Automotive Lidar Technology for Marine Applications – Determining and Increasing the Accuracy of Simultaneous Localisation and Mapping

Berin Đikić

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisors: Prof. Pedro Tiago Martins Batista

Thomas Gölles, PhD

Examination Committee

Chairperson: Prof. João Manuel de Freitas Xavier

Supervisor: Prof. Pedro Tiago Martins Batista

Member of the Committee: Prof. Bruno João Nogueira Guerreiro

July 2023

Acknowledgements

I would like to start by expressing my gratitude towards my family who were always there to support me, not only throughout the two years of this Master's programme, but also throughout my entire life. I would also like to thank my friends who were always ready to share all the good and the bad moments with me. A special mention goes to my late grandmother who was always encouraging me to push my limits every day.

I would also like to thank my mentors Dr. Thomas Gölles and Professor Pedro Batista who always offered guidance and showed me the way to solve the problems I encountered, even though sometimes I couldn't see it. Their expertise and commitment were essential in helping me to finish this thesis.

Finally, I would like to thank Stefan Muckenhuber for assisting with the fieldwork and providing guidance throughout the practical work for this thesis. Big thanks to Michael Stolz who devoted time and knowledge to guide me in the topic selection process and who made the initial connection between me and my supervisor Dr. Thomas Gölles. Also, I would like to thank Professors Ricard Marxer and Nicolas Boizot for their involvement and guidance in the initial stages of my search for the thesis topic and placement in the industry. Last but not least, I would like to thank my colleagues Birgit Schlager and Christoph Gaisberger for their help and friendliness.

The project was funded by the program "IKT der Zukunft" of the Austrian Federal Ministry for Climate Action (BMK). The publication was written at Virtual Vehicle Research GmbH in Graz and funded within the COMET K2 Competence Centers for Excellent Technologies from the Austrian Federal Ministry for Climate Action (BMK), the Austrian Federal Ministry for Labour and Economy (BMAW), the Province of Styria (Dept. 12) and the Styrian Business Promotion Agency (SFG). The Austrian Research Promotion Agency (FFG) has been authorised for the programme management.

Abstract

This thesis aims to develop a way of determining the accuracy of 3D Simultaneous Localisation And Mapping (SLAM) generated maps. The datasets for generating these maps are gathered by a novel mobile lidar sensor package called MObile LIdar SENsor System (MOLISENS), which uses inexpensive automotive lidar sensors. This makes it cheap, mobile and robust, thus it is suitable to be used in a multitude of different environments, including marine ones. The gathered data is then fed to four different SLAM algorithms that are tested, and their performance is evaluated based on a set of different criteria. Out of the tested algorithms, only the LIO-SAM algorithm was able to successfully complete the mapping task on all the datasets, and it also achieved the best map accuracy. However, even LIO-SAM had some deficiencies, mainly an inefficient and pretty basic loop-closure functionality. This deficiency was fixed by combining LIO-SAM with another algorithm called Scan Context, yielding an algorithm called SC-LIO-SAM. Scan Context specializes in loop detection, which makes the loop-closure functionality of the SC-LIO-SAM more accurate and robust. Besides making LIO-SAM more robust, by using SC-LIO-SAM, the accuracy of the generated 3D maps was also improved by incorporating GNSS data into the SLAM process, as well as by tuning the parameters of the algorithm itself. In conclusion, SC-LIO-SAM emerges as the optimal algorithm for the specific use cases addressed in this thesis, while the MOLISENS sensor package demonstrates its capability to accurately survey a wide range of environments, including marine settings.

Keywords

Map accuracy, Marine mapping, SLAM, MOLISENS, lidar, LIO-SAM

Resumo

Esta tese tem como objetivo desenvolver um método para determinar a precisão dos mapas gerados pelo sistema de Localização e Mapeamento Simultâneo (SLAM) em 3D. Os conjuntos de dados para gerar esses mapas são coletados por um novo pacote de sensores móveis lidar chamado MObile LIdar SENsor System (MOLISENS), que utiliza sensores lidar automotivos de baixo custo. Isso torna o sistema barato, móvel e robusto, adequado para ser usado em diversos ambientes, incluindo ambientes marinhos. Os dados coletados são então processados por quatro algoritmos SLAM diferentes, cujo desempenho é avaliado com base em critérios diversos. Apenas o algoritmo LIO-SAM conseguiu concluir com êxito a tarefa de mapeamento em todos os conjuntos de dados, alcançando também a melhor precisão nos mapas. Contudo, o LIO-SAM apresentou algumas deficiências, especialmente em relação à funcionalidade de fechamento de loops, que era ineficiente e básica. Essa deficiência foi corrigida ao combinar o LIO-SAM com outro algoritmo chamado Scan Context, resultando no algoritmo SC-LIO-SAM. O Scan Context é especializado em detecção de loops, tornando a funcionalidade de fechamento de loops do SC-LIO-SAM mais precisa e robusta. Além de aumentar a robustez do LIO-SAM, o uso do SC-LIO-SAM também melhorou a precisão dos mapas 3D gerados, incorporando dados GNSS ao processo SLAM e ajustando os parâmetros do algoritmo. Em conclusão, o SC-LIO-SAM emerge como o algoritmo ideal para os casos específicos abordados nesta tese, enquanto o pacote de sensores MOLISENS demonstra sua capacidade de mapear com precisão uma ampla gama de ambientes, incluindo ambientes marinhos.

Palavras Chave

Precisão do mapa, Mapeamento marítimo, SLAM, MOLISENS, lidar, LIO-SAM

Table of Contents

List of Figures

List of Tables

Abbreviations

AC/DC	Alternating Current/Direct Current
ADNN	Average Distance to the Nearest Neighbor
AHRS	Attitude and Heading Reference System
APE	Absolute Pose Error
ASV	Autonomous Surface Vessel
CHM	Canopy Height Model
CPU	Central Processing Unit
CSF	Cloth simulation-based construction of pit-free canopy height models
D2D-NDT	Distributions to Distributions Normal Distributions Transform
DARPA	Defence Advanced Research Projects Agency
DC/DC	Direct Current/Direct Current
DoF	Degree of Freedom
DTM	Digital Terrain Model
EKF	Extended Kalman Filter
EKF-SLAM	Extended Kalman Filter SLAM
EM	electromagnetic
FoV	Field of View
GICP	Generalised ICP
GLONASS	GLObalnaya NAvigazionnaya Sputnikovaya Sistema
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
GPU	Graphics Processing Unit
HAT	Hardware Attached on Top
HFoV	Horizontal Field of View
I/O	Input/Output

ICP Iterative Closest Point

IMU Inertial Measurement Unit

INS Inertial Navigation System

IP Ingress Protection

IQR Interquartile Range

KISS-ICP Keep It Small and Simple ICP

L-M Levenberg-Marquardt

LAN Local Area Network

lidar Light Detection and Ranging

LO Local Oscillator

LOAM Lidar Odometry and Mapping

LRF Laser RangeFinder

LTE Long Term Evolution

LTS Long Term Support

M3C2 Multiscale Model to Model Cloud Comparison

MCU Micro Controller Unit

MIMO Multiple Input, Multiple Output

MOLISENS MObile LIdar SENsor System

NDT Normal Distributions Transform

NMEA National Marine Electronics Association

NTRIP Networked Transport of RTCM via Internet Protocol

OLED Organic Light-Emitting Diode

OS Operating System

pub/sub publisher/subscriber

RAM Random Access Memory

RANSAC RANdom SAmple Consensus

RJ45 Registered Jack 45

RMS Root Mean Square

ROS Robot Operating System

RPE Relative Pose Error

RTC Real Time Clock

RTCM	Radio Technical Commission for Maritime
RTE	Relative Translation Error
RTK	Real Time Kinematics
SAIL	Stanford Artificial Intelligence Laboratory
SAL	Synthetic-Aperture Lidar
SLAM	Simultaneous Localisation And Mapping
SLAMMOT	Simultaneous Localization, Mapping, and Moving Object Tracking
SMA	Sub-Miniature A
SSD	Solid State Drive
surfel	surface element
SWIR	Short Wave InfraRed
TLS	Terrestrial Laser Scanner
ToF	Time of Flight
UAV	Unmanned Aerial Vehicle
USB	Universal Serial Bus
ViF	Virtuelles Fahrzeug
vSLAM	Visual SLAM

1 Introduction

Contents

The increase in popularity of Light Detection and Ranging (lidar) in the last few decades has been closely linked with the increased research and popularity of autonomous vehicles. During the Defence Advanced Research Projects Agency (DARPA) challenges in the 2000s lidar showed its worth and has been on the rise in the autonomous vehicle industry ever since [1]. The main reason for this correlation is that some autonomous vehicles use lidar, usually in conjunction with other sensors, to localise themselves, as well as to detect and avoid obstacles.

Although the main user and initiator of development in lidar technology is the autonomous (self-driving) vehicle industry, the usage of lidar has also increased in other industries. Mainly, it has been adopted in the maritime, consumer electronics, and aerospace industries, as well as in archaeology, architecture and geography [2]. This thesis focuses on lidar applications in the marine industry or closely related to the marine industry.

Due to the increase in demand, Autonomous Surface Vessels (ASVs) are being developed for both civilian and military purposes. According to Wang et al. [3] they are used for hydrology surveying, marine resource exploitation, and ocean monitoring. ASVs can also be used for passenger and goods transport in coastal cities such as Amsterdam or Venice [3]. In these ASV applications lidar plays a vital part. Other important marine applications of shipborne lidar include monitoring weather conditions around the ship, detecting obstacles and enemies above and below water, environmental monitoring, and many others [4]. 3D mapping is an application of lidar, that is not strictly a marine application, but is closely related to it. These 3D maps can depict the coastline, partially submerged caves, sea ice fronts, etc. [5].

1.1 Motivation

There are many techniques to produce a 3D map; one is by using lidars. A commonly adopted form of lidar for 3D mapping is known as Terrestrial Laser Scanner (TLS), which is widely used in the field. Although they produce detailed 3D maps, they have some disadvantages. For example, a new Riegl VZ-6000 TLS costs approximately $160\,000$ € [5]. In general, TLS systems are expensive (in the order of $100\,000$ €), heavy ($5 - 15$ kg), not very robust (typically IP64) and, in some use cases, not easy to handle, and they need to be stationary for a longer period of time in order to complete a measurement [5].

On the other hand, automotive lidar technology has been rapidly advancing in the last decade, and the current automotive lidars have become cheaper, lighter, more accurate, and more robust than before. This has prompted the company *Virtual Vehicle Research GmbH*, in cooperation with *University of Graz*, Austria, to develop MObile LIdar SENsor System (MOLISENS) [5]. Their goal was to develop a system that can complement and possibly replace TLSs in some specific use cases that require robust, lightweight, mobile, and modular sensor systems.

As mentioned in [5], MOLISENS is a standalone modular system that builds on the latest advances in high-resolution environment perception for the automotive industry. At the moment MOLISENS only contains lidar, but, since it is modular, it can be upgraded to also include both cameras and radars. Furthermore, MOLISENS is equipped with a Global Navigation Satellite System (GNSS) and an Inertial Measurement Unit (IMU) for georeferenced orientation and positioning. The system also works in a standalone version, ie., there is no need for it to be set up on a vehicle. This feature allows it to be used for measurements even at remote locations, and still provide very high spatial and temporal resolution at a relatively low cost per system [5].

According to [5], small industrial automotive lidars produce high-resolution point clouds with high acquisition frequencies ($10 - 20$ Hz). The acquisition frequency is this high because these lidars were designed to operate in highly dynamic environments, such as highways. The price of these systems is quite low, compared to the price of TLS, and stands at around $5000 - 10000$ €. The robustness is also better than that of TLSs (typically IP68) [5]. All of these features made automotive lidar sensors appealing to be included in MOLISENS. More details about MOLISENS are provided in Section 5.1.

1.2 Objectives

The main focus of this thesis is to determine and try to increase the accuracy of 3D maps produced by the MOLISENS setup, which is currently using the LIO-SAM Simultaneous Localisation And Mapping (SLAM) algorithm [6], and to compare it to the ground truth which is produced by the Riegl VZ-6000. The reason for doing this is that, although MOLISENS has been used for multiple measurements so far (coastal mapping in Rijeka, Croatia, Lurgrotte cave mapping near Graz, Austria [5] and glacier cave mapping in Longyearbreen, Norway [5]), this kind of analysis for SLAM generated 3D maps has not yet been performed. After determining the accuracy of the current setup, the goal is to analyse the accuracy results and then try to improve them, either by tuning the parameters of the already mentioned LIO-SAM SLAM algorithm or by using a completely different algorithm. Multiple SLAM algorithms are implemented and tested, see Section 6.4, and their results are compared, see Section 7.1.

1.3 Contributions

Several lidar based SLAM algorithms are presented in this work. Some of them are based on completely different operating concepts and require different auxiliary sensors to operate. One is based purely on Iterative Closest Point (ICP), two of them are based on Lidar Odometry and Mapping (LOAM), while the last one uses a graph-based SLAM approach to create a map. Another thing to consider is that some of the chosen algorithms are developed for earlier versions of Robot Operating System (ROS) and haven't been maintained since, so some modifications might be needed to make them work with the latest version of ROS, which is ROS Noetic.

To accomplish the objectives of the thesis multiple things need to be done:

1. Different mapping scenarios need to be created to test the algorithms in different conditions. This is reflected in the usage of 3 different datasets, explained in Section 5.4,
2. Modify the source code and dependencies of some SLAM algorithms to make them compatible with ROS Noetic,
3. Establish a way of measuring the accuracy of a 3D map by comparing it to the ”ground truth” map,
4. A metric to compare different SLAM algorithms needs to be established,
5. Methods for optimising the accuracy of the finished 3D map produced by the best-performing algorithm need to be devised and tested.

1.4 Outline

This thesis is organised as follows:

- **Chapter 1** - presents the main motivation and objectives of this thesis.
- **Chapter 2** - includes a basic introduction into lidars, since they are the main tool used to generate point clouds. It also introduces the software used in this thesis.
- **Chapter 3** - introduces the concept of SLAM, as well as the main issues facing SLAM and then divides it into two groups, Visual SLAM (vSLAM) and lidar SLAM. Later, an insight into lidar based SLAM method is provided.
- **Chapter 4** - provides a concise description of the problems that are being addressed in this thesis.
- **Chapter 5** - presents all the equipment used to gather and process the data, as well as how to process the raw data. It also introduces each dataset that is used in this thesis, as well as their specifics.
- **Chapter 6** - presents ways to measure the accuracy of a SLAM map, as well as a method to remove vegetation from an already completed map. Furthermore, it introduces a set of criteria on which all the used SLAM algorithms are evaluated. Then, a description of each used algorithm is provided alongside the ways of optimising the best-performing algorithm.
- **Chapter 7** - presents the results of comparing the performance of different SLAM algorithms to each other, as well as to the ”ground truth” map. Also presents the results of optimisation of the best performing SLAM algorithm.
- **Chapter 8** - presents the discussion of the results presented in the previous chapter, as well as the conclusions found from the said discussion. Ultimately, the possibilities for future work, and the overall impact of the findings in this thesis, are presented.

2
Background

Contents

This chapter aims to introduce concepts that are not necessarily the topic of this thesis, but are essential for a better understanding of the topics that are. Thus, the concept of lidar is introduced, alongside two software packages, ROS and Cloud Compare.

2.1 Lidar

The purpose of this section is to provide a brief introduction to lidar technology, history, basic operating principles, and types. Lidar is a remote sensing technology that uses laser impulses to measure distances. As McManamon [2] explains, lidar uses electromagnetic (EM) waves in the visible and infrared spectrum for its operation (overview of the EM spectrum given in Figure 2.1). Since the used wavelength is quite short (750 nm to $1.5\,\mu$m), the achieved resolution is good, but the drawback is the inability to penetrate through clouds or fog. Lidar is an active sensor, which means that it sends a laser impulse and receives the reflected signal back. The advantage of this approach is that it can be used at any time of day as it has its own source of light. The development of lidar, as we know it today, started in the early 1960s, not long after lasers gained wider adoption. First lidars were using a visible laser, later they started using lasers in near-infrared (Nd:YAG laser) and thermal infrared (CO_2 laser) spectra. In recent years, developments have been made in the eye-safe Short Wave InfraRed (SWIR) spectrum ($\sim 1.5\,\mu$m and beyond) [2].

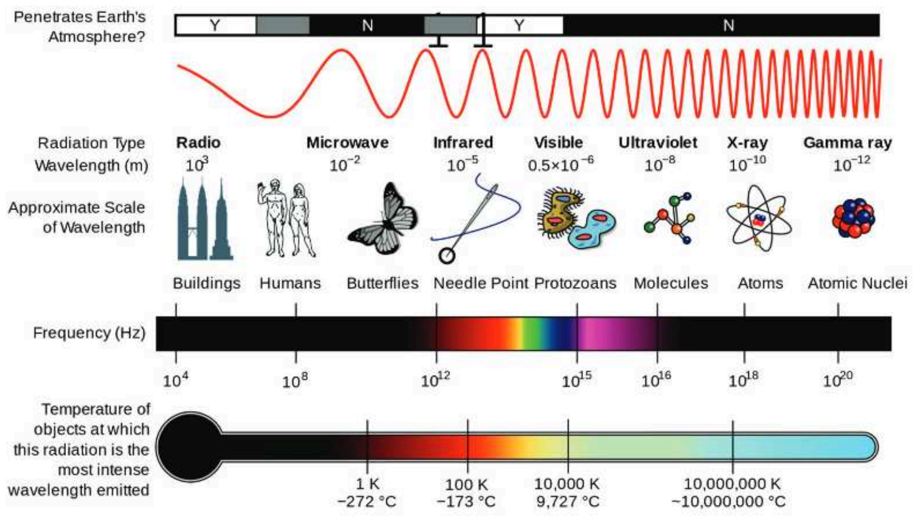


Figure 2.1: EM spectrum and the size of common objects compared to the wavelength [2]

2.1.1 History of Lidar

Since the prerequisite for the existence of lidar was the invention of the laser, it is hard to differentiate between the development of laser technologies and applications, and lidar development itself. Generally speaking, laser technologies, thus lidar as well, were primarily developed for military usage. Later, these technologies were adapted and adopted for civilian use.

Laser RangeFinders (LRFs), first developed in the years before WW2, were the first primitive versions of lidar. A single detector was used to determine the distance (range) to the selected object (target), based on the time it takes a laser pulse to go from the source to the target and back. This paved the way to lidar and laser usage in military technologies such as range finding, weapon guidance, and laser target designation. Further research and development led to the appearance of laser imaging systems that are based on 2D gated viewing and, lately, 3D imaging, which is still in the development process [2].

Figure 2.2: Military usage of lidar: (a) KTD 1 LRF - T-55 tank (1974) [7]; (b) Thales Talios Targeting Pod - Dassault Rafale (2021) [8]

After certain developments were made in military usage of lidars and the technology became declassified, civilian and dual-usage (both military and civilian) applications and technologies were developed. McManamon [2] lists some of these technologies. They include environmental lidar, used for remote sensing of the atmosphere and the ocean, and 3D Mapping using lidars, which is currently capable of producing 3D maps of large areas in many countries. There are many other applications of lidar, such as obstacle avoidance, altitude measurement, or lidar imaging. Decreasing size and price of lasers, coupled with the increase in efficiency, have opened new application fields, such as usage in unmanned vehicles. McManamon [2] believes that the first widespread commercial application of lidars will be in self-driving vehicles, followed by Unmanned Aerial Vehicles (UAVs). In addition to the aforementioned use cases, lidar is also used in medicine, primarily in ophthalmology [2].

The last 50 years saw huge technological improvements in lidar technology and lasers in general, which enabled us to make highly detailed maps of our environment as well as to measure distance with great accuracy [2]. There is little doubt that new technologies, such as autonomous vehicles and vessels, will further drive the development of lidar technology, as well as the development of SLAM technologies, which will be introduced in Chapter 3.

2.1.2 Operating Principle

The basic operating principle and components of lidar, as stated by McManamon [2], are shown in Figure 2.3.

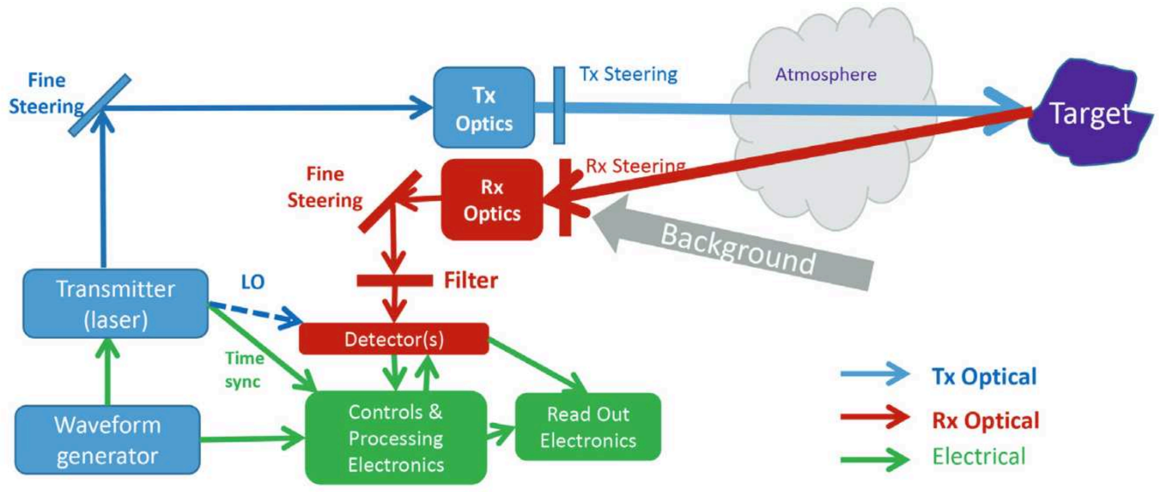


Figure 2.3: Lidar operating principle [9]

Firstly, a waveform generator generates a laser waveform that is needed to obtain range or velocity measurements. A laser provides the necessary light that illuminates the target. Theoretically, lidar can operate with any internal light source, but since it was invented, the laser is an essential part of the lidar setup. As far as the type of laser is concerned, there are many possibilities, such as a single laser, a seeded laser, or a master oscillator with one or more following laser amplifiers. It could also be an array of lasers. After the laser light has been generated, it passes through an optical transmission aperture and gets emitted. The same aperture can be used to receive the light afterwards, but this is not always the case. If the same aperture is used for both receiving and transmitting light, it is a monostatic lidar, otherwise, if the transmit and receive apertures are different, then it is a bistatic lidar [2]. In Figure 2.3 a bistatic lidar is depicted.

Laser light must travel through a medium, usually the atmosphere, to reach the target. Other possible mediums are vacuum, water or even human tissue. When operating in the water, lidar only works well at short distances due to the high absorption of EM waves. Some types of light, like blue and green light, perform better in water due to their wavelengths being optimal for transmission through water. After reaching the target, lidar light bounces from it, and travels back, through the same medium, to the receiving aperture of the lidar, where it is captured [2].

One issue that appears when detecting lidar light is that its carrier frequencies are very high (in the range of 200 - 600 THz), so it is not possible to measure the phase directly. This is solved by using a Local Oscillator (LO) that interferes with the return light signal on the detector (LO beating against the return signal). With this approach, it is possible to measure the beat frequency, and thus also the frequency and intensity of the returning light (coherent lidar). With coherent lidar we can capture the phase of the returning light, only if LO is perfectly stable, because, in that case, the phase change in the beat frequency is the same as the phase of the carrier frequency (returned light). To measure the distance, a concept of Time of Flight (ToF) is used. It refers to the time that passes from the firing of the laser beam to its return to the lidar after bouncing off the target. Once we know ToF and the speed of light in different mediums, calculating the distance is straightforward. Finally, the signal generated by the detector is digitised and processed to generate an image, or other information, such as velocity, or frequency of vibrations, based on the Doppler shift of the return [2].

Doppler shift, or Doppler effect, as stated by Šalaka et al. [10], describes the change in wave's frequency in relation to the observer when it is moving relative to the source of the wave. This phenomenon is also present whenever the target, the lidar sensor, or both move during the measurement process. The following equation introduces the Doppler shift, describing the relation between observed frequency f and emitted frequency f_0,

$$f = \left(\frac{c \pm v_\mathrm{r}}{c \pm v_\mathrm{s}}\right) f_0$$

where:

- c is the wave (light) propagation speed in a given medium,
- v_r is the observer (receiver) speed relative to the medium, and
- v_s is the source speed relative to the medium.

Angular information is detected by one or more separate detectors that are moving in angle, or by an array of detectors that are sampling the Field of View (FoV) [2].

2.1.3 Types of Lidar

As briefly mentioned in Section 2.1.1, there are many different types of lidar that exist today. McManamon [2] differentiates three main groups of lidars. The first one of them, a 1D lidar or Range-only lidar, is capable of measuring only one dimension, in this case, the range. 2D lidar, like all other types of lidars, provides its own illumination and it can be range gated to a certain range region of interest. This means that it is able to reduce the effect of noise and distractions from other, non-interesting, ranges. This is especially useful, for example, in foggy conditions, when backscatter from closer ranges could obscure the returning signal from the range of interest. The most common type of lidar today is the 3D lidar, which measures angle/angle/range. Almost all modern lidars are beam-steering sensors. Beam-steering refers to lidar systems' capability of steering the laser beam to different places in the environment. This steering can be achieved mechanically and electronically. All of the areas towards which a lidar system is able to emit its lase signal is called the Field of View (FoV). Furthermore, 2D and 3D lidars can be divided into conventional scanning lidars and flash lidars. The former works by sending the laser pulse to a single point in space at any given moment of time, whereas the latter illuminates the entire FoV with laser light in a single moment of time [2].

McManamon [2] also notes that velocity can be measured using a coherent lidar, which is capable of detecting the Doppler shift. This is possible because it measures both the phase and amplitude of the return signal. If we want to measure angular information, we need to use an aperture that is synthesised in a three-step process. Firstly, the pupil plane field is sampled in many locations, then, secondly, a larger pupil plane field sample is constructed, and then, finally, the Fourier transform is applied to get an image. One of the examples of lidar using this synthesised aperture is a Synthetic-Aperture Lidar (SAL). Other types of this lidar may use multiple physical apertures for either only receiving or for both receiving and transmitting signals. These lidars, that use multiple apertures, are referred to as Multiple Input, Multiple Output (MIMO) lidars [2].

2.1.4 Basic Lidar Concepts

One of the first concepts to be introduced is the range resolution, ΔR. It can be calculated in accordance with the following equation

$$\Delta R = \frac{c}{2B},$$

where B is the bandwidth of the system, which can be limited by the transmitter or receiver bandwidth (limited by the receiver or the electronics) [2]. It represents the minimal angular or linear distance between the two points that can be detected by the lidar system. This means that as the resolution increases, the point clouds become denser [1].

Precision represents the repeatability of a measurement. Lower precision sensors produce point clouds of noisy data, whereas precise sensors produce well-defined points [1].

Accuracy is best defined as the closeness of a measurement to the actual value. Therefore, an accurate point cloud will be as close as possible to the position of its environment [1].

The Field of View (FoV) is the angle at which a lidar sensor emits its signals, and for a full 3D representation of the system's surroundings, lidar sensors must provide both a sufficient vertical FoV and a full $360°$ Horizontal Field of View (HFoV) [1].

Roriz et al. [1] also consider the pulse rate, which can be defined as the rate at which the sensor measures pulses, as a metric that characterises particularly beam-steering sensors. High pulse rates usually produce denser point cloud data. Newer systems can also collect multiple returns from the same laser pulse, allowing for instance the detection of elevated surfaces. The scan rate corresponds to the time it takes to scan the entire FoV and is directly related to the imaging technique used by the sensor. In flash lidar systems, this value is often expressed as a frame rate because their operation is very similar to a standard digital camera: they capture an image with every surface they pass over [1].

2.2 Robot Operating System (ROS)

ROS is a name for a free and open-source software framework used to build robotic systems. Since its development in 2007 at Stanford Artificial Intelligence Laboratory (SAIL), it has become a widely used tool in the robotics sector. Thanks to its networked architecture and modular design, ROS makes it simple to construct complex robotic systems from smaller, reusable components. Additionally, it offers powerful tools for testing and debugging and supports a variety of hardware [11].

The idea of nodes lies at the core of ROS. Independent programs known as nodes carry out particular activities and send messages to other nodes to exchange information. A publisher/subscriber (pub/sub) communication mechanism is available in ROS, in which nodes post messages on particular topics and subscribe to messages on different topics. As a result, scalable communication between nodes is made possible, enabling the development of sophisticated robotic systems [11].

In a ROS system, the roscore serves as the primary hub for communication, controlling the many nodes and offering a standard framework for communication. The roscore offers services including time synchronisation, message passing between nodes, and node name and registration. Rostopic, a potent command-line tool offered by ROS, allows users to inspect and modify topics and messages [11].

Two key notions in ROS are publishers and subscribers. Nodes that send messages on a certain subject are called publishers, while nodes that receive messages on that subject are called subscribers. Since publishers and subscribers do not need to be aware of each other's internal state, the pub/sub model enables decoupling between nodes [11].

