Master's thesis

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

Henrik Senderud Storli

Timing and time synchronization within LiDAR- and IMU-based simultaneous localization and mapping (SLAM)

Master's thesis in Cybernetics and Robotics
Supervisor: Torleiv Håland Bryne
June 2023

NTNU
Norwegian University of
Science and Technology

Henrik Senderud Storli

Timing and time synchronization within LiDAR- and IMU-based simultaneous localization and mapping (SLAM)

Master's thesis in Cybernetics and Robotics
Supervisor: Torleiv Håland Bryne
June 2023

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

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of Science and Technology

Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

MASTER'S THESIS DESCRIPTION SHEET

Name:	Henrik Senderud Storli
Department:	Engineering Cybernetics
Thesis title (Norwegian):	Tidsstempling og tidssynkronisering innen multi-sensor kartlegging og lokalisering
Thesis title (English):	Timing and time synchronization within multi-sensor simultaneous localization and mapping (SLAM)

Thesis Description: SLAM is well studied field within robotics using several different sensor modalities as cameras, lidars and IMU. Some studies also include usage of global navigation satellite systems (GNSS).

The following tasks should be considered:

1. Perform a short literature review on
    a. graph-based SLAM where SLAM fuses IMU with lidar, and GNSS sensor data.
    b. timing and time synchronization within sensor fusion for navigation and SLAM
2. Set up a sensor platform using an ethernet-based lidar, an IMU and multiple GNSS. Interface the sensor setup using SentiBoard1.
3. Choose a time synchronization primitive and implement a sensor timing and synchronization algorithm for the multisensor platform. Justify your chosen synchronization primitive.
4. Deploy an excisting graph-based SLAM framework based on the sensors listed above. Justify your choice.
5. Modify the SLAM framework to accommodate a more flexible and improved GNSS integration.
6. Study the effect of timing and time synchronization in the SLAM framework. Use RTK-GNSS as ground truth.
7. Study the effect of the improved GNSS/SLAM integration.
8. Consider sensor mounting and mounting calibration.
9. Carry out experiment with a vehicle.
10. Present and discuss your results.
11. Conclude on your results and suggest further work.

Start date:	2023-09-01
Due date:	2023-16-06
Thesis performed at:	Department of Engineering Cybernetics, NTNU
Supervisor:	Adjunct associate professor Torleiv H. Bryne, Dept. of Eng. Cybernetics, NTNU
Co-Supervisor(s):	Frederik S. Leira, SentiSystems AS Sigurd M. Albrektsen, SentiSystems AS

Abstract

This thesis presents the construction of a sensor payload consisting of an Ouster16 LiDAR, a STIM300 inertial measurement unit (IMU), and three f9p u-blox global navigation satellite systems (GNSS) receivers, and investigates the performance of a particular LiDAR-inertial simultaneous localization and mapping (SLAM) algorithm, Liorf, when different timestamping primitives are used for sensor synchronization. To date, the field of applied sensor fusion has not extensively studied how the accuracy of synchronization between sensor measurements affects the performance of various SLAM algorithms. The primitives used are based on hardware timestamping from a SentiBoard, and software timestamping based on the Robotic Operating System (ROS). A synchronization module for synchronizing the Ouster LiDAR, based on encoder pulses, with the SentiBoard through its ROS driver is presented. Data is collected by mounting the sensor payload on top of a car and driving in urban areas as well as on the highway. The position and orientation estimates from Liorf are compared to estimates obtained by a Real-time kinematic positioning (RTK) solution from GNSS measurements. Full orientation estimates based on RTK GNSS are achieved through the use of three GNSS receivers. The result indicates that there is no significant difference between hardware and software synchronization when it comes to Liorf pose estimation accuracy. However, the accuracy of software synchronization is highly dependent on the deployed software, and the hardware components the software is running on. The experiment performed in this thesis used high-quality sensors, computers, and software, which resulted in accurate software timestamps, that would not necessarily be possible in other systems with low-quality components. In addition to the synchronization analysis of Liorf, an enhanced approach for integrating GNSS measurements into the algorithm is presented. This modification involves compensating for the relative mounting between the IMU and the GNSS antennas, as well as incorporating relative position measurements when multiple antennas are used. The implementation of this modification required the development of two new classes within the Georgia Tech Smoothing and Mapping (GTSAM) `C++` library.

