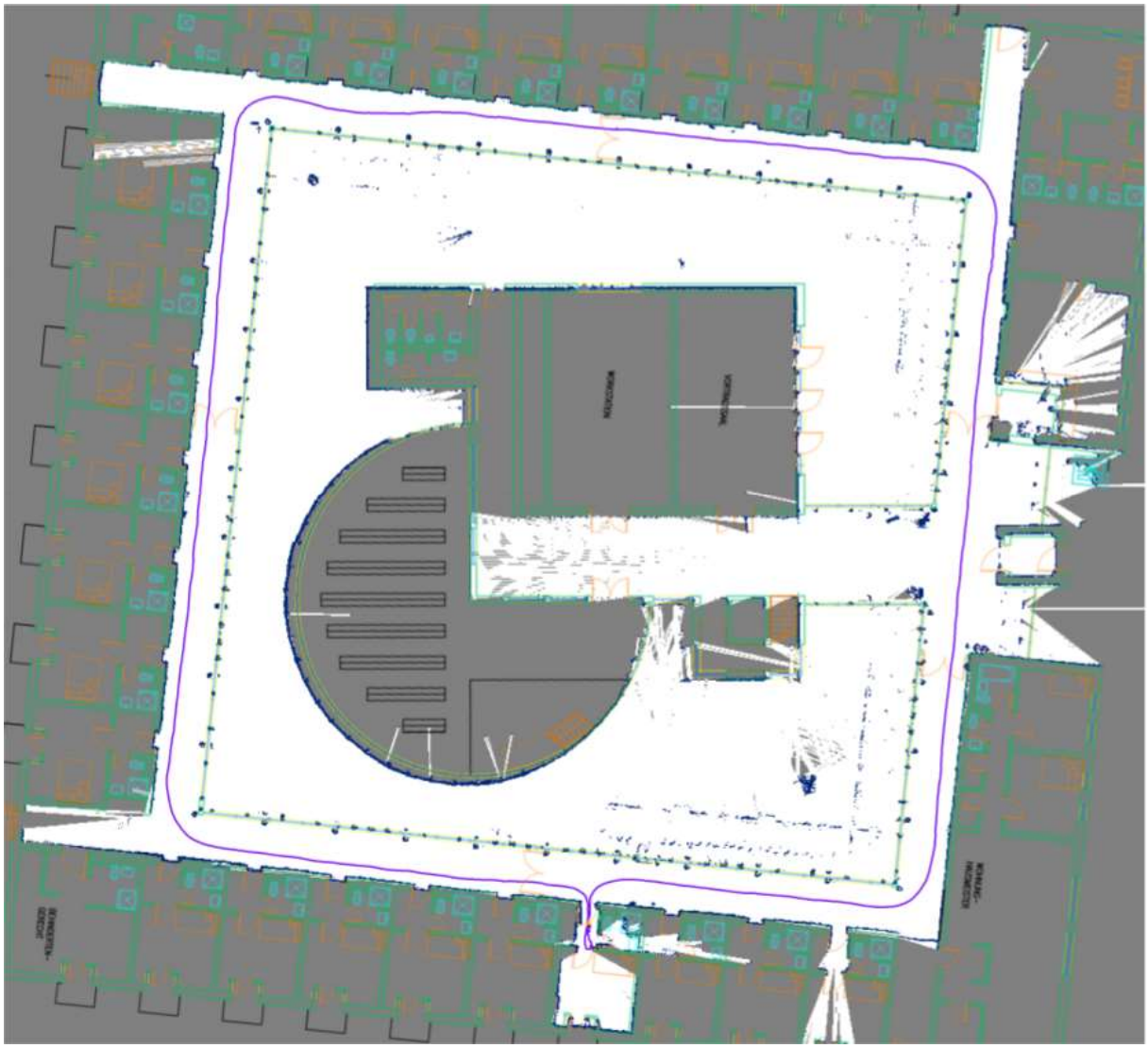


Figure 3.3: Example of Hector-SLAM map output overlayed with floor plan data. Courtesy of [13]

3.2 LOAM

Laser Odometry and Mapping (LOAM) was presented by Zhang and Singh [11] in 2014, and is a real-time algorithm for state estimation using a 3D Lidar. To date, LOAM achieves the best odometry-estimate using the KITTI benchmark data set of all range-only SLAM solutions. Thus, it should be considered state of the art. This SLAM solution achieves low/drift and low/computational complexity [11] through dividing the SLAM problem into two concurrent algorithms. One algorithm performs odometry at a high frequency and low fidelity to estimate the velocity of the 3D Lidar. The other runs at a $\frac{1}{10}$ of said frequency for fine matching and point cloud registration.

Both are based on feature extraction and feature matching.The features extracted are points located on sharp edges and planar surfaces. The odometry algorithm returns a 6-DOF rigid motion through finding correspondences between edges and edge line segments, and correspondences between planar surfaces and planar surface patches. To ensure fast computation, the features extracted are limited.

LIDARs are spinning as they emit laser beams. If the LIDAR experiences motion, as they will if mounted om mobile robots or vehicles, a range measurement at 0 degrees will be made at e different robot location than a range measurement at say 359 degrees. This leads to a distortion of the resulting point cloud registered when a LIDAR revelation is complete. The odometry algorithm uses said 6-dof rigid motion to create an undistorted version of the point cloud, and feeds this to the mapping algorithm.

The mapping algorithms then performs feature extraction on the point cloud, only this time 10 times as many. The correspondences this time, are found by examining geometric distributions of local point clusters, through the associated eigenvalues and eigenvectors. A rigid transformation is obtained from the mapping algorithm as well, and the lidar pose with respect to the map is a combination of the two transformations, calculated at the same frequency as the odometry algorithm. Further information on the feature matchers are found in [11].

LOAM can be supplemented with IMU data if available. This can help reduce the error by fusing simulated orientation data and orientation integrated from gyro sensor data, as well as disposing unnecessary frames. However, unless the LIDAR experiences very large motions, this is stated by (loam-paper) as unnecessary.

Additional advances has been made to the LOAM approach since the release of the 2014 paper, and more sophisticated SLAM algorithms have been incorporated into the solution. However, the author has since commercialized the findings through self-founded KAARTA, formerly Real Earth. The company delivers products for real-time mobile mapping [37]. A result of this is that open-source code related to LOAM has been removed, and is officially unavailable to the public. What is available, is laboshinl's Git repository linked to [12], who wrote his master thesis on the subject of Quantifying Aerial LIDAR Accuracy of LOAM for Civil Engineering Applications. The repository contains

a copy of the source code for LOAM, modified to the Velodyne VLP-16 LIDAR, as it originally was designed for a custom made rotating 2D-LIDAR. Finally, the source code used in this thesis is slightly further modified version daobilige-su's git, which contains a small bug-fix related to the odometry algorithm.

The "available" LOAM-version shows promising map building both for UGVs indoors and outdoors [11] and for UAV. Note that the goal of the algorithm is primarily mapping, and loop closure is not accounted for. However, since the pose of the LIDAR is provided through the transformations, the algorithm was deemed worth testing.

Figure 3.4: Example of map built by LOAM. Courtesy of [59].

3.3 BLAM

BLAM is in structure quite similar to LOAM, with a scan matcher determining odometry running concurrently with a scan matcher adding scans to the map. No real documentation of the method is available other than instructions in the git repository [53]. Thus, the following description is based very much on code. The algorithm is mainly chosen for testing due to its inclusion of loop closure mechanisms and impressive demonstrations in [54].

The BLAM SLAM-solution can be viewed as a series of sub tasks performed each time a new point cloud is received from the LIDAR sensor or the bag. First, a series of filters are applied, including downsampling through a random downsampling filter and a voxel grid filter, and removal of outliers through identifying statistical outliers and points without a sufficient number of neighbors within a given radius. Once filtered, odometry is estimated using ICP. The estimated rigid transformation is then transformed to the world frame,

which is then used to transform the filtered point cloud. With this point cloud as basis, a nearest neighbour search in the existing map is performed as well.

Further those nearest neighbours are transformed back to the sensor frame, and localization in the map is found again using ICP. Before updating the map, the pose is checked for loop closures. For this, BLAM uses GTSAM [5] as a back-end. The transformed pointcloud is attempted matched with point clouds from nearby poses using ICP. A pose-graph is maintained along with current loop-closure radius. If a loop closure is found, the map is regenerated before inserting new points to the map.

Contrary to LOAM, BLAM computes loop closures. This is done using GTSAM [5] as a back-end, ICP scan matching with nearby poses.

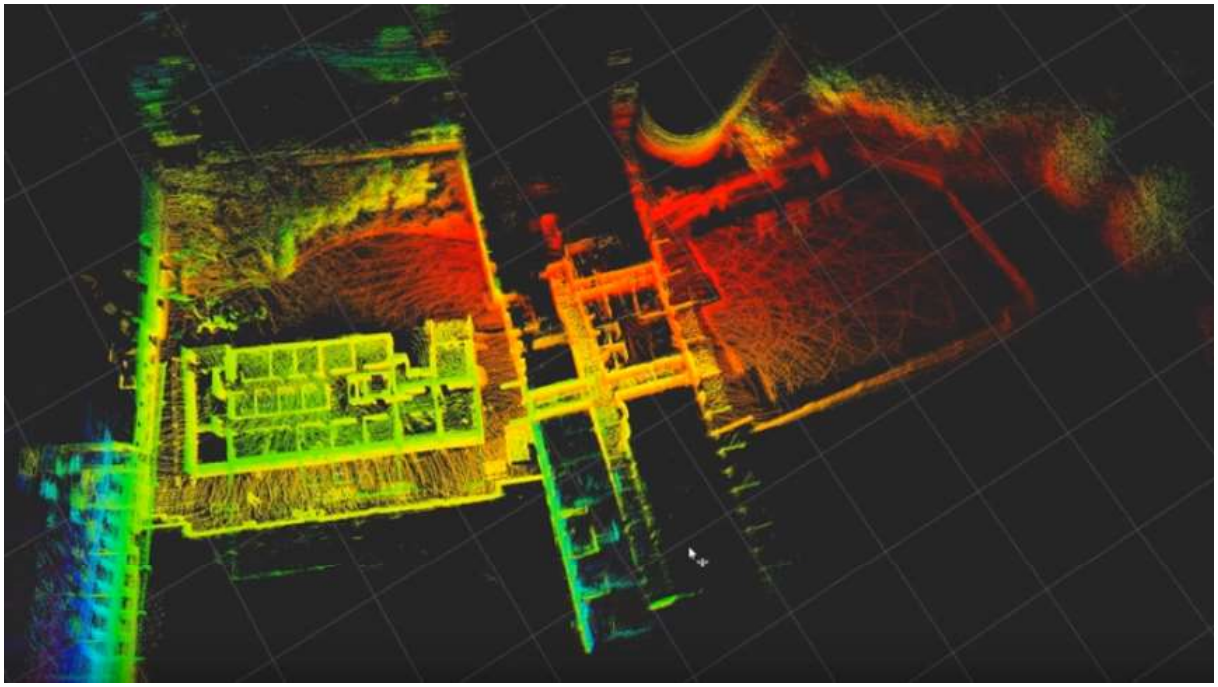


Figure 3.5: Screenshot of map built by BLAM using a Velodyne VLP-16 lidar, placed on top a mans head while traversing a building. Courtesy of [54].

3.4 Other Explored Algorithms

3.4.1 Cartographer

Cartographer is a system that provides real-time SLAM in 2D and 3D across multiple platforms and sensor configurations. It is a standalone C++ library, most popularly used in integration with ROS. The theory behind the algorithms therein are described in (Hess). To summarize, the scheme uses a *Ceres*-based scan matcher to match scans (point clouds

to sub-maps fitting the best estimated position). This is assumed sufficiently accurate for short periods of time, but the pose error accumulates as the matcher only depends on recent scans. To cope with this, a back-end pose optimization is run regularly. All scans and finished sub-maps are considered for loop closure in the back-end, and are added as a loop closing constraint to the non-linear optimization problem if a match is found. A Branch-and-bound based scan-matcher is used for this. The scheme can use 2D scans or down-projected 3D point clouds for 2D mapping and 3D point clouds for 3D mapping.

The open source software from cartographers git was extensively explored and experimented with, but no reasonable results were achieved. The reason is believed by the author of this thesis to be related to the many option parameters available. In any case, the approach was left at that due to lack of time.

3.4.2 Lego-LOAM

Lego-LOAM is presented in [46], and is implemented in particular for UGVs. The SLAM solution is based on LOAM, and has the same concurrent algorithm structure. Compared to LOAM, improvements include, but are not limited to the introduction of a pose-graph optimization, a segmentation step prior to feature extraction, reduction in features required in both algorithms and exploit of the presence of a ground plane. The latter is according to [46] easily removed, and should thus not introduces a problem for a USV, where no ground plane is present. Results of the method run on the KITTI benchmark data-set indicates that Lego-LOAM is much superior to LOAM both in run-time and pose estimation error.

Needless to say, the solution is highly interesting, and extensive attempts were made to try it. Being as freshly introduced as it is, the open-source code from RobustFieldAutonomyLab seem to be having some teething trouble, as the visualization program would crash every time the program was run on different LIDAR data. Message boards proved many others were experiencing the same issues, and attempts to fix the problem was unsuccessful. Showing as much promise as [46] does, it is not unlikely that the authors will fix these issues in the near future, or perhaps they already have at the time this thesis is read.

3.4.3 hdl_graph_slam

hdl_graph_slam is another SLAM algorithm estimating 6DOF using a 3D LIDAR. It is based on GraphSLAM and uses NDT scan matching-based odometry estimation and loop detection. It allows for addition of several graph constraint such as IMU data, GPD-data and floor plane, and seems to have been tested by the developers using the Velodyne VLP16, among other LIDARs.

However, the related paper, *A Portable Person Behaviour Measurement System using a 3D LIDAR* was not made available on request. In addition, as with BLAM, the code is not well documented. For these reasons, testing of this algorithm is not included in the scope of this thesis.

3.4.4 Gmapping

Gmapping is an algorithm based on [49], and uses Rao-Blackwellized particle filters to solve the SLAM problem. Being dependant on odometry input and highly sensitive to errors therein, the method is inferior to Hector-SLAM(Santos). As they both take 2D laser scans and produces grid maps, Gmapping was left alone.

3.4.5 Tiny-SLAM

TinySLAM [27] is another particle filter based 2D SLAM algorithm creating an occupancy grid map. The C code is written in less than 200 lines, which seems inadequate. As the result presented were not satisfying compared to Hector-SLAM, Tiny-SLAM was also discarded.

Chapter 4

Data Logging Setup

This chapter presents the surface vehicle, sensors and hardware used to log the data needed to run the algorithms described in Chapter 3. Also included is a description of the area the data was logged in, and its environments.

4.1 MilliAmpere

The surface vessel MilliAmpere is a half-scale prototype for the ferry supposed to eventually sail over the canal, and at the same time works as a research platform for implemented and under-implementation required for navigation, autonomy and security.

The vessel is 5.0 metres long and 2.80 metres wide in the centre, which is the widest point. Two azimuth thrusters is equipped, one in on each side of the centre along the inertial origin of the boat. They can thrust both ways, and can be rotated to any horizontal angle by the help of a servo motor each. This gives the ferry good maneuverability compared to a fixed propeller and rudder system, which is an important feature when maneuvering close to both mobile and immobile obstacles with that much inertia present.

The motors are electrical, and are powered by a few hundred kilos of led battery stationed below deck around sea level, along the inertial y-axis of the vessels right-handed system. This pushes the ferry's centre of gravity downwards, allowing for increased ship stability.

4.2 Sensors

The sensors used in the data logging include a LIDAR, an IMU and a GNSS/RTK-system.

Figure 4.1: Photo of MilliAmpere taken by the author some time during Spring 2018.

4.2.1 Velodyne VLP-16 LIDAR

The Velodyne VLP-16 outputs distances, calibrated reflectivities, rotation angles and synchronized time stamps with a resolution of micro-seconds, and has among others, the following attributes(vlp16manual):

- Measurement range up to 100
- Produces scans at 5-20 Hz
- 360 degrees horizontal field of view (FOV)
- 30 degrees vertical FOV (+15 degrees to -15 degrees)
- 0.1-0.4 degrees horizontal angular resolution
- 2 degrees vertical angular resolution
- Typically +/- 3 cm accuracy

The sensor is hardwired to an interface box, which in turn connects to a power source, and a computer through an Ethernet cable.

Choosing the optimal spot to mount the LIDAR is no straight-forward decision. For best utilization of the 3D capabilities of the LIDAR, it should be placed horizontally above the roof of the vessel, so that no laser beams are blocked by the roof. Specifically, as the roof was measured to be 200 cm long and 106 cm wide, the minimum elevation to ensure this is 30.27 metres. However, mounting the sensor on a pedestal atop the middle of the roof might block signals for other important instruments, or even take the place more suitable for another. In any case, trade-offs will likely have to be made between necessary instruments. The plan for the ferry is to carefully design a sensor rig atop the vessel, ensuring optimal utilization of the instruments.

Furthermore, with MilliAmpere's roof height to length ratio, the laser beams of the VLP-16 can never see the edge of dock while sufficiently close if the sensor is placed horizontally on the roof. If we assume the floor of the dock is horizontally level with the deck of the vessel and the roof is 2 metres above the deck, the laser channel pointing 15 degrees downwards along the ferry's inertial axis would hit said level 5 metres away from the boat. This is an issue for LIDAR-aided docking or LIDAR-SLAM-aided docking.

The problem can be mitigated by using one or more other immobile objects visible and recognizable for the LIDAR with known position relative to the dock. An illustrative example could be a star-shaped standing sign placed near the docking station.

Another possibility is to tilt the LIDAR and place it on the edge of the roof towards the stern/bow. This would allow the LIDAR to measure distance the dock while adjacent to

it, but only on one end. Hence, to maintain the ability to dock with both "stern" and "bow", a LIDAR would need to be mounted on the other end of the roof as well. Further, some stereo vision technique would need to be applied with means to produce a single point cloud at each measurement for SLAM purposes.

In any case, no safe mounting device was available for either tilting or elevating the VLP-16 at the time of data logging. As can be seen in figure 4.2, the sensor was duct-taped to sit atop the edge of the roof, facing the bow.

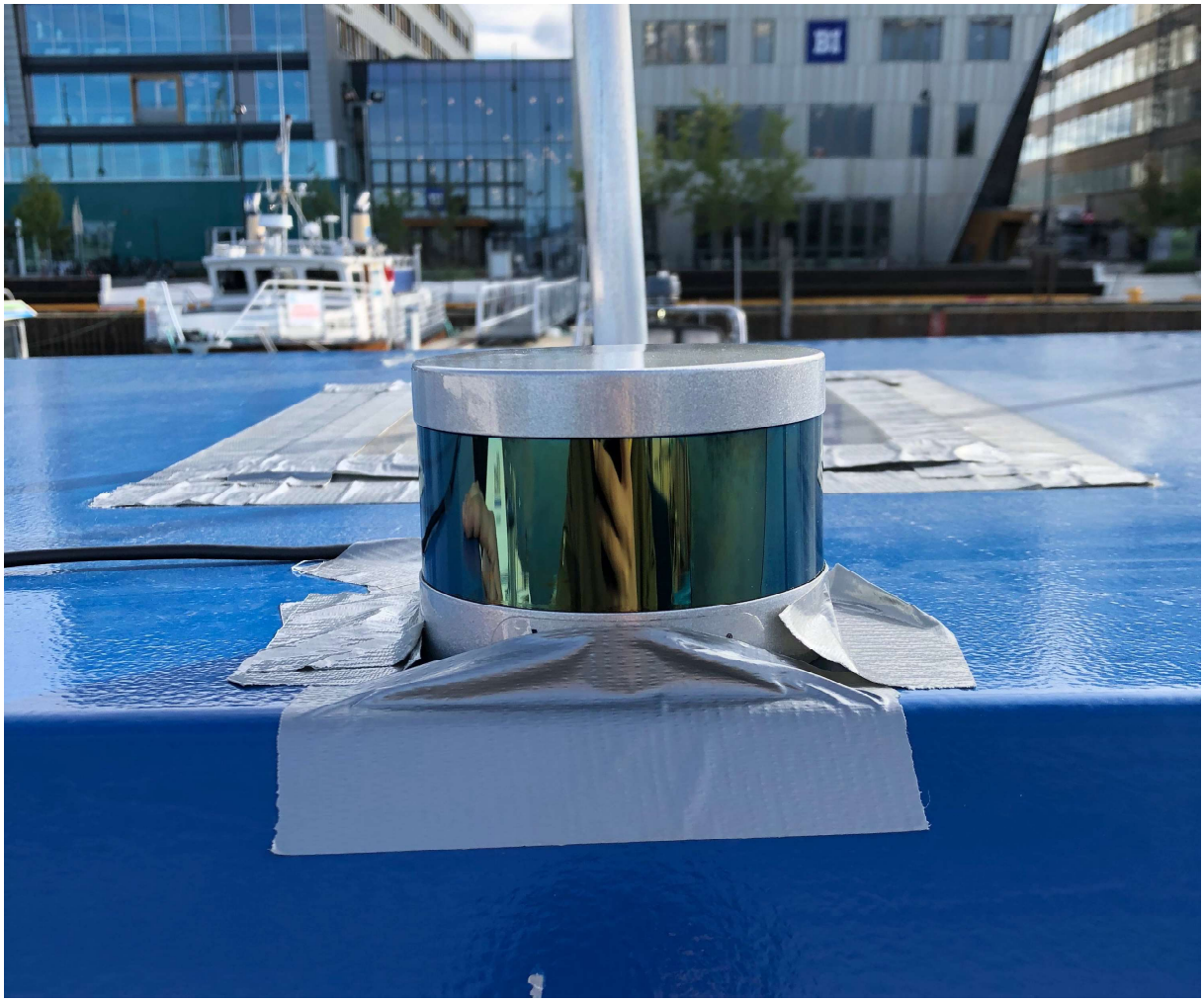


Figure 4.2: Photo taken by the author showing the placement of the LIDAR during the data-logging procedure.