Services are another key idea in ROS. Services give nodes a means of sending requests to other nodes, just like function calls do in conventional programming. The node delivering the service receives the request, completes the necessary work, and then gives the requesting node a response. Services are helpful for activities like localisation and mapping because they enable more complicated interactions between nodes [11].

In addition, ROS offers assistance with simulation and testing, both of which are necessary for creating sophisticated robotic systems. Without the usage of actual robots, code may be tested and iterated upon quickly thanks to simulation environments. Rosbag, a potent tool provided by ROS, can be used to capture and replay data from a ROS system. Rosbags can also be considered as a native ROS file format for saving data and they can be used to create datasets for machine learning, as well as for debugging and testing [11].

However, for those who are just starting out in robotics development, the complexity of the ROS system can be difficult. Before beginning to design sophisticated robotic systems, developers may need to invest a lot of time studying ROS due to its steep learning curve. The performance overhead brought on by ROS utilisation is another difficulty. Real-time applications may be concerned about the latency and overhead introduced by the pub/sub communication mechanism utilised in ROS [11].

The *sensor_msgs/PointCloud2* message type in ROS is a versatile data structure used for representing 3D point clouds. It provides a standardised format for capturing and transmitting point cloud data, making it easier to share and process such information across different ROS nodes and systems. The *PointCloud2* message encapsulates a collection of points, where each point is defined by its 3D coordinates (x, y, z) and additional optional fields, such as RGB colour information or intensity values. This message type also supports various data encodings, allowing flexibility in representing different point cloud representations, such as organised or unorganised point clouds. With its rich feature set and wide adoption in robotics and perception applications, the *PointCloud2* message type serves as a fundamental building block for many ROS-based point cloud processing tasks, including mapping, localisation, object recognition, and sensor fusion [11].

To sum up, ROS is an effective and popular framework for creating robot software. Complex robotic systems can be built on top of it thanks to its distributed and modular architecture, support for a variety of hardware, and robust testing and debugging tools. The system's complexity and the performance overhead connected to its use, however, present difficulties for developers. In general, ROS has the potential to speed up innovation in the field of robotics and to support the creation of fresh, cutting-edge robotic systems [11].

2.3 CloudCompare

Cloud Compare is an open-source software, used for point cloud and mesh manipulation, visualisation, and processing. Due to its stability and possibility to open and export almost any known type of point cloud format, it is widely used by professionals [12]. Cloud Compare includes many advanced algorithms for point cloud processing, such as registration, resampling, statistics computation, sensor management, colour/normal/scalar fields handling and many others [13]. Also, the software architecture is Plug-in based, which allows for easy integration of new features required by the user [12]. It is available on Windows, Mac OS and Linux [13].

3
State-of-the-Art in SLAM

Contents

Since this thesis is heavily focused on SLAM, this chapter gives an introduction to SLAM, some of the main SLAM types and challenges facing it, as well as its development history.

3.1 Simultaneous Localisation And Mapping

Simultaneous Localisation And Mapping (SLAM) represents a method, commonly used by autonomous vehicles and robots, to build a map of their environment and to, also, localise themselves in it. The concept of SLAM algorithms is fairly new, appearing only in the late 1980s [14] and gaining popularity after being used in DARPA challenges in the 2000s [1]. One of the main issues of SLAM has always been its need for a lot of computing power and fast results (in dynamic environments), as well as the price of sensors needed to acquire data for it. Only after vast improvements in computer processing speed and capacity, as well as a decrease in the price of sensors, such as cameras and lidars, SLAM has become widely adopted, making its way even into our homes, as a fundamental part of autonomous vacuum cleaners [14].

Actually, using a robot vacuum cleaner as an example is a great way to explain the difference between mapping and localisation. Firstly, a robot without any SLAM algorithm would move randomly throughout the room and use the battery excessively, without any guarantee of cleaning the entire room. On the other hand, a robot with SLAM capabilities can use the inputs from different sensors to localise itself and limit the movement needed. Furthermore, it can use the data from the same sensor to make a map of its surroundings, and thus prevent cleaning the same place twice. This action is called mapping. SLAM algorithms have many other use cases in addition to robot vacuum cleaners. They are used for autonomous vehicle navigation, warehouse robots, package delivery drones, and many other applications.

The general workflow of a SLAM algorithm is shown in Figure 3.1. It can be divided into a sensor-dependent part and a sensor-independent part. In the sensor-dependent part, also referred to as front-end processing, motion estimation and obstacle location estimation are performed, while in the sensor-independent part, also referred to as back-end processing, pose graphs are registered and optimised. Sensor data is fed to the front-end, where it is processed and fed to the back-end. The output of the back-end is a Pose graph, as well as a 2D or a 3D point cloud map [15].

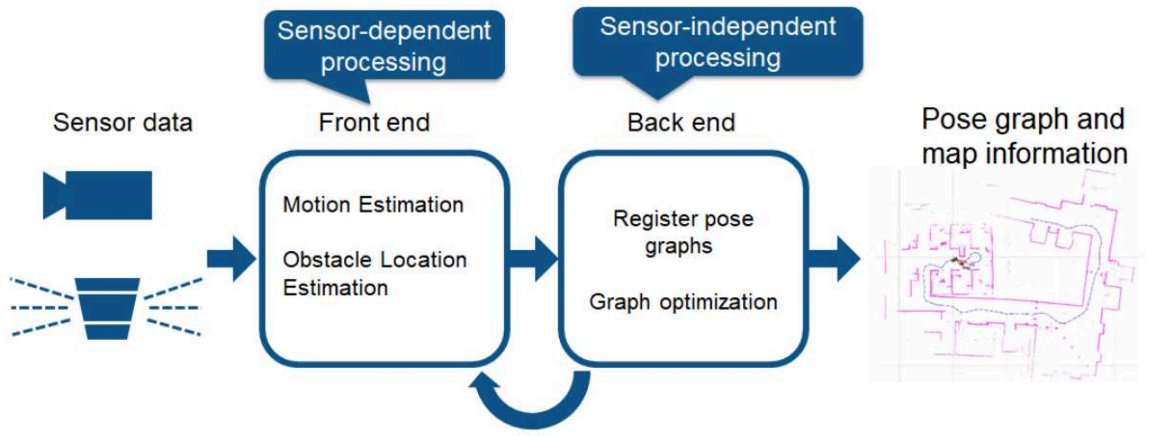

Sensor-dependent processing
Sensor-independent processing
Sensor data
Front end
Back end
Pose graph and map information
Motion Estimation
Obstacle Location Estimation
Register pose graphs
Graph optimization

Figure 3.1: General SLAM Workflow [15]

3.1.1 Types of SLAM

There are two main front-end types of SLAM: vSLAM and lidar SLAM [14]. Given that this thesis is focused on lidar SLAM, a more detailed overview of different technologies and algorithms used to achieve it is given in Section 3.2.

VSLAM uses images acquired by cameras to build a map and perform localisation, as explained by Debeunne and Vivet [14]. These cameras can be either simple cameras, compound eye cameras (both mono and stereo), or more complex RGB-D cameras. The price of implementing vSLAM is fairly low because inexpensive cameras can be used. Also, because cameras provide an image with a lot of information, this method is quite widely adopted for landmark detection. In the event that only a mono camera is used, determining depth becomes more difficult because there is not enough information in the image itself; camera information needs to be fused with information from other sensors, for example, an IMU, which in itself is a totally different challenge [14].

Debeunne and Vivet [14] also explain that the input for lidar SLAM is a 2D or a 3D lidar point cloud. Compared to the camera, lidar point clouds contain more precise measurements [14], but may lack sufficient features for detection, making it harder to align point clouds and achieve good SLAM results. This happens in cases where the environment is not rich with different distinct features. Generally, maps are made by estimating movement using sequential matching of point clouds. Furthermore, point cloud matching is computationally challenging, so optimisation is required to make the process faster and smoother. This is the reason why point clouds are sometimes fused with data from other sensors such as wheel odometry, IMU, or GNSS data [14].

3.1.2 Challenges Facing SLAM

Although SLAM is a powerful tool used by autonomous robots, it faces three major issues that prevent its adoption in more general-purpose cases, as summarised in [16]. These issues are: i) *the accumulation of localisation errors*; ii) *failed localisation causing the loss of position on a map*; and iii) *the high computational cost of input data processing*.

Accumulation of localisation errors occurs because SLAM algorithms estimate sequential movement and, in these estimations, some margin of error is present. This error accumulates over time and causes substantial deviation from actual values. This deviation then causes the map to distort and eventually fail, making subsequent searches difficult. The way to mitigate this effect is to remember some characteristic from a previous pose (image or point cloud) as a landmark and search for it in a subsequent pose, aligning them as best as possible and thus minimising the localisation error [16].

Since image and point cloud mapping does not consider any characteristics of the robot's movement, it can lead to discontinuous position estimates (failed localisation) which leads to the loss of the robot's position in a map. This problem can be prevented either by using a recovery algorithm or by fusing the motion model with some other auxiliary sensors. Most common recovery algorithms try to locate a currently observable landmark in one of the previously visited places. When searching for this landmark feature, the extraction process needs to be applied in a way that works as fast as possible. On the other hand, if we want to fuse sensor data with a motion model, we need to use a Kalman filter or some other type of Bayes filter. Commonly used sensors include IMU, Attitude and Heading Reference System (AHRS), Inertial Navigation System (INS), and encoders attached to wheels for odometry [16].

The computational cost is always a problem when implementing SLAM algorithms in a vehicle or a robot. The reason is that the hardware in these cases is usually not that powerful since it needs to be compact and have low energy consumption. However, to achieve localisation, it is essential to achieve point cloud matching and image processing at a high frequency. Furthermore, optimisation algorithms, such as loop-closure, require a lot of computation power. The solution to these problems is to run some of the processes, such as feature extraction, in parallel threads [16].

3.2 Overview of Lidar SLAM Algorithms

This section introduces SLAM technologies that are usually paired with a lidar as a sensor, as well as some challenges they face. Visual SLAM techniques will not be considered, as they are outside of the scope of this thesis.

Besides the general SLAM challenges introduced in Section 3.1.2, lidar SLAM faces additional challenges, such as the large amount of computational power required to process large lidar point clouds, sparse point clouds (sparsity increases with the range to target), and motion distortion [17]. Some of the problems mentioned earlier in this section are addressed with solutions in Section 3.1.2. However, in this section, algorithms that propose different approaches to tackle these problems, are introduced.

Yang et al. [17] explain that lidar SLAM is used for the construction of outdoor and indoor maps. When it comes to outdoor environments, the application generally is in the field of autonomous vehicles, drones, etc. Usually, they benefit from high precision localisation and mapping using GNSS, but in cases when the GNSS signal becomes weak or is lost completely, they rely on lidar based SLAM to localise themselves and build a map of the environment. In these cases, the lidar data can be fused with the data from other sensors to achieve higher precision or faster SLAM. One of these sensors is the IMU, which provides the sensor carrier positions, which are fused with the point clouds to greatly improve the accuracy and speed of obtaining the 3D map. This improvement is present because, in the case where no IMU is used, the initial value of the pose transformation is unknown in subsequent poses, resulting in more required iterations and thus a longer time to obtain a map [17].

Some simple mobile robotics applications can be satisfied using 2D SLAM, but autonomous vehicles require 3D SLAM, i.e., high safety requirements, complex road conditions and undulations on the road topography, etc. dictate that autonomous vehicles need to be positioned in 3D maps. 3D lidar SLAM technology is still in development and is still trying to solve the issues mentioned above. Currently, the main efforts are focused on improving robustness, precision, and real-time capabilities [17].

Since its invention in the 1980s, the development of SLAM technology has been closely tied with the development of sensors it uses to gather data, as well as the development of probabilistic and nonlinear optimisation methods it relies on. Lidar SLAM algorithms can be divided into two groups: i) *probabilistic based SLAM scheme*; and ii) *SLAM scheme based on nonlinear least squares* [17]. An overview of the history and algorithms of both schemes is given in the following sections. Only standalone lidar-based SLAM algorithms are listed, i.e., if an algorithm requires any other input sensor (mono camera, stereo camera, depth camera, etc.) in addition to lidar, IMU, and GNSS, it is not listed. Most of the overview is based on the work of Yang et al. [17], where the history of SLAM methods is nicely summarised.

3.2.1 Probabilistic-based SLAM Scheme

Bailey and Durrant-Whyte [18] summarised, in 2006, the development of SLAM technologies from its infancy in the 1980s up to 2006. They found that probabilistic methods were used and identified two predominant ones, Extended Kalman Filter SLAM (EKF-SLAM) and Fast SLAM. Both methods are using Lie algebra to express SLAM in 3D space [17].

After 2007, according to [17], representative SLAM schemes based on probabilistic methods have been used. Two methods using 1D lidar have been proposed. Grisetti et al. [19] used an improved Rao-Blackwellized particle filter method that reduced the number of sampled points during robot localisation and grid map construction. Similarly, Kohlbrecher et al. [20] used a motion estimation method for point scanning and local raster map registration, combining it with 3D navigation equipment, in order to complete localisation and raster map construction.

Yang et al. [17] conclude that, in general, probabilistic SLAM methods are computationally inefficient because they maintain a state matrix that grows over time. Also, there is no possibility of eliminating the cumulative error or performing closed-loop detection. However, these limitations do not mean that the development of this type of SLAM algorithms has completely stopped. The Extended Kalman Filter (EKF), still, has its use cases in new algorithms being developed, as recently as 2022, which is the case with the Simultaneous Localization, Mapping, and Moving Object Tracking (SLAMMOT) algorithm, developed by Simas et al. [21].

3.2.2 SLAM Scheme Based on Nonlinear Least Squares

SLAM based on the nonlinear least squares method gained momentum after 2012, as indicated in [17]. That year researchers discovered that using this method to reduce SLAM process error is sparse, the number of needed calculations is acceptable, and the SLAM problem in graph form becomes very intuitive. After this discovery, many lidar SLAM methods based on nonlinear least squares were proposed.

Figure 3.2 shows the classical framework of these methods, and the framework consists of 3 distinct parts. The first part is called the front-end and deals with the estimation of the position and the orientation of the sensor carrier in accordance with the real-time sensor data. It is reliant on the frame-to-frame registration method, and it constructs a graph model of constraints between frames received as an output of the frame-to-frame registration method. Currently, registration is performed between two frames and is accepted if the error is below a certain threshold, which leads to an accumulation of the pose error which grows exponentially and needs to be corrected from time to time. The second part is called the back-end and it deals with optimisation, while the third part is called closed-loop detection [17].

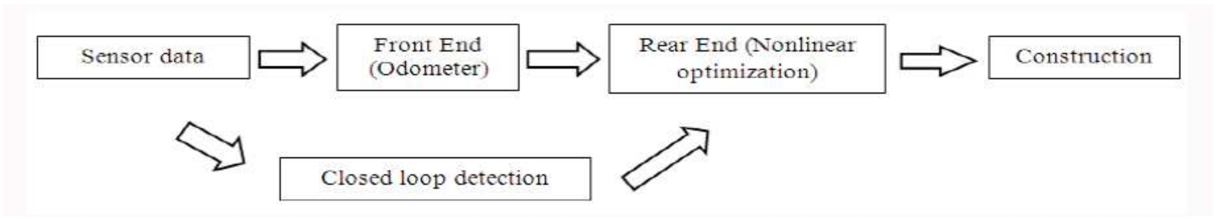


Figure 3.2: Framework of SLAM methods based on nonlinear least squares [17]

Yang et al. mention [17] that Iterative Closest Point (ICP) is the most classic algorithm proposed at this time. It associates data between two point clouds by searching for the nearest point, most often in a k-dimensional tree [22]. ICP, firstly, organises point cloud data, that accelerates the search [23], and then minimises the distance between two point clouds by using nonlinear optimisation techniques to solve the pose estimation problem. Generalised ICP (GICP) [24] is the mainstream variant of ICP methods. It combines the point-to-surface ICP method with the original ICP algorithm, thus making it a planar-to-planar ICP method [17].

After ICP methods, a new method called LOAM is introduced. It is a lidar-based 3D odometry scheme, proposed by Zhang and Singh [25]. It estimates inter-frame motion and the pose transformation between frames using point-to-surface and point-to-line ICP algorithms. LOAM makes the usage of descriptors redundant, making LOAM simpler and more efficient. Descriptors are replaced by edges and plane points that are registered in each frame of a point cloud. The k-dimensional tree is used to organise the point cloud and to complete the nearest neighbour search. If we assume uniform motion for a short period of time, then the interframe motion can be estimated by the ICP algorithm, and the point cloud motion distortion correction is completed by the result [17].

Alongside LOAM a set of methods based on Normal Distributions Transform (NDT) is covered by Yang et al. [17], most of them not using closed-loop detection. The method using a normal distribution based on NDT, along with a 1D lidar to divide a frame of point clouds obtained by lidar into 2D grids, assuming each grid, was proposed by Biber and Strasser [26]. Points inside the grid obey the rules of the normal distribution, also the mean and covariance matrices of point coordinates in each grid are counted. To represent the points in the first grid, a normal distribution is used, while the points in the frame point cloud are obtained by transforming them using the initial pose transformation estimate. Afterwards, the probability density values of each normal distribution are calculated, summed, and used as criteria to determine the registration result. Pose transformation is solved using nonlinear optimisation methods [27]. Using the NDT methods for 3D lidar point clouds was first proposed by Magnusson et al. [28]. They used the same evaluation criteria as 2D NDT. Only in 2012, Stoyanov et al. [29] proposed to represent the entire point cloud as a normal distribution model during the registration process. This meant that a lot of storage space was saved by describing the entire frame with a normal distribution. The registration process called Distributions to Distributions Normal Distributions Transform (D2D-NDT), evaluates whether the two frames, represented with normal distribution, are similar and estimates the pose transformation matrix between them. The drawback of both D2D-NDT and a mapping method called NDT Fusion [30], even though they can be applied to large-scale dynamic environments, is the fact that neither of them uses closed-loop detection. Droeschel et al. [31] tried improving the previous methods by recording the normal distribution in a grid as a surface element (surfel) called a bin. They also divided the grid maps into multiple resolutions, depending on the distance between the selected points and the lidar itself. This made the algorithm useful to track the rapid movement of drones; however, it still lacks a closed-loop detection feature and there is no way to eliminate the cumulative error [17].

Afterwards, Droeschel and Behnke [32], expanded on their previous registration method and combined it with the continuous-time trajectory estimation method. This, new, SLAM algorithm uses a 3D method to interpolate the pose. Furthermore, hierarchical optimisation of the pose map offers improvement to the correction of point cloud motion distortion. It even has a closed-loop detection, based on D2D-NDT performed on randomly selected frames [17].

The next milestone highlighted by Yang et al. [17] happened in 2018. This year saw a new SLAM scheme using 3D lidar data, called SuMa, be proposed by Behley and Stachniss [33]. Similarly to previous algorithms, it also uses surfel based map representation and solves the pose transformation matrix using ICP algorithm. The difference is that it uses a library called OpenGL to render point clouds. Furthermore, its closed-loop detection feature is very robust, so it even allows the virtualisation of maps that have only small overlapping regions between the point cloud and the map itself [17].

Another closed-loop detection method is covered in [17]. It is based on the branch and bound algorithm and was proposed by Hess et al. [34]. It searches for a point cloud and a sub-score in the pose space by introducing a small discrete pose space in the vicinity of the current pose estimate. If this pose value exists, it is used as the initial value. New pose transformations are still estimated with front-end estimation methods and are used as closed-loop detection constraints. This method is quite simple, which makes it useful in real-time applications; however, a drawback, is that when the accumulated error is higher than the distance threshold, closed-loop detection fails [17].

The three previously mentioned algorithms all use closed-loop detection methods based on geometric relationships. However, as stated in [17], in the field of vSLAM, contemporary closed-loop detection methods all use the bag of words model [35]. This model originated in 2D image comparison and it basically treats image features as words and uses those words to match features between different images. The same approach is nowadays used in vSLAM. This prompted Steder et al. [36] to propose an algorithm that detects closed loops in a 3D point cloud by combining the bag of words model with the point cloud features. Later, a closed-loop detection method based on 3D point cloud object segmentation SegMatch was proposed by Dube et al. [37]. It combines the advantages of global and local point cloud features to allow reliable operation in large-scale, unstructured environments with a rate of change up to 1 Hz [17].

Ultimately Yang et al. [17] mention that, in 2018, a new map representation method called SegMap was introduced by Dube et al. [38]. Alongside it, a positioning method that is based on incremental point cloud segmentation in the point cloud itself was proposed [39]. SegMap method extracts point cloud features using descriptors that are created with the help of machine learning algorithms. Afterwards, these feature points are used to make a point cloud map, which is the main input for the segmentation-based incremental positioning method. The operating frequency of this algorithm is 1 Hz, which means it has somewhat solved the slow operating speed issue, that the previous algorithms faced [17].

The year 2020 saw the introduction of a new SLAM algorithm called LIO-SAM, proposed by Shan et al. [6]. LIO-SAM allows the usage of data from other sensors such as IMU and GNSS alongside lidar point clouds to enhance the accuracy of the constructed map. It also supports loop-closure. The algorithm also focuses heavily on real-time performance, which is enhanced by marginalising old lidar scans for pose estimation, rather than matching lidar scans to a global map. This local-scale scan matching gives this algorithm high performance and makes it suitable for real-time usage [6].

Near the end of 2022, Vizzo et al. [40] introduced a new SLAM algorithm that, again, focuses on the basics, which is ICP. Keep It Small and Simple ICP (KISS-ICP), unlike other state-of-the-art SLAM algorithms, removes complexity in the motion estimation process and reduces the types of required input data to only lidar point clouds. This simplification yields an algorithm that is simple to use and can operate under various environmental conditions and using multiple types of lidar sensors. Pose estimation of KISS-ICP relies on point-to-point ICP that is combined with adaptive thresholding for matching correspondences, a simple motion compensation approach, a robust kernel and a strategy for point cloud sampling. All of this combined gives an algorithm with only a few parameters, that in most cases do not even need to be tuned for different lidar sensors and applications [40].

4

Problem Statement

This thesis aims to provide an alternative to TLSs when it comes to producing comprehensive 3D lidar point cloud maps of larger areas. An alternative is needed, because TLSs need to be fully stationary for a longer period of time, up to a few dozen minutes, in order to complete the measurement process, which is virtually impossible in marine environments. Although this thesis is focused on marine applications, it is not restricted to them, thus other use cases, besides the marine one, will be considered. The aim is to use Automotive lidar to capture 3D point clouds of the environment as it moves through it, and a SLAM algorithm to produce a 3D map of the said environment.

Regarding the point cloud capturing aspect, it is outside of the scope of this thesis because the setup, named MOLISENS, was already developed and tested, as explained in [5]. A detailed explanation of the MOLISENS setup is provided in Section 5.1.

Therefore, the real focus of this thesis is on the creation of the 3D lidar point cloud map and its evaluation. This is a multi-step process that involves solving several issues.

The first problem to solve involves gathering the datasets that are suitable to test different SLAM algorithms in multiple circumstances. Such datasets include marine datasets and land-based datasets. More about the datasets used is mentioned in Chapter 5.

After the datasets have been created, suitable SLAM algorithms need to be chosen. The choice depends on multiple factors, but the main criterion is that it is capable of outputting a 3D point cloud map. Also in the selection, algorithms using multiple different SLAM methods are tested with the aim of finding the one that suits the mapping needs the best. Another criterion for the choice of the algorithms is the fact that they need to be compatible with the post-processing setup, i.e. the laptop used for the SLAM 3D map generation. This setup is introduced in Section 5.1.5. The SLAM algorithms, that were chosen, are introduced in Section 6.4.

Once the maps are created, another problem to solve is to find a way to compare the results. This problem is divided into two different ones. Since some of the datasets have a reference 3D map to compare the SLAM results to, a SLAM map accuracy measurement needs to be chosen. Furthermore, this can not be the only criterion, so some other criteria are also introduced. These other criteria are mainly based on the SLAM algorithms' computational requirements, as well as the goodness of the created 3D map. These criteria are introduced in Sections 6.1 and 6.3.

In the end, an attempt to improve the accuracy and/or the computational performance of the ”winning” SLAM algorithm needs to be performed. This includes analysing the core concept of the algorithm and then, suggesting and trying methods of improving the performance. This is introduced in more detail in Section 6.5.

5

Data acquisition and analysis

Contents

This chapter introduces the tools used in the process of gathering data, the process itself, as well as the datasets that have been gathered and are used, in this thesis. In Section 5.1 the architecture of the MOLISENS setup, both hardware and software aspects, is introduced. Also, some features of the TLS Riegl VZ-6000 are mentioned. Section 5.2 describes the process of gathering the data, while Section 5.3 explains what tasks need to be performed in order to prepare the raw data for processing. Finally, Section 5.4 briefly explains the specifics of each dataset used in this thesis.

5.1 Experimental Setup

Creators of MOLISENS, Goelles et.al [5], explain that it is a modular framework that allows the integration of small industrial sensors, such as lidars, radars and cameras, that support ROS framework, into a single sensor unit. The MOLISENS hardware setup follows Ingress Protection (IP) standards for small industrial sensors, for example, Ouster OS2-64 lidar has IP class of 69K with the Input/Output (I/O) cable attached [41]. This makes the MOLISENS setup suitable for fieldwork in rougher environments. Figure 5.1 shows the hardware components and basic operation of MOLISENS. Sensors and the data logger are connected by an in-house developed wire harness, which eliminates the need to have multiple wired connections. MOLISENS can be powered by either an Alternating Current/Direct Current (AC/DC) adaptor or by batteries, depending on the application conditions. The sensor unit scans the environment and transmits the data to the data logger where it is saved. These data can later be downloaded using a Local Area Network (LAN) interface for further post-processing on a computer. Post-processing, usually, can be done in multiple ways. The first way includes running a SLAM algorithm to create a 3D map and then processing the created 3D map with Cloud Compare software [5]. Alternatively, the raw sequence of point clouds can be replayed in the order it was recorded using ROS, or the raw data can be analysed using *pointcloudset*. It is a Python package that extracts and packages useful data from the raw lidar data [42].

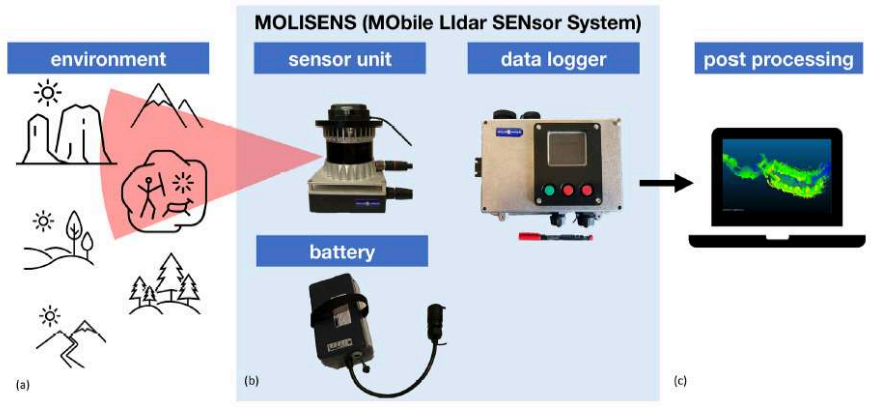


Figure 5.1: MOLISENS hardware components and basic operating principle [5]

Goelles et.al [5] later clarify that raw lidar data contains, besides the x, y, and z coordinates of each point in a 3D point cloud, also, range, intensity, reflectivity, ambient near-infrared, and timestamp for each point. This leads to data set sizes in the order of gigabytes, in the form of 3D point clouds recorded over time. This makes *pointcloudset* package needed to organise the data. It organises the raw data into a *pointcloudset* data set. This data set contains multiple *PointCloud* objects, timestamps and metadata. Furthermore, *pointcloudset* has the ability to analyse the data and compute statistics, and even process each individual point cloud [5].

5.1.1 MOLISENS Hardware

This section introduces all the individual hardware components of the MOLISENS setup. The sensor unit, with all the currently available sensors, data logger, and power supply systems are explained.

Sensor unit

The sensor unit of the MOLISENS setup contains 3 different sensors. It incorporates a small industrial lidar sensor Ouster OS1-64 or OS2-64, ANN-MB series ublox active multi-band GNSS antenna, and a 9-axis Xsens MTi 630 IMU. The sensor unit can be mounted on any standard camera handle or tripod, because it is equipped with a standard $\frac{1}{4}$in camera thread [5].

According to its datasheet [43], Xsens MTi 630 is an AHRS, an advanced version of an IMU, that provides referenced inertial data as well as roll, pitch, and yaw data. MTi 630 contains a 3-axis gyroscope, a 3-axis magnetometer, and a 3-axis accelerometer, a barometer, a high accuracy crystal oscillator, and a low power Micro Controller Unit (MCU). MTi 630 fuses the data of all the sensors at a high frequency and produces a real-time stream of devices' 3D orientation data at rates up to 400 Hz. The device supports three communication protocols, RS232, CAN, and UART. If Universal Serial Bus (USB) is needed UART/RS323 to USB converters are available. Figure 5.2 shows the external look of the sensor [43].