4.3 Xsens MTI-G-710

The Xsens MTI-G-710 is an Inertial Navigation system (INS) with a GNSS-receiver, magnetometer and a barometer integrated with the IMU.. The unit provides BODY-frame accelerations and angular velocities both based on raw IMU data and Kalman filtered with GNSS-data. Thus, it also outputs geodetic ECEF position estimates. The

system has an Attitude and Heading Reference System (IAHRS) to provide drift-free 3D orientation as well. This is done by integrating gyroscopes and fusing this data with accelerometer data and magnetometer data. This orientation estimate has an accuracy of ± 0.3 deg in roll and pitch, and an accuracy of ± 0.8 deg in yaw.

With the position accuracy stated as $\pm 1 - 2$m, [43] found the INS to be an insufficient navigation system for Dynamic Positioning (DP) on the model ship DNV-GL Revolt. Still, the unit was ported to MilliAmpere, was used as the source of IMU data during logging. The IMU has an update rate of 2000HZ, and the GNSS receiver an update rate of 4HZ. The sensor is mounted upside down on the underside of the ferry's roof, such that the IMU data must be rotated 180 degrees about the BODY-frame x-axis.

4.4 Hemisphere Vector VS330

As precision and accuracy in both heading and position measurement is crucial for DP, a dual antenna Real Time Kinematics (RTK) GNSS was acquired. RTK is a variant of differential GNSS, and is based on correction fed to the robot from a GNSS base station in a known position. The method gives extremely accurate results compared to regular GNSS.

The chosen receiver was the Hemisphere Vector VS330, which was later also chosen for MilliAmpere. The receiver supports both RTK and SBAS, and communicates with a local GNSS base station through a Satel Radio Link also mounted on the underside of the roof. It uses two antennas and an internal gyro stabilizer, allowing it to measure heading at an accuracy of 0.05 degrees. The horizontal accuracy is specified as ± 0.30 metres without RTK, and ± 0.01 metres with. As RTK was up and running at the time of data logging, both heading and position are assumed ground truth in evaluation of the SLAM algorithms performed.

The GNSS update rate is specified as 20Hz. The receiver does not, however, support the in-progress satellite constellation Galileo, whose main focus area will be Europe. This is not yet a problem though, as the constellation is some time off to being operational.

4.5 OBC

The On-Board Computer (OBC) installed on MilliAmpere is an Axiomtek eBOX670-883-FL-DC. It is a fan-less embedded computer with Linux Ubuntu 16.04 installed as operative system. Both the Hemisphere and the Xsens was connected to this during logging.

(a) (b) (c)

Figure 4.3: (a) shows the Xsens MTI-G-710, (b) shows the Hemisphere Vector VS330 and (c) shows the Satel Radio Link.

4.6 Lenovo G50

At the time of data logging, the ferry was getting ready to be shipped to Haugesund for a demo a few days later. For that reason, it was not desirable to introduce another sensor to the system and the OBC, in fear of messing with the current IP-addresses and ports. For practical reasons the LIDAR was also connected to the authors private laptop, and data was stored on two different computers.

A small issue with this was that the two computers was not linked, such that the machine times were not synchronized. Specifically, the ROS-time differed. The issue, however, was considered at the time, and the ROS-time difference between the two was logged and compared, and accounted for in post-processing of the data.

As the OBC, the laptop runs Linux Ubuntu 16.04. It possesses an Intel Core i5 processor and 4GB memory. The same computer was used to run the algorithms described in Chapter 3.

4.7 Logging Area

The map in figure 4.4 shows the location where the data logging operation was performed. The desire on the day was to sail in the canal, allowing for work with data as similar to the data present when the ferry is operational in the future. However, after some on-site debugging with the instruments prior to shipping-out, the tide would not let it be so. MilliAmpere was docked in the area marked as A, and it was decided that it require high tide to get into the canal, for safety reasons. Thus, the sailing took place in Brattørabassenget.

Figure 4.4: 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.

The areas are to the eye in many senses very similar. While on the water one would at all times be surrounded by a rich environment of immobile structures, such ass docked boats, buildings and docks. For a LIDAR and its placement, however, some differences are important to be aware of:

- The geometry of the surroundings are different. While the ferry in Brattørabassenget is almost at all times enclosed with structures 360 degrees horizontally, it would in the canal at all times have a canal-shaped field without range-measurements cutting across the point cloud measured. Intuitively, specifically from a 2D perspective, localization is simpler in the former surroundings, and many scan-matchers fail when traversing corridor-like environments.
- The long, gray structure stretching from the left towards the top in figure 4.4 is relatively low, and not much is visible for the LIDAR at the current placement, unless faced towards it. This fact makes quite a large portion of the point cloud measured by the LIDAR very sparse. This in combination with the former point causes problems, evident in section (RESULTS)
- The visible buildings surrounding the canal-area would be more and closer to the ferry in the canal compared to in Brattørabassenget. This is especially important when the downward FOV for the robot is as limited as it is, and is evident by inspection of the point cloud. In addition to being most visible with the current setup, they allow for many edges to extract as features.
- Also, the thin gray structures in the water at Brattørabassenget, the floating docks, are very low and invisible to the LIDAR in most cases. However, they are in most cases covered with other vessels, which on the other hand, most often are.

Figure 4.5 shows the trajectory for the ferry during the logging, based on logged GPS-coordinates. Prior to shipping out, the bow was defined as facing the floating dock, which was the same end the LIDAR was mounted. Shortly after, during the first curve, the vessel was slowly spun 180 degrees, such that the ferry would sail in the direction the LIDAR the best view. All change of vessel heading after this, was done in a slow, comfortable manner.

The reason the bow was defined as facing the floating dock, was to utilize the DP-system implemented the prior summer. The DP-system worked in the way that GPS-location reference and a heading reference was manually defined on the OBC, and thruster-allocation ensured the ferry would automatically go there, slowly and controlled. If SLAM is to be responsibly for the docking process, the same smooth execution should be the goal. It was therefore assumed it would be interesting to see how various SLAM algorithms behaved during such a scenario. Thus, the reference was defined as the exact position and heading as just prior to ship-off.

Moreover, making a large data-set was desired, so the whole area was attempted covered. 11 minutes though, which is the time it took to sail this rather small route, was perhaps

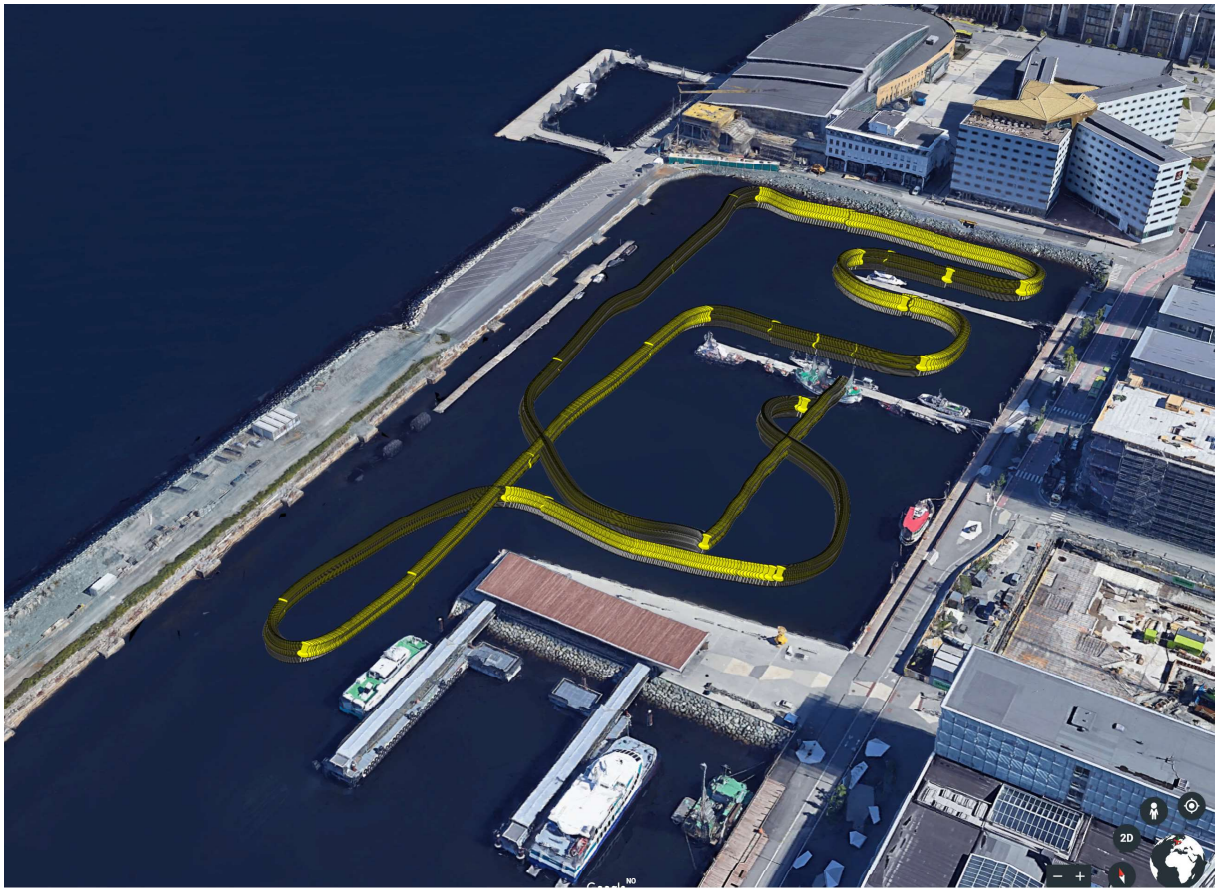


Figure 4.5: Screen-shot of the ferry's trajectory from which the data is logged. The almost straight line from the middle floating dock is the return, while the curved one is from shipping out. Longitude and Latitude was from a rosbag in MATLAB, converted to a .KML-file and uploaded in GOOGLE earth. 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.

a bit longer than intended, but it was considered important start and stop at the same location in the same data-set, if only for the sake of testing possible loop-closing mechanisms. Still, shrinking a data set in post-processing is much easier than enlargening it. The loop-closure aspect was also the reasoning behind the crossing of paths, rather than sailing in a circular trajectory.

Chapter 5

Software Implementation

This chapter describes the sensor drivers used during data-logging, software implemented and software utilized.

5.1 ROS

A part of the pre-project, consisted of getting acquainted with Robot Operating System (ROS). Hence, a short introduction to the software framework is given in (project). All code produced during this thesis is written in the programming language C++. As is most of the open source software utilized and existing software on board, with a small portion written in Python.

The data present during sailing was collected in *rosbags* [55]. In these, any ROS topic is recorded and stored in the same order as they are present in the system. This way, the rosbag can be played, and simulate the ferry publishing the saved messages. All available topics were stored during logging.

RVIZ (ROS Visualization) [23] is a 3-D visualizer for displaying sensor data and state information from ROS. This tool is used both for visualizing the map and trajectory resulting from the SLAM algorithms, as well inspection of the lidar data.

5.2 Sensor Drivers

Although suggested in the pre-project to this thesis [39], no changes were made to any of the following drivers by neither the author or anybody else, as far as the author can tell.

However, since more information has been apparent, a more thorough description follows, rather than a reference.

5.2.1 Velodyne Driver

The driver for the Velodyne VBLP-16 is adapted from [17]. No modifications were made to it, and thus the relevant outputs are the point cloud and the scan, available at the topics /velodyne_pointcloud and scan. The scan output originates from the ring of laser beams reflected at -1 degree resolution. Both topics are published at approximately 10Hz.

5.2.2 IMU Driver

The IMU driver installed on the MilliAmpere OBC is written in Python and originates from work in [43], and is a modified version of a third party driver. The driver allows for publishing of raw gyroscope and acceleration measurement data, raw GPS position, Kalman filtered GPS position, Kalman filtered velocity estimate, Kalman filtered orientation estimate, magnetic field data, temperature data, barometric pressure data and diagnostics.

Since [39] came short in including IMU data in its work, a bug regarding the time-stamps in the IMU-messages was discovered while working only after the logging was performed. The time-stamps simply stated a whole other time than the data received from the lidar and the GNSS-receiver, making them essentially incompatible as they were. This was fixed in post-processing after some debugging, but still rather annoying.

Of the possible topics stated above, only raw gyroscope, and acceleration data and Kalman filtered orientation estimate data was enabled. The former two are combined in a standard sensor_msgs/imu-message, while the latter formatted as a custom message. The topics are named /xsens/imu and /xsens/orientation, respectively, and both measurements has a frequency of ca 100Hz.

5.2.3 Hemisphere Driver

The GNSS driver originates from work in [42], and is a modified version of [56]. The driver parses the NMEA sentences consisting of raw GPGGA, heading, and latitude and longitude. All are formatted in custom messages, instead of ROS standards.The topics are named /vectorVS330/GPGGA, /vectorVS330/heading and /vectorVS330/fix, respectively.

5.3 Implemented Nodes

- plc_to_scan projects a 3D point cloud down to the 2D-plane. It does so by keeping the closest point in each horizontal resolution. The node is adapted from [61], and modified to include intensities of reflected laser beams, as Hector-SLAM depends on them.

- imu_preprocessor combines the two IMU-messages into one sensor_msgs/imu, as this is the formats most open-source tools expect.

- pcl_node defines a box and filters points inside it. This is to remove points originating from MilliAmpere from the point cloud used in the SLAM algorithms. The point cloud library [62] was utilized to implement this node.

- scan_to_sca_filter_chain can utilize a set of filters compatible with lasers. It was used to run a filter removing measure points on MilliAmpere from the scans fed into Hector-SLAM.

5.4 Hector-SLAM

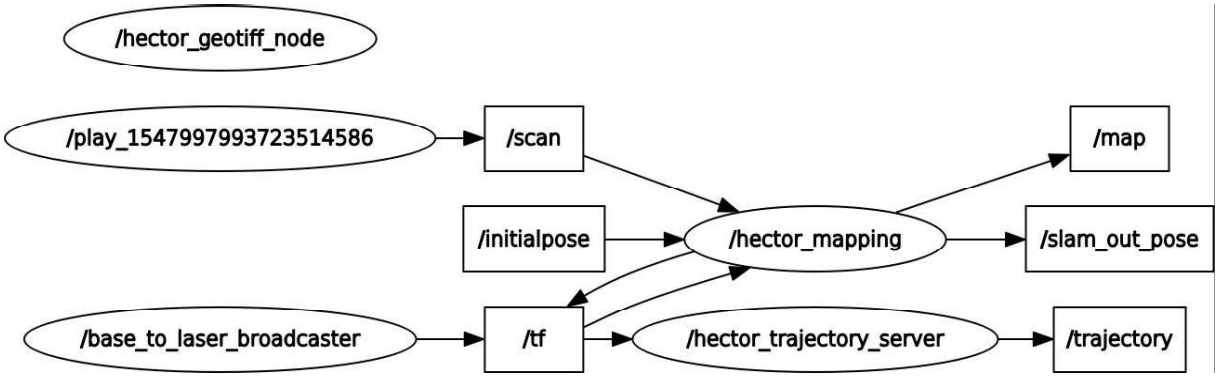


Figure 5.1: ROS-graph when running Hector-SLAM on a rosbag, without any filters used. The oval boxes are executable nodes, while the square boxes are topics.

The Hector-SLAM source code is adapted from [24], with only small changes made. These are made in the launch file, regarding map size, grid map starting point and subscribed topics. Figure 5.1 shows the flow of the system. The executable node /hector_mapping is the SLAM node. For every scan received, the node performs the tasks described in Section 3.1, and outputs the grid map updates on the /map topic and the pose on the slam_out_pose topic. The /hector_trajectory_server is responsible for saving trajectories and publishing the green curve seen in Figure 6.3. When running filters on the incoming scans, the /scan topic is the launch file replaced by the output topic of the last filter.

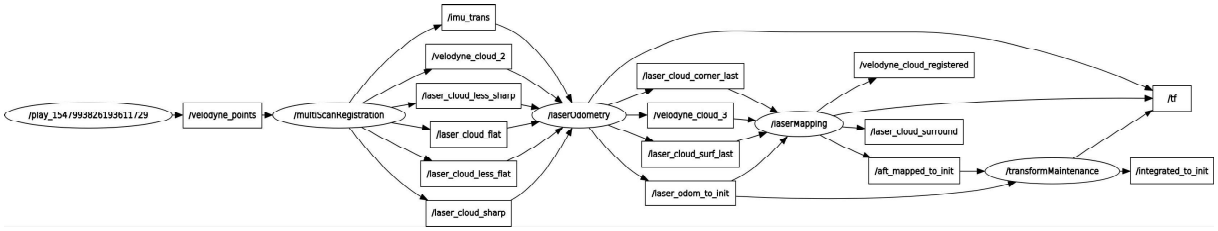


Figure 5.2: ROS-graph when running LOAM on a rosbag, without any filters used. The oval boxes are executable nodes, while the square boxes are topics.

5.5 LOAM

The LOAM code is adapted from [63], and is ultimately only altered in terms of parameters. That is, both the odometry update and map update is set to 10Hz. This is because point clouds are built the driver only once per revelation, which makes odometry calculations more often than this, pointless. As with the Hector-SLAM-implementation, the input topic /velodyne_points is changed to the filter output if any are used.

Looking at Figure 5.2,the /multiScanRegistration node is responsible for all actions performed on the new point cloud. That is, extracting the different feature points,and creating new point clouds consisting of the different sorts, explicitly. The different clouds have different types of features, as well as different amounts of features. The original point cloud is also republished as /velodyne_points_2.

The /laserOdometry node then performs the scan matching scheme after the created point clouds as described in Section 3.2, and outputs the transformation along with republishing the original point cloud and the richest of the feature point clouds.

The /laserMapping node takes the map point cloud and downsamples it. The two incoming feature clouds are transformed according to the odometry transform, and are matches towards the map through the mapping scan matching scheme described in Section 3.2. This matching forms the basis for the points to be added to the cloud, and the hopefully improved odometric transformation.

5.6 BLAM

BLAM is adapted from [56], and is run only modifying the launch file.. Note that the GTSAM library must be installed seperately, using [64]. Even though the BLAM source code is divided into task-based modules, the program is run as a single executable node. Thus, the ROS-graph generated during execution contains very limited information, and an algorithm flowchart was made showing the modules relation to each other. point_cloud_odometry outputs an odometry estimate through ICP, point_cloud_localization produces the new pose estimate based on odometry and ICP, handle_loop_closure checks

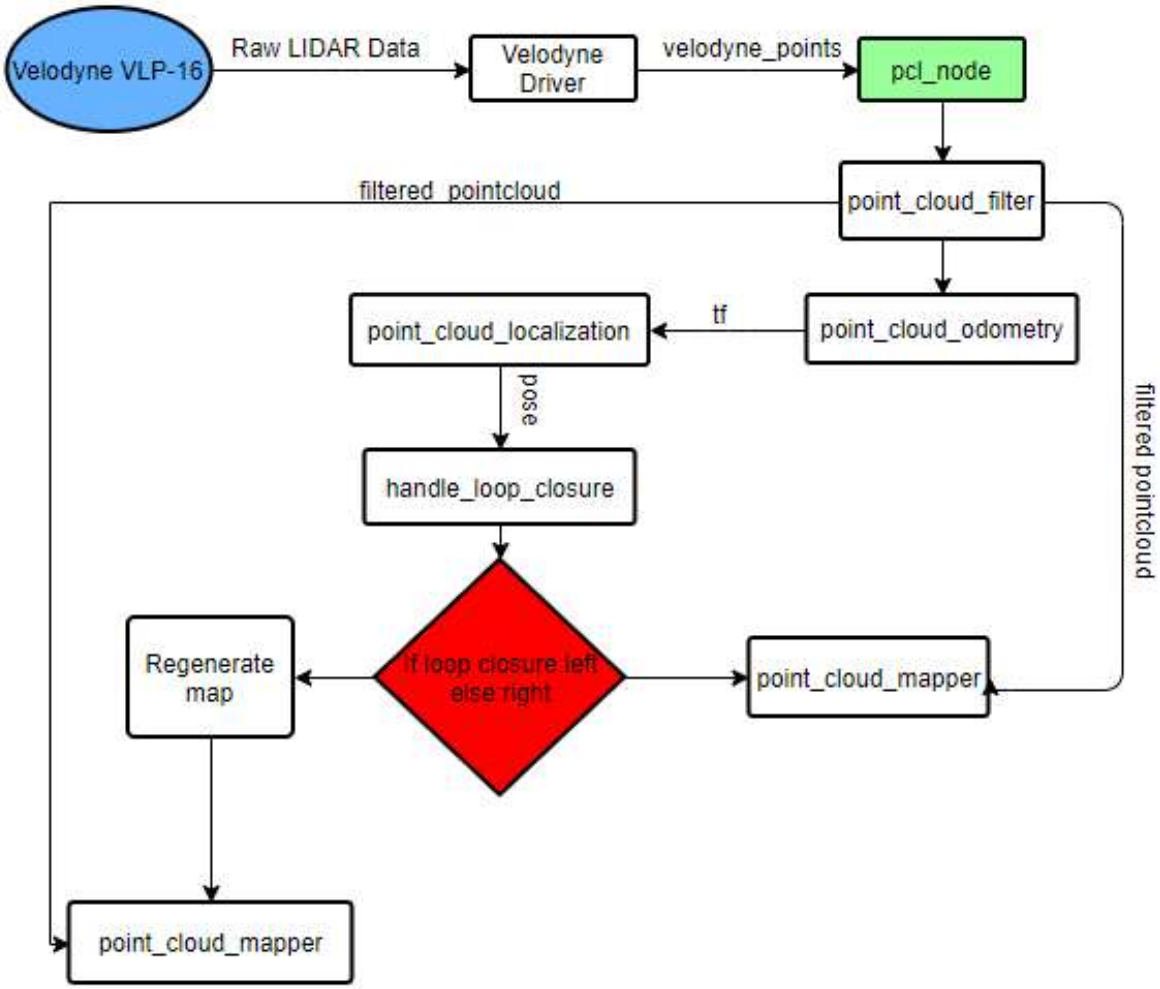


Figure 5.3: Flowchart of the BLAM algorithm.

for loop closure and point_cloud_mapper inserts the new points in the map.

Chapter 6

Results

6.1 Evaluation Methodology

When trying to run the algorithms on the logged data, it became apparent that they should all be run on different segments of the data-set. By looking at the resulting maps, it was obvious that all three algorithms had difficulties in the same area, namely the bottom left area seen in 4.5. The area is from here on referred to as the "corridor", because the ferry traverses through what resembles said corridor. The scan matching-algorithms give erroneous position estimates, resulting in both pose drift and map inconsistencies. By inspecting the resulting maps and trajectories, it it obvious that the errors have consequences, corrupting the results. On another note, in hindsight, the route travelled during the data-logging is quite different from that in the proposed operation of the ferry. While the logged scenario is quite long both in duration and distance travelled, the autoferry should only sail about 100 metres for a minute or so. Further, while the logged trajectory is rich in rotations and cutting corners, the ferry should aim for motion as close to a straight line as possible with minimal rotations for passenger comfort, bar no obstacles must be avoided. For these reasons, data-subsets with two different starting points were defined in addition to the full 636 seconds set:

- One subset starts just after traversing the problematic corridor described above. That is, the last 446 seconds. This allowed for evaluating the algorithms as if the problematic area is not present, as well as giving un-corrupted SLAM results for the rest of the trajectory.
- The other subset starts just after the last 90 degree turn made by the vessel. The turn can be seen in the middle of the image in Figure 4.5. This allowed for evaluating pose and map in a more operational-like scenario, as well as giving a better look at what map is built should the algorithm be used for autonomous docking purposes. Though short in distance, the subset lasts for the last 161 seconds of the journey. This is because the newly implemented autonomous DP docking mode was activated.

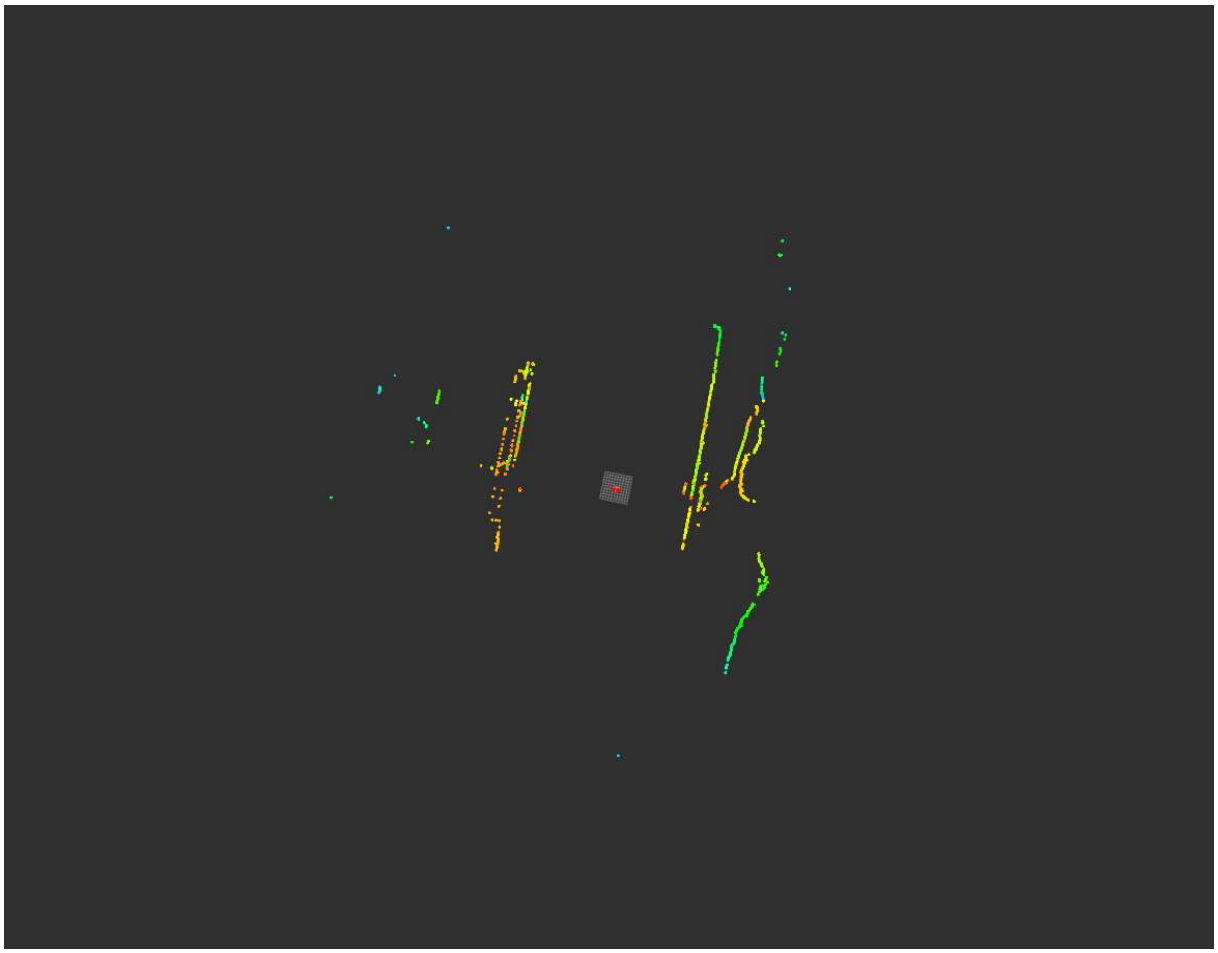


Figure 6.1: The typical point cloud measured by the lidar in the corridor.

This thesis main criteria for evaluating the SLAM algorithms in the different mentioned scenarios, is the error of their estimated poses with respect to the accurate estimates given by the RTK-GNSS-receiver. That is, horizontal position and heading. For the 3D solutions, vertical drift is included as well.

Using MATLAB, the errors were calculated from the transformation output of the SLAM algorithms and the logged messages from the Hemisphere. The GPS compass heading measurement was kept as they were, while the geodetic coordinate measurements where transform to local cartesian coordinates in the ENU-frame using the MATLAB function *geodetic2enu*. The trajectory is shown in Figure 6.2. As no initial pose was defined for the SLAM algorithms, all start with the lidar situated in the map origin facing [-1 0], constituting a heading of 0 degrees. Thus, the resulting position vector was rotated by the most recent GPS compass heading measurement, aligning the two trajectories and giving them equal starting heading. Error in position is defined as SLAM-position minus GPS-position. The heading-error is given as the absolute value of the difference between SLAM-output and GPS-output.

As the GPS antenna was located at (0,-0.975) and the LIDAR was located at (0.47,0) in MilliAmpere's NED body-frame, lever-arm compensation was required. This was dealt with by translating all points to the vessel center prior to calculating the errors.

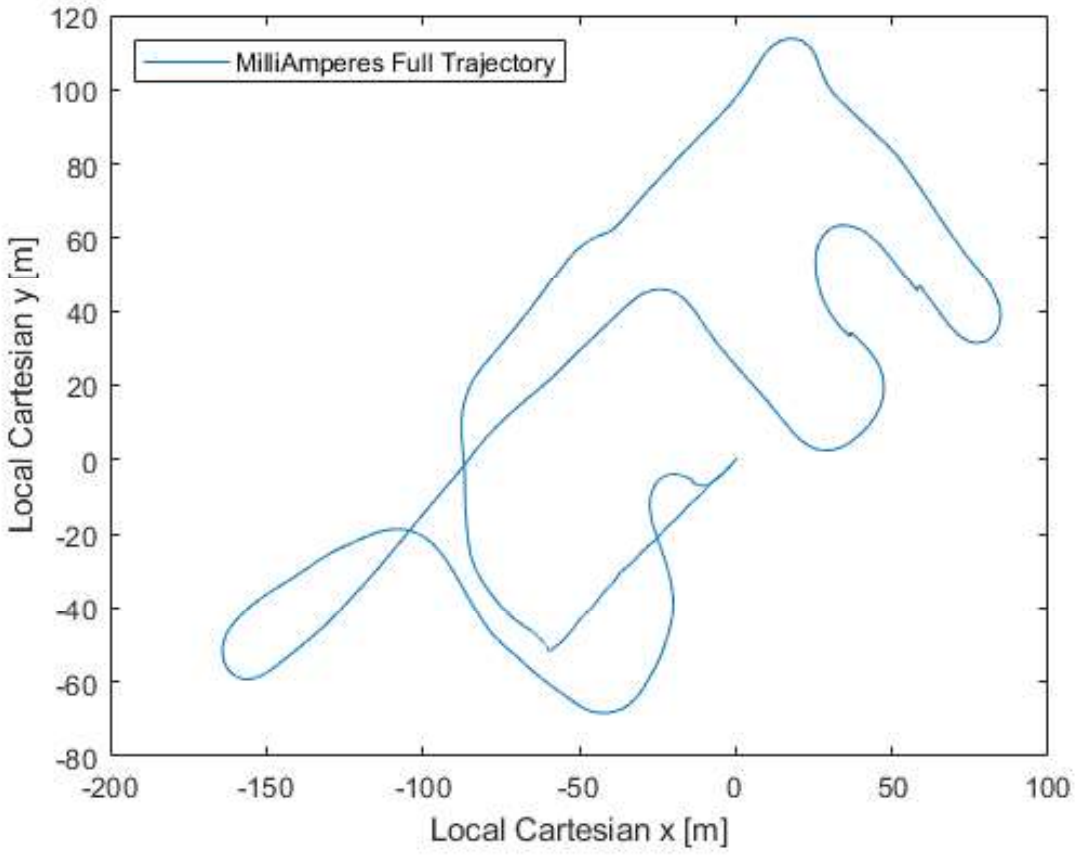


Figure 6.2: Logging trajectory in local cartesian coordinates.

For each algorithm run, a screen-shot image of the resulting map is provided. For the 3D maps, these are given from a top-down point of view, for easier comparison with the actual maps of the environment. No concrete evaluation criteria for map evaluation is included in the scope of this thesis. However, comparing the output map along with its trajectory gives a reasonable indication of the result quality.

6.2 Hector-SLAM

6.2.1 2D Scan-Matching

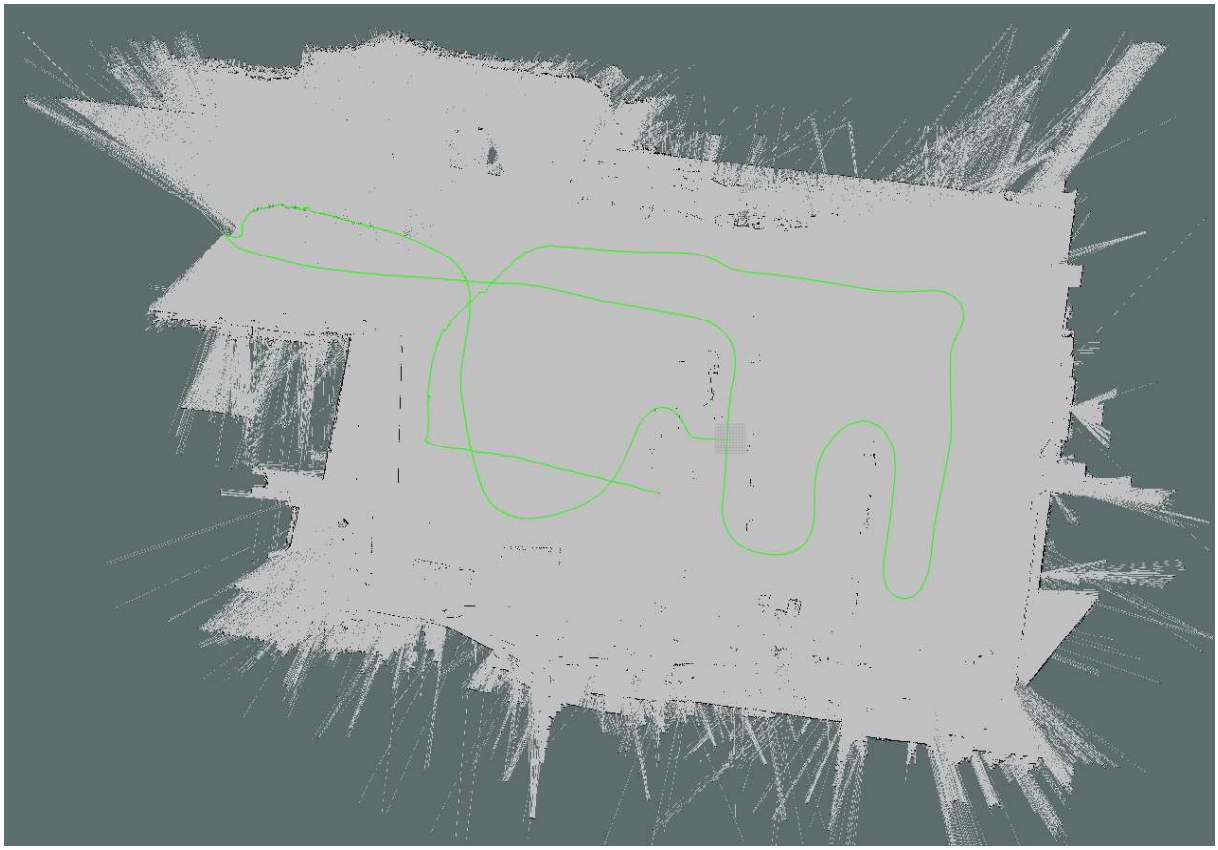


Figure 6.3: Resulting map and trajectory from running Hector-SLAM on the full data-set. Scans were filtered through the laser filter beforehand to get rid of rays hitting MilliAmpere.

Figure 6.3 shows the resulting map and the shape of the trajectory from running Hector-SLAM on the full data-set. The trajectory seems quite similar to the factual in Figure 6.2, but the scan-matcher has failed on at least one occasion. This is obvious by inspecting the trajectories in the corridor that Hector-SLAM has failed to register both some translation and rotation while traversing there. Looking at the position-error graphs in Figure 6.4 and 6.5, we see some noisy sparks at around 150 seconds, followed by periods of drift. At 6.6, we see can also identify the same time as the beginning of drift in the mean heading error.

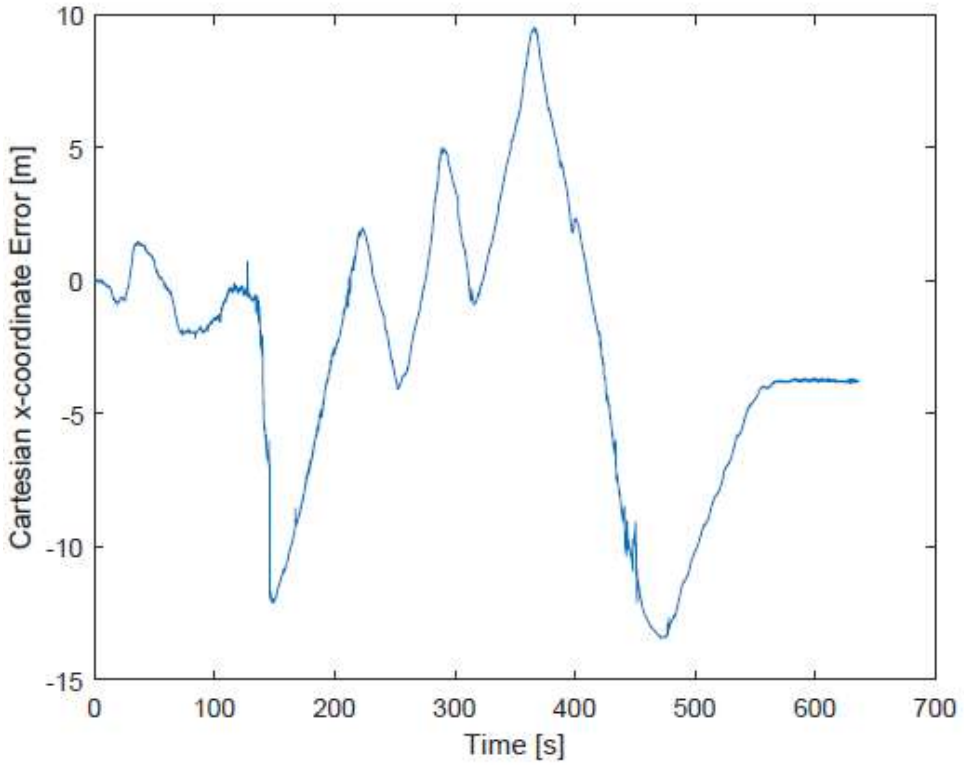


Figure 6.4: Resulting x-position error from Hector-SLAM run on the full data-set.