Figure 5.2: Xsens MTi 630 9-axis IMU [43]

A GNSS antenna is used to amplify the received signal that is transmitted by GNSS satellites. This signal contains the required data to perform accurate localisation of the receiving sensor using triangulation. Datasheet [44] of the ANN-MB series ublox antenna, chosen for MOLISENS, works with multiple GNSS services, such as Global Positioning System (GPS), GLObalnaya NAvigazionnaya Sputnikovaya Sistema (GLONASS), Galileo, and BeiDou. The antenna provides the amplification gain of 28 ± 3.0 dB. This antenna has an IP class of 67, it is waterproof, and can operate in temperatures from -40 to 85 °C. Furthermore, it is lightweight, weighing at only 173 g, and offers versatile mounting and connecting options. Figure 5.3 depicts this ublox antenna [44]. This antenna is coupled with the Long Term Evolution (LTE) stick that is used to retrieve Real Time Kinematics (RTK) data [5]. The connection between them is explained in Section 5.1.2

Figure 5.3: ANN-MB series ublox GNSS antenna [44]

MOLISENS uses both Ouster OS1-64 and OS2-64 lidars. However, since OS1-64 has been explained in detail by Goelles et al. [5], this paragraph focuses on the Ouster OS2-64. It is a mechanical rotating lidar, which was originally intended for usage in the automotive industry and costs around $21\,000$€. According to the datasheet [41], even with the I/O cable, it is dust and waterproof, which is guaranteed by its IP level of IP69K. OS2-64 is a long-range lidar with a maximum operating distance of up to 210 m and a minimum detection distance of 1 m. The wavelength of the laser is 865 nm and its eye safety class is 1, according to IEC/EN 60825-1:2014, which makes it safe to operate without any concerns regarding the eye safety of the operator or anyone else within the operating range. The range resolution of this sensor is 0.3 cm, which means it can detect individual objects when the distance between them, in the scanning direction, is equal to or greater than 0.3 cm. The range accuracy is ± 10 cm for retroreflectors and ± 3 cm for Lambertian targets. The precision of OS2-64 depends on the range and varies between ± 2.5 cm and ± 8 cm. The horizontal resolution is configurable and can be either 512, 1024 or 2048 scanning points across the entire horizontal FoV of $360\,°$. The vertical resolution depends on the variant of the sensor and can be either 32, 64 or 128 scanning points across the entire $22.5\,°$ ($-11.25\,°$ to $-11.25\,°$) of the vertical FoV. Both vertical and horizontal angular sampling accuracy is $\pm 0.01\,°$, while the rotation rate of the entire sensor can be configured to be either 10 Hz or 20 Hz. Ouster OS2-64 is shown in Figure 5.4 [41].

Figure 5.4: Ouster OS2-64 lidar [41]

Data logger

As stated by Goelles et.al [5], the data logger consists of a Raspberry Pi 4 as a processing unit, three Raspberry Pi Hardware Attached on Tops (HATs), an LTE stick for retrieving RTK data, an Ouster interface board for transferring data and powering the Ouster OS2-64 lidar, and two Direct Current/Direct Current (DC/DC) converters. One is a 24 V/24 V converter and the other one is a 24 V/5 V used for internal power supply. The first HAT has a Real Time Clock (RTC), the second one has 1 TB Solid State Drive (SSD) for saving data, and the third one is a Raspberry Pi HAT for GNSS. The data logger unit provides a multitude of different interfaces for connecting it with different apparatus. It has a connection for the setup's power supply, a 24-pin connector for the Ouster power supply and data link, a USB connector for the IMU, a Sub-Miniature A (SMA) connector to connect GNSS, a Registered Jack 45 (RJ45) connector for an Ethernet connection, and a generic multipurpose USB jack. Alongside all these connectors, the data logger also has an on/off button, two red coloured buttons for selecting the measuring mode, and a green button to start or stop the measurement. It is also equipped with an Organic Light-Emitting Diode (OLED) display, that shows the file name of the current measurement, as well as the status of the LTE connection and the selected measurement mode. The aluminium casing of the data logger provides sufficient protection and cooling to the aforementioned hardware components [5].

Power supply

MOLISENS setup can be powered by either a $24\,\mathrm{V}$ nominal voltage AC/DC adaptor, or by batteries, as explained in [5]. Usually, batteries are used during the survey, because they provide more mobility and freedom, and later, while transferring the data from MOLISENS to the computer for further processing, the adaptor is used. In the current sensor unit and data logger composition, MOLISENS draws around $1\,\mathrm{A}$ of current while the data, of all three sensors, are recorded. There are two different battery packs that can be used at the moment. The first one is a single Li-ion battery with a capacity of $10.4\,\mathrm{Ah}$, which gives an operating time of $10.4\,\mathrm{h}$. The other is a parallel connection of 2 $LiFePO_4$ batteries with $3.6\,\mathrm{Ah}$ capacity each. It gives the battery pack capacity of $7.2\,\mathrm{Ah}$, or the operating time of $7.2\,\mathrm{h}$. The operating temperature for discharging the battery pack is in the same range for both battery packs (-20 to $60\,^{\circ}\mathrm{C}$) [5].

5.1.2 MOLISENS Software

Figure 5.5 shows the software stack of the MOLISENS data logger. The Operating System (OS), installed on the Raspberry Pi 4, is an Ubuntu Server 20.04 ARM64. Raspbian, the native Raspberry Pi OS, was replaced because ROS integration is easier with Ubuntu.

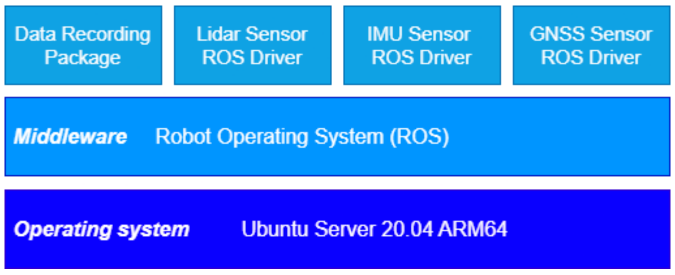


Figure 5.5: Software stack of the data logger (adapted from Goelles et al. [5])

As stated in Section 2.2, as well as in [11], ROS is an open-source middleware, mainly used for robotic applications. The main advantage of ROS is a wide range of third-party, open-source packages, and tools for a variety of different applications. In ROS, a master, named roscore, controls all nodes running in the system. Each node, an entity that performs tasks, can exchange data with other nodes by subscribing or publishing messages using topics. A topic is a communication channel defined by a unique name and the specific type of message being transferred. Writing code in ROS is possible in many different programming languages. At the moment ROS supports C++, C, and Python [11].

The MOLISENS specific packages installed on the Raspberry Pi 4 are: i) data recording package; ii) lidar ROS driver package; iii) IMU ROS driver package; and iv) GNSS ROS driver package. The data recording package and the GNSS ROS driver package are self-developed by the creators of MOLISENS [5].

Lidar sensor package uses the ROS package provided by the manufacturer of the sensor. The software package processes data from the sensor and converts it into a point cloud representation. It also includes tools to visualize the output and verify that the sensor is accurately capturing the environment and check the light intensity of the points. However, due to the limitations in the computational power of the Raspberry Pi, only raw lidar data are recorded. These data are recorded as a custom ROS message [5]. More details about the exact message type the data are saved in, as well as the way to convert it to a more standard message type are provided in sections 5.2 and 5.3.

IMU sensor package, also uses the ROS package provided by the manufacturer of the sensor. Some modifications are made to the configuration and topic selection to make it work with the MOLISENS setup. All of the unneeded topics are omitted, in order to increase the system's performance [5].

The data recording package is a ROS package, coded in Python, that provides an easy interface to start and stop the data recording, as well as a flexible configuration for the specific requirements of the use case. It allows the user to select the sampling rate of either 10 Hz, or 20 Hz for the Ouster OS1-64, or the Ouster OS2-64 lidar, and the number of points in the horizontal direction of either 512, 1024, or 2048, using the previously mentioned red coloured buttons on the Data logger case. The IMU data are recorded with the frequency of 200 Hz, while GNSS data with the frequency of 1 Hz. All messages from specified topics are recorded and saved as a time-synchronous rosbag file. This rosbag file can later be used as an input to a SLAM algorithm that generates 3D maps of the surveyed areas [5].

GNSS sensor package is another ROS package, coded in Python, that is used to retrieve National Marine Electronics Association (NMEA) messages from the ublox GNSS module. Through the integrated Networked Transport of RTCM via Internet Protocol (NTRIP) client, this driver can also receive correction data from Radio Technical Commission for Maritime (RTCM) messages. Using this NTRIP client, correction data from external services are included in the GNSS module to improve measurement accuracy. The application of these correction data is known as GNSS RTK. When the signal from the satellites is clear and the base station from the correction data service is not far away, i.e., less than 10 km from the GNSS module, the precision error is less than 2.5 cm with fixed RTK. When the conditions are not ideal, the module can still achieve a precision of $10 - 45$ cm, using RTK float [5].

5.1.3 Riegl VZ-6000

Based on Riegl's exclusive V-Line technology, the 3D VZ-6000 TLS provides unmatched long-range reflectorless measurement performance of more than 6000 m. This scanner works well even in difficult circumstances, like reduced vision brought on by haze, rain, snow, or dust, making it a good choice for measuring snowy and icy terrains. The reason for such good performance in snowy and icy terrain is the fact that the laser wavelength is 1064 nm, which is hardly absorbed by snow. The scanner can be used in several different ways, such as standalone mode with a touchscreen interface, remote control using VNC Viewer on a tablet or mobile device, or customised mode with third-party tools based on Riegl's interfaces and scanner libraries [45]. Figure 5.6 shows the external look of the Riegl VZ-6000.

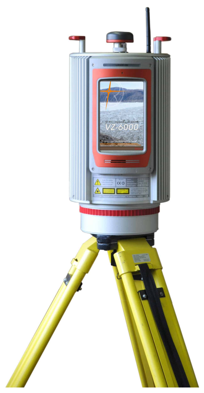


Figure 5.6: Riegl VZ-6000 TLS [45]

The scanner is equipped with a number of noteworthy features, such as a broad FoV of 60° (vertical) by 360° (horizontal), high-speed data collecting of up to 222,000 measurements per second, high accuracy and precision ranging based on echo digitization and online waveform processing, and the capacity to track numerous targets. Additionally, it has improved camera choices, a built-in calibrated digital camera, onboard inclination sensors, an integrated compass, an optional waveform data output, an integrated L1 GNSS receiver with an antenna, and a small and durable build [45].

Topography and mining, glacier mapping, snow field monitoring, long-range monitoring, civil engineering, and archaeology are some typical uses for the 3D VZ-6000 TLS. The Riegl VZ-6000 scanner is an invaluable tool for a variety of applications that call for accurate and dependable 3D measurements in a variety of settings due to its greater measuring range, ability to function in difficult circumstances, and advanced capabilities. When it is combined with the Riegl software package, called RISCAN PRO, it can output accurate 3D point cloud maps that can be used as ground truth data for other measurement techniques [45].

5.1.4 Ouster Lidars and Riegl VZ-6000 Comparison

Table 5.1 shows a direct comparison between the VZ-6000 TLS and two lidars used in this thesis, OS1-64 and OS2-64. After analysing the values, it is quite clear that the Ouster lidars are much more robust and mobile, but lack in terms of accuracy compared to the Riegl. This justifies the intention to use the Riegl VZ-6000 for creating a reference dataset, as is explained in section 5.2.

Parameter	**Riegl VZ-6000**	**Ouster OS1 - 64 (Gen 6)**	**Ouster OS2 - 64 (Gen 6)**
Price	160,000 €	10,000 €	21,000 €
Range	5m - 6000m (R $\geq$ 90%, 50 kHz PRR) 5m - 4200m (R $\geq$ 90%, 150 kHz PRR) 5m - 3300m (R $\geq$ 90%, 300 kHz PRR) 5m - 3600m (R $\geq$ 20%, 50 kHz PRR) 5m - 2400m (R $\geq$ 20%, 150 kHz PRR) 5m - 1800m (R $\geq$ 20%, 300 kHz PRR)	0.3m - 100m (R = 80 %, $>$ 0.9 DP) 0.3m - 120m (R = 80 %, $>$ 0.5 DP) 0.3m - 45m (R = 10 %, $>$ 0.9 DP) 0.3m - 55m (R = 10 %, $>$ 0.5 DP)	1m - 210m (R = 80 %, $>$ 0.9 DP) 1m - 240m (R = 80 %, $>$ 0.5 DP) 1m - 80m (R = 10 %, $>$ 0.9 DP) 1m - 100m (R = 10 %, $>$ 0.5 DP)
Range Accuracy	15 mm	30 mm (for Lambertian targets) 100 mm (for retroreflectors)	30 mm (for Lambertian targets) 100 mm (for retroreflectors)
Range Precision	10 mm	0.7 cm - 5 cm dependent on distance	2.5 cm - 8 cm dependent on distance
Vertical FoV	60°	45°	22.5°
Horizontal FoV	360°	360°	360°
Minimum Range	5 m	0.3m	1 m
Maximum Range	6000 m	120 m	240 m
Max. measurement rate (pts/s)	37,000 (50 kHz PRR) 113,000 (150 kHz PRR) 222,000 (300 kHz PRR)	1.310.720	1.310.720
Laser Wavelenght	1064 nm	865 nm	865 nm
Beam divergence	0.006875°	0.18°	0.09°
Beam diameter (range)	15mm (0 m) 60mm (500m) 120mm (1000m) 240mm (2000m)	9.5mm (0 m)	19mm (0 m)
Angular step width vertical (θ)	0.002° $\leq \Delta\theta \leq$ 0.280°	0.7 ° at 64 channels	0.35 ° at 64 channels
Angular step width horizontal (ϕ)	0.002° $\leq \Delta\phi \leq$ 3°	0.176° (2048 pts) 0.35° (1024 pts) 0.7° (512 pts)	0.176° (2048 pts) 0.35° (1024 pts) 0.7° (512 pts)
Weight	14.5 kg	0.377 kg	1.1 kg
Dimensions	248 x 226 x 450 mm	Diameter: 85 mm; h: 58.35 mm	Diameter: 119.6 mm; h: 98.9 mm
Power supply	11 - 32V DC / 75W – 90W	22 – 26V DC; 24V nominal	22 – 26V DC; 24V nominal
Eye Safety Class	Class 3B	Class 1	Class 1
User Interface	7" VWGA Touch-Display	external computer needed	external computer needed
GNSS Receiver	integrated	not integrated	not integrated
Compass	integrated	not integrated	not integrated
Camera	SMP campera integrated	not integrated	not integrated
Data storage and transfer	Integrated SSD	No storage built in	No storage built in
Peak operating temperatures	0°C to + 40 °C	-40°C (OS1 cold start) to + 60°C	-40°C (OS2 cold start) to + 64°C
IP Level	IP 64	IP 68	IP 68
Shock	not specified	IEC 60068-2-27: 100 g, 3 shocks x 6 directions	IEC 60068-2-27: 25 g, 400 shocks x 6 directions
Vibration	not specified	IEC 60068-2-64: 3 Grms, 3 axes x 8 h duration	IEC 60068-2-64: 2 Grms, 3 axes x 8 h duration

Table 5.1: Comparison of automotive lidars Ouster OS1-64, OS2-64 and TLS Riegl VZ-6000 (from OS1 datasheet [46], OS2 datasheet [41] and VZ-6000 datasheet [45]), R = reflectance, DP = detection probability, PRR = pulse repetition rate (adapted from Hammer [47])

5.1.5 Post-processing Setup

All data post-processing, as well as the creation of 3D point cloud SLAM maps, is performed on the same Dell Inspiron 14 5000 laptop that was bought in May 2020. Hardware specifications of the laptop are the following:

- **Central Processing Unit (CPU):** Intel Core i7-10510U CPU with the clock speed of 1.80 GHz
- **Random Access Memory (RAM):** 20 GB DDR4
- **Graphics Processing Unit (GPU):** NVIDIA GeForce MX230 with 2 GB of dedicated GDDR5 RAM
- **Internal storage:** PC SN740 NVMe WD SSD with 512 GB of storage capacity

On this laptop, Ubuntu 20.05 Long Term Support (LTS) (Focal Fossa) is installed. It is coupled with ROS Noetic Desktop Full version. This combination is chosen because Focal Fossa is the last version of Ubuntu that supports ROS Noetic, which, itself, is the final version of ROS. Furthermore, this setup offers the ability to, later, upgrade from ROS to ROS2.

All of the operations performed on the data, both post-processing and SLAM algorithm runs, are performed under the same conditions, i.e. the only programs running on the computer are the algorithms for data processing. No other applications and programs are running in the background. This is done to ensure fair and equal testing conditions for each of the individual SLAM algorithms tested.

5.2 Data Gathering Process

The data gathering process on the survey site can be divided into two different parts. The first part consists of gathering the ”ground truth” dataset, using the Riegl VZ-6000 TLS, and the second part is making the ”working” dataset, using the MOLISENS setup.

5.2.1 Cretaing the ”Ground Truth” Dataset

Firstly, the Riegl tripod needs to be positioned on the surveying location and the VZ-6000 needs to be mounted onto it. The next step is to ensure that the TLS is level with the ground and to connect the power source and power it on. Once it is powered on, the scanner itself needs to be positioned as desired, the FoV and both angular resolutions need to be adjusted so that the features that need to be captured are inside the scanner’s FoV. The scanning itself can last from a few minutes up to a few dozen minutes, depending on the level of detail of the scene captured, as well as the chosen scanning settings. It is important to note that nobody should look into the laser opening of the scanner during the operation, as it can cause permanent eye damage.

After the scanning is done, the created files can be transferred from the Riegl VZ-6000 to a portable USB memory stick and then post-processed. The post-processing is done on a Laptop that has RISCAN PRO software already installed. The data created by VZ-6000 are uploaded to the software, which is then used to create a full 3D point cloud map in the desired format. This map can be later processed in some other point cloud processing software such as Open 3D or Cloud Compare. Figure 5.7 shows how the Riegl VZ-6000 is used to capture a 3D point cloud map of the landscape.

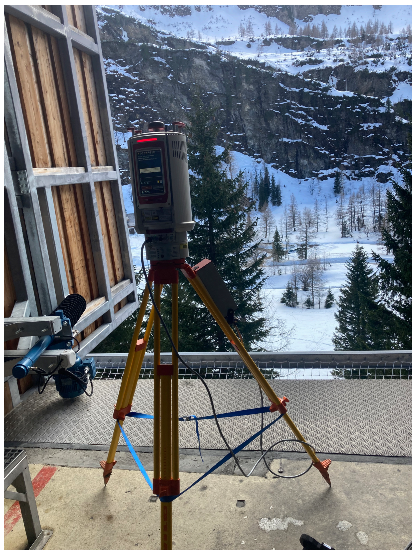


Figure 5.7: Riegl VZ-6000 TLS in the field - Sonnblick Observatory

Additionally, in case accurate georeferencing of the 3D point cloud map or alignment between two different point cloud maps is required, retroreflective targets can be placed inside the surveyed area. The placement of targets should be such that they offer extra features in places that are not rich in natural features (trees, buildings, vehicles, etc.). Also, it should be avoided to place the targets in a straight line, as well as that the targets are obscured by some other features. It should be noted that the Ouster OS2-64 has a reduced range accuracy when measuring retroreflective targets, as mentioned in Section 5.1.1. The targets are made from a plastic drainage pipe that is wrapped with a retroreflective foil. The outer diameter of the pipe is 110 mm and the length of the target is 100 mm. Figure 5.8 shows the reflective target in the field during a survey.

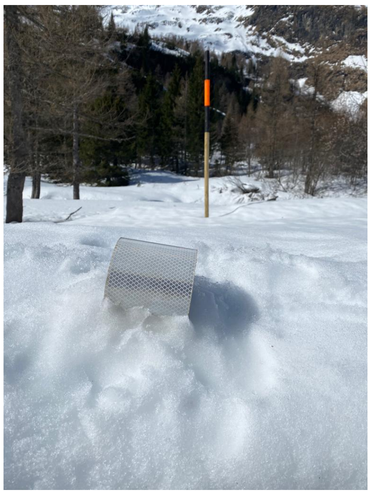


Figure 5.8: Reflective target in the field - Sonnblick Observatory

5.2.2 Creating the ”Working” Dataset

The ”working” dataset is created using the MOLISENS setup. The first step is to position the reflective targets at desired locations, if they are required. Afterwards, the MOLISENS itself needs to be set up. It is important that it is safely secured to the vehicle it is being transported on. Figure 5.9 shows how the MOLISENS sensors are set up on a gondola during our expedition to the Sonnblick Observatory. In this case, the Data logger and the Power supply are positioned inside the gondola.

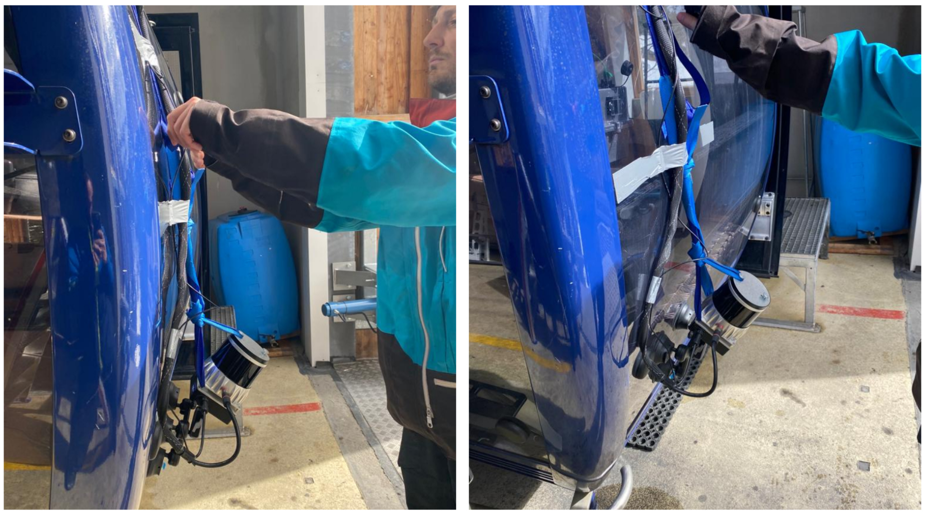


Figure 5.9: MOLISENS setup on a gondola - Sonnblick Observatory

After the MOLISENS system has been properly positioned, it is time to check that all the software functionalities operate properly. This is done by testing each feature individually using the laptop that is connected to the Raspberry Pi 4 of the Data logger. Upon ensuring the satisfactory functionality of all components, the commencement of the surveying process may ensue.

Prior to commencement, it is imperative to activate the system and carefully determine the rotation frequency and horizontal resolution of the Ouster OS2-64. Once that is selected, the survey can start. All the data are saved in a rosbag file. Once the survey is completed, the data recording is stopped and the finalised rosbag file, containing all the data, can be transferred to a computer for later post-processing. It is important to note that due to the extremely big difference in the size of the bag files (2-5 times), lidar data are saved in a custom Ouster message format (*ouster_ros/PacketMsg*) instead of the standard 3D point cloud message format (*sensor_msgs/PointCloud2*). Later, in the Data post-processing step, which is explained in Section 5.3, lidar data are converted into the standard format.

5.3 Data Post-processing

Since the lidar point clouds are not saved in their standard format (*sensor_msgs/PointCloud2*), they have to be converted into it. This process is rather simple and, also, offers another benefit, by allowing the visual inspection of the saved point clouds, at the same time as they are being converted into the standard format. This is beneficial because it allows the visual control of the goodness of the saved data.

To complete the conversion, an internally developed, open-source ROS package is installed on the laptop used for all post-processing steps [48]. Both hardware and software specifications of the laptop are discussed in detail in Section 5.1.5. Before starting the conversion process, appropriate lidar configuration files need to be used. They determine the settings of the package with regards to the type of lidar used (Ouster OS1 or Ouster OS2) and the horizontal resolution (1024 or 2048). Subsequently, the package is launched, the rosbag containing the raw data from MOLISENS with the clock time published is played, and, finally, the desired topics are saved into another rosbag. The desired topics include *clock time, lidar point clouds, IMU and GNSS information*, but further topics can also be saved. Finally, the new rosbag contains the converted data that can then be easily fed as input data to different SLAM algorithms.

5.4 Datasets

Throughout this thesis, three different datasets will be used to provide variability to the test conditions of each algorithm. Each dataset depicts a different environment, as well as a different mounting vehicle for the MOLISENS system. Each individual dataset poses different challenges that the SLAM algorithms need to overcome. More details about each dataset are available in the following Sections 5.4.1, 5.4.2, and 5.4.3. Finally, Section 5.4.4 explains how the analysis of IMU and GNSS data of each dataset is performed. It also introduces the IMU and GNSS data specifics for each dataset, as well as the comparison between the datasets. The analysis of the data is important because it can explain some of the phenomena later encountered, during the testing of different SLAM algorithms.

Table 5.2 offers a brief overview of all the datasets used in this thesis. It can be seen that each dataset has the same settings regarding the MOLISENS lidar sensor rotation frequency of 10 Hz and the horizontal resolution of 1024 points in the entire $360\,^\circ$ FoV.

Dataset name	Date	Sensor carrier	Ouster sensor	Ground truth	Horizontal resolution	Rotation frequency
ViF building	12/12/2022	Handheld	OS1-64	NO	1024	10 Hz
Rijeka Harbour	15/10/2021	Boat	OS1-64	NO	1024	10 Hz
Sonnblick Observatory	21/03/2023	Gondola	OS2-64	YES	1024	10 Hz

Table 5.2: Overview of the datasets used in the thesis

5.4.1 Sonnblick Observatory Dataset

This dataset was gathered on the 21st of March 2023 near the site of Sonnblick observatory in Rauris Valley, Austria. Figure 5.10 shows the survey site viewed from the satellite on Google Maps. During the time of the survey, the ground was covered by snow, which presents an extra challenge to both the equipment as well as the SLAM algorithms, due to the fact that snow is highly reflective, which may affect the quality of the generated lidar point clouds.

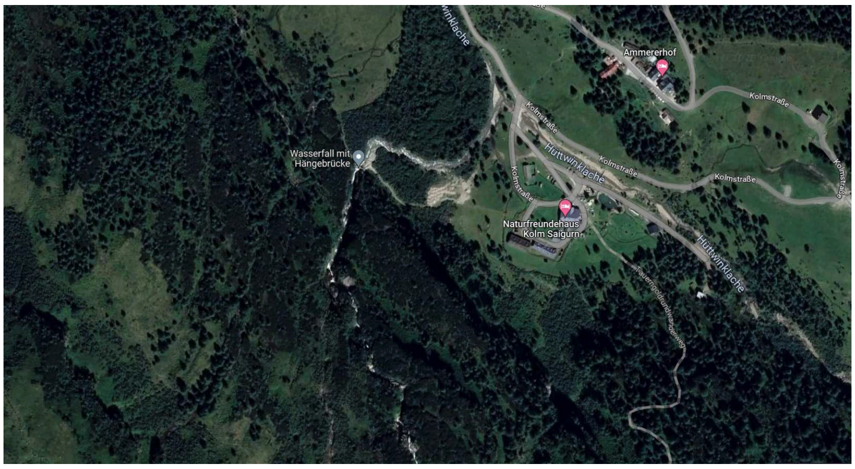


Figure 5.10: Google Maps representation of the *Sonnblick Observatory* dataset

This dataset consists of two distinct sets of data. The first one is the ”ground truth” dataset that will be used as a reference for the 3D point cloud maps generated by different SLAM algorithms. The Riegl VZ-6000 TLS is positioned at the base of the gondola in the valley station, while the exact surveying process is highlighted in Section 5.2.1.

The second set of data is the ”working” dataset which is collected by the MOLISENS setup. The exact way it was set up, in this particular expedition, is highlighted in the Section 5.2.2. For this survey, an Ouster OS2-64 lidar was used. The dataset itself is quite big, since the hardware was mounted on a gondola whose total length is 6108 m and the running time is approximately 20 minutes, only a part of the dataset is used in this work. The part chosen is the one closest to the valley station as it offers the most distinct features that can be used to produce the map. Furthermore, the ”ground truth” level of details is the highest in that part, since the TLS was positioned in the valley station, as well.

5.4.2 Virtuelles Fahrzeug (ViF) Building Dataset

This dataset was gathered on the 12th of December 2022 on the site of the Virtual Vehicle Research GmbH office building in Graz, Austria. Figure 5.11 shows the survey site viewed from the satellite on Google Maps.