Simulating a trajectory avoiding the corridor, the Hector-SLAM algorithm was run on the second subset. The result is shown in Figure 6.7. No immediate issues can be seen in the trajectory, and whatever map structure is built look reasonable. Looking at Figure 6.8, 6.9 and 6.6, one can see that the result is a massive improvement compared to the former attempt. Hence, it is clear that Hector-SLAM is a SLAM solution where the robot might find itself in environments similar to the corridor.

6.2.2 Point Cloud Projected Down to 2D Scan-Matching

As it seemed a shame to posses a high-end 3D lidar and basically only use a single vertical channel, attempts were made to project the full point cloud measured by the lidar down to the 2D plane, and feed the resulting scan to Hector-SLAM instead. There was also a theory that this would result in richer scans which could improve moth the general accuracy of the algorithm, as well as its ability to cope with corridor-like structures. Unfortunately, the algorithm failed and crashed both for the full data-set, and the middle sized one. Experimentation with different bag starting times indicated that the algorithm had problems when the lidar rotated. Examining the down-projected 2D scan during rotations revealed that the structure of them drastically changed when rotating, and it was evident that the different vertical FOV for different angles due to the lidar placement, was the issue. Thus, the approach was attempted on the smallest data-set, the one that barely includes rotations.

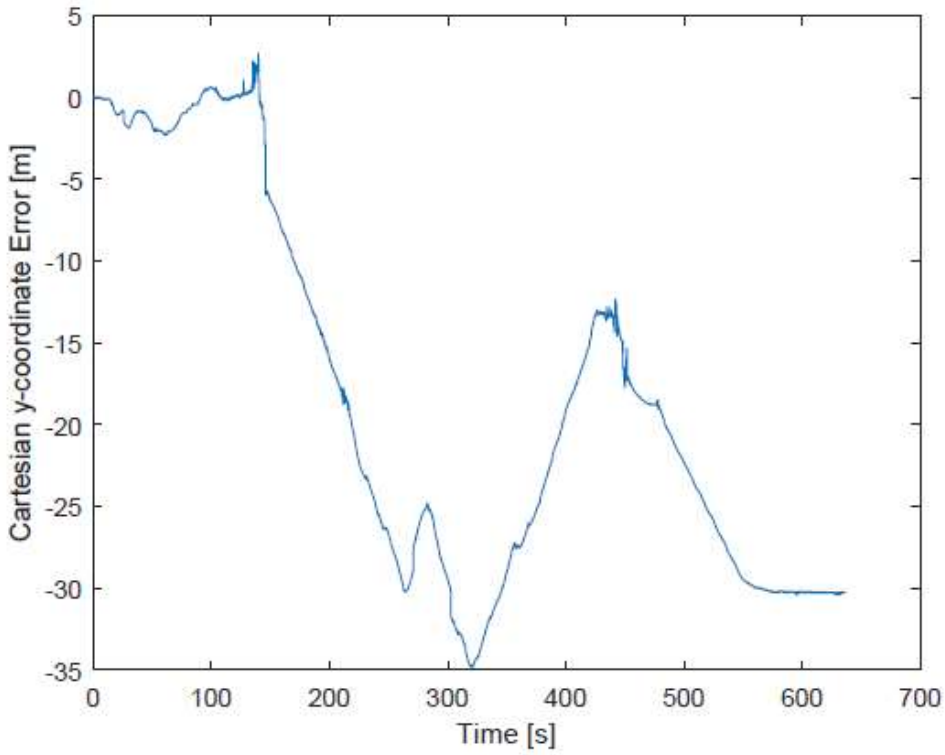


Figure 6.5: Resulting y-position error from Hector-SLAM run on the full data-set.

The resulting map is shown adjacent to the regular 2D version for comparison in Figure 6.11. Not much can be said of the trajectories simply by inspection, but it is evident that the down-projected version has managed to include a lot more measurements in its scans, resulting in a relatively significantly more detailed map. On the other hand, that version also experiences map inconsistencies in the form of double walls. Surprisingly, as seen in Figures 6.12, 6.13 and 6.14, the resulting position estimates are experiencing much more drift compared to the prior results. This could be because the algorithm is initialized during frequent oscillation in the heading of the vessel. In fact, while in DP-mode, the servo for one of the azimuths were constantly oscillating around a reference it could not reach, causing oscillations of the ferry. Moreover, the heading error has at most twice as large spikes in the down-projected result compared to the regular Hector-SLAM result.

6.3 LOAM

The LOAM algorithm was first attempted to run on the full data-set with the point cloud filter feeding it data, such that points on the boat would not be taken into account by the SLAM algorithms scan matchers. This works quite nice for a while, and a few impressively detailed structures become apparent in the map. After some thirty seconds though, the pose estimates become unstable, and start spinning and mapping structures all over the cloud. As both the paper and the online documentation states that IMU data

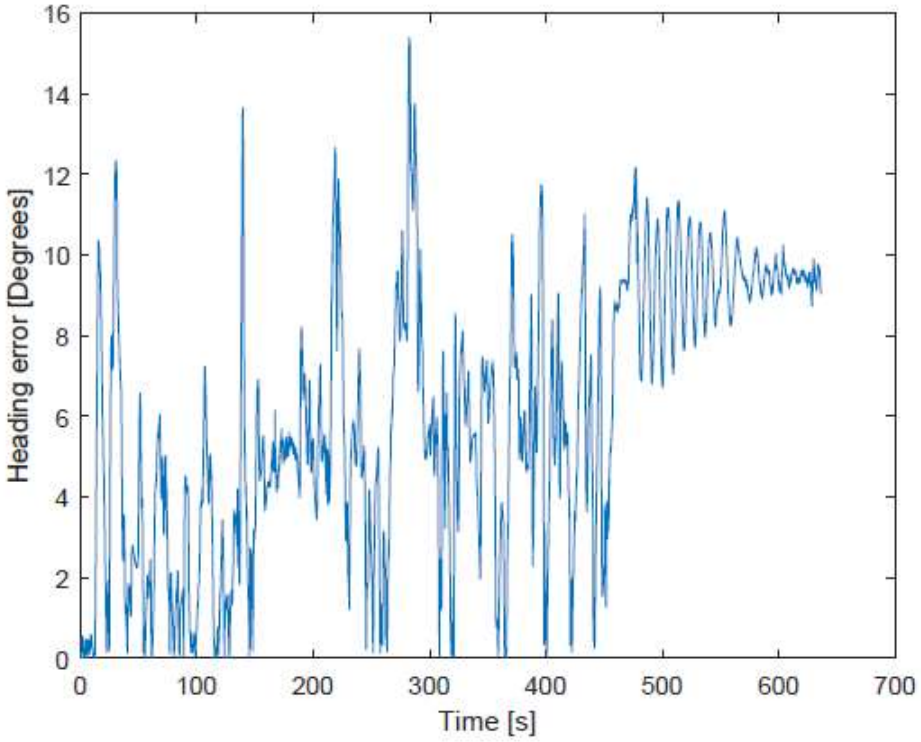


Figure 6.6: Resulting heading error from Hector-SLAM run on the full data-set.

can easily be included, attempts were made to include it to remedy the problem. These proved unsuccessful, and after some a Git issue thread found while searching for similar problems revealing that the author of the online documentation, at least, had not actually tried it himself.

Thus, no IMU data was fused with the SLAM solution. What proved somewhat successful, was leaving the point cloud filter out of the system, and running the algorithm on point clouds which included points of the boat. A theory is that these points worked as a constraint on change in roll and pitch, making spinning less likely to occur. Another theory is that without these points, the mapping algorithm has a too sparse point cloud to extract enough features from. The resulting map is somewhat messy, as the mapper algorithm at times concludes with changes in pitch and roll, leading to structure built at different elevations and angles. By inspecting the map in Rviz both during and after the algorithm run, it can be seen that the same map structures are built several places, at different angles and elevation. What also is seen, is the potential of the algorithm, as it starts off so well, at least regarding the map building.

The resulting map is shown in Figure 6.15, with corresponding error-graphs in Figure 6.16 and 6.17. The map contains many elements that seem correctly placed relative to the trajectory, and many that are not. Most of the latter are elevated above what can be recognized as the map from Figure 4.5, mapped with erroneous roll and pitch. The trajectory is evident through the trail of mapped MilliAmpere measurements, and definitely has similarities to the ground truth GPE-trajectory. Interestingly, the trajectory

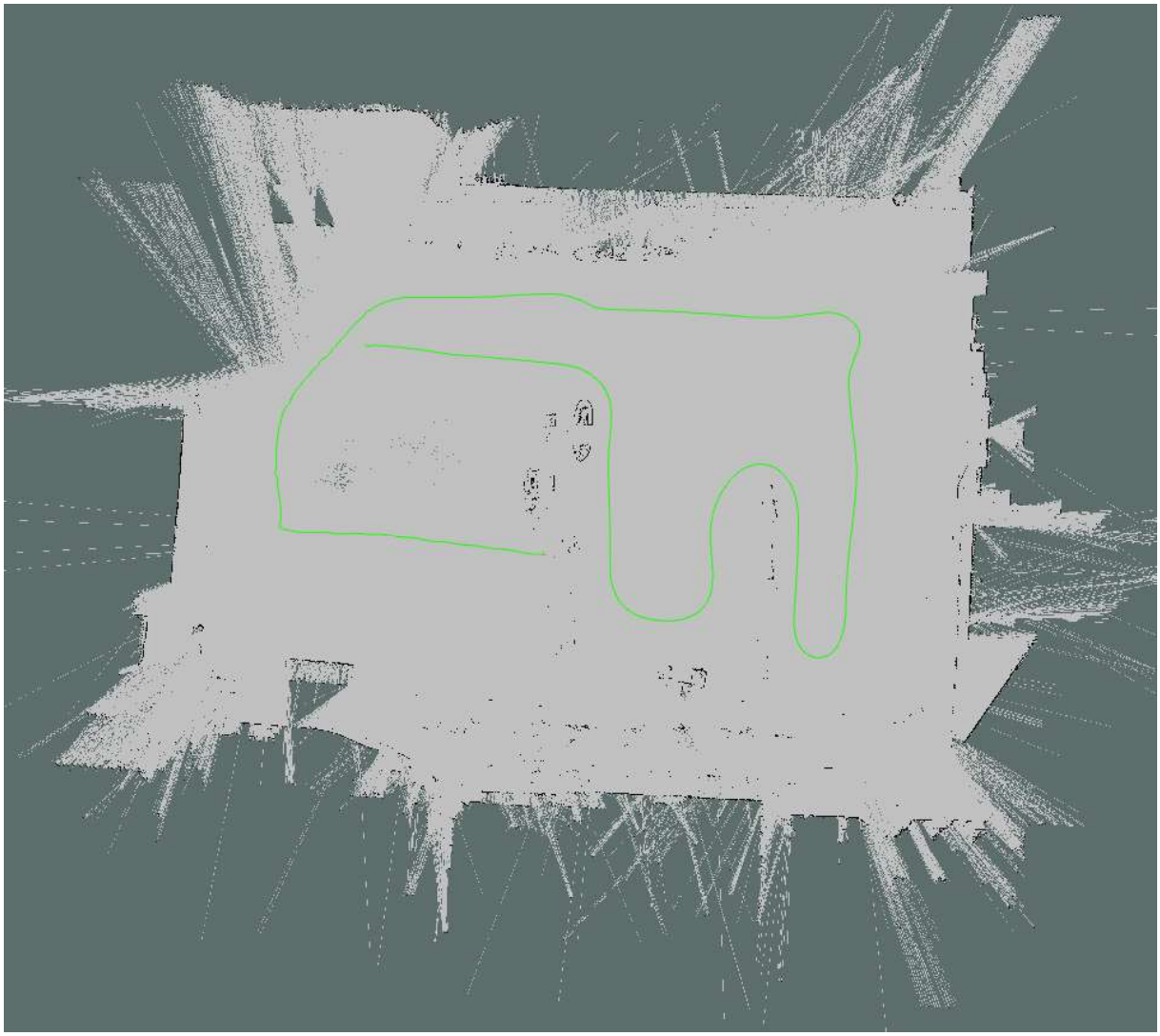


Figure 6.7: Resulting map and trajectory from running Hector-SLAM for the last 446 seconds.

exist the corridor with a rather impressive 2D-trajectory relative to the starting point. On the other hand, the time that proved problematic for Hector-SLAM, around 150 seconds, marks the start of a substantial vertical drift, ultimately breaking the map.

Inspecting the position errors, it is clear that the pose estimates start out quite well in the first 100 seconds or so, before venturing out on a substantial drift as the trajectory approaches the corridor. This coincides with the authors perception from watching the map building in action. However, when running the algorithm on the middle sized data-sized data-set and starting points around there, caused the program to eventually crash. The map can be seen in figure 1, and looks quite promising. Due to the crashes, however, no data was available to analyze.

The SLAM method was also performed on the smaller subset. Figure 2, 3 and 4 shows the map and the error-graphs. Both the trajectory and the map looks quite accurate on this occasion. The objects in the vicinity are recognizable and seemingly geometrically correctly located relative to the ferry. Based on this, the measured drift in position is

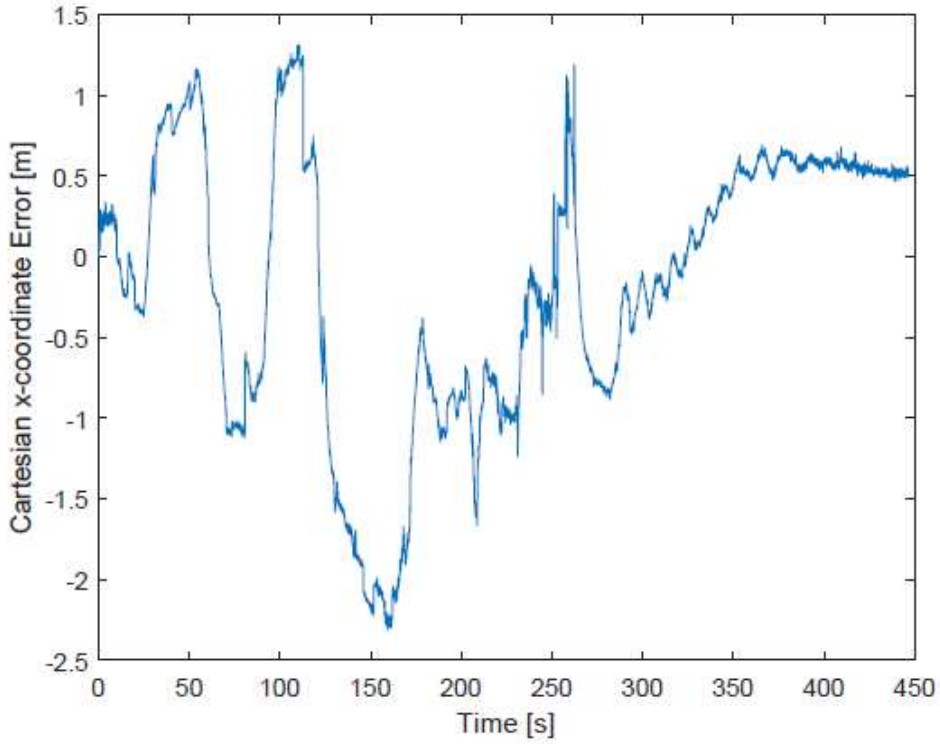


Figure 6.8: Resulting x-position error from Hector-SLAM run on the last 446 seconds.

surprising. However, we also have a relatively large and lasting heading error, outputting an angle which coincides with the position error. 6.18 Shows a close-up of the dock, after the smaller dataset-based map is finished building.

6.4 BLAM

Results of the BLAM algorithm run on the full data-set can be seen in from Figure 6.21 and on-wards. Even though the we can recognize the structure of the map and trajectory, neither are particularly good results. Inspecting the trajectory along with the position error-graphs, it is evident that the algorithm struggles way before approaching the corridor, and perhaps more so in the opposite end of the map. When observing the map building, no sign of loop closing was apparent. Being local loop closure, however, this can be explained by the vertical drift causing the pose to never be the same more than once. Moreover, algorithm utilizes a large amount of memory as the map grows large, causing Rviz to sometimes crash. The difference in color in Figure 6.21 is in fact caused by Rviz having crashed at the screen-shot time.

On the plus-side, the point cloud filter seems to help, as when used, the horizontal and vertical drift appear slightly smaller with all data-sets. Results from the other data-sets are included in the Appendix, starting with 5.

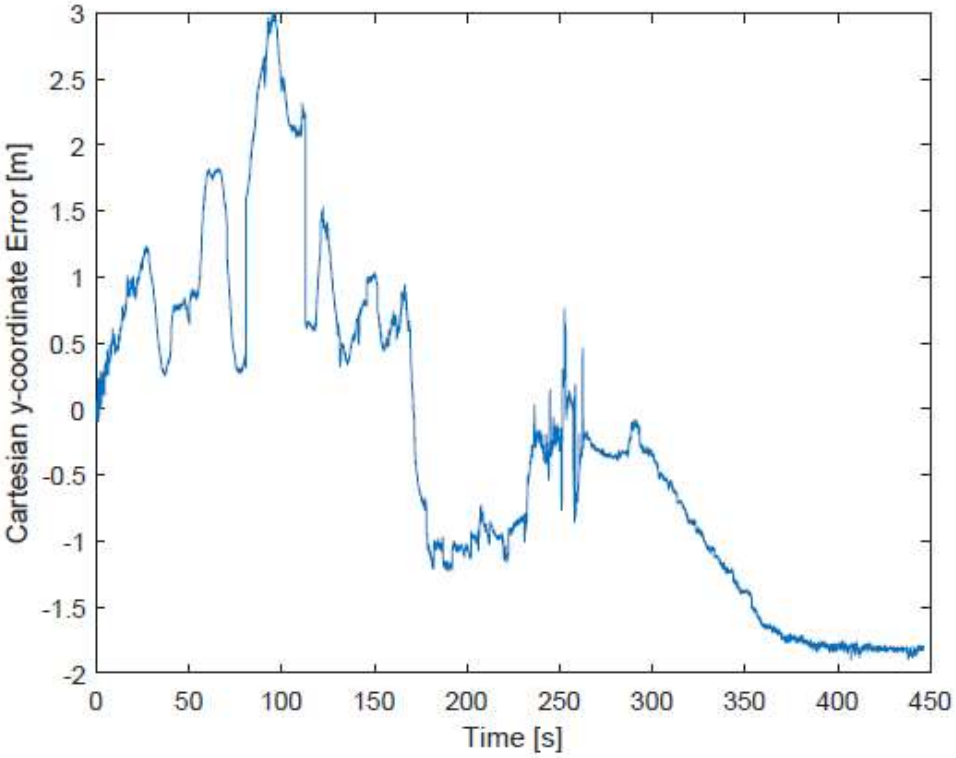


Figure 6.9: Resulting y-position error from Hector-SLAM run on the last 446 seconds.

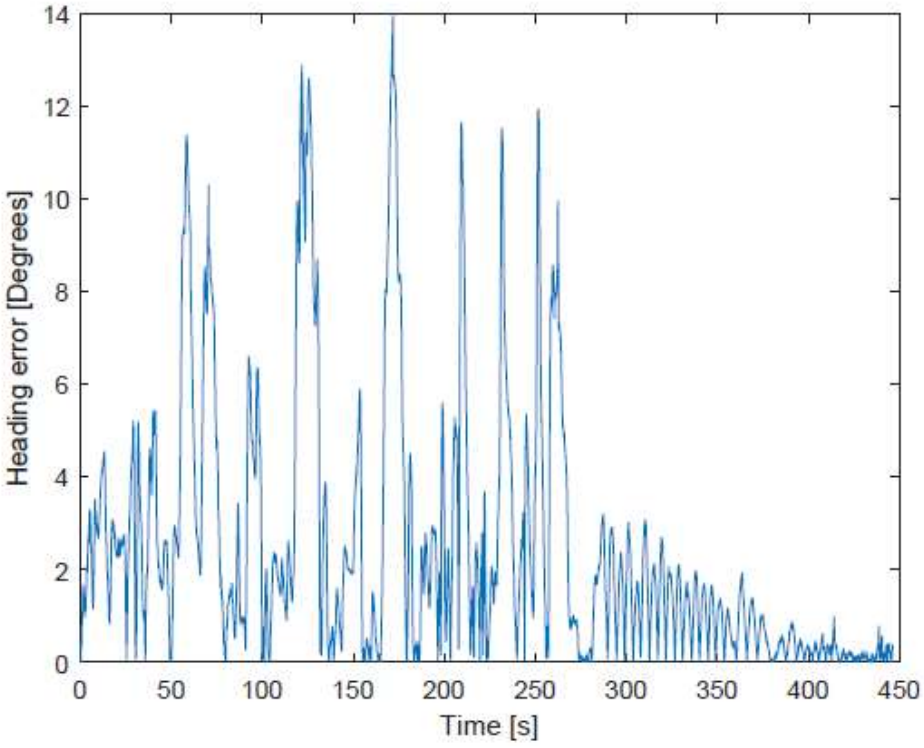
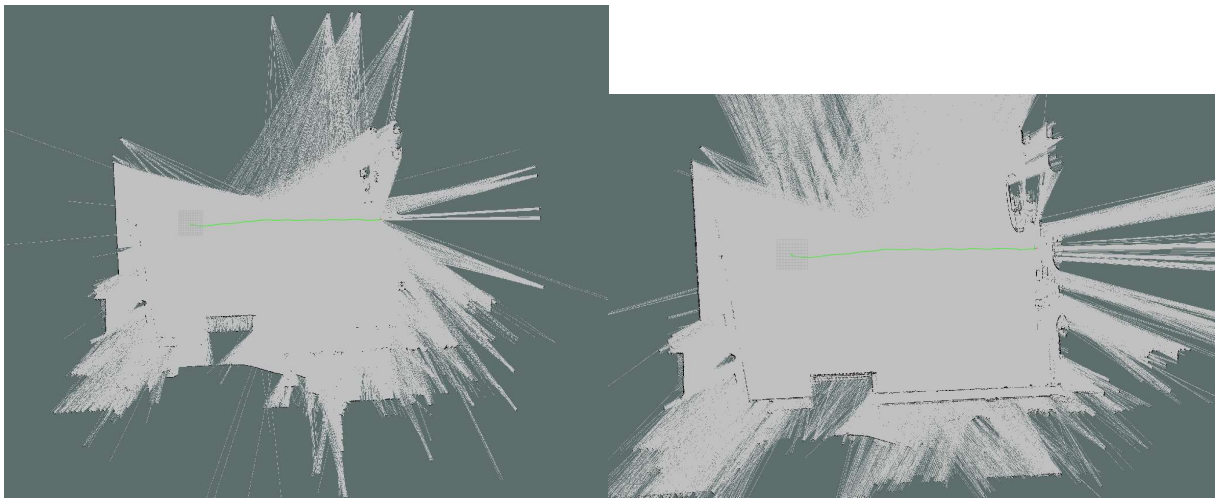
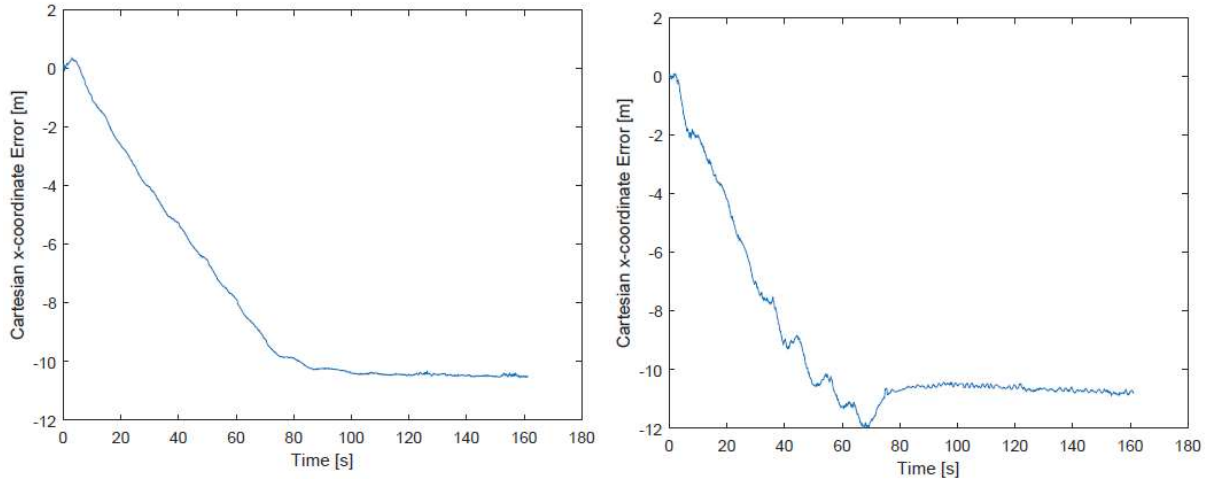


Figure 6.10: Resulting heading error from Hector-SLAM run on the last 446 seconds.



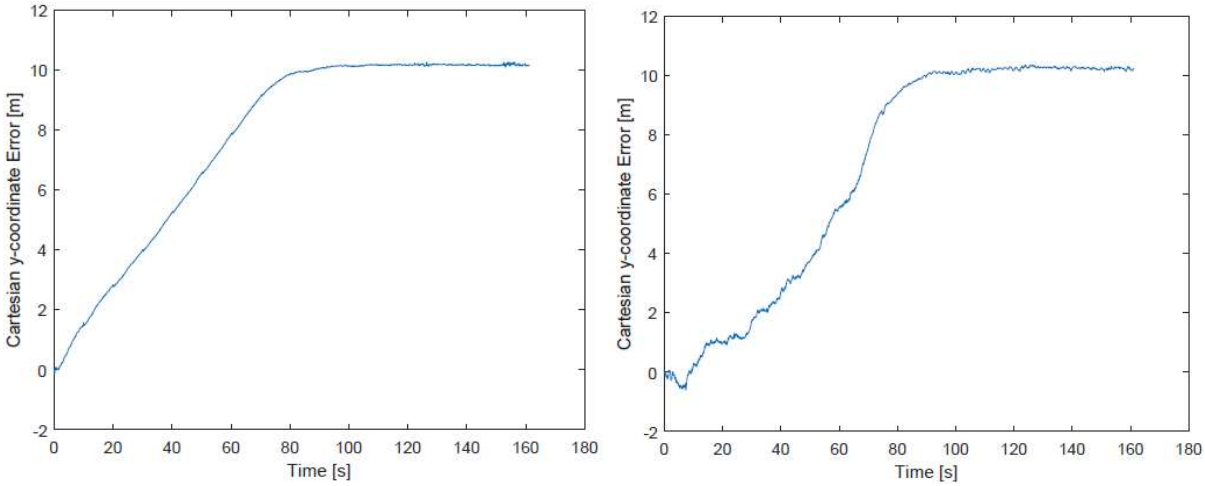
(a) (b)

Figure 6.11: (a) shows the result of regular Hector-SLAM run on the last 161 seconds. (b) shows the result of Hector-SLAM run on down-projected scans from the last 161 seconds.



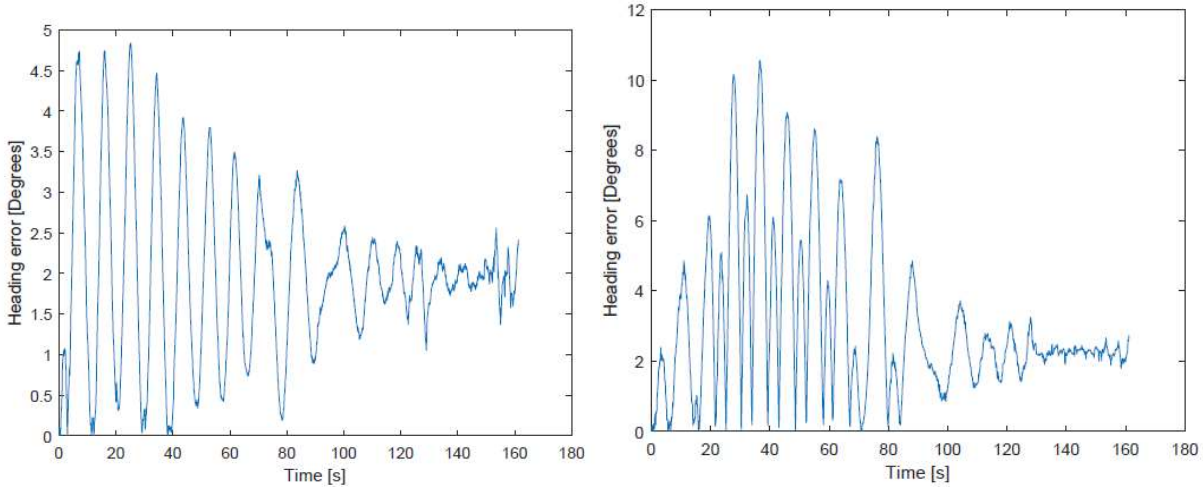
(a) (b)

Figure 6.12: (a) shows the error in x for regular Hector-SLAM run on the last 161 seconds. (b) shows the error in x for Hector-SLAM run on down-projected scans from the last 161 seconds.



(a) (b)

Figure 6.13: (a) shows the error on y for regular Hector-SLAM run on the last 161 seconds. (b) shows the error in y for Hector-SLAM run on down-projected scans from the last 161 seconds.



(a) (b)

Figure 6.14: (a) shows the error in heading for regular Hector-SLAM run on the last 161 seconds. (b) shows the error in heading for Hector-SLAM run on down-projected scans from the last 161 seconds.

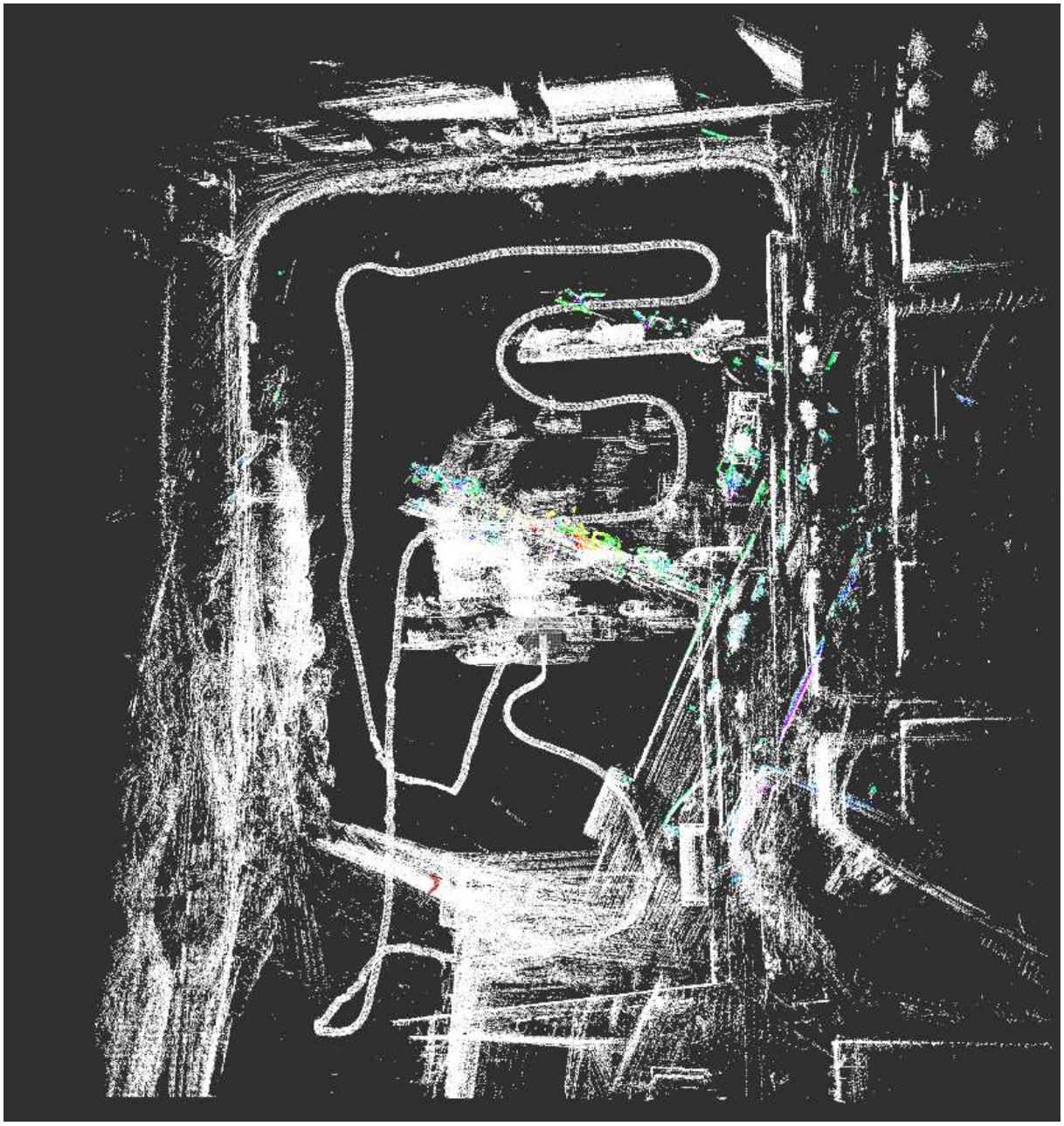
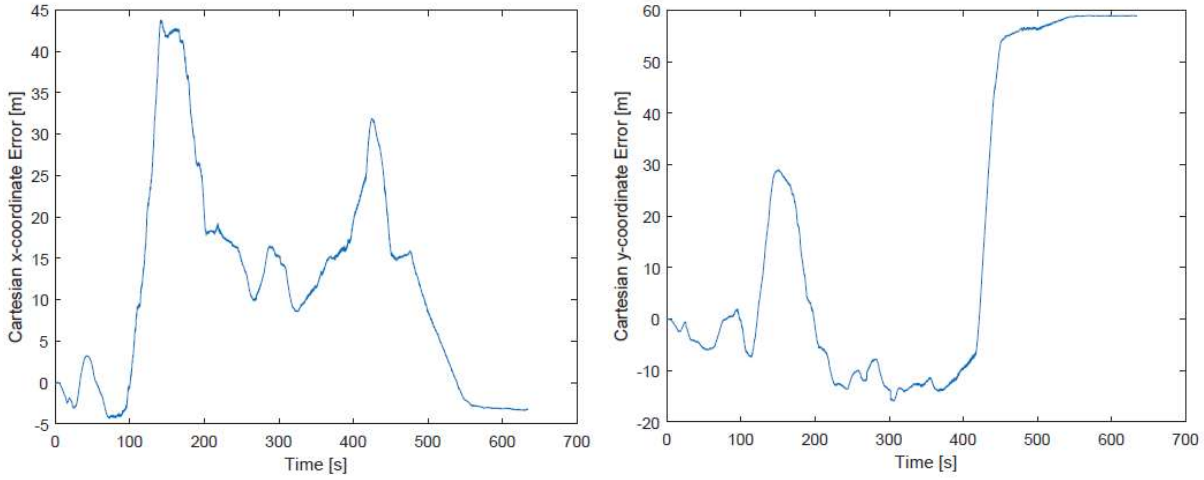
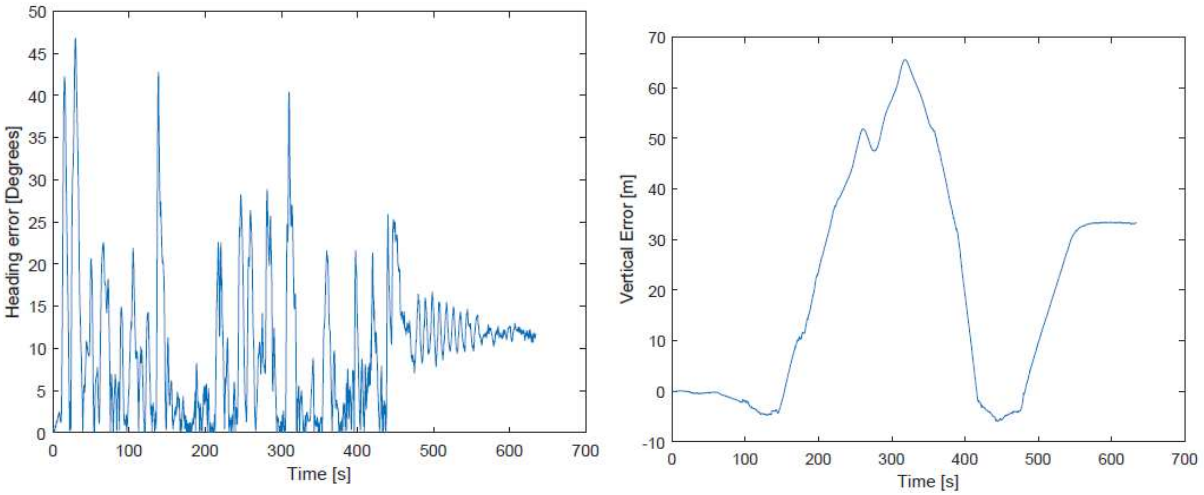


Figure 6.15: Result of LOAM run on the full data-set without filtering measurements of MilliAmpere



(a) (b)

Figure 6.16: (a) shows the error in x for LOAM run on the full data-set. (b) shows the error in y for LOAM run on the full data-set. The point cloud filter was not included in this test.



(a) (b)

Figure 6.17: (a) shows the error in heading for LOAM run on the full data-set. (b) shows the vertical drift for LOAM run on the full data-set. The point cloud filter was not included in this test.