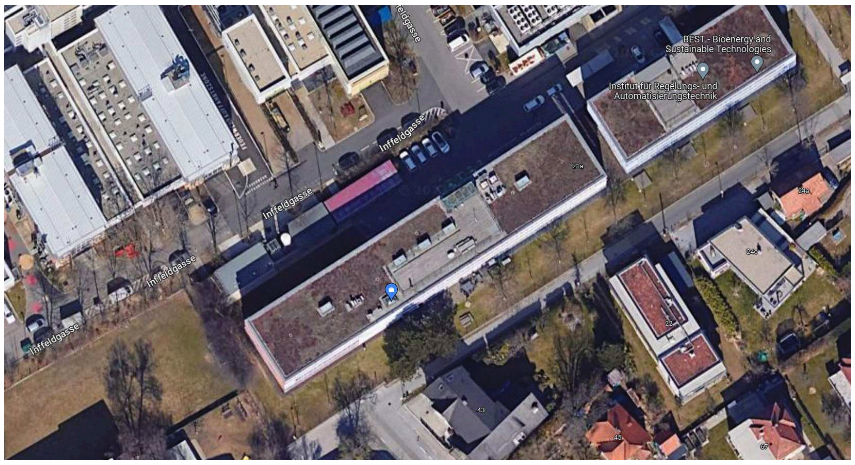

BEST - Bioenergy and Sustainable Technologies
Institut für Regelungs- und Automatisierungstechnik
Inffeldgasse
Inffeldgasse
Inffeldgasse
Inffeldgasse
Inffeldgasse

Figure 5.11: Google Maps representation of the *ViF Building* dataset

This dataset only contains the ”working” dataset as well. However, it depicts another type of environment, which is an urban environment with a lot of potentially dynamic features such as people, cars, etc. Also, it is worth noting that in this survey, the MOLISENS setup is handheld, which, introduces some constant rocking motion of the sensors. How the setup looks can be seen in figure 5.12. Ouster OS1-64 lidar was used in this survey. This dataset contains a loop, i.e. the survey started and ended at the same position, which is useful to test loop-closure capabilities of different SLAM algorithms.

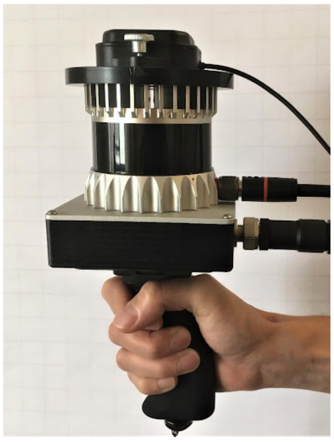


Figure 5.12: MOLISENS handheld setup

5.4.3 Rijeka Harbour Dataset

This dataset was gathered on the 15th of October 2021 on the site of Sušak Harbour in Rijeka, Croatia. Figure 5.13 shows the survey site viewed from the satellite on Google Maps. Since this Google Maps screenshot was not taken on the same day as the survey itself, it is highly possible that there is a difference in the number of ships and their position between the image and the actual dataset itself.

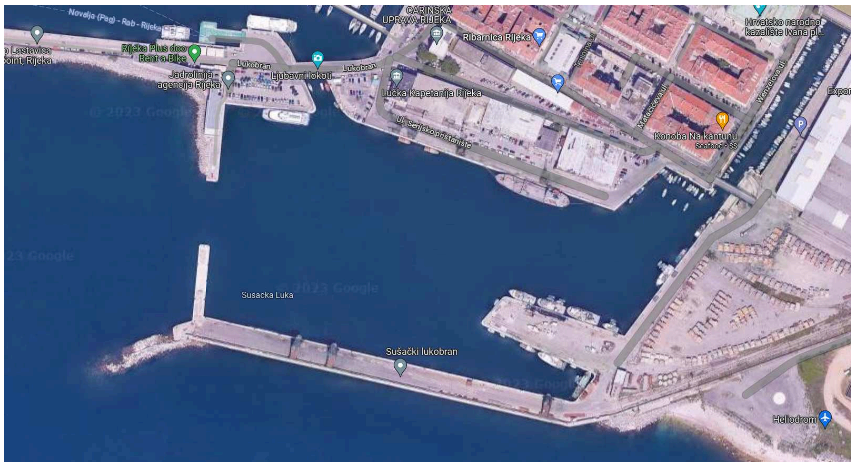


Figure 5.13: Google Maps representation of the *Rijeka Harbour* dataset

Even though this dataset only consists of the ”working” dataset, it is still highly valuable since it tackles a completely different environment compared to the first two. This dataset deals with the maritime environment where it is common to have the sensors move all the time, due to the movement of the sea itself. For this survey, the MOLISENS setup was fixed on a boat, as can be seen in Figure 5.14. Ouster OS1-64 lidar was used in this survey as well. Just like the previous one, this dataset also contains a loop.

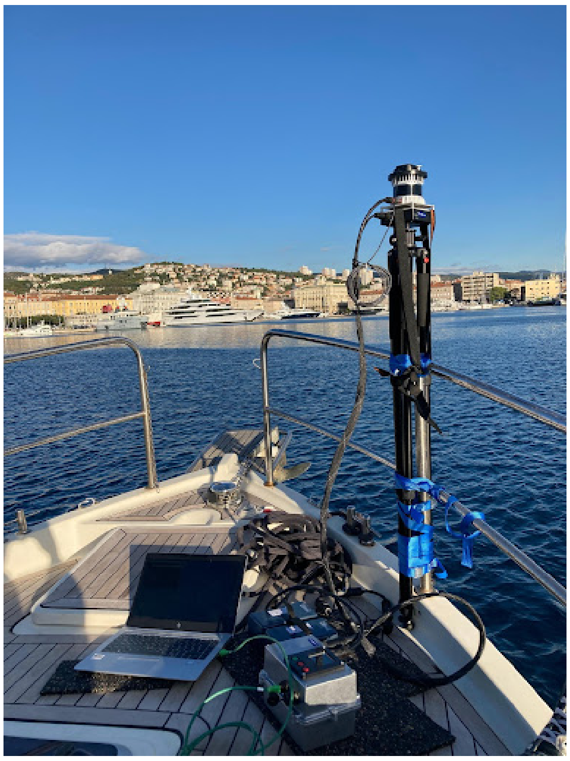


Figure 5.14: MOLISENS setup on a boat

5.4.4 IMU and GNSS Data Analysis

Analysing the data from the GNSS and, especially, the IMU can provide valuable information about the conditions under which the survey was completed. This specifically applies to external conditions affecting the motion and stability of the sensor, such as wind, rough seas, big waves, bumps in the road, jumps, etc., depending on the type of environment a survey is being performed in. The effects of these phenomena will be shown in the spikes on the linear acceleration, angular velocity and attitude graphs. These spikes represent swift changes in the sensor's movement and position. They can also be interpreted as noise in the data, and given the fact that some of the SLAM algorithms tend to perform integration of IMU data, then the noise amplifies the error, which is something that should be avoided, if possible.

Analysis of this data is performed using a software package that was developed in the scope of this thesis. It consists of two separate parts. The first part is a ROS package that converts the IMU and GNSS data from the post-processed MOLISENS rosbag into a format, that separates the complex sensor messages into simple ROS topics, that are then saved into a new rosbag. After the new bag file has been recorded, it is fed to the second part of the software package. The second part is written in MATLAB/Octave and converts the original IMU attitude data quaternions into Euler angles, as well as changing the angle unit from radians to degrees. Also, it saves all the data into a MATLAB mat file so it can be later visualised using MATLAB or Octave. Data visualisation is also performed by a MATLAB script.

Figure 5.15 shows the Cartesian coordinate frame with the 3 main axes of the MOLISENS setup. This information is important for the latter analysis of the IMU data.

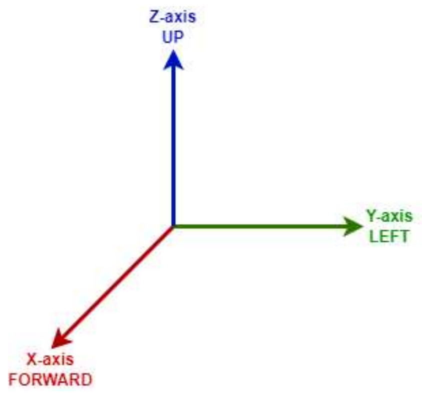


Figure 5.15: Coordinate frame of the MOLISENS setup

Sonnblick Observatory

Figure 5.16 shows the values of Angular velocity around each of the three axes of the MOLISENS coordinate system. Frequent changes in the direction of the angular motion can be observed. These changes are caused by the movement of the gondola itself. However, for this dataset, the emphasis should be placed on the graph representing the Y-axis. On this graph, the effect of the gondola starting to move is evident, as indicated by a short spike around the 40th second. Subsequent to that juncture, a multitude of frequent changes in both the value and direction of motion become apparent. This occurrence can be attributed to the influence of cross-wind impacting the gondola, thereby inducing a rocking motion. However, this rocking motion is not strong, as it peaks at an angular velocity of around $4\,^{\circ}$/s.

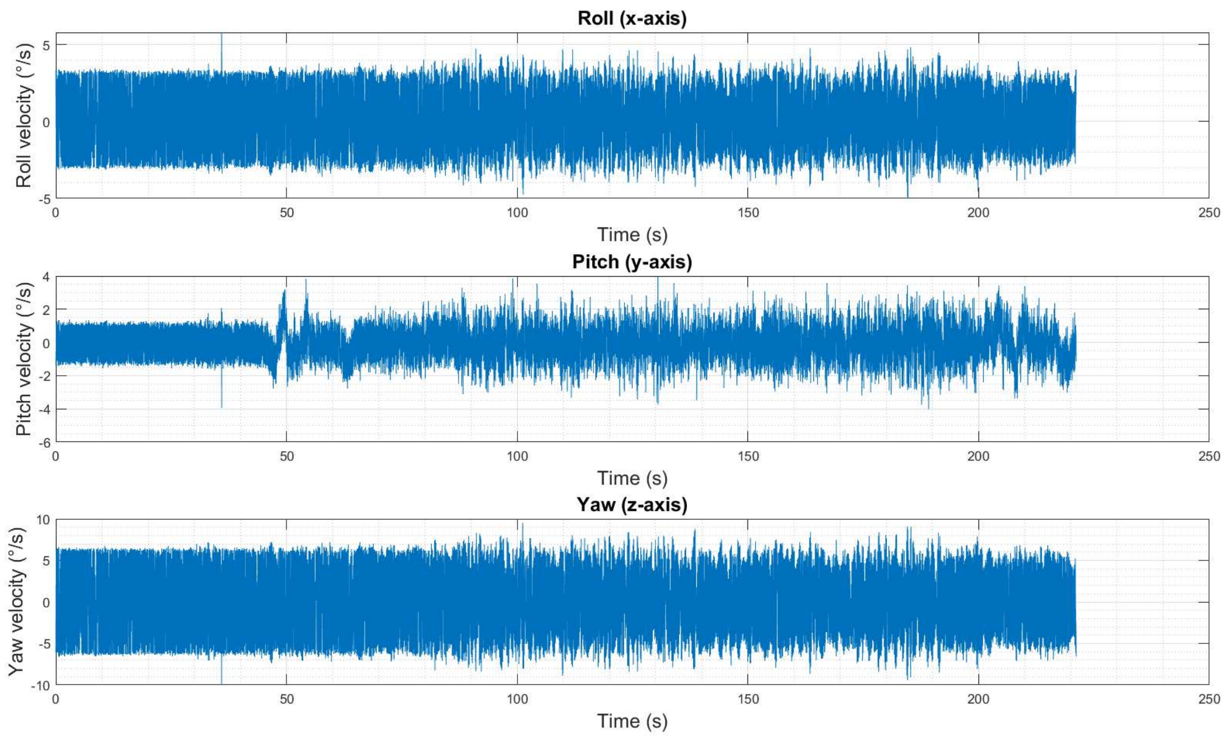


Figure 5.16: Angular velocity - Sonnblick Observatory

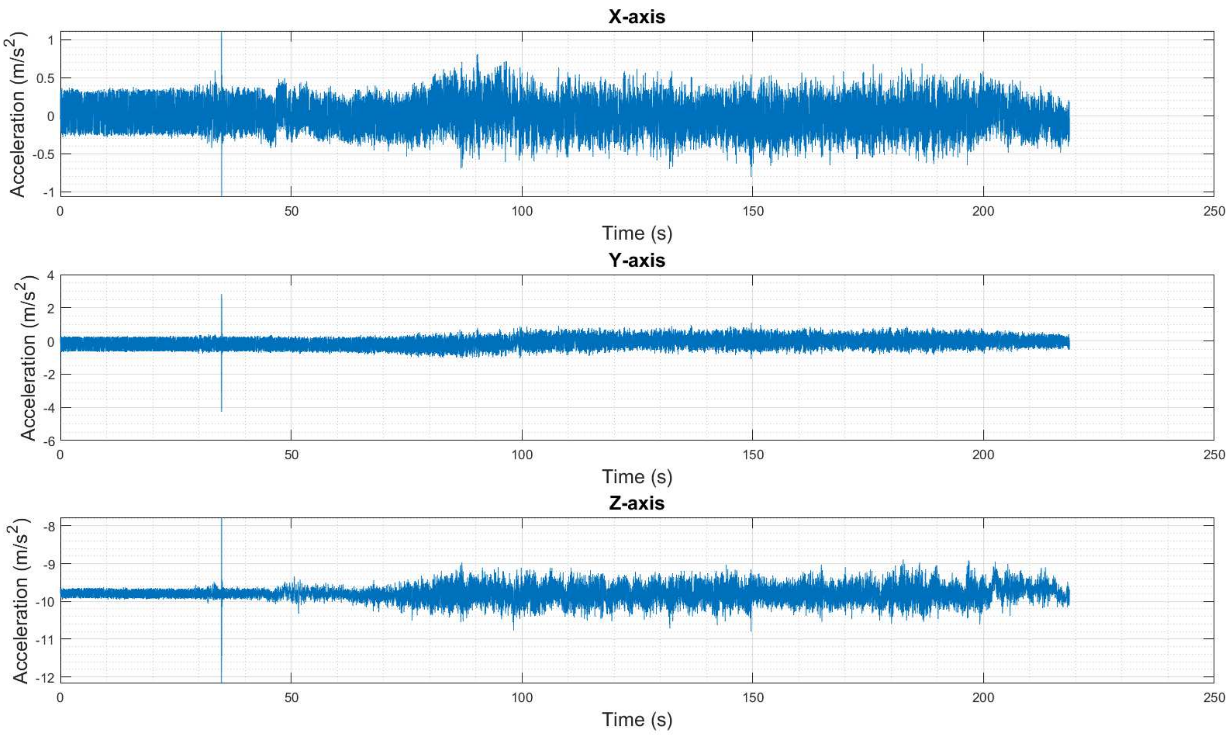


Figure 5.17: Linear acceleration - Sonnblick Observatory

A similar movement of the gondola can be noticed in Figure 5.17 which shows linear acceleration along all 3 axes of the MOLISENS system. In this depiction, a discernible spike around the 40th second can be observed, aligning precisely with the commencement of the gondola's motion. Subsequently, frequent alterations in the direction of acceleration become apparent along both the X-axis and the Z-axis. These variations depict a gradual, undulating rocking motion that coincides with the gondola's gradual vertical oscillation during its movement.

Rijeka Harbour

Figure 5.18 shows all the GNSS data from this survey. In the initial graph, the altitude displays a range between 2.4 m and 4 m, which aligns with the notion that the boat is in synchrony with the undulations of the sea waves. The subsequent two graphs illustrate the longitude and latitude values throughout the survey. By examining the values themselves, it becomes evident that the survey commenced and concluded at the same location.

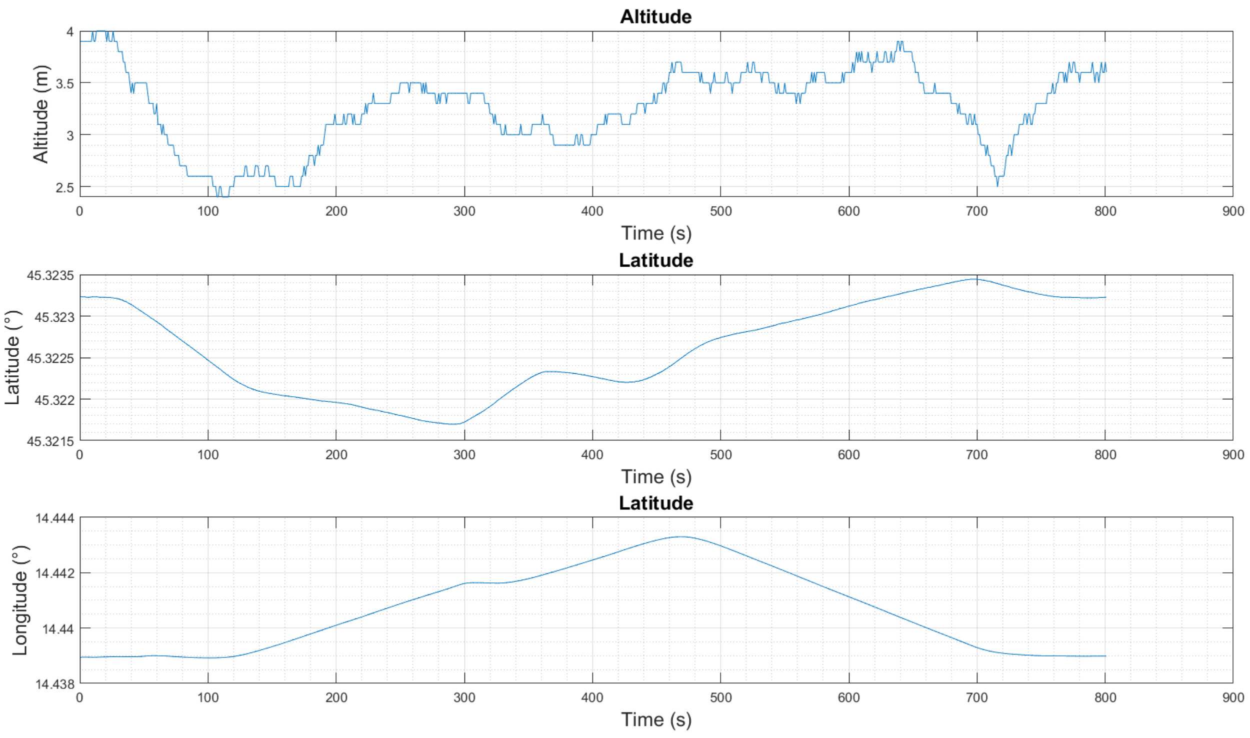


Figure 5.18: GNSS data - Rijeka Harbour

Figure 5.19 shows how the attitude of the MOLISENS setup changed during the survey. Attitude is represented by the Euler angles. It can be observed that the roll and pitch angles oscillate between $-5\,°$ and $5\,°$, which is consistent with the boat rocking on the wavy sea. The yaw angle shows the boat's orientation throughout the mission, as it made its looped path.

Similar behaviour of the boat can be observed based on the data in Figure 5.20, which shows linear acceleration along the 3 main axes of the MOLISENS system, thus the boat as well. Oscillations in the values and the direction of the linear acceleration can be observed on all axes. This happens because of the natural movement of the boat in the sea.

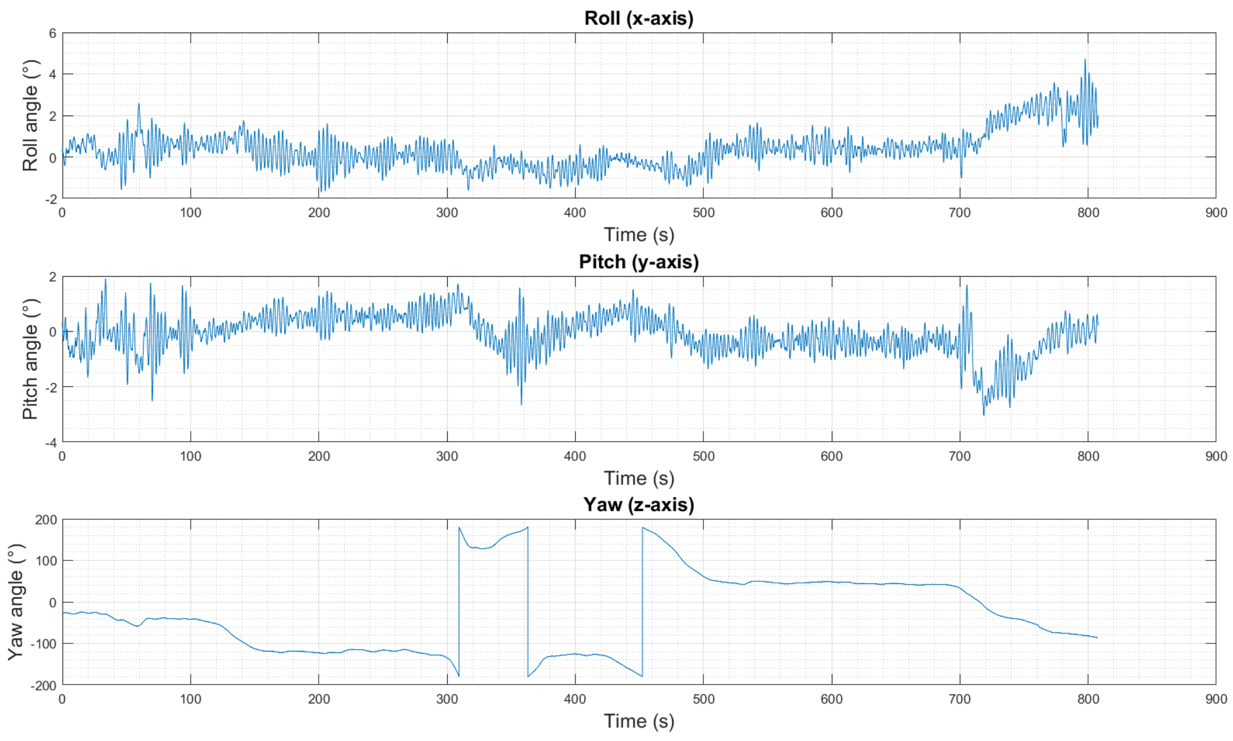


Figure 5.19: Euler angles - Rijeka Harbour

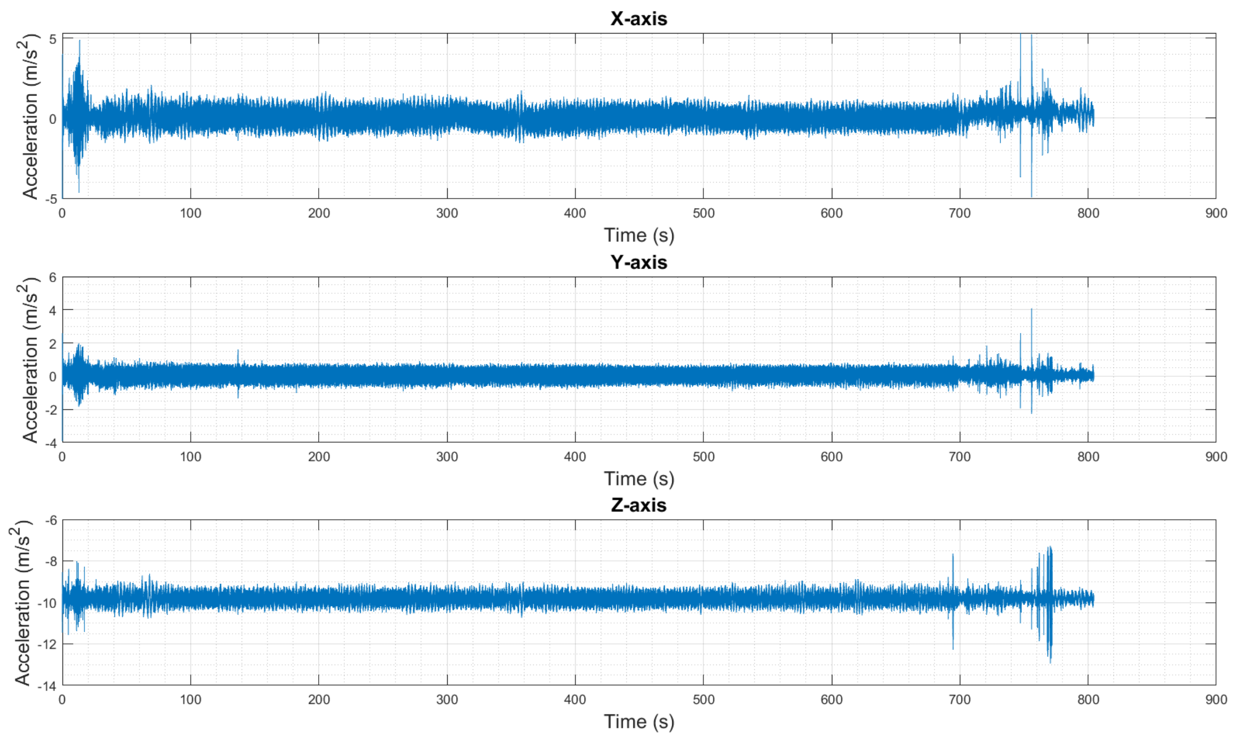


Figure 5.20: Linear acceleration - Rijeka Harbour

However, in Figure 5.20, some spikes, where the values are much bigger than the average, can also be noticed. This can be caused by multiple factors. Some of the possibilities are that a bigger wave hit the boat, or the boat is accelerating/decelerating or trying to dock. Whatever the cause is, these higher-than-average perturbations may cause some issues for the SLAM algorithm. Specifically, it may reduce the quality, or cause a full failure of the loop-closure functionality. It is particularly bad news if the goal of the survey is to produce a quality map of the area. One other issue may arise from this behaviour. If the fact that most IMU and GNSS sensors have an auto-calibration feature after they are turned on is considered, any unusually high intensity of oscillations may cause wrong calibration or wrong initial measurements. The effects of this behaviour on SLAM is depicted in more detail in Section7.1.

ViF Building

As mentioned before, this dataset was acquired by walking with the MOLISENS setup held in the hand. The oscillations in the roll and pitch of the sensor, seen in figure 5.21, are caused by the inability of the person holding the MOLISENS setup to keep it perfectly still. Furthermore, the yaw graph shows how the sensor orientation changed during the survey. From the same graph, the existence of the loop in the dataset may be observed.

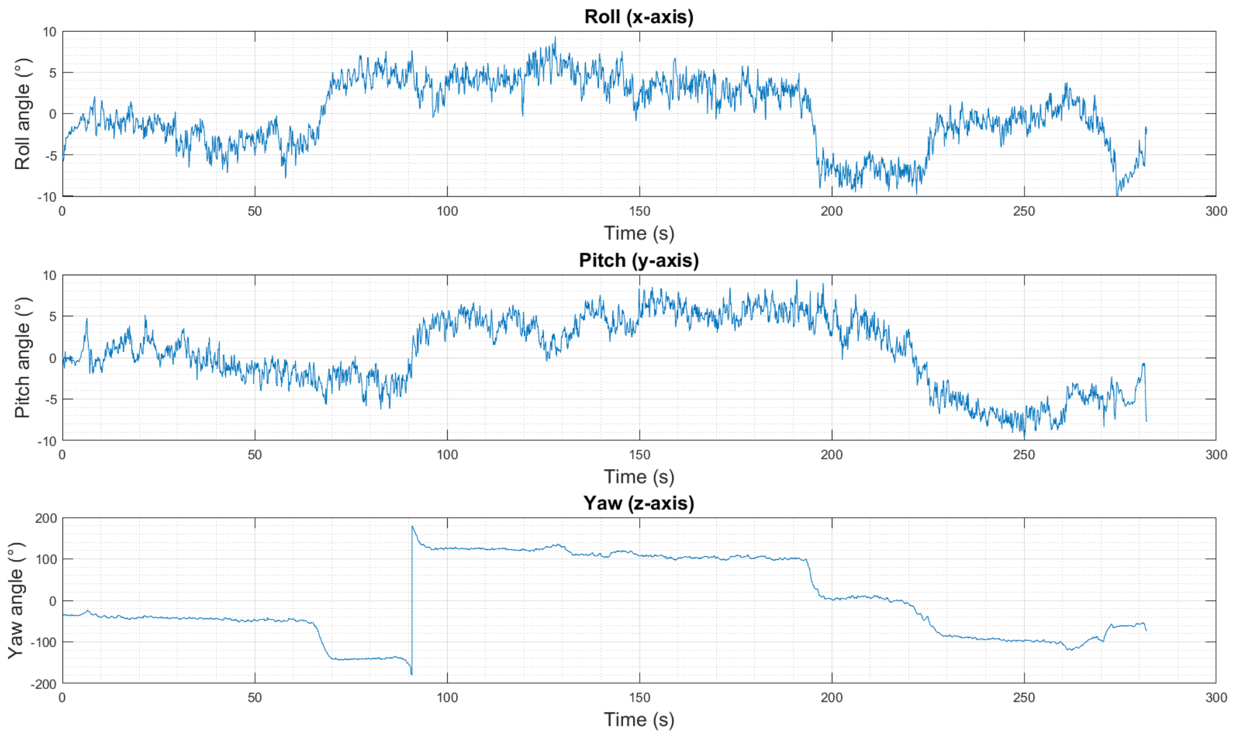


Figure 5.21: Euler angles - ViF Building

Figure 5.22 depicts the linear acceleration along the 3 main axes of the MOLISENS system, thus, also, the person who is carrying it. It is interesting to note how rough and oscillatory human walking actually is. Unlike in the previous dataset, no spikes of unusually high amplitude are noticed.