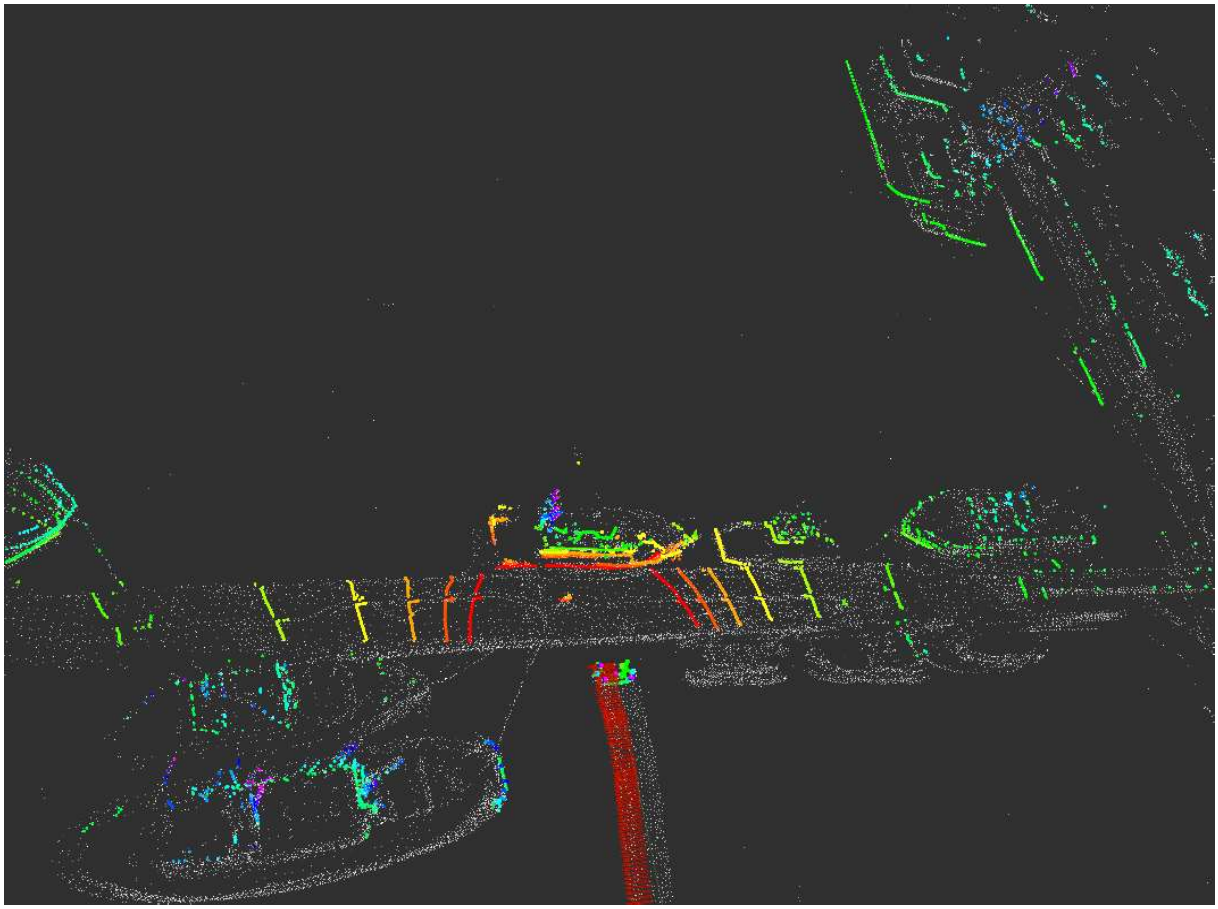
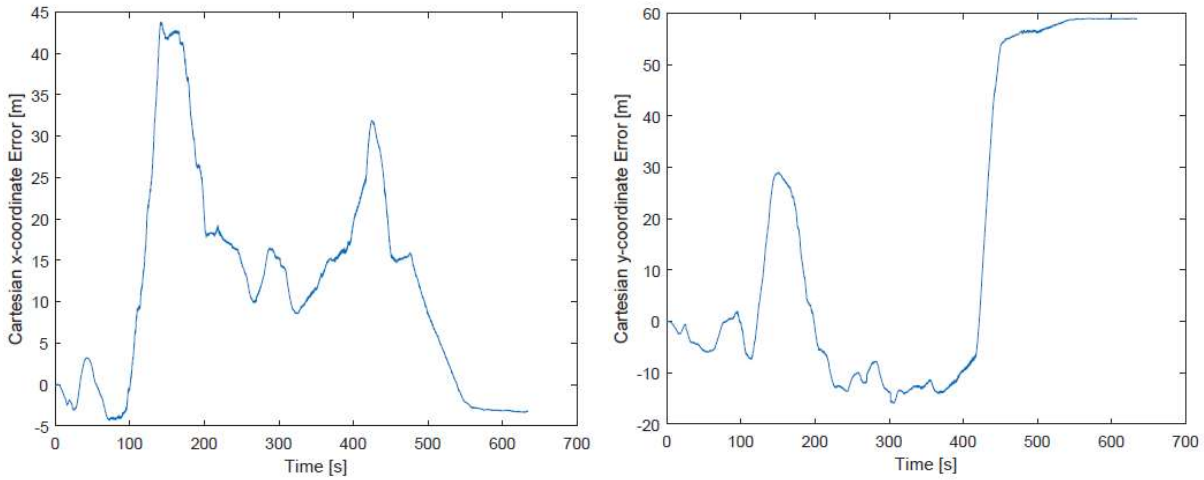
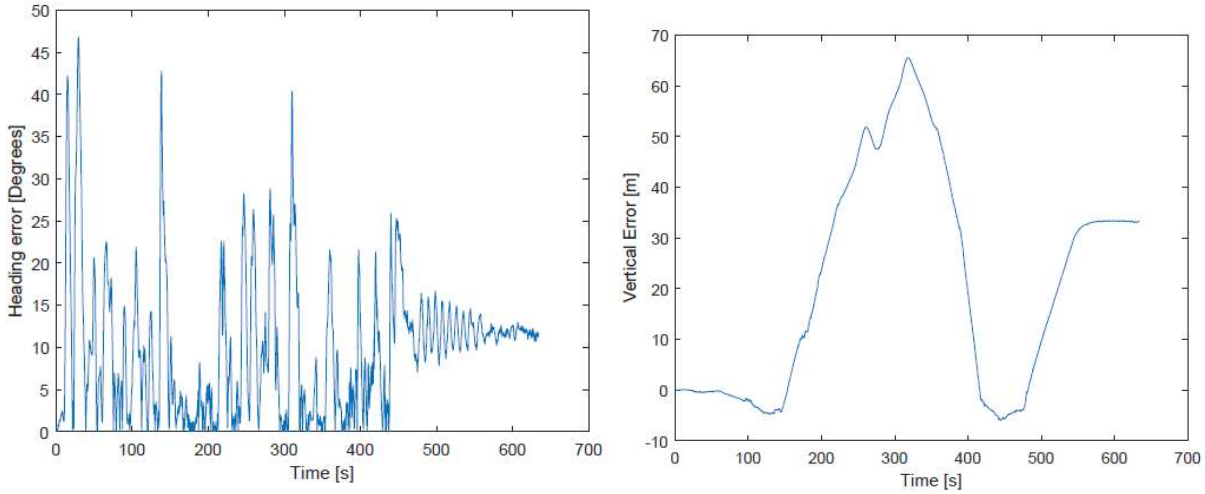


Figure 6.18: Close-up of the LOAMs mapping of the dock, as the map building if the small set is finished.



(a) (b)

Figure 6.19: (a) shows the error in x for LOAM run on the full data-set. (b) shows the error in y for LOAM run on the full data-set. The point cloud filter was not included in this test.

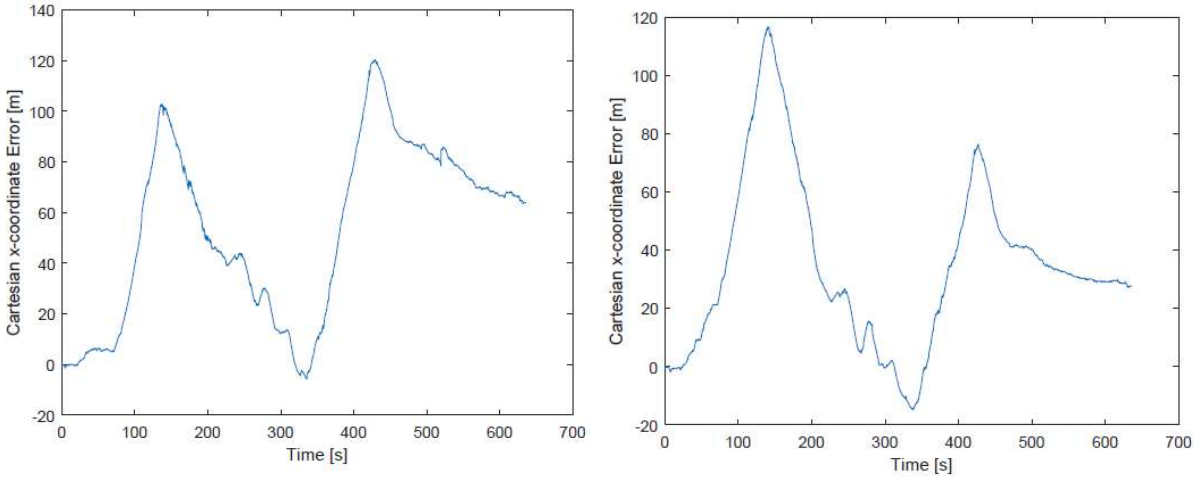


(a) (b)

Figure 6.20: (a) shows the error in heading for LOAM run on the full data-set. (b) shows the vertical drift for LOAM run on the full data-set. The point cloud filter was not included in this test.

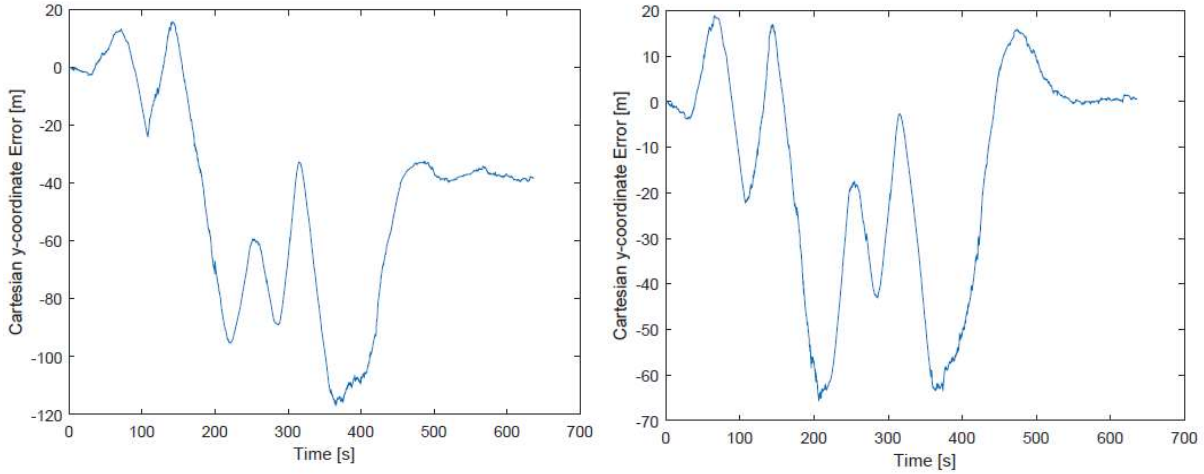
Figure 6.21: Resulting map when running BLAM on the full data-set with filtered point clouds

Figure 6.22: Resulting map when running BLAM on the full data-set with unfiltered point clouds



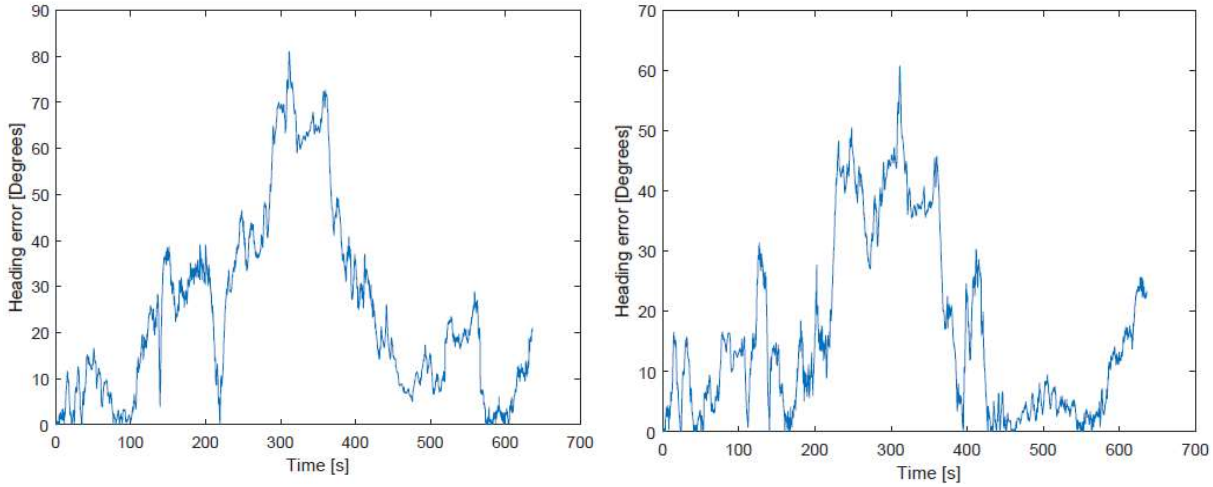
(a) (b)

Figure 6.23: (a) shows error in x-position from BLAM run on the full data-set with point cloud filter. (b) shows error in x-position from BLAM run on the full data-set without point cloud filter.



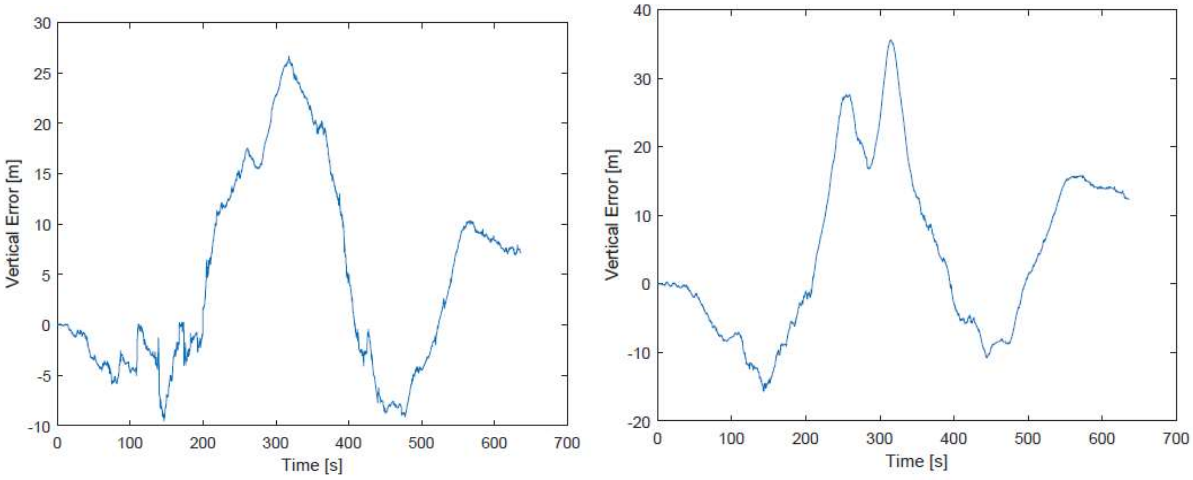
(a) (b)

Figure 6.24: (a) shows error in y-position from BLAM run on the full data-set with point cloud filter. (b) shows error in y-position from BLAM run on the full data-set without point cloud filter.



(a) (b)

Figure 6.25: (a) shows error in heading from BLAM run on the full data-set with point cloud filter. (b) shows error in heading from BLAM run on the full data-set without point cloud filter.



(a) (b)

Figure 6.26: (a) shows vertical drift from BLAM run on the full data-set with point cloud filter. (b) shows vertical drift from BLAM run on the full data-set without point cloud filter.

Chapter 7

Discussion

This chapter is devoted to discussion of the results presented in Chapter 6, together with discussion of elements of experimental setup and software implementation.

7.1 SLAM Performance and Implementation Improvements

Hector-SLAM was performed on data in [39], but only evaluated by means of recognizing that the trajectories look similar. When the corridor is avoided, the position estimates are rather impressive. However, a question remains if the lidar is located in the canal, will we see behaviour like in the canal? In any case, the issue should be addressed if this is a method decided to develop further.

IMU integration would likely help. However, discarding SLAM odometry when in a corridor-like environment and relying solely on integrated gyro- and acceleration data will cause the solution to drift as well. A likely better option is to implement the Covariance Intersection sensor fusion method described in [13], with perhaps also including GPS-data when available. Preliminary work was begun on the matter before it was discovered that Hector-SLAM failed when projecting the point clouds down to the 2D-plane, and focus shifted to other methods, like Cartographer, LeGO-LOAM, LOAM and BLAM. In hindsight, that is regarded as a mistake, with what results achieved from there being significantly poorer.

The output map from Hector-SLAM, however, is not very useful when based on only one channel. The floating docks were almost never included bar the only successful run with the down-projected point cloud. Thus, the current implementation could never be used for docking purposes except for if using maneuvering based on distance to visible target with known pose related to the place of docking.

Another apparent mistake was focusing on BLAM. As the inclusion of loop-closure using the GTSAM and a Youtube-video demonstrating it was the appealing factors with the algorithm, it is disappointing that it could not be seen in practice on the logged data-set. The lack of documentation should have been a cue to leave it alone.

LOAM can ultimately not show for particularly good results in the previous chapter. However, the beginning 30 seconds or so of building maps look quite promising, before eventually rotating pitch and error, and starting the vertical drift. Looking at the map structures in 6.18, they are impressively similar. The algorithm obviously suffer some bugs affecting the pose estimate, which should be investigated. It would also be interesting to see constraints introduced on the vertical position, pitch and roll, and then study the results. Though not achieved in the scope of this thesis, integration with IMU/data is do/able, and would most likely improve results.

7.2 Experiment Improvements

As mentioned in Chapter 4, the preferred logging area was the proposed operational area. As this was not possible on the actual day, perhaps efforts should have been made towards doing another run. Another run would be beneficial in another sense, namely to switch lidar placement. It is strongly believed that limiting the vertical FOV on half the LIDARS horizontal FOV causes confusion for all scan matchers applied in the scope of this thesis. Thus, neither SLAM solutions can really be discarded until they have been run with the lidar elevated above the roof. The 2D Hector/SLAM implementation is only minimally affected by this, as only a few points hit the roof and had to be filtered.

Another tweak that could have been tested, is changing the setup of the lidar, such that scan rate is 20Hz while horizontal resolution is halved. As (hector-paper) states that the algorithm expects scan rates of 20Hz, it would be intriguing to see which effect, if any, that measure would have had.

The error-graphs on on the smaller data-sets compared to the full data-set indicates that the initial heading measured by the GPS is perhaps not a very good match to the initial SLAM pose when the lidar has initial motion. This is particularly clear in the Hector-SLAM results, where the earlier poses are much more accurate on the full data-set than the smallest, even though the scans are from the same area and and the former includes much more rotational motion. Such a mismatch in the initial heading will cause the visualized result to drift, even though it might not actually be the case. Based on this, it would likely be smarter to ensure a brief motionless period when initializing the SLAM algorithms.

Chapter 8

Conclusions and Future Work

8.1 Conclusion

A series of assumed scenarios for an operational autonomous passenger ferry in a harbour environment have been planned and executed, resulting in a comprehensive data-set which can prove valuable for future work and research. On this data, the SLAM methods Hector-SLAM, LOAM and BLAM with varying results. These results shows proof of concept for Hector-SLAM, and the following section presents ideas moving toward a more complete solution. LOAM shows potential in particular map building, but and pose estimation based on the shape of the trajectories. With its current implementation, however, the algorithm suffers too much drift and difficulties in corridor-like environments. With BLAM not showing for any loop closure capabilities in the results, the algorithm comes off ass inferior to LOAM as they are quite similar in structure.

8.2 Suggestions for Future Work

8.2.1 Collecting New Data

As a part of moving forward towards achieving a SLAM solution for the autonomous ferry, there are incentives to collect new data.

Lidar Placement

There are at least two more interesting locations on the ferry to place the lidar. That is, elevated atop the roof to take full advantage of the LIDAR's 2.5D FOV, but also on the edge of the roof tilted down towards the water such that docks are guaranteed to be visible. From experience, it is advised to start considering mounting devices and how to acquire it/them. To avoid lever-arm compensation from lidar to IMU, the it is suggested to elevate the lidar above the IMUs horizontal position.

Sensor Drivers

Before collecting new data, the current sensor driver setups on the MilliAmpere OBC should be examined and modified. Much data from both the IMU and the GNSS-receiver was disabled while collecting the data-set for this thesis, data which could prove useful for i.e. sensor fusion. Also, the timestamp-bug in the Xsens-driver should be addressed, and both lidar-time and IMU-time should preferably be synchronized with the GPS-clock of the GNSS-receiver.

8.2.2 SLAM Algorithms Moving Forward

As mentioned in Chapter 7, results indicates that it is worth moving forward with the Hector-SLAM algorithm, both regarding fusion with IMU- and GPS-data, as well as attempting another down-projection in case new data is collected. In that case, running LOAM would not hurt either, to check its performance under a richer point cloud.

It could also prove worth it to have a go at LEGO-LoAM, to see if any updates have been made. The result presented in the thesis seem very promising. It is not mentioned as alternatives in Chapter 3 and has not been investigated other than briefly reading the paper [36], but SegMatch could me another interesting approach. The method uses deep learning to perform segmentation based loop closure for 3D point clouds, providing a different route than previously explored. Using deep learning approaches, the method requires training.

8.2.3 Investigating Point Cloud Features

The thesis work never got this far, but it could be worth it to experiment with different feature extraction principles, to see what type features are present in a typical point cloud in a harbor environment. This could be especially useful if an attempt to develop a purpose-built scan matcher.

Bibliography

[1] P. Biber and W. Strasser, "The normal distributions transform: a new approach to laser scan matching", in Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2003, vol. 3, pp. 2743–2748.

[2] F. Lindseth, Class Lecture, Topic: "Object Detection.", TDT4265, NTNU, Trondheim, Spring 2018.

[3] A. Neal, "LiDAR vs. RADAR", 2018. [Online]. Available: https://www.sensorsmag.com/components/lidar-vs-radar [Accessed: 18- Dec- 2018].

[4] A. Doucet, N. Freitas, K. Murphy, S. Russell "Rao-Blackwellised Particle Filtering for Dynamic Bayesian Networks," in UAI '00 Proceedings of the 16th Conference of Uncertainty in Artificial Intelligence, 2000, pp. 176-183.

[5] 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, 2011, pp. 216–235, 2012.

[6] M. Montemerlo, W. Whittaker, S. Thrun, A. Stentz, and D. Fox, "FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem With Unknown Data Association", in Eighteenth national conference on Artificial intelligence, 2003, pp. 593-598.

[7] Ø. U. Halvorsen, "Collaborating Robots Multi-robot Exploration of an Unknown.", Master Thesis, Norwegian University of Science and Technology, 2014.