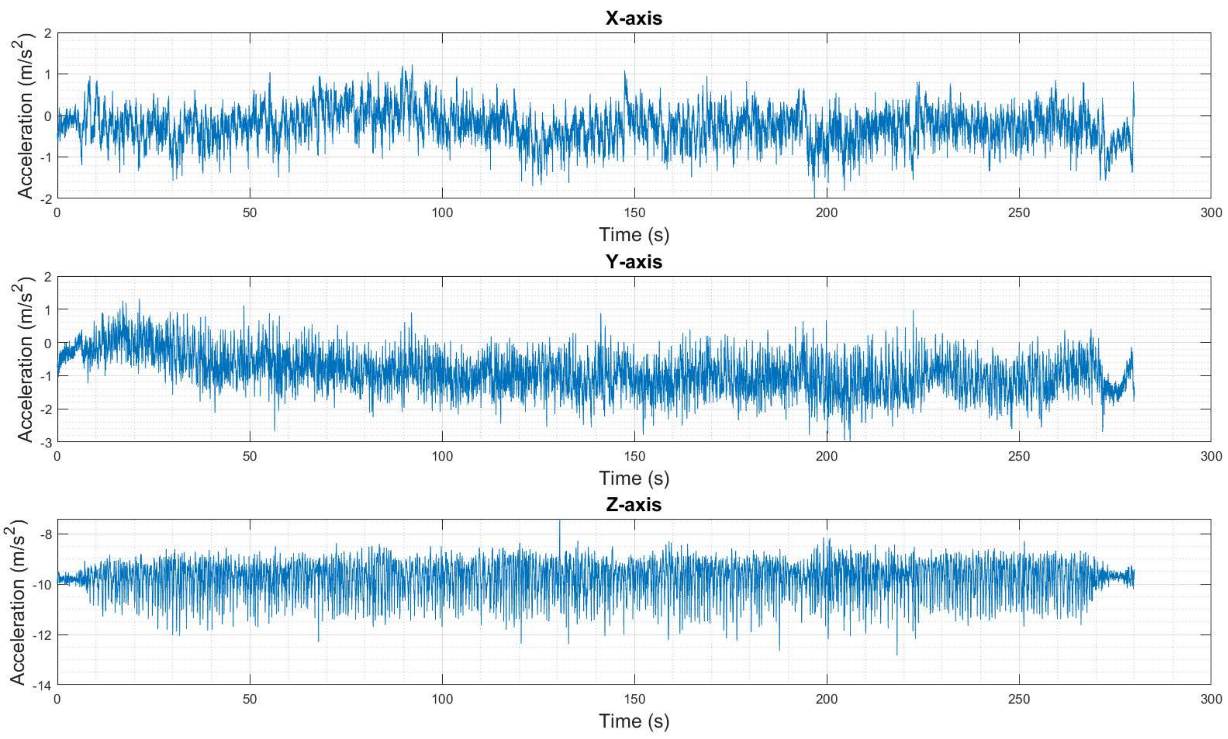


Figure 5.22: Linear acceleration - ViF building

Dataset comparison

According to [49], a box plot and a density plot are combined to create the violin plot, a graphic tool for exploratory data analysis that shows the distribution of a variable graphically. It enables a comparison of how one or more variables are distributed among several groups or categories. The violin plot displays the density of the data, in contrast to a box plot, which merely displays summary statistics [49].

The kernel density plot, which is superimposed over the box plot in the violin plot, depicts the distribution of the data. The density of the data at a specific position is indicated by the breadth of the plot there. The box's height represents the Interquartile Range (IQR), which covers the middle $50\,\%$ of the data, and the plot is symmetrical about the median. Any data points outside of this range are referred to as outliers. The whiskers extend from the box to the most extreme data points that are within 1.5 times the IQR from the box's edge [49].

In comparison to other graphical tools, violin plots provide a number of benefits. They offer a clear picture of the data distribution, making it simpler to compare data between groups. Additionally, they enable the detection of multimodal distributions, which could be overlooked when using a straightforward box plot. Finally, violin plots can be used to uncover patterns or trends in the data by detecting changes in the distribution over time [49].

Figure 5.23 shows the violin plots of linear acceleration along the 3 main axes for all three datasets considered in this thesis. A large number of outliers in the *Rijeka Harbour* dataset can be noticed. This goes in hand with the analyses presented previously, and the effects of this phenomenon will be shown and explained in Chapter 7.

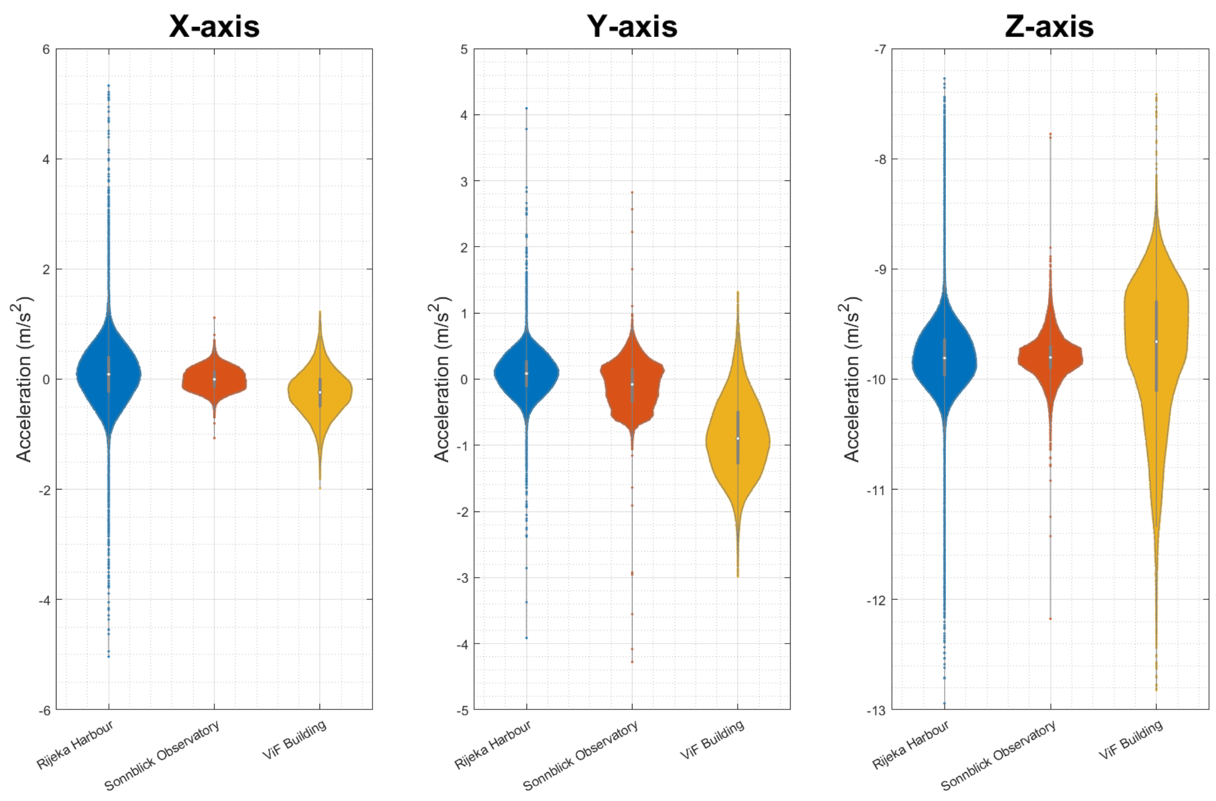


Figure 5.23: Violin plots of the linear acceleration - dataset comparison

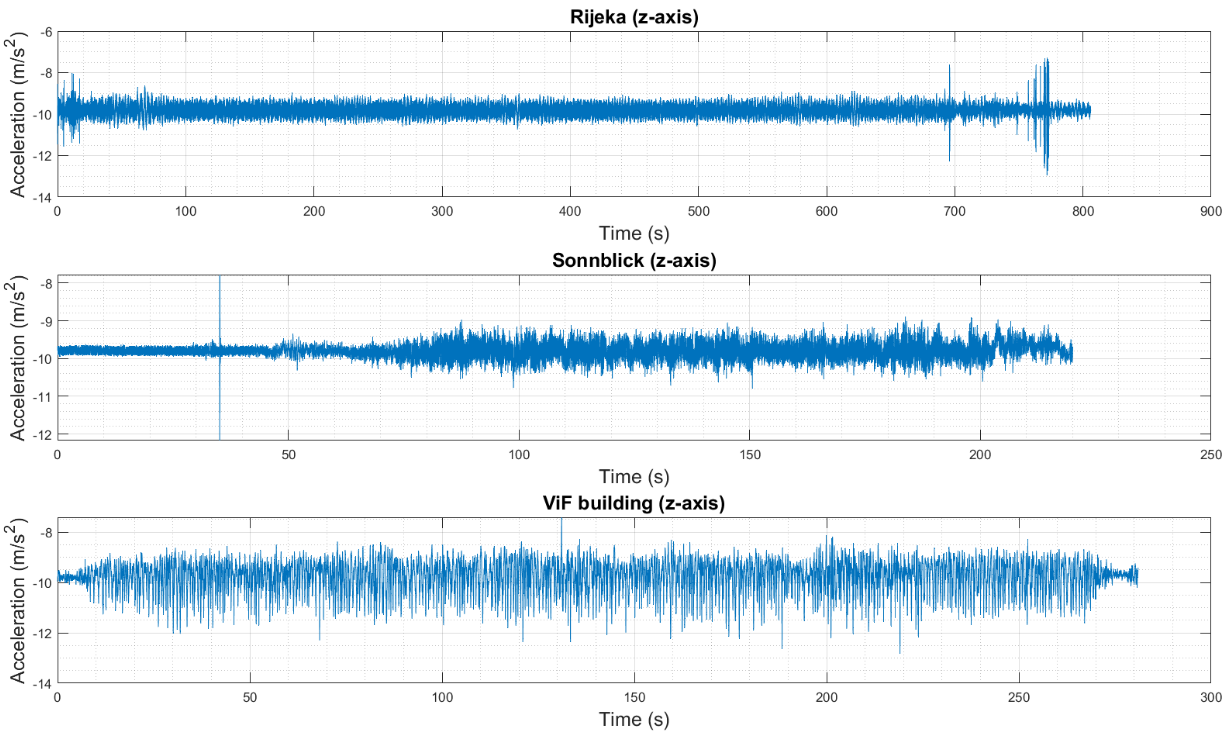


Figure 5.24: Linear acceleration along Z-axis - dataset comparison

Figure 5.24 is interesting because it shows the difference in steadiness between three different types of movement. It can be seen that the movement of the gondola is the smoothest in general, followed by the movement of the boat, and, finally, it is observed that the human walk is the least steady form of motion of the three forms considered.

6
Methodology

Contents

This chapter introduces different ways of measuring the accuracy of a 3D point cloud map, which is explained in Section 6.1. Since some of the datasets presented in Chapter 5 contain vegetation, which may introduce measurement uncertainty in the lidar point clouds, Section 6.2 introduces an algorithm that removes the vegetation from a previously generated 3D SLAM map. Furthermore, a metric for comparison of different SLAM algorithms is introduced in Section 6.3. Finally, different SLAM algorithms used in this thesis are introduced in Section 6.4, as well as the methods of improving the performance of the best one in Section 6.5.

6.1 Measuring the Accuracy of a SLAM Map

No SLAM algorithm in existence is capable of producing an absolutely perfect 3D map of the real world. There is always, at least some degree of error, no matter how small it might be. There are multiple ways of approximating and explaining this error, but as Kalenjuk and Leinhart [50] explain, the end user of the 3D map is usually not interested in the reasons behind the distortion of the map, but only in the order of magnitude of the distortion. The way to give a measurement of this distortion, thus also the accuracy of the 3D map, is to compute the distance between the corresponding points of the reference and the generated SLAM point cloud. To generate a reference point cloud, a TLS is used. One advantage of using a TLS to produce the reference point cloud is a denser and more accurate point cloud than the one produced by the automotive lidar, which gives more possibilities when computing distances between them [50]. Cloud Compare software, introduced in Section 2.3, is used to perform the comparison between the point clouds.

As mentioned in its Wiki page [51], Cloud Compare offers three different ways of measuring distances between two point clouds representing the same 3D space, a set of values chosen as a metric to determine the accuracy of a SLAM 3D map. The first way, which is also the simplest, is to compute the nearest neighbour distance. It means that for each point of the compared cloud, the software will find the nearest point in the reference point cloud and calculate their Euclidean distance. The nearest neighbour distance principle works well, only if the reference point cloud is dense enough, which it is in our case. The principle of nearest neighbour distance is shown in Figure 6.1 [51].

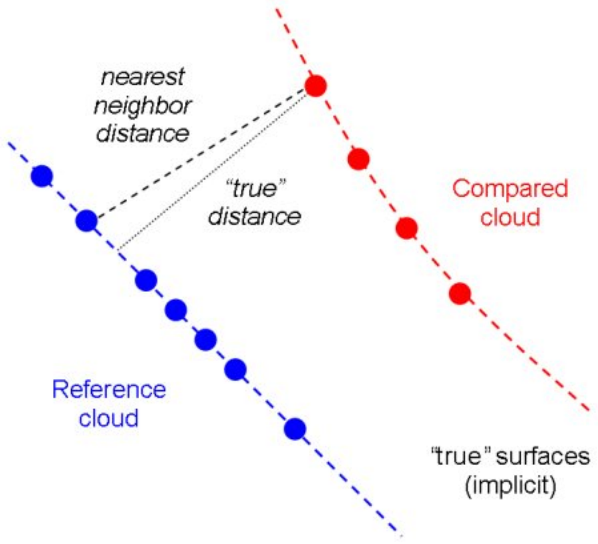


Figure 6.1: Nearest neighbour distance principle [51]

Unfortunately, the nearest neighbour distance principle is not the optimal solution, given that there are cases when the reference point cloud is not dense enough, making the nearest neighbour distance estimation imprecise. For this reason, a second method, called local modelling, is introduced. As well as global modelling, local modelling also tries to obtain a better model of the surface, in order to mitigate the effects of the lack of density in the reference point cloud. Unlike global modelling, local modelling is not trying to obtain a global model of the surface, but, instead, a partial surface model, which is less accurate, but also much easier to compute. So, after the software determines the nearest neighbour in the reference point cloud, instead of calculating the distance immediately, it replaces that point in the reference point cloud with a surface that is obtained by fitting a mathematical model of the surface to that reference point and its neighbours. Then the Euclidean distance is computed between the initially matched point in the compared point cloud and this locally modelled surface in the reference point cloud. This approach, statistically, gives more precise results on a global scale and is also less dependent on point cloud sampling. Figure 6.2 shows the principle of local modelling [51].

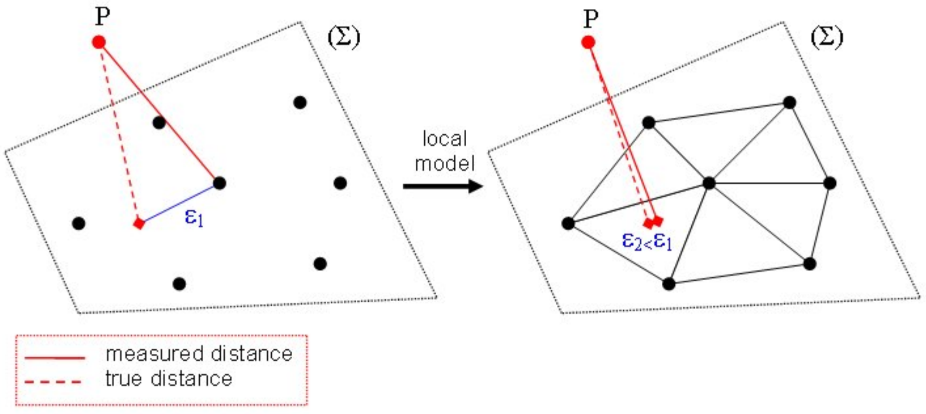


Figure 6.2: Nearest neighbour local model principle [51]

Ultimately, the user manual of Cloud Compare suggests using the Multiscale Model to Model Cloud Comparison (M3C2) algorithm [52], to compute more robust and also signed distances between two point clouds [51]. M3C2, created and explained by Lague et al. [52], only computes the distances between some specific points in the two point clouds, called core points. This approach speeds up the computations, and since the TLS point clouds are very dense, it is not needed to measure the distance at such high density. Generally, core points represent a sub-sampled version of the original reference point cloud and are regarded as regions of interest in the analysis process. This algorithm uses a two-step process to calculate the distance between points of two point clouds. This process is shown in Figure 6.3. The first step includes calculating a normal vector $\overrightarrow{N}$ for any given point i in the core point cloud. This vector is defined for each point cloud by fitting a plane to all the neighbours of i that are within the diameter D around i. As a measure of the roughness of the point cloud $\sigma_i(D)$ at a scale D in the vicinity of i, a standard deviation of the neighbours of i to the best-fit plane is used. In the second step, once $\overrightarrow{N}$ is defined, it is used to make a projection of i onto each point cloud at a projection scale d. This is done using a cylinder of diameter d, that is oriented along $\overrightarrow{N}$ and whose axis is passing through i. Projecting the two subsets of points on the cylinder axis, obtained by intersecting the cylinder with the two point clouds, yields two distributions of distances. The standard deviations of these two distributions give the estimate of the point cloud roughness $\sigma_1(d)$ and $\sigma_2(d)$ along the normal direction, while the means of the distributions i_1 and i_2 represent the average positions of the point clouds, also along the normal direction. Finally, the distance between the two point clouds $L_{\text{M3C2}}(i)$ is calculated as the distance between i_1 and i_2 [52]. Figure 6.4 demonstrates the application of the M3C2 algorithm on complex topographies.

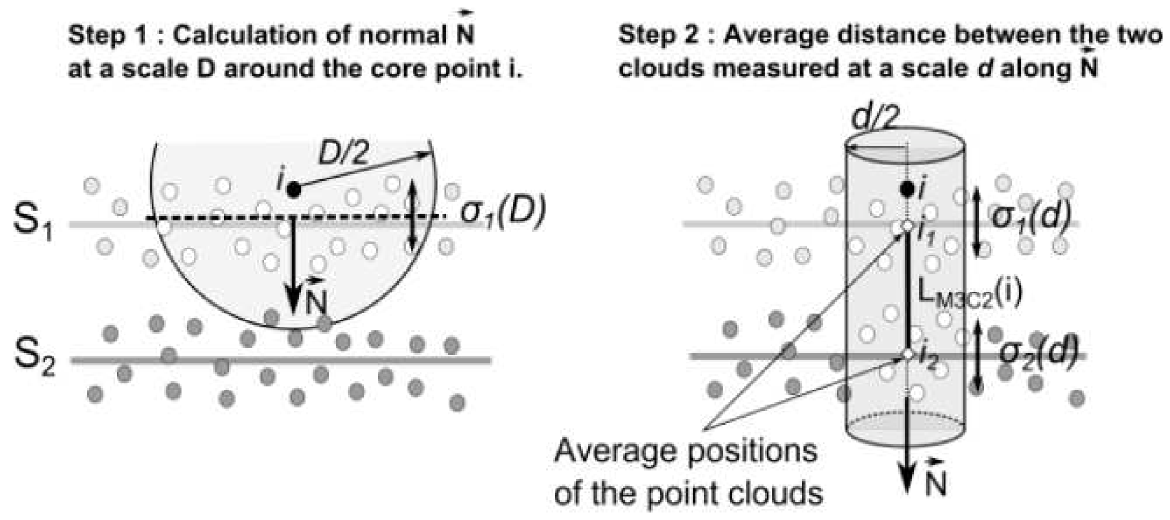

Step 1 : Calculation of normal $\vec{N}$ at a scale D around the core point i.
Step 2 : Average distance between the two clouds measured at a scale d along $\vec{N}$
D/2
i
$\sigma_1(D)$
S_1
S_2
$\vec{N}$
d/2
$\sigma_1(d)$
i_1
$L_{M3C2}(i)$
i_2
$\sigma_2(d)$
Average positions of the point clouds

Figure 6.3: M3C2 principle [52]

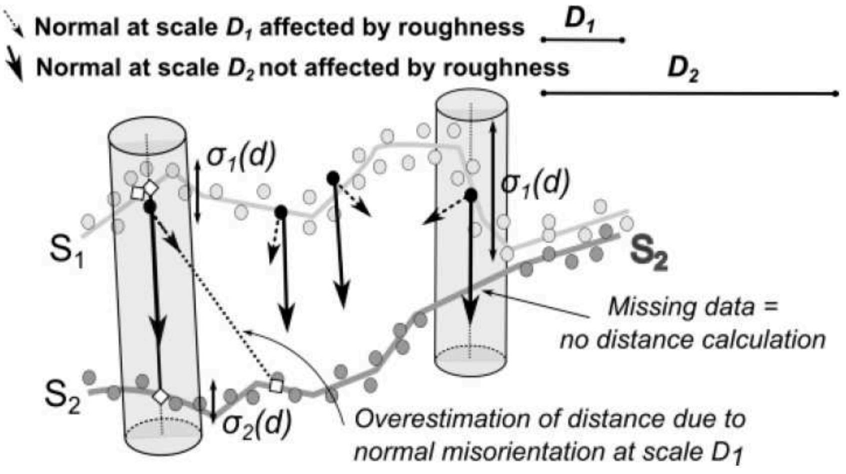

Normal at scale D_1 affected by roughness
Normal at scale D_2 not affected by roughness
D_1
D_2
$\sigma_1(d)$
S_1
S_2
$\sigma_2(d)$
Missing data = no distance calculation
Overestimation of distance due to normal misorientation at scale D_1

Figure 6.4: M3C2 in use on complex topographies [52]

6.2 Vegetation Removal

Zhang et al. [53] present a method for constructing accurate and complete Canopy Height Models (CHMs) from airborne lidar data, called Cloth simulation-based construction of pit-free canopy height models (CSF). CHMs are essential for various applications, such as forest management, environmental monitoring, and urban planning. In this thesis, they are used to remove the vegetation from a previously constructed 3D SLAM map. The goal is to have a 3D map of the ground without any other artefacts on it, because, then, those maps can be used for more accurate comparison between different maps. However, traditional CHM construction methods often result in data gaps or pits due to occlusion, penetration, or attenuation issues, which can significantly affect the accuracy and completeness of the resulting CHMs [53].

The cloth simulation algorithm used in the CSF method simulates the movement and deformation of a virtual cloth that represents the canopy. The algorithm aims to estimate the height of the canopy surface at any given location by simulating how the cloth would interact with the lidar data points. The algorithm's operation can be summarised in the following steps:

1. Data pre-processing: The lidar data is pre-processed to remove noise and outliers and to generate a Digital Terrain Model (DTM),
2. Cloth simulation: The canopy is represented as a virtual cloth made up of a grid of nodes. The cloth simulation algorithm simulates the movement and deformation of the cloth by applying forces to each node based on its position relative to the lidar data points. The forces include gravity, tension, and collision forces. The tension force keeps the cloth taut, while the collision force prevents the cloth from penetrating the lidar data points,
3. Height estimation: Once the cloth simulation is complete, the height of the canopy surface at each location can be estimated by interpolating the heights of the nodes in the virtual cloth,
4. Calibration: The cloth simulation algorithm's parameters are calibrated using a set of ground-truth data to ensure that the simulated canopy surface matches the actual canopy surface,
5. CHM generation: The final CHM is generated by subtracting the DTM from the estimated canopy surface height at each location [53].

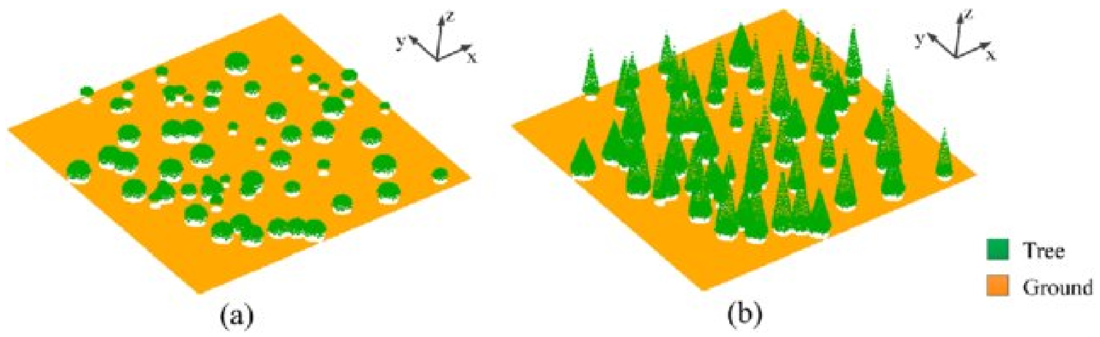


Figure 6.5: Examples of segmented 3D lidar point cloud maps using the CSF algorithm [53]

6.3 SLAM Algorithms Performance Metric

As was mentioned in Chapters 4 and 5, a multitude of different SLAM algorithms will be extensively tested on 3 different datasets, in order to determine which one is the most suitable for the desired application. In order to compare the algorithms, some comparison criteria need to be introduced. The list of criteria is presented in the following list in no particular order. More complex criteria are explained later.

- Visual inspection
- Map accuracy
- Map resolution - determined by the number of points in the 3D point cloud map
- Loop-closure availability - does a SLAM algorithm support loop-closure or not
- Playback factor
- Implementation difficulty

Visual inspection

Visual inspection is used to verify that the SLAM process has been completed successfully. It can detect anomalies in the map, like artefacts, that are not supposed to be there, or are missing, failed loop-closure, skewed or in other ways distorted map, etc.

Map accuracy

Map accuracy is calculated using the M3C2 method, explained in Section 6.1, in Cloud Compare. However, before using the said method, the reference point cloud, obtained from the ”ground truth” dataset, and the SLAM map need to be aligned. This is done by importing both of the point clouds into Cloud Compare and roughly aligning them by hand. This rough alignment is not precise and may still contain misalignment in terms of both rotation and translation. Afterwards, a cloud registration feature of Cloud Compare is used to finely register the two point clouds, thus aligning them as close as possible. This fine point cloud registration method is based on the ICP algorithm and offers the user to set some constraints, for example, rotations around some axes or translations along some axes can be blocked. Furthermore, the desired percentage of overlap between two aligned point clouds can be set, as well as the desired number of ICP iterations, or the Root Mean Square (RMS) difference. The number of iterations or the RMS difference can be used as a successful alignment criterion. After this alignment is done, the M3C2 method is used to calculate the distance between the two point clouds. M3C2 parameters, such as the diameter of the cylinder and the diameter around the core points, are automatically calculated by Cloud Compare. Since the reference point cloud is denser than the one obtained by the SLAM algorithm, it is subsampled at a certain scale to reduce the computational load of the procedure itself. Points of the subsampled reference point cloud are considered as core points. The output of this procedure is a coloured point cloud of distances between the SLAM map and the reference map, and a histogram classifying the points of the SLAM map according to their distance to the reference point cloud. The coloured point cloud is used to visually examine where the discrepancies appear and try to determine why. It is also used to visualise the results. On the other hand, the histogram is used to provide a numerical value of the SLAM map accuracy, by determining the percentage of points that satisfy a certain, predetermined, threshold, which can vary from application to application. One nice feature of Cloud Compare is that it stores the exact distance of each point, so there is a possibility to increase the resolution of the histogram by increasing the number of classification classes.

Playback factor

Some algorithms might not be able to run in real-time due to the computational demands of the SLAM process and restrictions in the hardware specifications. By playing back the rosbag containing the lidar data at a slower rate, for instance, 0.5 or 0.1, this issue is lessened. It is equivalent to using a higher-performance post-processing setup, and playing the rosbag in real time.

Implementation difficulty

This is a criterion that aims to measure how easy it is to use a specific SLAM algorithm. It aims to give judgement on both the ease of setting up the algorithm, as well as the later configuration of the algorithm to fit the specific use case.

These criteria have been chosen among many since it suits the work done in this thesis, which focuses on the map quality, the best. There exists many more, mainstream, criteria to judge the performance of a SLAM algorithm, as is stated by Valentin in [54]. Some of the other criteria mentioned by Valentin include Absolute Pose Error (APE), Relative Translation Error (RTE), and Relative Pose Error (RPE). Valentin also mentioned Average Distance to the Nearest Neighbor (ADNN) as a criterion. Measuring the map accuracy using the M3C2 method is just an improvement of the ADNN method used by Valentin. Finally, it is worth noting that it is proven, by Valentin, that all of these aforementioned criteria are heavily correlated, so there is no need to take all of them into consideration and choosing just one is sufficient.

6.4 Chosen SLAM Algorithms

As already mentioned in Chapter 3, there are a lot of different SLAM algorithms. To choose suitable algorithms for testing, multiple sources were considered. The basis for choosing the algorithms was a list of open-source SLAM algorithms, that are implemented in ROS, available on Autoware website [55]. Similar to the Autoware website, both Garigipati et al. [56] and Valentin [54], also, only considered algorithms available in ROS. Based on the information available in the aforementioned resources, a list of eight candidate algorithms is created. These algorithms, alongside their main features, are displayed in Table 6.1. Only the FD-SLAM algorithm is not tested, since it requires GNSS data, which might not be always available, to run. Out of the other seven algorithms, FAST-LIO-LC, ISCLOAM, and A-LOAM are unable to work with the post-processing setup used in this thesis, introduced in Section 5.1.5. Most likely, the reason for this is that they are created for an older version of ROS.

	Required sensors	Optional sensors	ROS version	Newest ROS release	Loop-closure	Tested version	Repository stars	Latest update
FAST-LIO-LC	Lidar & IMU	GNSS	ROS 1	ROS Melodic	NO	d059255	191	11/03/22
FD-SLAM	Lidar & GNSS	IMU	ROS 1	ROS Noetic	YES	-	13	13/04/22
HDL-Graph-SLAM	**Lidar**	**IMU & GNSS**	**ROS 1**	**ROS Noetic**	**YES**	**cb7af42**	**1600**	**28/02/23**
ISCLOAM	Lidar	-	ROS 1	ROS Melodic	YES	d6d7c61	458	13/03/21
LeGO-LOAM	**Lidar**	**IMU**	**ROS 1**	**ROS Melodic**	**YES**	**896a7a9**	**2000**	**02/07/20**
LIO-SAM	**Lidar & IMU**	**GNSS**	**ROS 1 & ROS 2**	**ROS Noetic & ROS Humble**	**YES**	**0be1fbe**	**2500**	**17/04/23**
KISS-ICP	**Lidar**	**-**	**ROS 1 & ROS 2**	**ROS Noetic & ROS Humble**	**NO**	**959e507**	**883**	**06/06/23**
A-LOAM	Lidar	-	ROS 1	ROS Melodic	NO	e51f88c	1700	28/03/19

Table 6.1: Considered SLAM algorithms

In the end, four, frequently used, algorithms, are selected. They are KISS-ICP, LeGO-LOAM, HDL-Graph-SLAM, and LIO-SAM. Figure 6.6 shows a quick overview of the selected SLAM algorithms, while each one of them is explained later in this section. The overall goal is to strike a balance between the quality of the 3D map and the speed and simplicity of its construction. To achieve this, algorithms based on different technologies, are implemented. Two different types of SLAM are visible in the figure, full SLAM, and odometry and mapping. The main difference between odometry and full SLAM approaches is that odometry estimates position incrementally, frame-by-frame, while full SLAM approaches aim to maintain global consistency by detecting revisited places and correcting pose estimate errors through loop-closure detection.

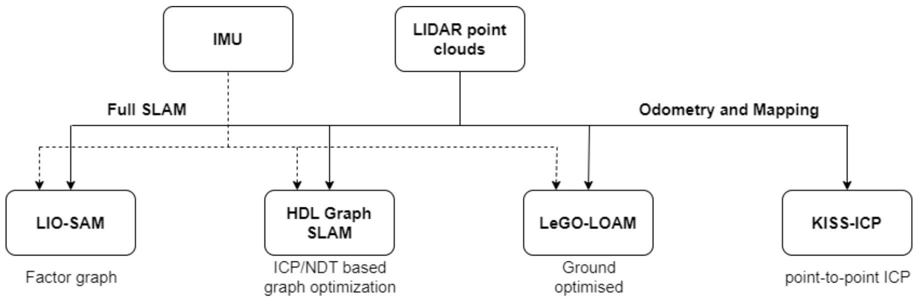


Figure 6.6: Selected SLAM algorithms and required input data (adapted from Garigipati et al. [56])

6.4.1 KISS-ICP

As stated by the creators of this algorithm, Vizzo et al [40], the proposed approach involves four main steps for registering point clouds in the context of mobile robots. Firstly, motion prediction and scan deskewing are performed using a constant velocity model to estimate the robot's motion and compensate for it when processing 3D scans. Secondly, point cloud subsampling is carried out using a voxel grid to reduce the number of points processed during registration. Thirdly, a local map is constructed using the subsampled point cloud, and correspondence estimation is performed using ICP with a distance threshold that depends on the expected initial pose error and sensor noise. Finally, alignment is achieved through robust optimisation, minimising point-to-point residuals until convergence [40].

By rethinking point cloud registration in this way, the proposed approach provides a fast and effective registration process that is well-suited for mobile robots operating in dynamic environments. The use of a constant velocity model and voxel grids allows for accurate motion compensation and efficient processing of point clouds, while the reliance on classic point-to-point ICP and robust optimisation enable accurate alignment. These steps are particularly useful for mobile robots that need to register point clouds in real time while navigating through complex and dynamic environments. This implementation uses a compact set of seven parameters, consisting of two parameters for correspondence search, four for map representation and scan subsampling, and one for ICP termination [40].

6.4.2 LeGO-LOAM

Shan and Englot [57] explain that the proposed framework is a 6 Degrees of Freedom (DoFs) pose estimation system that takes input from a 3D lidar and is divided into five modules. The first module is segmentation, where a point cloud is projected onto a range image, and ground points are extracted. In the second module, feature extraction, edge and planar features are selected from each row of the range image using roughness values. The third module is lidar odometry, which estimates sensor motion between consecutive scans by performing point-to-edge and point-to-plane scan matching. Correspondences between features in consecutive scans are found by looking for points with the same label in both scans, and the Levenberg-Marquardt (L-M) method is used to find the minimum distance transformation between two consecutive scans [57].

The fourth module is lidar mapping, which matches features in the point cloud map to the pose transformation using the L-M method. The map can be obtained by choosing the feature sets that are in the FoV of the sensor or by fusing the selected feature sets. The fifth module is transform integration, which can integrate pose-graph SLAM by adding spatial constraints between a new node and the chosen nodes using the transformations obtained after L-M optimisation [57].

Overall, this framework uses a range of techniques such as segmentation, feature extraction, lidar odometry, lidar mapping, and transform integration to provide accurate and reliable 6 DoF pose estimation. It provides a comprehensive solution for estimating the pose of objects in complex environments using 3D lidar sensors [57].

6.4.3 HDL-Graph-SLAM

Creators of HDL-Graph-SLAM, Koeide et al. [58], state that it is a real-time SLAM algorithm used for 3D laser scanners. It utilizes a 3D graph SLAM with an NDT-like scan matching process to determine the trajectory of the sensor, making it suitable for six DoFs. However, some other algorithms, like ICP, can be used for the scan matching process. Other sensor inputs, such as IMU or GNSS, as shown in Table 6.1, can be used as boundary conditions for trajectory determination. HDL-Graph-SLAM is made up of four nodes, namely *prefiltering*, *scan_matching_odometry*, *floor_detection*, and *hdl_graph_slam* [58].

In the *prefiltering* node, laser scan data is pre-processed by removing measurement points that are too close or too far away. Outliers can also be removed using either a radius method or a statistical method. In the *scan_matching_odometry* node, scan matching is performed using filtered points to estimate the sensor pose and determine the robot's trajectory. The *floor_detection* node is used to optimise the pose graph by taking into account the detected floor area, assuming that all floor surfaces in the scan lie on the same plane. RANdom SAmple Consensus (RANSAC) is used to estimate the ground plane, and a minimum number of points and plane angle threshold must be met for an estimated plane to be accepted [58].

Finally, the estimated odometry and floor areas are forwarded to the *hdl_graph_slam* node, which optimises the trajectory and outputs a registered 3D point map. Additional boundary conditions such as IMU data and GNSS can also be considered in this node [58].

6.4.4 LIO-SAM

The proposed system utilises a 3D lidar, an IMU, and, optionally, a GNSS, to estimate the robot's state and trajectory, using a factor graph to model the problem and solve a nonlinear least squares problem, as stated by its creators Shan et al. [6]. The factor graph comprises four types of factors: IMU pre-integration factors, lidar odometry factors, GNSS factors, and loop-closure factors. The system optimises the factor graph using incremental smoothing and mapping with the Bayes tree upon the insertion of a new node, helping maintain a sparse factor graph [6].

The IMU pre-integration factor deals with IMU measurements that are affected by slowly varying bias and white noise. The relative body motion, between two time steps, can be computed using the IMU pre-integration method. The lidar odometry factor involves feature extraction on a lidar scan, composing a lidar frame, by combining the extracted edge and planar features, and creating a voxel map from a sliding window approach. The system adopts the concept of key frame selection, which adds a new lidar frame to the factor graph when the change in robot pose exceeds a user-defined threshold [6].

The GNSS factor helps eliminate drift during long-duration navigation tasks by introducing sensors that offer absolute measurements. Finally, the loop-closure factor proposes a Euclidean distance-based loop-closure detection approach that is compatible with other methods. The system searches for prior states that are close to the new state in Euclidean space, tries to match them using scan matching, and adds loop-closure factors that can correct altitude errors approaching $100\,\mathrm{m}$ when GNSS is the only absolute sensor available. Overall, the proposed system combines different types of factors and techniques to achieve accurate state estimation and mapping for a robot [6].

6.5 Performance Optimisation

From the comparison results of the chosen SLAM algorithms which are introduced in Section 7.1 (the algorithms are introduced in Section 6.4), based on the criteria introduced in Section 6.3, it appears that the optimal SLAM algorithm, for the use cases presented in this thesis, is LIO-SAM. This section introduces three LIO-SAM improvement approaches used in this thesis.

Since the results, as well as the authors of the algorithm themselves, indicate that the loop-closure method implemented in the base LIO-SAM is not the best, improving it is the first step in the quest to enhance the algorithm's performance. More details on how is this done are given in Section 6.5.1.

Another option for improving the SLAM mapping results is including the available GNSS data in the map-making process itself. By default, LIO-SAM is relying only on IMU data for estimating the trajectory of the sensor carrier, thus the initial pose estimation in each frame, as well. Once the GNSS data is introduced, an accurate location of the sensor carrier is available, making the initial pose estimate more accurate. More details on the GNSS integration are provided in Section 6.5.2.

Finally, the last approach to increasing the LIO-SAM performance is the tuning of the algorithm's parameters. LIO-SAM uses more than 30 user-defined parameters, some of them should not be modified, while others can be modified, to achieve different performance. Section 6.5.3 provides additional details on the parameters that can be modified and are actually modified.

6.5.1 Loop-closure Improvement - SC-LIO-SAM

SC-LIO-SAM is an upgrade of the original LIO-SAM algorithm, by integrating it with another algorithm called Scan Context, which is developed by G. Kim and A. Kim [59]. Scan Context improves the loop detection capabilities of the original LIO-SAM algorithm, thus, also, its loop-closure capabilities. The specifics of the Scan Context algorithm are explained later, in the following paragraphs. Since the original SC-LIO-SAM repository on GitHub is no longer updated (the last update was on 19th of July 2021), and the current version (d43ca00) is not up to date with the original LIO-SAM algorithm, the original SC-LIO-SAM repository has been forked and the updates to the LIO-SAM part of the algorithm made, in the scope of this thesis. The updated SC-LIO-SAM is open-source and available on GitHub [60].

Scan Context

Place recognition is a critical component of various robotics applications, and recent advancements in SLAM have provided dense 3D maps of environments. While existing methods rely on diverse feature detectors and descriptors for visual scenes, incorporating structural information to describe a place remains relatively underexplored. The Scan Context method introduces a novel spatial descriptor for global place recognition using 3D lidar scans [59].

The Scan Context method directly captures the 3D structure of the environment from lidar scans without relying on histograms or prior training. It partitions the 3D scan into azimuthal and radial regions, leveraging the scan's centre as a global key point. This approach provides a concise representation of the vertical shape of surrounding structures, enabling efficient place recognition. To construct the Scan Context, the point cloud is divided into mutually exclusive regions, based on a regular grid. Each region is assigned a value representing the maximum height of points within it. This process summarizes the 3D structure of the environment, without using explicit bins or matrices [59].

To ensure robust recognition against translation, the proposed method augments the Scan Context through root shifting. By storing Scan Contexts from neighbouring locations, it assumes that similar point clouds can be obtained even after translation, improving recognition robustness [59].

A cosine distance, between the column vectors at the same index, is used to compute the similarity score between two Scan Contexts. However, since lidar viewpoint changes for different places, column shifting can occur, even within the same location. To address this, distances are computed by comparing all possible column-shifted Scan Contexts, ensuring robustness to viewpoint changes [59].

Scan Context introduces a two-phase search algorithm for efficient loop closure detection. It combines pairwise similarity scoring and hierarchical nearest neighbour search. A rotation-invariant descriptor, called the ”ring key,” is extracted from the Scan Context to enable fast searching for potential loop closure candidates. The closest candidate satisfying a threshold is selected as the revisited place [59].

6.5.2 GNSS Data Inclusion

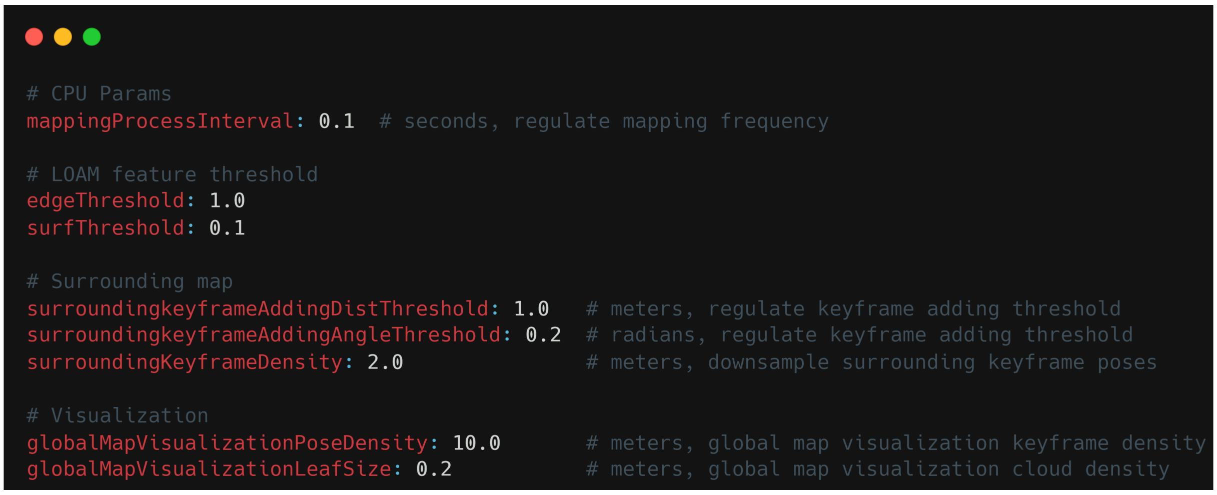
Including GNSS data in the SLAM process can yield several advantages. Firstly, GNSS provides global position information, typically in latitude and longitude coordinates. Integrating this data into SLAM aids in establishing an initial global frame of reference for the map. This enables the robot to determine its starting position within the map and align subsequent measurements to the correct global coordinates. Moreover, SLAM algorithms often require an initial estimate of the robot's pose to initiate mapping and localization. GNSS data can effectively serve as an initialization step by providing an approximate position estimate. This allows the SLAM system to commence mapping from a known location, thereby enhancing the efficiency and accuracy of the algorithms. Furthermore, SLAM algorithms can suffer from drift over time due to sensor noise and accumulated errors, affecting the accuracy of pose estimation in local coordinates. GNSS data can serve as a global reference to correct this drift and periodically relocalize the robot. By periodically integrating GNSS data with SLAM estimates, the robot's pose can be refined and aligned with the actual global position, ensuring long-term localization accuracy [61].

Additionally, the inclusion of GNSS data can improve the accuracy of the generated maps. By combining GNSS measurements with other sensor data, such as lidar scans, SLAM algorithms can effectively reduce uncertainties and enhance the overall quality of the maps. GNSS data can act as a constraint to align local maps with the global reference frame, resulting in more precise and consistent maps. Moreover, GNSS measurements can assist in addressing data association challenges encountered during SLAM. In scenarios with limited visual features or ambiguous sensor measurements, GNSS data can provide valuable information for associating sensor readings with specific locations. This aids in correctly aligning sensor measurements with the corresponding map elements, thus improving the overall robustness of the SLAM system [61].

In summary, the integration of GNSS data into the SLAM process should offer various benefits, including establishing a global frame of reference, improving initialization, ensuring long-term localization accuracy, enhancing mapping precision, and assisting in data association. By leveraging GNSS data alongside other sensor inputs, SLAM systems can achieve more accurate and reliable mapping and localization results. Lastly, it needs to be noted that both LIO-SAM and SC-LIO-SAM come with included GNSS data integration capabilities, as can be seen in Table 6.1. Enabling this functionality is a straightforward process that involves activating two specific features and adjusting the values of two parameters within the algorithm's configuration settings.

6.5.3 Parameter Optimisation

Since LIO-SAM uses more than 30 user-defined parameters, a selection had to be made, to choose the ones that are going to be optimised to increase the accuracy of the final algorithm's output, which is the generated 3D point cloud map. Unfortunately, the documentation and information on the purpose of each parameter are extremely limited, with only some brief, one-sentence explanations, or even worse, no explanation at all. This made it quite difficult to identify which parameters to modify. Finally, eight different parameters were selected and divided into four groups: i) *CPU parameters*; ii) *LOAM feature threshold parameters*; iii) *Surrounding map parameters*, and iv) *Visualisation parameters*. These four groups contain parameters whose influence on the mapping results should be directly proportional, i.e. the effect of increasing the value of one, goes together with the effect of increasing the values of other parameters. Furthermore, these groups are also suggested by the Shan et al. [6], since they divided the parameters into the same groups in the configuration file. The first group of parameters determines the mapping frequency. The second group deals with the thresholds for the LOAM features since LIO-SAM uses parts of the LOAM algorithm. The third group of parameters deals with the thresholds for adding key frames, as well as the key frame density. Finally, the last group, called Visualisation parameters, determines the finished 3D global point cloud map density. Figure 6.7 displays all four groups of parameters, together with the parameters themselves, and the explanation of each parameter, provided by the algorithm's authors.



```
# CPU Params
mappingProcessInterval: 0.1  # seconds, regulate mapping frequency

# LOAM feature threshold
edgeThreshold: 1.0
surfThreshold: 0.1

# Surrounding map
surroundingkeyframeAddingDistThreshold: 1.0  # meters, regulate keyframe adding threshold
surroundingkeyframeAddingAngleThreshold: 0.2 # radians, regulate keyframe adding threshold
surroundingKeyframeDensity: 2.0              # meters, downsample surrounding keyframe poses

# Visualization
globalMapVisualizationPoseDensity: 10.0      # meters, global map visualization keyframe density
globalMapVisualizationLeafSize: 0.2          # meters, global map visualization cloud density
```

Figure 6.7: SC-LIO-SAM - selected configurable parameters

To determine whether the accuracy of the finished 3D map is increased, each group of parameters is individually tested and evaluated. The default values of parameters in each group are both increased and decreased and the finished map is saved for each different configuration. The accuracy of each saved map is then determined and a judgment is made, whether tuning that set of parameters is affecting the accuracy of the map. Finally, once it is determined which parameter groups have the potential to increase the accuracy, the best-performing values of parameters in those groups are combined and tuned to achieve the best possible accuracy.

7 Results

Contents

This chapter presents and discusses the results acquired by generating 3D point cloud maps using the SLAM algorithms presented in Chapter 6. Section 7.1 introduces the comparison results of all the chosen SLAM algorithms. Section 7.2 presents the results of performance optimisation attempts performed on the best performing SLAM algorithm, as determined in Section 7.1.

7.1 SLAM Algorithms Comparison Results

Initially, some general characteristics of the chosen SLAM algorithms, as well as the implementation and usage difficulty, are analysed. They are displayed in Table 7.1.

	Loop-closure	**Installation difficulty**	**Usage difficulty**
KISS-ICP	NO	Medium	Medium
HDL-Graph-SLAM	**YES**	Easy	Medium
LeGO-LOAM	**YES**	Hard	Hard
LIO-SAM	**YES**	Medium	Easy

Table 7.1: Chosen SLAM algorithm characteristics

It can be noted that only KISS-ICP does not have some form of loop-closure implemented. LeGO-LOAM is, in general, the hardest algorithm to both, implement and use. The latest version of ROS it supports is ROS Melodic, which reached its end of life in April of 2023, so implementation in ROS Noetic required substantial modifications to the source code. In the end, the map saving functionality did not function properly, so no 3D point cloud maps were saved. However, it did not affect the quality of the comparison, which will be shown later. Usage of LeGO-LOAM also proved to be very complex, since all the configurable parameters are contained inside a C++ header file, and any changes made to it require the rebuilding of the entire ROS workspace, which is highly inconvenient. The most straightforward installation procedure is with the HDL-Graph-SLAM, where only the instructions need to be followed. However, usage of HDL-Graph-SLAM is difficult and not intuitive, since it has more than 150 changeable parameters, whose function is hardly explained. Finally, both KISS-ICP and LIO-SAM require some minor modifications to the source code in order to install them. On the other hand, the usage of KISS-ICP is a bit more complex. Configuring the parameters is simple since there are only seven of them, but saving the map is rather difficult, requiring the creation of a rosbag file with all the output point clouds, and then taking the last point cloud and converting it into a file format suitable for a 3D point cloud map. Unlike KISS-ICP, usage of LIO-SAM is rather straightforward, despite it having more than 30 configurable parameters, since the effects of changing most of them are somewhat documented. Furthermore, the 3D point cloud map is saved using a single command.

Table 7.2 shows the results for each algorithm, tested on every dataset. Firstly, it is important to note that all the runs were performed with the rosbag playback factor of 1, i.e. in real-time. This was done to test the computational requirements of each algorithm, as well as to ensure the same, fair conditions throughout the comparison. As can be seen in Table 7.2, LIO-SAM algorithm consistently provides 3D SLAM generated point cloud maps with at least two times (2x) better resolution than any of the other SLAM algorithms tested. Finally, Table 7.2 provides an insight into the results of applying the ”visual inspection” criterion to 3D point cloud maps generated by each one of the tested SLAM algorithms for each dataset. It can be noted that only LIO-SAM was able to generate maps of every dataset, that are deemed successful, as per the criteria. More details about the performance of each SLAM algorithm, on each dataset, are given later in this section.

		Map resolution (points)	Playback factor	Visual inspection
Rijeka Harbour	KISS-ICP	2,190,438	1	Fail
	HDL-Graph-SLAM	1,662,611	1	Fail
	LeGO-LOAM	-	1	Fail
	LIO-SAM	**5,372,780**	1	**Fail/Success**
ViF Building	KISS-ICP	640,608	1	**Success**
	HDL-Graph-SLAM	372,573	1	**Success**
	LeGO-LOAM	-	1	Fail
	LIO-SAM	**1,248,135**	1	**Success**
Sonnblick Observatory	KISS-ICP	1,316,906	1	**Success**
	HDL-Graph-SLAM	1,711,318	1	Fail
	LeGO-LOAM	-	1	Fail
	LIO-SAM	**11,132,938**	1	**Success**

Table 7.2: General SLAM algorithm comparison results

Rijeka Harbour dataset

As already mentioned in Section 5.4, the Rijeka Harbour dataset is a marine environment dataset collected in Croatia. One of its interesting features is the existence of big spikes in the IMU data during the initial part of the dataset. These spikes later contributed to the failure, of some tested SLAM algorithms, to successfully generate 3D point cloud maps of the surveyed area.

Figure 7.1 shows maps generated by the LeGO-LOAM and the KISS-ICP algorithms. It can be seen that the LeGO-LOAM mapping process failed rather quickly, without even managing to map the entire survey area at all. On the other hand, the KISS-ICP mapping process was completed. However, the inexistence of a loop-closure algorithm within the KISS-ICP algorithm is visible, since the map is distorted and parts of it intersect with each other. Due to these facts, both algorithms are considered to have failed the mapping process for this particular dataset, as per the ”visual inspection” criterion.

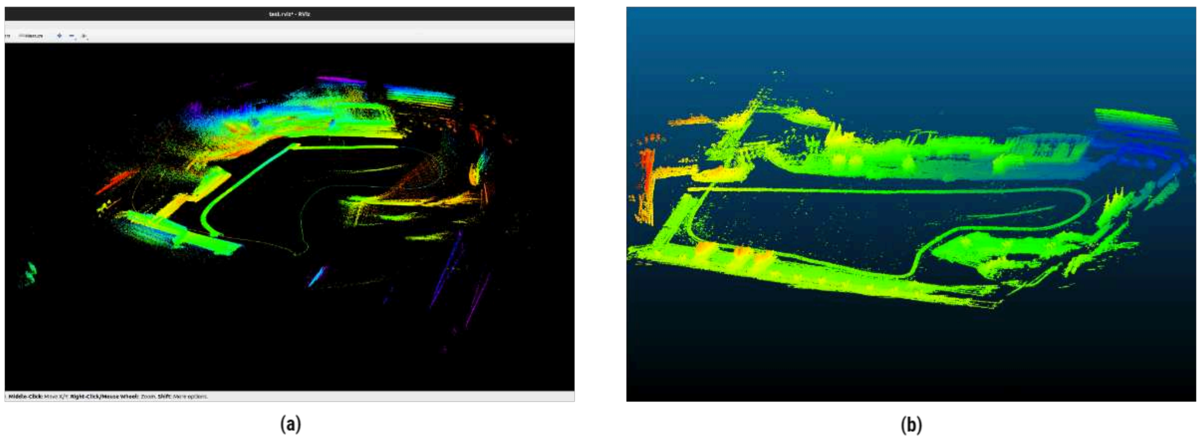


Figure 7.1: Rijeka Harbour dataset: (a) LeGo-LOAM generated map; (b) KISS-ICP generated map

Figure 7.2 shows 3D maps that are generated using the HDL-Graph-SLAM and the LIO-SAM algorithms. It can be seen that the mapping process was completed when using both algorithms. However, it is also visible that the loop-closure algorithm failed to properly close the loop in both cases as well, thus some parts of the generated 3D maps are not merged properly. Because of this failure, the mapping process is considered to have not achieved complete success when using any of the two algorithms, as per the ”visual inspection” criterion.

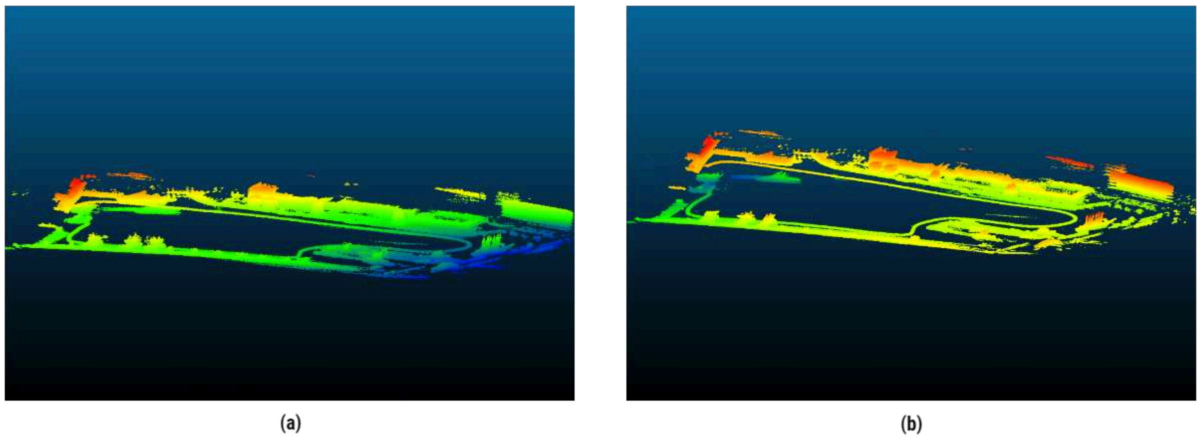


Figure 7.2: Rijeka Harbour dataset: (a) HDL-Graph-SLAM generated map; (b) LIO-SAM generated map

Since the mapping process did not fully succeed when using any of the tested algorithms, and considering the fact that the initial few seconds of recorded IMU data contain large and unnatural spikes, which can be considered as noise, another experiment was run. The dataset was modified by removing the noisy initial data, both lidar and IMU, all the way up to the point when the boat carrying the surveying equipment started moving. This modification resulted in the LIO-SAM algorithm managing to successfully generate a 3D map of the said dataset, as is shown in Figure 7.3. However, modifying the data in this way might not be possible for all future applications, so some, proposed ways of mitigating the effect of noisy initial data, are presented in Section 6.5 and the results of their implementation in Section 7.2.