[8] M. W. M. Gamini Dissanayake, P. Newman, S. Clark, H. F. Durrant-Whyte, and M. Csorba, "A solution to the simultaneous localization and map building (SLAM) problem," IEEE Transactions on Robotics and Automation, vol. 17, pp. 229-241, 2001.

[9] J. Fuentes-Pacheco, J. Ruiz-Ascencio, and J. M. Rendón-Mancha, "Visual Simultaneous Localization and Mapping: A Survey," Artificial Intelligence Review, vol. 43, no. 1, 2015, pp. 55–81,.

[10] B. Ferris, D. Fox, N. Lawrence, "WiFi-SLAM Using Gaussian Process Latent Variable Models", Proceedings of the 20th International Joint Conference on Artificial Intelligence, 2007, pp. 2480-2485.

[11] J. Zhang and S. Singh, "Lidar Odometry and Mapping in Real-time.", in Robotics: Science and Systems Conference, 2014.

[12] D. A. Wolfie, "Quantifying Aerial LiDAR Accuracy of LOAM for Civil Engineering Applications", Project Thesis, Brigham Young University, 2016.

[13] S. Kohlbrecher, O. Von Stryk, J. Meyer, and U. Klingauf, "A flexible and scalable SLAM system with full 3D motion estimation," in 9th IEEE International Symposium on Safety, Security, and Rescue Robotics, 2011, pp. 155–160.

[14] A. Diosi and L. Kleeman, "Laser scan matching in polar coordinates with application to SLAM," in 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2005, pp. 1439–1444.

[15] D. Hahnel, W. Burgard, D. Fox, and S. Thrun, "An efficient fastslam algorithm for generating maps of large-scale cyclic environments from raw laser range measurements," in 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems, vol. 1, 2003, pp. 206–211.

[16] G. Costante, M. Mancini, P. Valigi, T. A. Ciarfuglia, and T. A. Ciarfuglia, "Exploring Representation Learning With CNNs for Frame-to-Frame Ego-Motion Estimation" no. 1, 2015.

[17] J. Whitley, J. O'Quin, "velodyne - ROS Wiki." [Online]. Available: http://wiki.ros.org/velodyne. [Accessed: 11-Jun-2018].

[18] C. Fox, M. Evans, M. Pearson, and T. Prescott, "Tactile SLAM with a biomimetic whiskered robot," in IEEE International Conference on Robotics and Automation, 2012, pp. 4925–4930.

[19] Sebastian Thrun, Dieter Fox, Wolfram Burgar, and Frank Dellaert, "Robust Monte Carlo localization for mobile robots," Artificial Intelligence, vol. 128, no. 1–2, pp. 99–141, 2000.

[20] T. Bailey, J. Nieto, and E. Nebot, "Consistency of the FastSLAM Algorithm.", in IEEE International Conference on Robotics and Automation, 2006.

[21] R. C. Smith and P. Cheeseman, "On the Representation and Estimation of Spatial Uncertainty," International Journal of Robotic Research, vol. 5, no. 4, pp. 56–68, 1986.

[22] P. Besl and N. McKay, "A Method for Registration of 3-D Shapes," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 14, no. 2. pp. 239–256, 1992.

[23] J. F. Dave Hershberger, David Gossow, "rviz - ROS Wiki," ROS.org, 2009. [Online]. Available: http://wiki.ros.org/rviz. [Accessed: 15-Jan-2019].

[24] J. Meyer and S. Kohlbrecher, "hectorslam - ROS Wiki." [Online]. Available: http://wiki.ros.org/hector slam. [Accessed: 11-Jun-2018].

[25] D. Filliat, A. Angeli, S. Bazeille, N. Beaufort and Z. Ni "Loop Closure Detection", [Online]. Available: http://cogrob.ensta-paristech.fr/loopclosure.html. [Accessed: 06-Jan-2019].

[26] K. P. Murphy, "Bayesian map learning in dynamic environments", in Proceed of the 12th International COnference on Neural Information Processing Systems, pp. 1015–1021, 1999.

[27] B. Steux and O. El Hamzaoui, "tinySLAM: A SLAM algorithm in less than 200 lines C-language program," in 11th International Conference on Control, Automation, Robotics and Vision, 2010, pp. 1975–1979.

[28] H. K. Melbø, "Autonomous Multi-Robot Mapping," Master Thesis, Norwegian University of Science and Technology, 2017.

[29] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection", IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 779-788.

[30] K. Naus and Ł. Marchel, "SLAM aided Inertial Navigation System", Scientific Journal of Polish Naval Academy, vol. 200, 2015.

[31] T. Magnussen, "Fjernstyring av Legorobot.", Master Thesis, Norwegian University of Science and Technology, 2008.

[32] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem.", in Proceedings of the AAAI National COnference on Artificial INtelligence, 2002, pp. 583-598.

[33] D. Kurth, "Range-only robot localization and SLAM with radio," in Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems, Techincal Report, Carnegie Mellon University, 2004.

[34] T. Bailey and H. Durrant-Whyte, "Simultaneous localization and mapping (SLAM): Part I," IEEE Robotics and Automation Magazine, vol. 13, no. 3, pp. 108–117, 2006.

[35] E. Ackerman and E. Guizzo, "iRobot brings visual mapping and navigation to the Roomba 980," IEEE Spectrum, 2015. [Online]. Avalable: https://spectrum.ieee.org/automaton/robotics/home-robots/irobot-brings-visual-mapping-and-navigation-to-the-roomba-980

[36] R. Dube, D. Dugas, E. Stumm, J. Nieto, R. Siegwart, and C. Cadena, "SegMatch: Segment based place recognition in 3D point clouds," in Proceedings - IEEE International Conference on Robotics and Automation, 2017, pp. 5266–5272.

[37] "KAARTA" [Online]. Available: http://www.kaarta.com/. [Accessed: 15-Jan-2019].

[38] K. Kovach, P. J. Mendicki, and B. Renfro, "GPS Receiver Impact from the UTC Offset (UTCO) Anomaly of 25-26 January 2016", Proceedings of the 29th International Technical Meeting og The Satellite Institute of Navigation, 2016, pp. 2887-2895.

[39] M. Ødven, "Hector-SLAM for Autonomous Ferry", Project Report, Norwegian University of Science and Technology, 2018.

[40] M. Berg, "Navigation with Simultaneous Localization and Mapping For Indoor Mobile Robot", Master Thesis, Norwegian University of Science and Technology, 2013.

[41] E. S. Ueland, "Marine Autonomous Exploration using a Lidar", Master Thesis, Norwegian University of Science and Technology, 2016.

[42] H. Alfheim and K. Muggerud, "Development of a Dynamic Positioning System for the ReVolt Model Ship", Master Thesis, Norwegian University of Science and Technology, 2017.

[43] H. Alfheim and K. Muggerud, "Dynamic Positioning of the ReVolt Model-Scale Ship," Project Report, Norwegian University of Science and Technology, 2017.

[44] S. Riisgaard and M. R. Blas, "SLAM for Dummies, A Tutorial Approach to Simultaneous Localization and Mapping By the 'dummies'", 2004.

[45] A. Eliazar and R. Parr, "DP-SLAM: Fast, Robust Simultaneous Localization and Mapping Without Predetermined Landmarks.", in Proceedings of the 18th International Joint Conference on Artificial Intelligence, 2003, pp. 1135-1142.

[46] T. Shan and B. Englot, "LeGO-LOAM: Lightweight and Ground-Optimized Lidar Odometry and Mapping on Variable Terrain", 2018.

[47] T. J. Chong, X. J. Tang, C. H. Leng, M. Yogeswaran, O. E. Ng, and Y. Z. Chong, "Sensor Technologies and Simultaneous Localization and Mapping (SLAM)," in Procedia Computer Science, vol. 76, pp. 174–179, 2015.

[48] S. Thrun and M. Montemerlo, "The GraphSLAM Algorithm with Applications to Large-Scale Mapping of Urban Structures", in INternational Journal of Robotics Research, vol. 25, pp. 403-430, 2006.

[49] G. Grisetti, C. Stachniss, and W. Burgard, "Improved techniques for grid mapping with Rao-Blackwellized particle filters," IEEE Transactions on Robotics, vol. 23, pp. 34–46, 2007.

[50] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Cartographer - Cartographer Documentation," 2018. [Online]. Available: https://google-cartographer.readthedocs.io/en/latest/. [Accessed: 15-Jan-2019].

[51] Z. Zhang, "Iterative point matching for registration of free-form curves and surfaces," International Journal of Compututer Vision, vol. 13, pp. 119–152, 1994.

[52] M. Kaess, A. Ranganathan, F. Dellaert, "iSAM: Incremental Smoothing and Mapping", in IEEE Transactions on Robotics, vol. 24, pp. 1365-1378, 2008.

[53] E. Nelson, "B(erkeley) L(ocalization) A(nd) M(apping)", [Online]. Available: https://github.com/erik-nelson/blam [Accessed: 22-Nov-2018]."

[54] E. Nelson "Wide-Area Indoor and Outdoor Real-Time 3D SLAM", [Online]. Available: https://www.youtube.com/watch?v=08GTGfNneCIfeature=youtu.be [Accessed: 22-Nov-2018].

[55] T. Field, J.Leibs, J. Bowman, "rosbag - ROS Wiki", [Online]. Available: http://wiki.ros.org/rosbag. [Accessed: 15-Jan-2019].

[56] E. Perko, S. Martin, "nmea_navsat_driver - ROS Wiki", [Online]. Available: http://wiki.ros.org/nmea_navsat_driver. [Accessed: 15-Jan-2019].

[57] P. Bovbel, T. Foote, "pointcloud_to_laserscan - ROS Wiki", [Online]. Available: http://wiki.ros.org/pointcloud_to_laserscan. [Accessed: 15-Jan-2019].

[58] P. Bovbel, K. Wada, "perception_pcl - ROS Wiki", [Online]. Available: http://wiki.ros.org/perception_pcl. [Accessed: 15-Jan-2019].

[59] D. A. Wolfie, "loam_velodyne", [Online]. Available: https://github.com/ laboshinl/loam_velodyne [Accessed: 16-Nov-2018].

[60] F. Dellaert et. al., "Georgia Tech Smoothing and Mapping library", [Online]. Available: https://bitbucket.org/gtborg/gtsam [Accessed: 22-Nov-2018].

Appendix

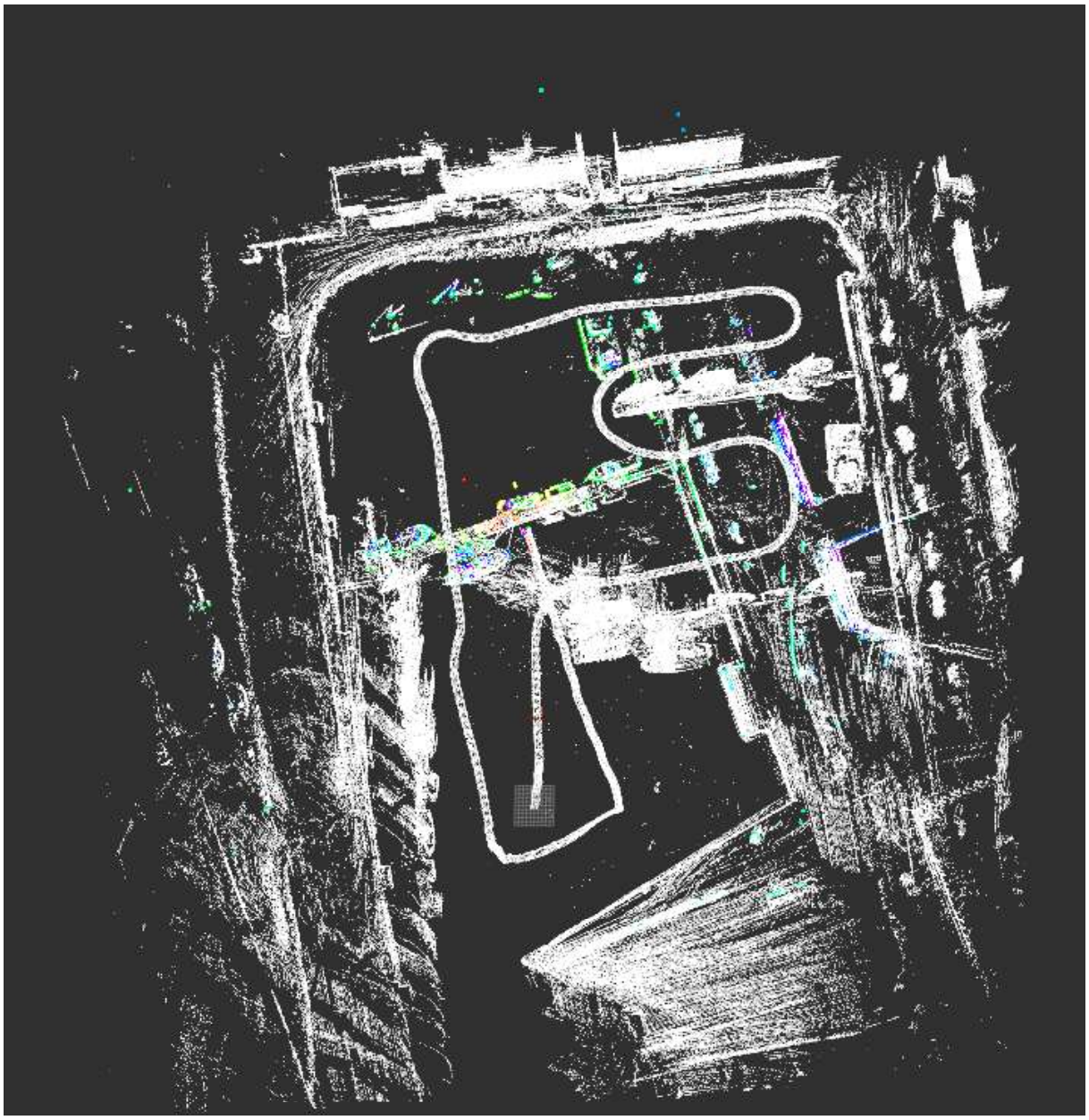


Figure 1: Result of LOAM run on the mid-sized without filtering measurements of MilliAmpere, before ultimately crashing.

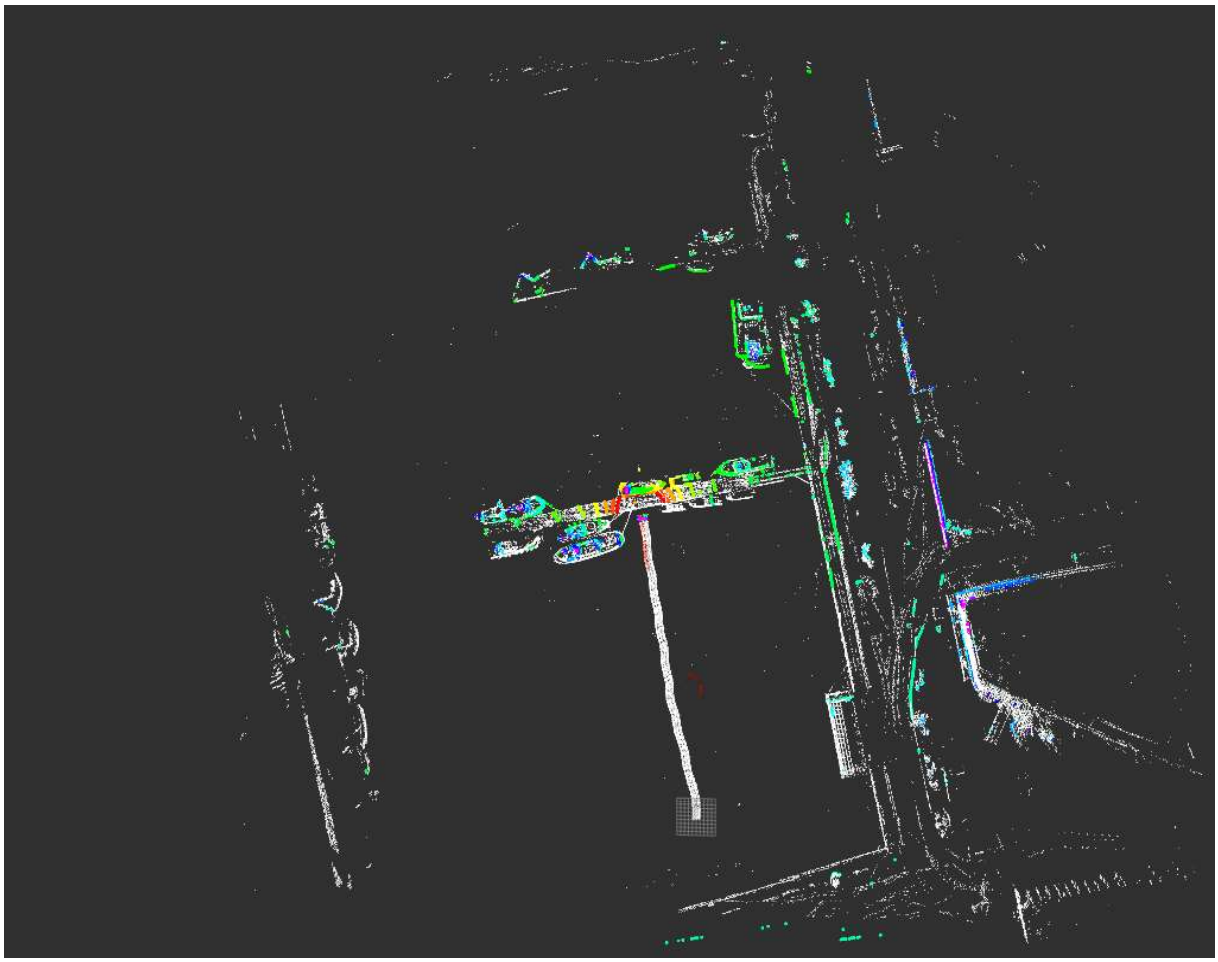
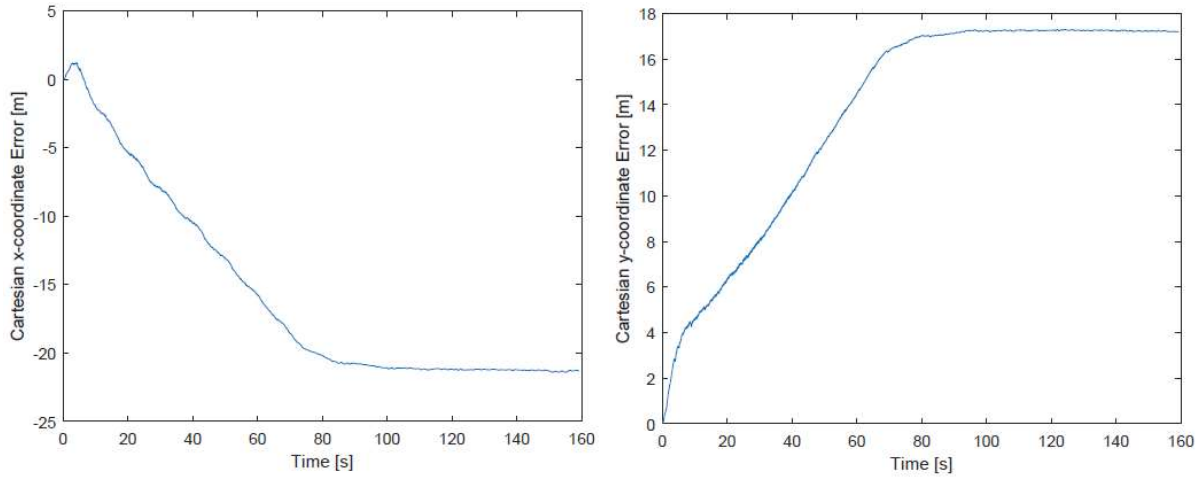
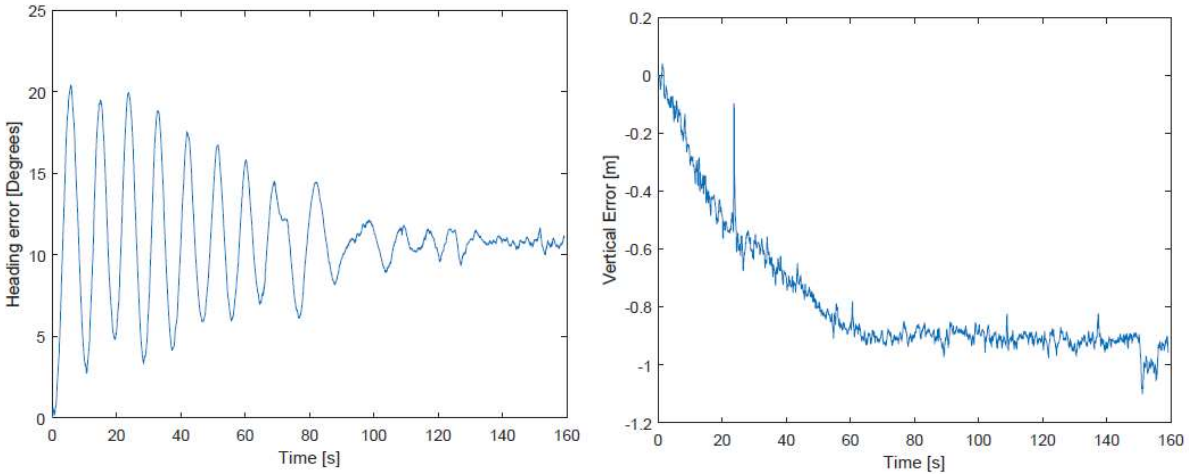


Figure 2: Resulted map after running LOAM on the small subset.