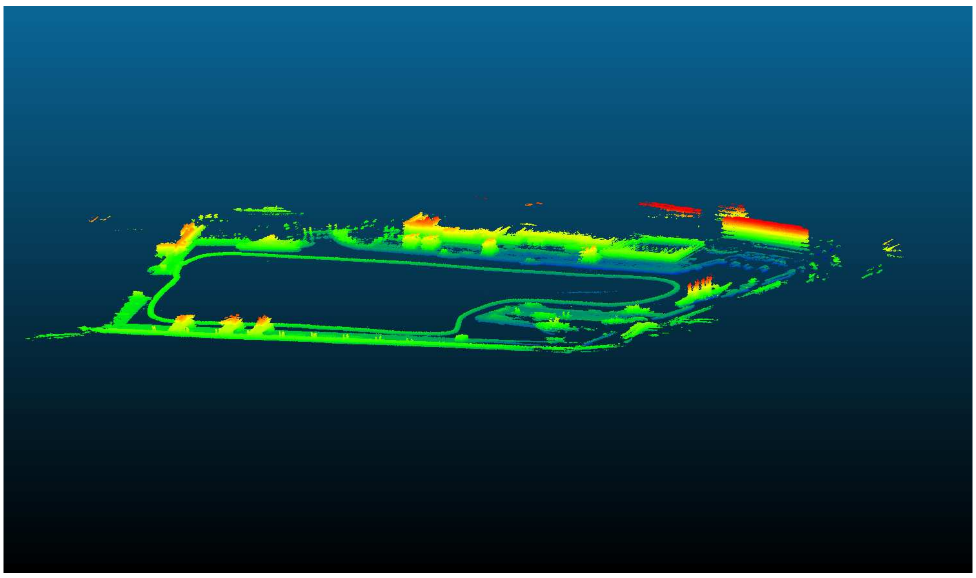


Figure 7.3: Successful LIO-SAM generated map - Rijeka Harbour dataset

ViF Building dataset

As already mentioned in Section 5.4, the ViF building dataset is an urban environment dataset collected in Graz, Austria. This dataset is supposed to be the simplest one, used in this thesis, for the SLAM algorithms to successfully generate a 3D map. Figure 7.4 shows maps generated by the LeGO-LOAM and the KISS-ICP algorithms. It can be seen that the LeGO-LOAM mapping process was completed. However, the loop-closure feature failed completely, leaving the map distorted in a way that one of the building's outside walls is piercing the building itself. In spite of the KISS-ICP generated map looking good at first sight, a closer look reveals that it is distorted and some features are doubled since there is no loop-closure feature implemented. Despite the aforementioned issues, the KISS-ICP algorithm successfully completed the mapping process, according to the "visual inspection" criterion, unlike LeGO-LOAM.

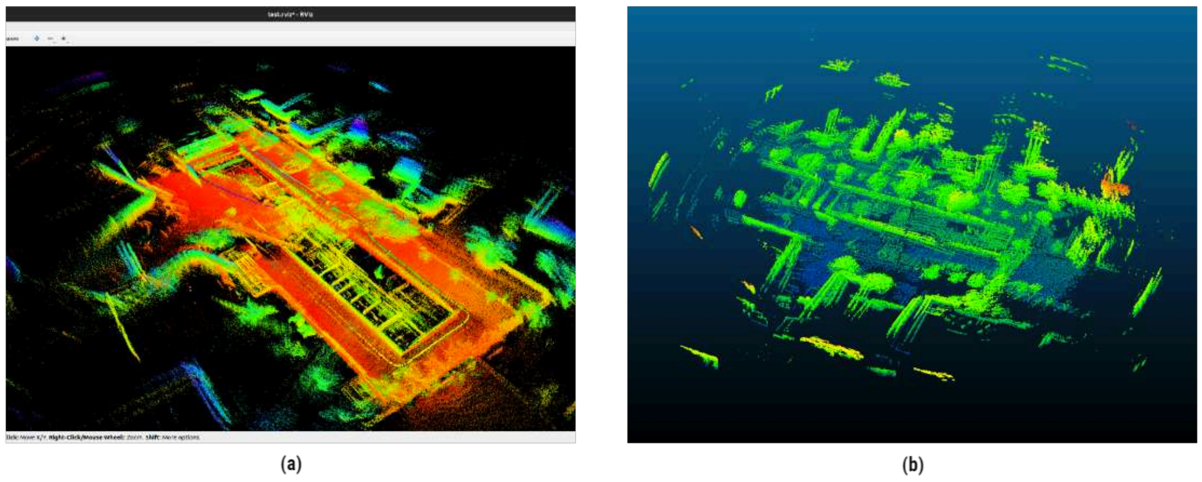


Figure 7.4: ViF Building dataset: (a) LeGo-LOAM generated map; (b) KISS-ICP generated map

As can be seen in Figure 7.5, both HDL-Graph-SLAM and LIO-SAM algorithms were able to successfully generate 3D point cloud maps of the ViF building and its surroundings. It can also be noted that the loop-closure was performed reasonably well by both algorithms, so the mapping process is considered to have been completed successfully by both algorithms.

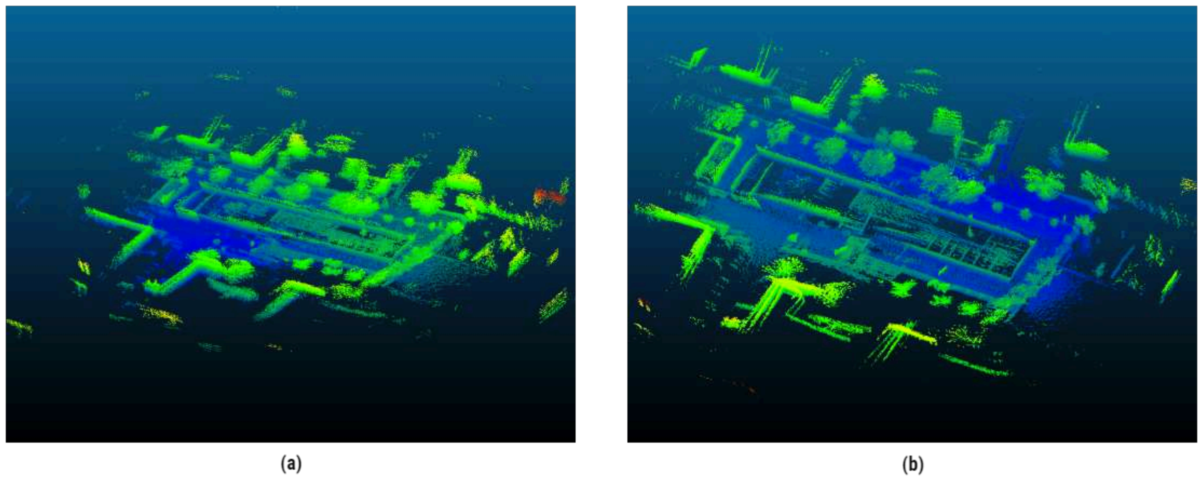


Figure 7.5: ViF Building dataset: (a) HDL-Graph-SLAM generated map; (b) LIO-SAM generated map

Sonnblick Observatory dataset

Out of the three datasets used in this thesis, the Sonnblick Observatory dataset is the only one to contain the ”ground-truth” data. It was gathered in the Rauris Valley, Austria, in a snowy environment, and as a sensor carrier, a gondola was used, as previously explained in Section 5.4. The snowy environment makes this dataset particularly challenging for the lidar sensor and the SLAM algorithms, due to the absorbent nature of snow at the lidar sensor operating wavelengths, as well as the reduced number of features in the environment, due to the snowfall.

Figure 7.6 shows maps generated by the LeGO-LOAM and the KISS-ICP algorithms. It can be seen that the LeGO-LOAM mapping process failed completely, without even managing to produce a plausible map of the surveyed area. On the other hand, KISS-ICP mapping process was completed and it yielded a plausible 3D point cloud map of the surveyed area, which is later tested to determine its accuracy. Due to these facts, LeGO-LOAM is considered to have failed the mapping process for this dataset, while the KISS-ICP has succeeded, as per the ”visual inspection” criterion.

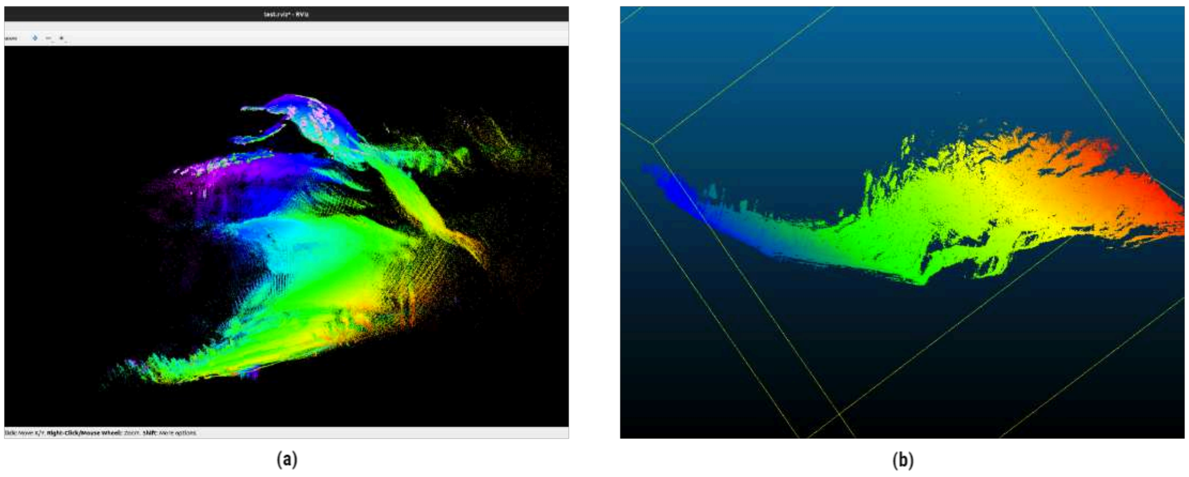


Figure 7.6: Sonnblick Observatory dataset: (a) LeGo-LOAM generated map; (b) KISS-ICP generated map

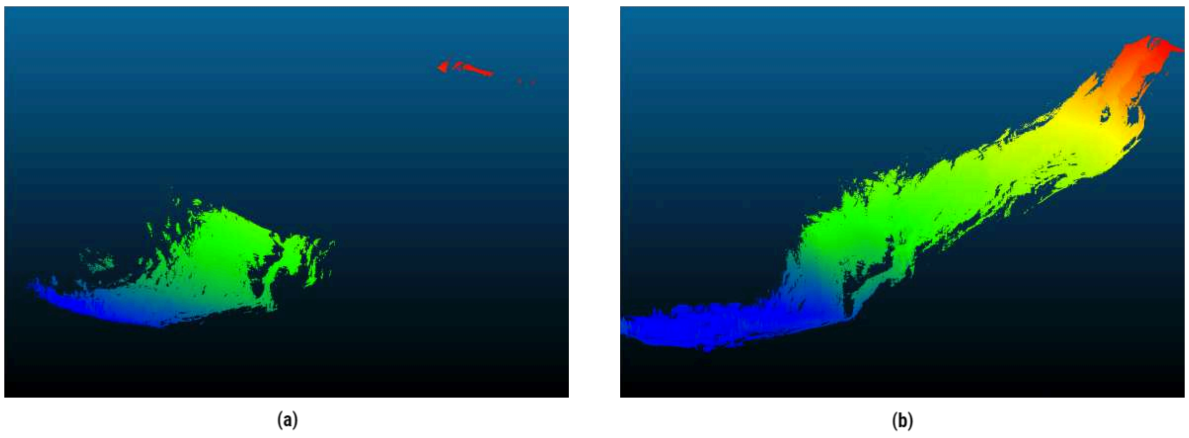


Figure 7.7: Sonnblick Observatory dataset: (a) HDL-Graph-SLAM generated map; (b) LIO-SAM generated map

As can be observed in Figure 7.7, the HDL-Graph-SLAM algorithm failed to produce a 3D of the full surveyed area. It was only successful to map the lower part of the environment, near the start of the survey. However, the LIO-SAM algorithm was able to successfully generate a 3D point cloud map of the surveyed area. Even though the map generated by the HDL-Graph-SLAM algorithm is not complete, it is tested, alongside the map generated by the LIO-SAM algorithm, to determine its accuracy.

Table 7.3 presents the results of the map accuracy measurement performed on the maps generated by the KISS-ICP, HDL-Graph-SLAM, and LIO-SAM algorithms. Each number in the table represents a percentage of points in the SLAM generated 3D map, whose distance from the corresponding point in the reference map, falls within a certain range, i.e. ± 5 m, or ± 3 m, ± 2 m, or ± 1 m. For example, this means that, for a 3D map generated by the KISS-ICP algorithm, 42.9% of all points are no more than 5 m distance away from their real position, according to the reference map. Two different types of maps were considered. In the first case, the original map was used, while in the second one, the vegetation was removed from the map. The vegetation was removed in order to reduce the measurement uncertainty introduced by the existence of vegetation in the surveyed scene, as previously explained in Chapter 6. From the data, it can be seen that the LIO-SAM algorithm provides the best results in both test cases.

	Sonnblick Observatory			**Sonnblick Observatory - vegetation removed**		
	KISS-ICP	**HDG-Graph-SLAM**	**LIO-SAM**	**KISS-ICP**	**HDG-Graph-SLAM**	**LIO-SAM**
Map accuracy (+-5m)	42.90%	76.26%	**88.07%**	55.41%	31.29%	**95.13%**
Map accuracy (+-3m)	29.38%	56.83%	**81.02%**	19.54%	19.06%	**92.93%**
Map accuracy (+-2m)	20.68%	47.17%	**73.37%**	7.36%	12.93%	**89.57%**
Map accuracy (+-1m)	11.99%	28.32%	**56.48%**	3.23%	7.17%	**73.54%**

Table 7.3: Map accuracy - SLAM algorithm comparison results

Figure 7.8 explains the colour schematic used in all of the SLAM generated and reference map comparisons. In all the tests, values are saturated at -5 m for purple, and at $+5$ m for brown, i.e. any point with a distance value outside of the aforementioned range, will be coloured either purple or brown, depending on the sign of its distance to the corresponding reference point.

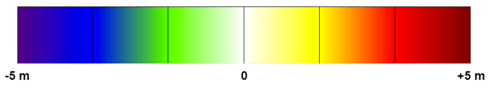


Figure 7.8: Map accuracy colorbar

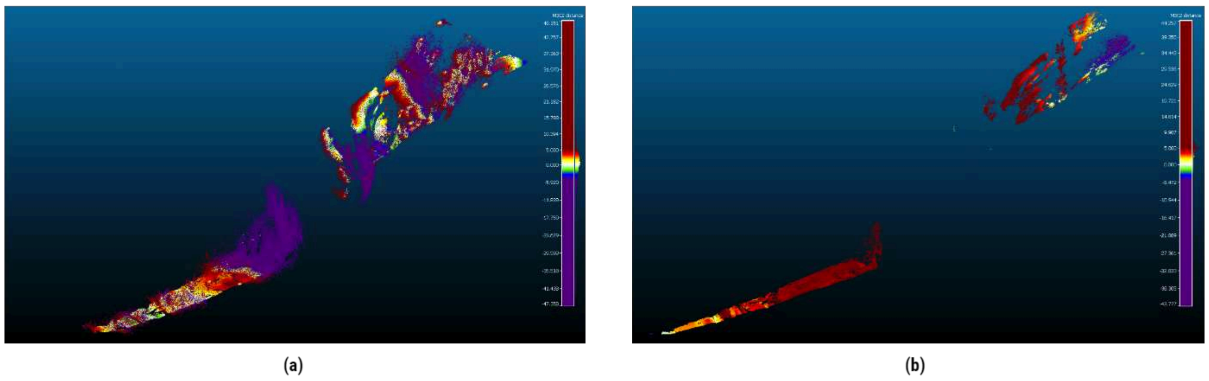
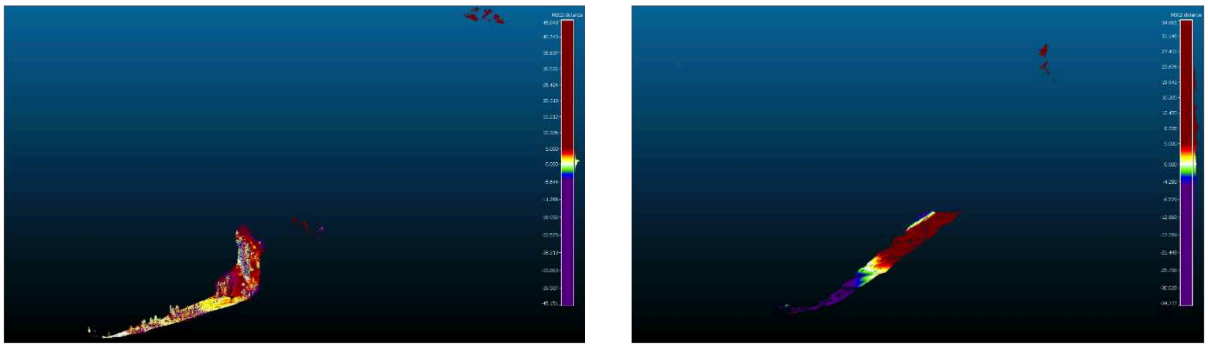


Figure 7.9: 3D SLAM generated and "ground-truth" map comparison - Sonnblick Observatory dataset - KISS-ICP algorithm: (a) Original map; (b) Map without vegetation

Figure 7.9 shows the comparison results between a 3D map generated using the KISS-ICP algorithm and the reference map. On the left, the original maps with vegetation are compared, while on the right, the maps without vegetation are compared. Based on the colour of the points it can be concluded that the KISS-ICP map accuracy is rather poor since most of the points are more than $5\,\mathrm{m}$ away from their corresponding point in the reference map.

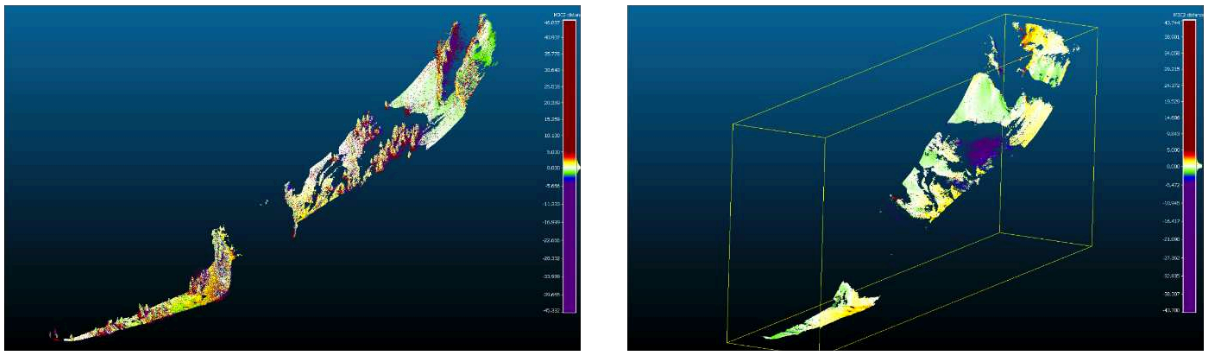
Accuracy measurement results of the map generated by HDL-Graph-SLAM algorithm are shown in Figure 7.10. Even though the results in Table 7.3 indicate that HDL-Graph-SLAM is capable of generating more accurate maps than the KISS-ICP algorithm, after examining this figure it becomes apparent that it is not true. These comparison maps just confirm the conclusion made according to the ”visual inspection” criterion: the map generation process failed when using the HDL-Graph-SLAM algorithm for this dataset.



(a) (b)

Figure 7.10: 3D SLAM generated and ”ground-truth” map comparison - Sonnblick Observatory dataset - HDL-Graph-SLAM algorithm: (a) Original map; (b) Map without vegetation

Finally, Figure 7.11 presents the comparison results between a 3D map generated using the LIO-SAM algorithm and the reference map. In the original map, with the vegetation present, it can be noted that most of the points, with a high distance to their corresponding reference points, are located in areas with a lot of vegetation. This confirms a well-known limitation of lidar sensors when it comes to surveying vegetation. Furthermore, when the comparison map, where the vegetation was removed, is analysed, it is observed that the ground itself is mapped rather accurately.



(a) (b)

Figure 7.11: 3D SLAM generated and ”ground-truth” map comparison - Sonnblick Observatory dataset - LIO-SAM algorithm: (a) Original map; (b) Map without vegetation

7.1.1 Comparison Summary

After looking at the results, based on all the criteria, and also considering the core purpose of this thesis, which is creating accurate SLAM generated 3D point cloud maps, it is safe to assume that out of all the tested algorithms, LIO-SAM is the best one. However, it also displayed some shortcomings during the testing on the three datasets used in this thesis, such as a not robust loop-closure algorithm, so some improvements of the LIO-SAM algorithm are suggested in Section 6.5, while the results of those improvements are displayed in Section 7.2.

7.2 LIO-SAM Algorithm Optimisation Results

This section presents the results of all the optimisation methods tested in the scope of this thesis. These methods are explained in Section 6.5. Only the first method, an improvement of the loop-closure capabilities, using another SLAM algorithm called SC-LIO-SAM, is tested on all three datasets. This was done to ensure that it provides the same, if not better, results compared to the original LIO-SAM algorithm. The second method, GNSS data inclusion, is tested on the Rijeka Harbour, and the Sonnblick Observatory datasets. Finally, the last method, the optimisation of certain configurable parameters, is tested on the Sonnblick Observatory dataset, because it is the only dataset with the ”ground-truth” data, making it suitable for analysing even the smallest changes that may happen, as a result of changing certain parameters.

7.2.1 Loop-closure Improvement - SC-LIO-SAM

As already shown in Section 7.1, the loop-closure segment of LIO-SAM can fail under some conditions, thus its improvement is needed. An algorithm called SC-LIO-SAM, presented in Section 6.5.1, claims to fix these issues, present in the original LIO-SAM algorithm. Figure 7.12 shows 3D point cloud maps generated by the SC-LIO-SAM algorithm of the Rijeka Harbour, and the ViF building datasets. The first thing to notice is that, unlike with the LIO-SAM algorithm, the map of Rijeka Harbour is successfully constructed. Furthermore, the map of the ViF building is also successfully constructed, making the SC-LIO-SAM algorithm successful in constructing a 3D map of the datasets that require loop-closure.

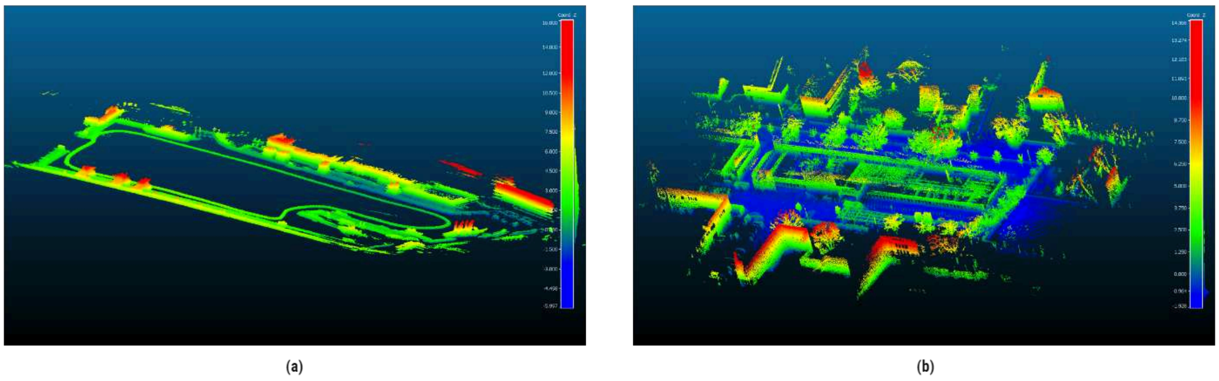


Figure 7.12: 3D SLAM generated maps by SC-LIO-SAM algorithm: (a) Rijeka Harbour dataset; (b) ViF building dataset

	LIO-SAM	SC-LIO-SAM	Relative Difference
Map accuracy (+-5m)	**88.07%**	87.86%	0.24%
Map accuracy (+-3m)	81.02%	**81.20%**	**0.22%**
Map accuracy (+-2m)	73.37%	**73.60%**	**0.31%**
Map accuracy (+-1m)	**56.48%**	55.01%	2.60%

Table 7.4: Map accuracy comparison - LIO-SAM and SC-LIO-SAM algorithms - Sonnblick Observatory dataset

Table 7.4 presents the map accuracy of the 3D SLAM generated maps by both the LIO-SAM and the SC-LIO-SAM algorithms, as well as the relative difference between the two accuracies. After analysing the values in the table, it can be concluded that the difference in the accuracy between the two maps is small. This conclusion is also confirmed by analysing the comparison results between a 3D map generated using the SC-LIO-SAM algorithm and the reference map, shown in Figure 7.13, and the same comparison results for the LIO-SAM algorithm, shown in Figure 7.11. No noticeable difference can be observed between them.

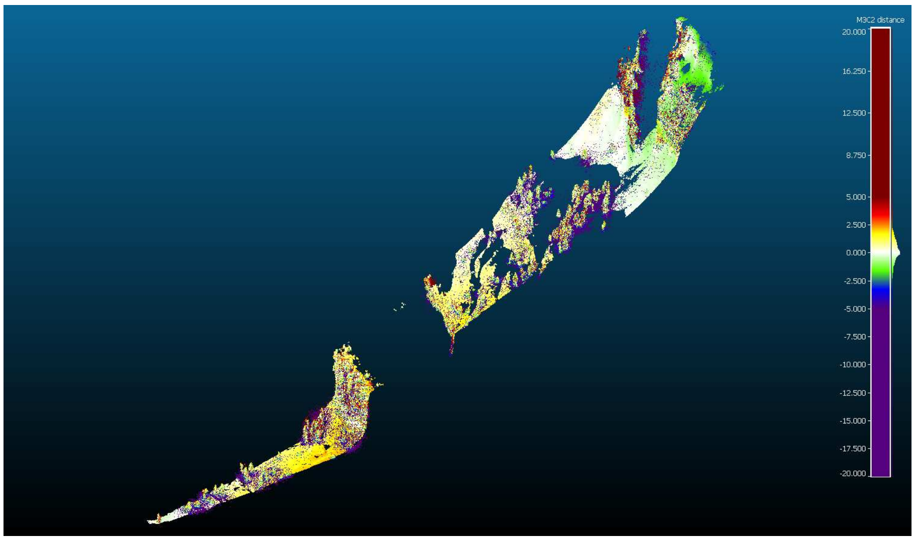


Figure 7.13: 3D SLAM generated and ”ground-truth” map comparison - SC-LIO-SAM algorithm - Sonnblick Observatory dataset

Based on the results presented in this section, it is decided that from this point onwards, the SC-LIO-SAM algorithm is tested with the remaining performance improvement methods.

7.2.2 GNSS Data Inclusion

As already mentioned in Section 6.5.2, including the data from the GNSS sensor into the SLAM process can have multiple benefits. Since LIO-SAM and SC-LIO-SAM have almost the same performance regarding map accuracy, as shown in Section 7.2.1, SC-LIO-SAM is used in this section. Table 7.5 displays the comparison results between map accuracy of maps generated by LIO-SAM and by SC-LIO-SAM with included GNSS data. From the data, it can be observed that the accuracy stays almost the same with the vegetation present. However, the accuracy improves when the maps without vegetation are compared. This is particularly true as the distance threshold reduces. Finally, based on the data presented, it can be concluded that the inclusion of GNSS data in the SLAM process can lead to increased map accuracy.

	Sonnblick Observatory			Sonnblick Observatory - vegetation removed		
	LIO-SAM	**SC-LIO-SAM-GPS**	**Relative Difference**	**LIO-SAM**	**SC-LIO-SAM-GPS**	**Relative Difference**
Map accuracy (+-5m)	**88.07%**	87.93%	0.16%	95.13%	**95.14%**	**0.01%**
Map accuracy (+-3m)	**81.02%**	80.98%	0.05%	92.93%	**93.03%**	**0.11%**
Map accuracy (+-2m)	73.37%	**73.60%**	**0.31%**	89.57%	**90.88%**	**1.46%**
Map accuracy (+-1m)	**56.48%**	54.02%	4.36%	73.54%	**77.26%**	**5.06%**

Table 7.5: Map accuracy comparison - LIO-SAM and SC-LIO-SAM GNSS algorithms - Sonnblick Observatory dataset

Another benefit of GNSS data inclusion, in the SLAM process, is that generated 3D maps have the proper compass orientation, as soon as they are generated. If no GNSS data is used, it is quite common that the maps are not oriented properly in the beginning, and they need to be oriented properly later, in the post-processing step, which is inconvenient, and in some applications, might be impossible. When the map is oriented properly from the start, it makes the process of overlaying a 3D map, over a 2D map of the same area, quite simple. Figure 7.14 shows an overlayed 3D map over a Google Maps screenshot of the Sušak Harbour in Rijeka, Croatia.