(a) (b)

Figure 3: (a) shows the error in x for LOAM run on the last 161 seconds. (b) shows the error in y for LOAM run on the last 161 seconds.



(a) (b)

Figure 4: (a) shows the error in heading for LOAM run on the last 161 seconds. (b) shows the vertical drift for LOAM run on the last 161 seconds.

Figure 5: Resulting map when running BLAM on the middle sized data-set with filtered point clouds

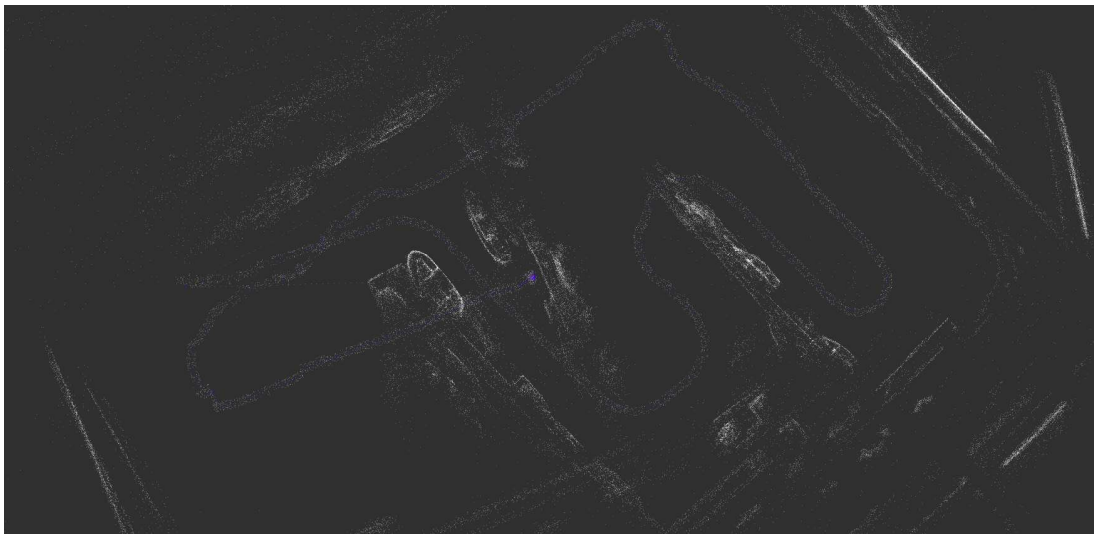
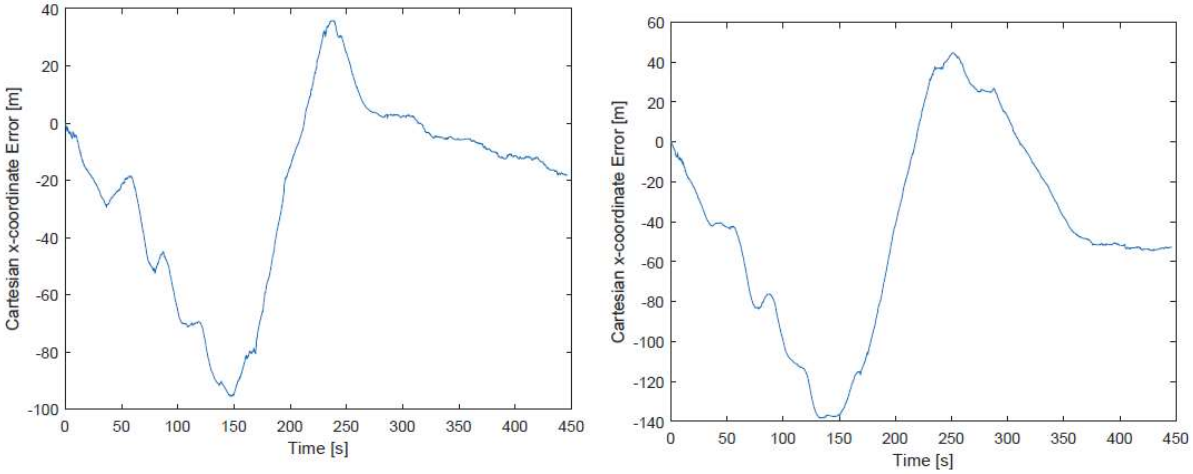
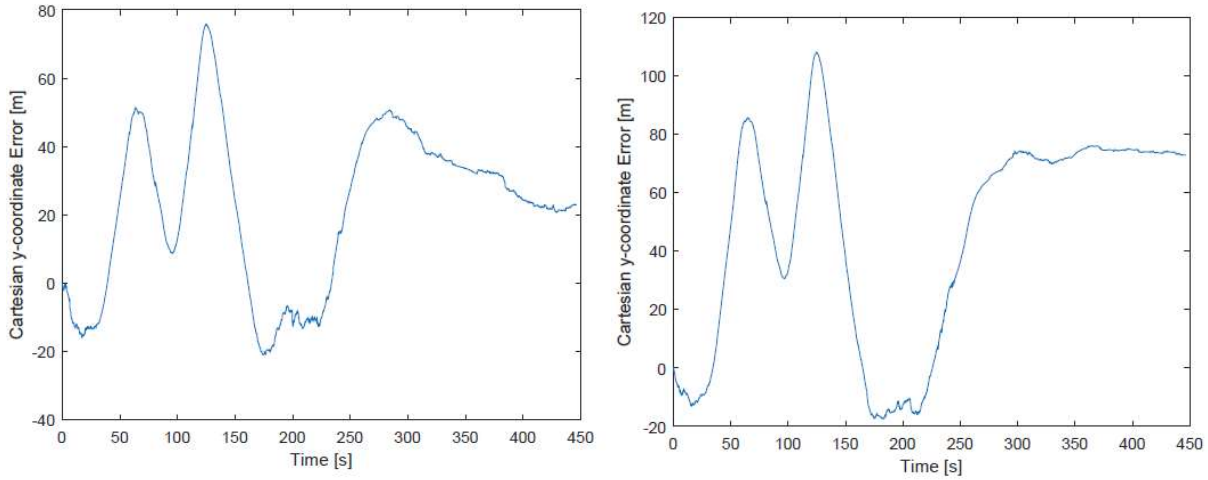


Figure 6: Resulting map when running BLAM on the middle data-set with unfiltered point clouds



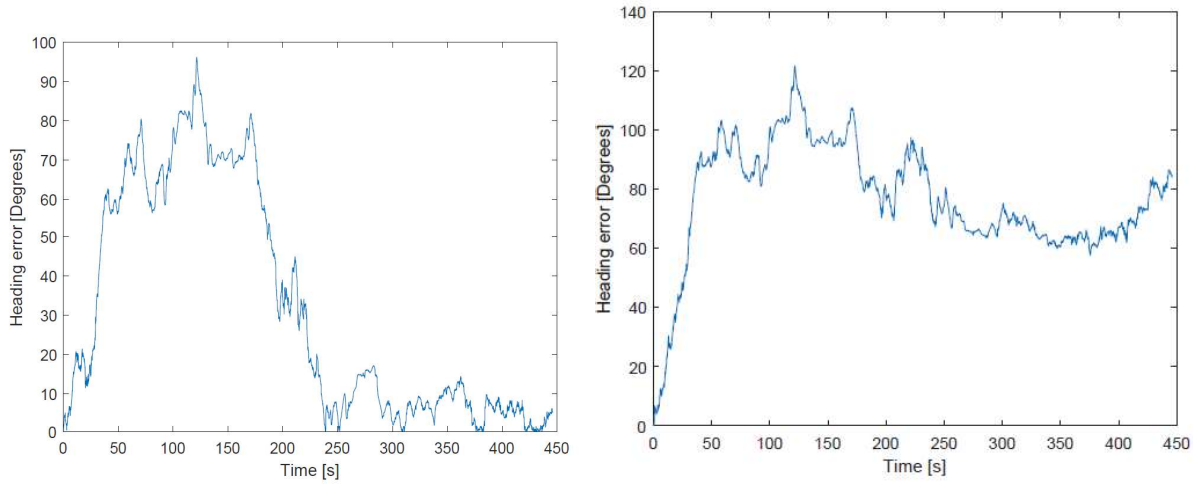
(a) (b)

Figure 7: (a) shows error in x-position from BLAM run on the middle sized data-set with point cloud filter. (b) shows error in x-position from BLAM run on the middle sized data-set without point cloud filter.



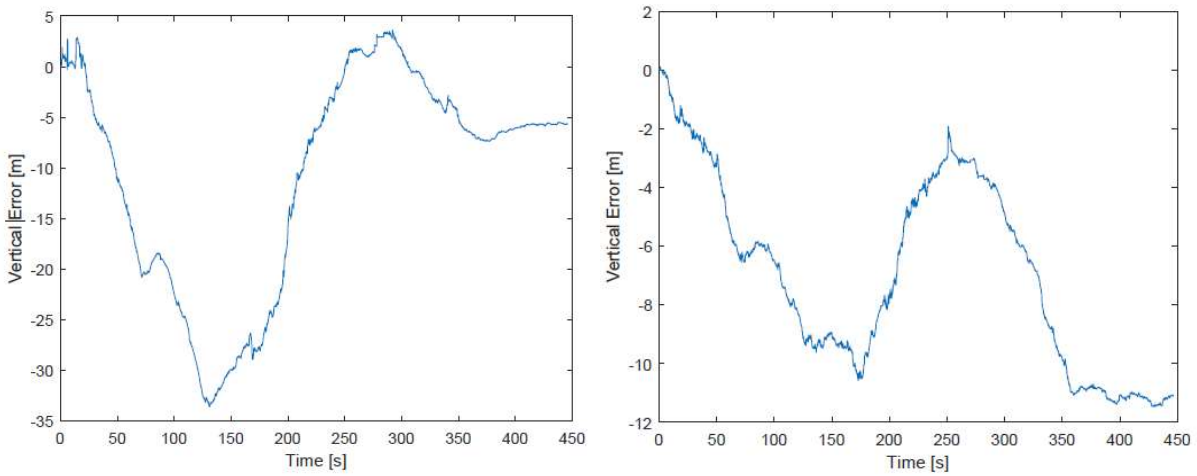
(a) (b)

Figure 8: (a) shows error in y-position from BLAM run on the middle sized data-set with point cloud filter. (b) shows error in y-position from BLAM run on the middle sized data-set without point cloud filter.



(a) (b)

Figure 9: (a) shows error in heading from BLAM run on the middle sized data-set with point cloud filter. (b) shows error in heading from BLAM run on the middle sized data-set without point cloud filter.

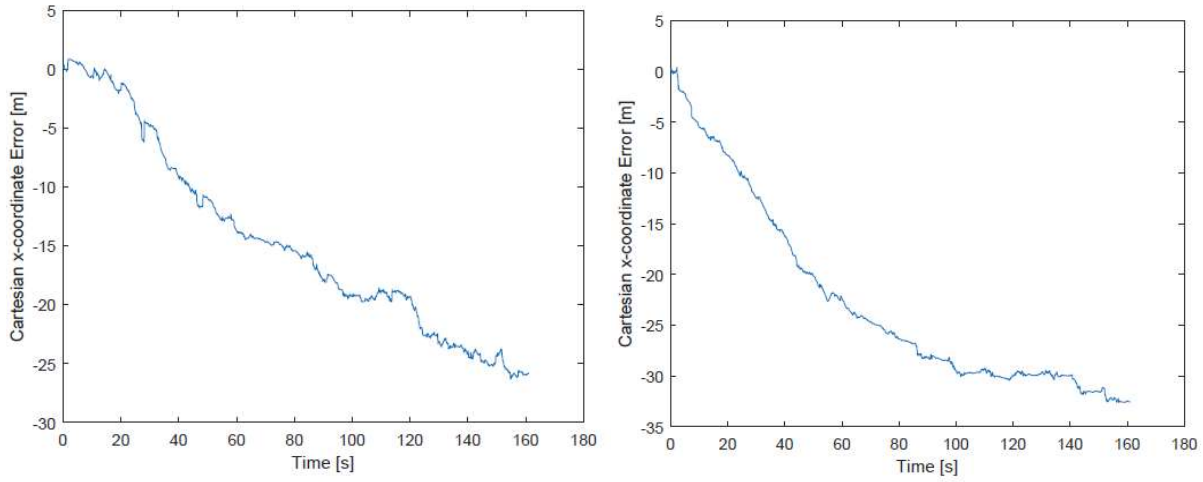


(a) (b)

Figure 10: (a) shows vertical drift from BLAM run on the middle sized data-set with point cloud filter. (b) shows vertical drift from BLAM run on the middle sized data-set without point cloud filter.

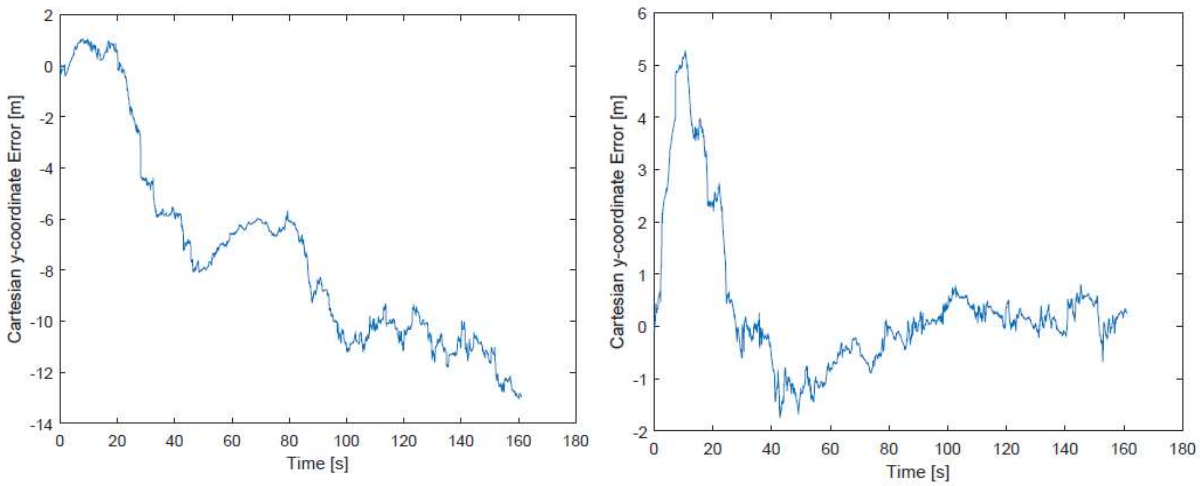
Figure 11: Resulting map when running BLAM on the small data-set with filtered point clouds

Figure 12: Resulting map when running BLAM on the small data-set with unfiltered point clouds



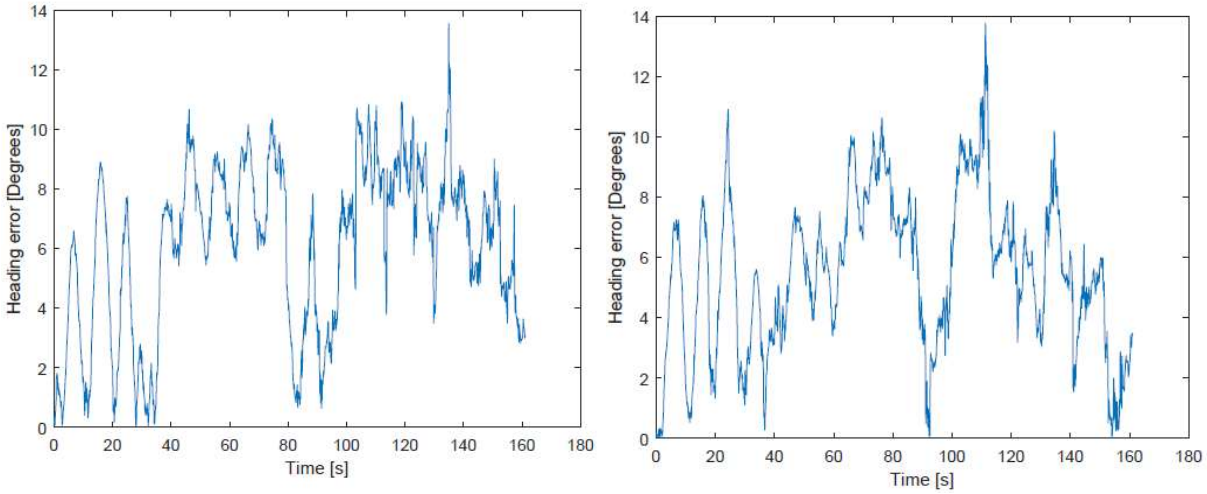
(a) (b)

Figure 13: (a) shows error in x-position from BLAM run on the small data-set with point cloud filter. (b) shows error in x-position from BLAM run on the small sized data-set without point cloud filter.



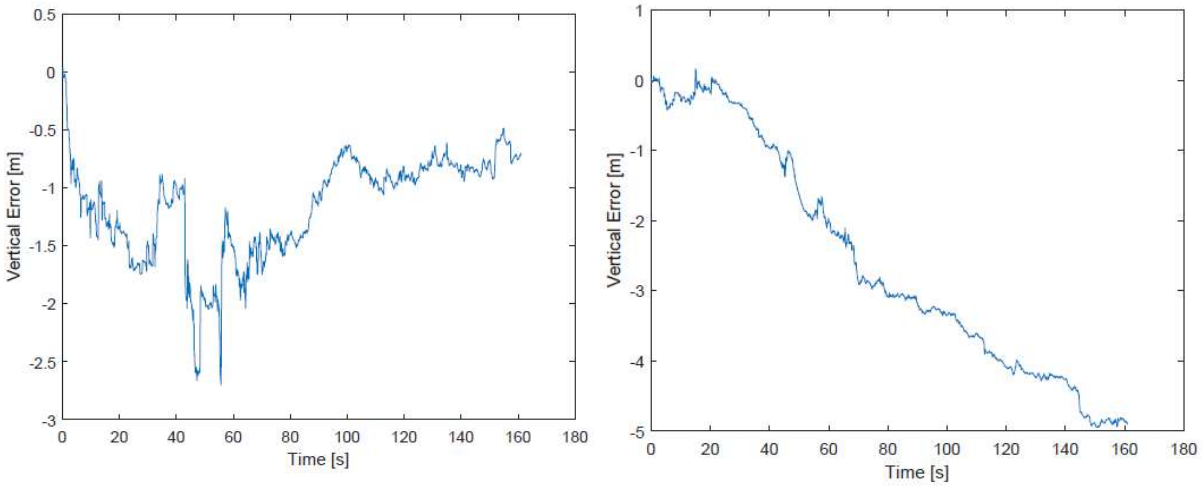
(a) (b)

Figure 14: (a) shows error in y-position from BLAM run on the small sized data-set with point cloud filter. (b) shows error in y-position from BLAM run on the small data-set without point cloud filter.



(a) (b)

Figure 15: (a) shows error in heading from BLAM run on the small data-set with point cloud filter. (b) shows error in heading from BLAM run on the small sized data-set without point cloud filter.



(a) (b)

Figure 16: (a) shows vertical drift from BLAM run on the small sized data-set with point cloud filter. (b) shows vertical drift from BLAM run on the small data-set without point cloud filter.