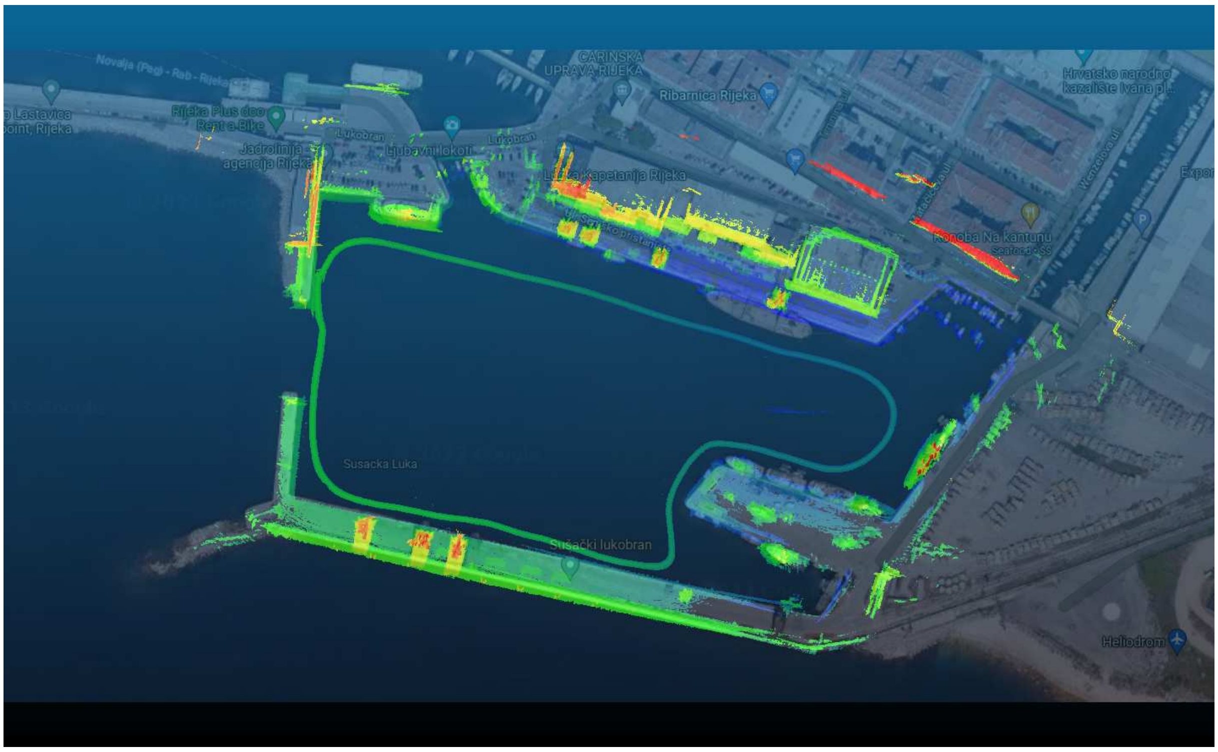


Figure 7.14: 3D SLAM generated map overlayed with Google Maps screenshot - Rijeka Harbour dataset

Similar to the previous figure, Figure 7.15 shows an overlayed 3D map over a Google Maps screenshot of the Sonnblick Observatory gondola base station in Rauris, Austria.

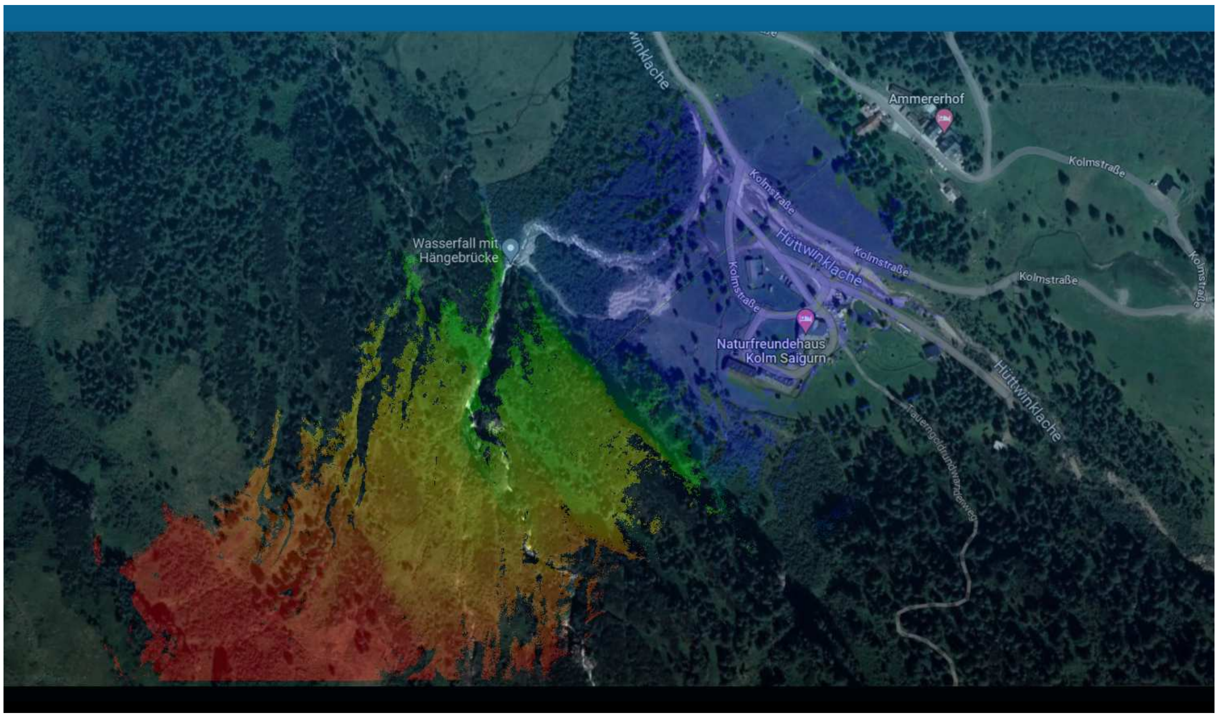


Figure 7.15: 3D SLAM generated map overlayed with Google Maps screenshot - Sonnblick Observatory dataset

7.2.3 Parameter Optimisation

As already explained in Section 6.5.3, the effects of tuning parameters in the four groups are tested individually. After that, all the best parameter values from each group, whose tuning showed an improvement in the map accuracy, are combined and tested. The tuned parameters are tested on the SC-LIO-SAM algorithm and the resulting map accuracies are compared to the accuracy values of maps generated using the original LIO-SAM algorithm. The tuning process is considered successful only if the accuracy increases for all four distance thresholds. All the individual group tests are performed on the Sonnblick Observatory dataset, with all the vegetation present.

Table 7.6 shows the map accuracies while tuning the *CPU parameters*. From the data values, appears that the default value is the optimal one, since in both cases, increasing and decreasing the parameter values, the map accuracy dropped.

	Relative difference	Decreased values	Default values	Increased values	Relative difference
Map accuracy (+-5m)	21.28%	69.33%	**88.07%**	86.69%	1.57%
Map accuracy (+-3m)	37.46%	50.67%	**81.02%**	79.00%	2.49%
Map accuracy (+-2m)	43.81%	41.23%	**73.37%**	67.26%	8.33%
Map accuracy (+-1m)	54.23%	25.85%	**56.48%**	43.92%	22.24%

Table 7.6: CPU parameters tuning - SLAM generated map accuracy results - Sonnblick Observatory dataset

Table 7.7 shows the map accuracies while tuning the *LOAM feature threshold parameters*. Similar to the previous one, the default values of parameters in this group seem to be close to optimal. Unlike the *CPU parameters* group, tuning the parameters in this group did yield some better results in certain accuracy ranges. However, for the same ”tuned” parameters, accuracy dropped substantially in another accuracy range, which made the improvements in certain ranges irrelevant.

	Relative difference	Decreased values	Default values	Increased values	Relative difference
Map accuracy (+-5m)	0.15%	87.94%	**88.07%**	87.94%	0.15%
Map accuracy (+-3m)	**0.26%**	**81.23%**	81.02%	80.97%	0.06%
Map accuracy (+-2m)	0.04%	73.34%	73.37%	**73.44%**	**0.10%**
Map accuracy (+-1m)	7.29%	52.36%	**56.48%**	51.56%	8.71%

Table 7.7: LOAM feature threshold parameters tuning - SLAM generated map accuracy results - Sonnblick Observatory dataset

Table 7.8 shows the map accuracies, while tuning the *surrounding map parameters*. Unlike with the previous two parameter groups, tuning the parameters in this group actually yielded improved map accuracy across all the ranges. The improvement is not substantial, peaking at 0.20 % compared to the values achieved by the original LIO-SAM algorithm. This implies that the default values of parameters in this group are close to the optimal values.

	Relative difference	Decreased values	Default values	Increased values	Relative difference
Map accuracy (+-5m)	**0.02%**	**88.09%**	88.07%	85.95%	2.41%
Map accuracy (+-3m)	**0.02%**	**81.04%**	81.02%	78.93%	2.58%
Map accuracy (+-2m)	**0.20%**	**73.52%**	73.37%	72.01%	1.85%
Map accuracy (+-1m)	**0.07%**	**56.52%**	56.48%	52.41%	7.21%

Table 7.8: Surrounding map parameters tuning - SLAM generated map accuracy results - Sonnblick Observatory dataset

Table 7.9 shows the map accuracies while tuning the *visualisation parameters*. The behaviour of parameters in this group is similar to the ones in the previous group. A minor improvement in the accuracy across all distance ranges is achieved. However, the original parameter values are close to the optimum.

	Relative difference	Decreased values	Default values	Increased values	Relative difference
Map accuracy (+-5m)	**0.01%**	**88.08%**	88.07%	87.99%	0.09%
Map accuracy (+-3m)	**0.12%**	**81.12%**	81.02%	80.90%	0.15%
Map accuracy (+-2m)	**0.15%**	**73.48%**	73.37%	72.95%	0.57%
Map accuracy (+-1m)	**0.05%**	**56.51%**	56.48%	50.77%	10.11%

Table 7.9: Visualisation parameters tuning - SLAM generated map accuracy results - Sonnblick Observatory dataset

After performing the accuracy tests on each parameter group, it can be concluded that tuning the parameter values, by decreasing them, in the Surrounding map and the Visualisation parameter groups, can improve the accuracy of the SLAM generated 3D map. For the final test, values of the tuned parameters from both groups are combined, and finetuned again, in order to get the best possible map accuracy values. For this, final test, the Sonnblick Observatory dataset is used. However, testing is performed both on maps with and without vegetation.

Table 7.10 presents the comparison of map accuracy values for maps generated using the optimised parameters of the SC-LIO-SAM and the original parameters of the LIO-SAM algorithm. When the parameters are optimised, the SC-LIO-SAM algorithm performs better than the original LIO-SAM algorithm. The accuracy improvement is negligible for the dataset with the vegetation, peaking at 0.28 %, while, in the case when the vegetation is removed, the improvement is noticeable, peaking at 8.53 %.

	Sonnblick Observatory			Sonnblick Observatory - vegetation removed		
	LIO-SAM	**SC-LIO-SAM**	**Relative Difference**	**LIO-SAM**	**SC-LIO-SAM**	**Relative Difference**
Map accuracy (+-5m)	88.07%	**88.11%**	**0.05%**	95.13%	**95.31%**	**0.19%**
Map accuracy (+-3m)	81.02%	**81.25%**	**0.28%**	92.93%	**93.11%**	**0.19%**
Map accuracy (+-2m)	73.37%	**73.51%**	**0.19%**	89.57%	**91.01%**	**1.61%**
Map accuracy (+-1m)	56.48%	**56.54%**	**0.11%**	73.54%	**79.81%**	**8.53%**

Table 7.10: Optimised parameters - SLAM generated map accuracy results - Sonnblick Observatory dataset

8

Discussion and Conclusions

The primary objective of this thesis is to analyze and enhance the accuracy of SLAM generated maps of the environment. The initial section provides a concise overview of lidar sensors, emphasizing their fundamental types and operational principles. In this thesis, lidar sensors serve as the primary data collection tools for environmental information. The thesis then proceeds to present a comprehensive overview of SLAM methods, with a specific emphasis on lidar SLAM. The historical background and specific characteristics of lidar SLAM are covered in detail.

To evaluate the performance of different SLAM algorithms, four specific algorithms were selected and tested on diverse datasets representing various environments, each presenting its own unique challenges. These datasets were acquired using a novel sensing setup known as MOLISENS, which is complemented by a TLS providing ”ground truth” data. A set of criteria was established to compare the performance of the four selected algorithms. Subsequently, the algorithm that yielded the best results among the tested algorithms was further optimized to enhance the accuracy of the generated maps.

During the analysis of the GNSS and IMU data, it was observed that these sensors exhibit significant oscillations once activated and initialized on a boat in the water. These oscillations, characterized as noise, are likely a result of the boat’s movement in the water and the initial self-calibration process undergone by each sensor upon activation. Although the specific reasons behind the appearance of this noise are not extensively discussed in this thesis, its presence can lead to SLAM process failure, as demonstrated in Section 7.1. While one possible solution is to simply remove the noisy data, this approach may not always be feasible, particularly in cases where real-time SLAM is required, such as with autonomous vessels and vehicles. Thus, it is crucial to explore alternative methods for addressing or compensating for this noise to ensure the effectiveness and reliability of the SLAM process in practical applications.

Based on the analysis of the results presented in Section 7.1, it can be concluded that LeGO-LOAM is not suitable for the applications discussed in this thesis. This is primarily due to its limitations in saving the generated map and performing loop closure. These shortcomings significantly hinder its practical applicability. One potential reason for its poor performance may be the lack of official support for the software versions used in the post-processing setup of this thesis, as the algorithm was last updated in July 2020. This suggests potential compatibility issues with the software versions employed in this study, underscoring the importance of using algorithms that are regularly maintained and updated to align with the latest software developments and requirements.

SLAM algorithms without loop closure cannot produce plausible maps in complex environments. This is mainly concerning the KISS-ICP algorithm that, as shown in Section 7.1, failed to produce a plausible map of the Rijeka Harbur dataset and also produced a slightly distorted map of the ViF building dataset.

Furthermore, during the analysis of the Sonblick Observatory dataset in Section 7.1, it was observed that several tested SLAM algorithms encountered difficulties in generating undistorted maps when the ground was not level. Only the LIO-SAM algorithm successfully accomplished this task. Specifically, HDL-Graph-SLAM exhibited problems in producing a complete map of the environment, with the central part of the surveyed area missing. Similarly, the KISS-ICP algorithm encountered issues, resulting in a partially distorted and improperly oriented map. Due to these limitations, especially with HDL-Graph-SLAM, the ”map resolution” criterion values presented in Table 7.2 should not be considered as reliable indicators since parts of the HDL-Graph-SLAM and KISS-ICP maps are incomplete or distorted.

Another phenomenon to note is that trees, and vegetation in general, are a great source of measurement uncertainty with lidar sensors, since there is no way to know which exact leaf will be detected. This was shown as a reason for the ”low” accuracy of the maps representing the Sonnblick Observatory dataset. However, once the vegetation is removed, the accuracy increases substantially, up to above 95%.

Based on the comprehensive comparison tests conducted, it has been determined that the LIO-SAM algorithm is the most suitable choice for the specific application addressed in this thesis. However, recognizing the limitations of LIO-SAM, efforts were made to enhance its performance. As an improvement over the original algorithm, SC-LIO-SAM, developed by G. Kim and A. Kim[59], was introduced to address the limitations related to loop-closure capabilities. Based on the results presented in Section 7.2.1, it is evident that the SC-LIO-SAM algorithm indeed represents an improvement over LIO-SAM. It effectively resolves the issue of failed loop-closure observed with the Rijeka Harbour dataset, thereby enhancing the robustness of the SLAM process. However, it should be noted that SC-LIO-SAM comes with increased computational requirements, necessitating a reduction in the playback rate of the rosbags from 1 to 0.8 as a trade-off.

In Section 7.2.2 it was shown that the inclusion of GNSS data in the SLAM process has its benefits. These benefits include increased map accuracy, peaking at just over 5% for maps without vegetation. However, the main benefit of including GNSS data in the SLAM process is that the generated 3D maps have the proper compass orientation, as soon as they are generated. This makes comparing, for example, 3D maps with 2D maps rather simple, since they are already aligned properly.

In a final attempt to improve the accuracy of the map generated by the LIO-SAM algorithm, modifications were made to specific customizable parameters, as discussed in Section 6.5.3. However, after a comprehensive analysis of the results, it can be concluded that although optimizing the default parameter values can yield better map accuracy, the improvements achieved are often marginal. In many cases, the incremental gains obtained through fine-tuning each parameter are so small that they do not justify the time and effort invested. The maximum accuracy improvement observed is less than 0.3% for maps containing vegetation and less than 8.6% for maps without vegetation.

Future work may include gathering more diverse datasets, with the ”ground-truth”, so that more rigorous evaluation criteria can be applied to the SLAM generated 3D maps. Furthermore, the analysis of each individual configuration parameter in the LIO-SAM or SC-LIO-SAM algorithm, and their effect on the mapping process, is needed, because, currently, the information about it is extremely sparse. Ultimately, the analysis of the external forces affecting the sensor carrier in different environments could provide valuable information for improving the robustness of SLAM algorithms in general.

In conclusion, this thesis establishes the viability of utilizing automotive lidars, specifically the MOLISENS setup, for accurate environmental mapping. The findings of this research open up numerous potential applications that rely on the availability of a precise environmental map. In the marine field, it has the potential to enhance the accuracy of coastal and sea ice maps, thereby improving navigational safety. Additionally, automotive lidar technology can be applied to autonomous ships, particularly in urban transportation scenarios. These examples represent only a fraction of the possible applications that require an accurate environmental map for effective and safe operations. By leveraging the capabilities of automotive lidars, significant opportunities for innovation in various domains can be realized.

Bibliography

[1] R. Roriz, J. Cabral, and T. Gomes, “Automotive lidar technology: A survey,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 7, pp. 6282–6297, 2022.

[2] P. F. McManamon, *Lidar Technologies and Systems*. SPIE Press, 2019.

[3] “Roboat ii: A novel autonomous surface vessel for urban environments.” Institute of Electrical and Electronics Engineers Inc., October 2020, pp. 1740–1747.

[4] A. Pantazis, “Lidars usage in maritime operations and eco-autonomous shipping, for protection, safety and navigation for nato allies awareness,” 2019.

[5] “Molisens: Mobile lidar sensor system to exploit the potential of small industrial lidar devices for geoscientific applications,” *Geoscientific Instrumentation, Methods and Data Systems*, vol. 11, pp. 247–261, August 2022.

[6] T. Shan, B. Englot, D. Meyers, W. Wang, C. Ratti, and D. Rus, “Lio-sam: Tightly-coupled lidar inertial odometry via smoothing and mapping,” July 2020. [Online]. Available: http://arxiv.org/abs/2007.00258

[7] “Музей геодезических приборов,” Avaialble: https://theodoliteclub.com/?page_id=1136 (Accessed: 20/10/2022).

[8] “The rafale carries a wide range of smart and discrete sensors,” Avaialble: https://www.dassault-aviation.com/en/defense/rafale/a-wide-range-of-smart-and-discrete-sensors/ (Accessed: 20/10/2022).

[9] P. F. McManamon, *Field Guide to lidar*. SPIE Press, 2015.

[10] Z. Šalaka, S. Dervišbegović, and D. Milošević, *Fizika sa zbirkom zadataka za 3. razred srednje škole*. Svjetlost, 1998.

[11] M. Quigley, “Ros: an open-source robot operating system,” in *IEEE International Conference on Robotics and Automation*, 2009.

[12] J. Będkowski, M. Pełka, K. Majek, T. Fitri, and J. Naruniec, “Open source robotic 3d mapping framework with ros — robot operating system, pcl — point cloud library and cloud compare,” in *2015 International Conference on Electrical Engineering and Informatics (ICEEI)*, 2015, pp. 644–649.

[13] D. Girardeau-Montaut, “Cloudcompare - presentation,” Avaialble: https://www.cloudcompare.org/ (Accessed: 23/10/2022).

[14] C. Debeunne and D. Vivet, “A review of visual-lidar fusion based simultaneous localization and mapping,” *Sensors*, vol. 20, no. 7, 2020. [Online]. Available: https://www.mdpi.com/1424-8220/20/7/2068

[15] “What is slam (simultaneous localization and mapping),” Avaialble: https://www.mathworks.com/discovery/slam.html (Accessed: 10/10/2022).

[16] R. B. Rusu, *Andreas Nüchter (2009): 3D Robotic Mapping: The Simultaneous Localization and Mapping Problem with Six Degrees of Freedom (Springer Tracts in Advanced Robotics 52)*, Sep 2010, vol. 24, no. 3. [Online]. Available: https://doi.org/10.1007/s13218-010-0036-0

[17] J. Yang, Y. Li, L. Cao, Y. Jiang, L. Sun, and Q. Xie, “Survey of slam research based on lidar sensors,” p. 1003, 2019.

[18] T. Bailey and H. Durrant-Whyte, “Simultaneous localization and mapping (slam): Part ii,” *Robotics Automation Magazine, IEEE*, vol. 13, pp. 108 – 117, October 2006.

[19] G. Grisetti, C. Stachniss, and W. Burgard, “Improved techniques for grid mapping with rao-blackwellized particle filters,” *Robotics, IEEE Transactions on*, vol. 23, pp. 34 – 46, March 2007.

[20] S. Kohlbrecher, O. von Stryk, J. Meyer, and U. Klingauf, “A flexible and scalable slam system with full 3d motion estimation,” *2011 IEEE International Symposium on Safety, Security, and Rescue Robotics*, pp. 155–160, 2011.

[21] M. Simas, B. J. Guerreiro, and P. Batista, “Earth-based Simultaneous Localization and Mapping for Drones in Dynamic Environments,” vol. 104, Apr. 2022.

[22] P. Besl and N. D. 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] F. Pomerleau, F. Colas, and R. Siegwart, “A review of point cloud registration algorithms for mobile robotics,” *Foundations and Trends® in Robotics*, vol. 4, pp. 1–104, May 2015.

[24] A. Segal, D. Hähnel, and S. Thrun, “Generalized-icp,” June 2009.

[25] J. Zhang and S. Singh, “Low-drift and real-time lidar odometry and mapping,” *Autonomous Robots*, vol. 41, pp. 401–416, February 2017.

[26] P. Biber and W. Straßer, “The normal distributions transform: a new approach to laser scan matching,” *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, vol. 3, pp. 2743–2748 vol.3, 2003.

[27] R. Cottle and M. N. Thapa, *Linear and nonlinear optimization*. Springer, 2017.

[28] M. Magnusson, A. Lilienthal, and T. Duckett, “Scan registration for autonomous mining vehicles using 3d-ndt,” *Journal of Field Robotics*, vol. 24, pp. 803–827, October 2007.

[29] T. Stoyanov, M. Magnusson, H. Andreasson, and A. J. Lilienthal, “Fast and accurate scan registration through minimization of the distance between compact 3d ndt representations.” *Int. J. Robotics Res.*, vol. 31, no. 12, pp. 1377–1393, 2012. [Online]. Available: http://dblp.uni-trier.de/db/journals/ijrr/ijrr31.html#Stoyanov0AL12

[30] J. P. Saarinen, H. Andreasson, T. Stoyanov, and A. J. Lilienthal, “3d normal distributions transform occupancy maps: An efficient representation for mapping in dynamic environments,” *The International Journal of Robotics Research*, vol. 32, no. 14, pp. 1627–1644, 2013. [Online]. Available: https://doi.org/10.1177/0278364913499415

[31] D. Droeschel, J. Stückler, and S. Behnke, “Local multi-resolution representation for 6d motion estimation and mapping with a continuously rotating 3d laser scanner,” *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5221–5226, 2014.

[32] D. Droeschel and S. Behnke, “Efficient continuous-time slam for 3d lidar-based online mapping,” *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1–9, 2018.

[33] J. Behley and C. Stachniss, “Efficient surfel-based slam using 3d laser range data in urban environments,” June 2018.

[34] W. Hess, D. Kohler, H. Rapp, and D. Andor, “Real-time loop closure in 2d lidar slam,” in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 1271–1278.

[35] D. Gálvez-López and J. D. Tardós, “Bags of binary words for fast place recognition in image sequences,” *IEEE Transactions on Robotics*, vol. 28, pp. 1188–1197, 2012.

[36] B. Steder, M. Ruhnke, S. Grzonka, and W. Burgard, “Place recognition in 3d scans using a combination of bag of words and point feature based relative pose estimation,” in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011, pp. 1249–1255.

[37] R. Dubé, D. Dugas, E. Stumm, J. I. Nieto, R. Siegwart, and C. Cadena, “Segmatch: Segment based loop-closure for 3d point clouds,” *CoRR*, vol. abs/1609.07720, 2016. [Online]. Available: http://arxiv.org/abs/1609.07720

[38] R. Dube, A. Cramariuc, D. Dugas, J. Nieto, R. Siegwart, and C. Cadena, “Segmap: 3d segment mapping using data-driven descriptors,” June 2018.

[39] R. Dubé, M. G. Gollub, H. Sommer, I. Gilitschenski, R. Y. Siegwart, C. Cadena, and J. I. Nieto, “Incremental-segment-based localization in 3-d point clouds,” *IEEE Robotics and Automation Letters*, vol. 3, pp. 1832–1839, 2018.

[40] I. Vizzo, T. Guadagnino, B. Mersch, L. Wiesmann, J. Behley, and C. Stachniss, “Kiss-icp: In defense of point-to-point icp - simple, accurate, and robust registration if done the right way,” *ArXiv*, vol. abs/2209.15397, 2022.

[41] “Os2 long-range high-resolution imaging lidar - ouster,” Avaialble: https://data.ouster.io/downloads/datasheets/datasheet-revd-v2p0-os2.pdf (Accessed: 30/04/2023).

[42] T. Goelles, B. Schlager, S. Muckenhuber, S. Haas, and T. Hammer, “pointcloudset: Efficient analysis of large datasets of point clouds recorded over time,” *The Journal of Open Source Software*, vol. 6, p. 3471, September 2021.

[43] “124600-xsens20-d-leaflet mti-series v5 d20200129,” Avaialble: https://www.xsens.com/hubfs/Downloads/Leaflets/MTi%20600-series%20Datasheet.pdf (Accessed: 10/12/2022).

[44] “Ann-mb series - u-blox,” Avaialble: https://content.u-blox.com/sites/default/files/ANN-MB_DataSheet_%28UBX-18049862%29.pdf (Accessed: 10/12/2022).

[45] “Riegl vz-6000 - 3d ultra long range terrestrial laser scanner with online waveform processing,” Avaialble: http://www.riegl.com/uploads/tx_pxpriegldownloads/RIEGL_VZ-6000_Datasheet_2020-09-14.pdf (Accessed: 30/04/2023).

[46] “Os1 mid-range high-resolution imaging lidar - ouster,” Avaialble: https://data.ouster.io/downloads/datasheets/datasheet-revd-v2p0-os1.pdf (Accessed: 30/04/2023).

[47] T. Hammer, “New applications of automotive lidar sensors in geosciences,” Master’s thesis, Technische Universität Graz, November 2021.

[48] Virtual Vehicle Research GmbH, “Mapping dataset styria,” Avaialble: https://github.com/virtual-vehicle/mapping_dataset_styria (Accessed: 31/03/2023).

[49] J. L. Hintze and R. D. Nelson, “Violin plots: A box plot-density trace synergism,” *The American Statistician*, vol. 52, no. 2, pp. 181–184, 1998. [Online]. Available: http://www.jstor.org/stable/2685478

[50] S. Kalenjuk and W. Lienhart, “A method for efficient quality control and enhancement of mobile laser scanning data,” *Remote Sensing*, vol. 14, no. 4, 2022. [Online]. Available: https://www.mdpi.com/2072-4292/14/4/857

[51] “Distances computation - cloudcomparewiki,” Avaialble: https://www.cloudcompare.org/doc/wiki/index.php/Distances_Computation (Accessed: 28/12/2022).

[52] D. Lague, N. Brodu, and J. Leroux, “Accurate 3d comparison of complex topography with terrestrial laser scanner: Application to the rangitikei canyon (n-z),” *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 82, pp. 10–26, 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0924271613001184

[53] W. Zhang, S. Cai, X. Liang, J. Shao, R. Hu, S. Yu, and G. Yan, “Cloth simulation-based construction of pit-free canopy height models from airborne lidar data,” *Forest Ecosystems*, vol. 7, p. 1, December 2020.

[54] B. Valentin, “Evaluation and comparison of 3d lidar based slam algorithms,” Master’s thesis, Royal Military Academy, January 2021. [Online]. Available: https://www.aia-polytech.be/wp-content/uploads/2021/01/EvaluationAndComparisonOf3DLidarBasedSLAMAlgorithms.pdf

[55] Autoware Documentation, “Available open source slam,” Avaialble: https://autowarefoundation.github.io/autoware-documentation/main/how-to-guides/integrating-autoware/creating-maps/open-source-slam/ (Accessed: 09/06/2023).

[56] B. Garigipati, N. Strokina, and R. Ghabcheloo, “Evaluation and comparison of eight popular lidar and visual slam algorithms,” 2022.

[57] T. Shan and B. Englot, “Lego-loam: Lightweight and ground-optimized lidar odometry and mapping on variable terrain,” in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 4758–4765.

[58] K. Koide, J. Miura, and E. Menegatti, “A portable three-dimensional lidar-based system for long-term and wide-area people behavior measurement,” *International Journal of Advanced Robotic Systems*, vol. 16, February 2019.

[59] G. Kim and A. Kim, “Scan context: Egocentric spatial descriptor for place recognition within 3d point cloud map,” in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 4802–4809.

[60] B. Dikic, “Sc-lio-sam-updated,” Avaialble: https://github.com/Bera97/SC-LIO-SAM-updated (Accessed: 01/06/2023).

[61] H. Chen, W. Wu, S. Zhang, C. Wu, and R. Zhong, “A gnss/lidar/imu pose estimation system based on collaborative fusion of factor map and filtering,” *Remote Sensing*, vol. 15, no. 3, 2023. [Online]. Available: https://www.mdpi.com/2072-4292/15/3/790