Sammendrag

Denne avhandlingen presenterer konstruksjonen av en sensorsammensetning bestående av en Ouster16 LiDAR, en STIM300 inertial måleenhet (IMU) og tre f9p u-blox globalt navigasjonssatellittsystem (GNSS)-mottakere, og undersøker ytelsen til en spesifikk LiDAR-inertial Samtidig posisjonering og kartlegging (SLAM) algoritme, Liorf, når ulike tidsstemplingsteknikker brukes for sensor-synkronisering. Hittil har forskningsfeltet innen anvendt sensorfusjon ikke omfattende studert hvordan nøyaktigheten av synkroniseringen mellom sensormålinger påvirker ytelsen til ulike SLAM-algoritmer. De benyttede teknikkene i denne avhandlingen er basert på maskinvarebasert tidsstempling fra et SentiBoard og programvarebasert tidsstempling basert på Robotic Operating System (ROS). En synkroniseringsmodul for å synkronisere en Ouster LiDAR, basert på enkodersignaler, med et SentiBoard gjennom dens ROS-driver, blir presentert. Data samles inn ved å montere sensorsammensetningen på toppen av en bil og kjøre i byområder og på motorveien. Posisjons- og orienteringsestimatene fra Liorf blir sammenlignet med estimatene som er oppnådd gjennom en sanntidskinematisk posisjonering (RTK) basert på GNSS-målinger. Fullstendige orienteringsestimat basert på RTK GNSS oppnås ved bruk av tre GNSS-mottakere. Resultatene indikerer at det ikke er noen betydelig forskjell mellom maskinvare- og programvaresynkronisering når det gjelder nøyaktigheten av estimatene fra Liorf. Imidlertid er nøyaktigheten av programvaresynkronisering sterkt avhengig av den anvendte programvaren og maskinvarekomponentene. Eksperimentet som ble utført i denne avhandlingen brukte høykvalitetssensorer, datamaskiner og programvare, noe som resulterte i nøyaktige programvaretidsstempler som ikke nødvendigvis vil være mulig i andre systemer. I tillegg til synkroniseringsanalysen av Liorf blir det presentert en forbedret tilnærming for å integrere GNSS-målinger i algoritmen. Denne modifikasjonen innebærer kompensering for den relative monteringen mellom IMU-en og GNSS-antennene, samt innlemming av relative posisjonsmålinger når flere antenner brukes. Implementeringen av denne modifikasjonen krevede utviklingen av to nye klasser innenfor Georgia Tech Smoothing and Mapping (GTSAM) `C++` biblioteket.

Preface

This thesis represents the final milestone towards the completion of a Master of Science degree in Cybernetics and Robotics at the Norwegian University of Science and Technology (NTNU). The research has been conducted in collaboration with SentiSystems AS, who generously provided the necessary hardware and software technology for sensor timing and data integration. Throughout the project, Associate Professor Torleiv H. Bryne has served as the dedicated supervisor, offering continuous guidance and support.

This thesis builds upon an unpublished specialization project thesis (Storli, 2022) undertaken by the same author in the autumn of 2022. It incorporates and rephrases essential findings and information from the specialization project, ensuring their relevance and coherence within the context of this thesis. The following is a list of the materials included in the specialization project:

- Parts of Section 2.1 and Section 2.2.
- Parts of Section 3.3 and Section 3.4.
- Parts of Section 4.3.
- Section 6.1

The material derived from the specialization project and utilized in this thesis will be indicated by the symbols * and † at the beginning and end, respectively. It is worth noting that all illustrations and figures included in this thesis are original creations of the author unless explicitly stated otherwise.

Acknowledgments

The author extends appreciation to Associate Professor Torleiv H. Bryne for providing guidance throughout this project and for being a valuable conversation partner. Special thanks are also due to Dr. Frederik S. Leira and Dr. Sigurd M. Albrektsen for their valuable information and assistance in utilizing the SentiBoard and SentiSystems software. This project would not have been possible without their support.

Additionally, the author would like to express sincere gratitude to Glenn Angel, Anders R. Petersen, and Daniel Bogen at the NTNU workshop for their invaluable assistance and exceptional service-oriented approach.

Contents

List of Figures

List of Tables

Listings

Acronyms

1PPS 1-Pulse-Per-Second. 6, 32, 41, 51, 53, 54, 83

ECEF Earth Centered Earth Fixed. 10, 11, 14, 18, 19

ECI Earth Centered Inertial. 10, 11

ENU East North Up. 10–12, 14–16, 18–20, 24, 27, 46, 47, 60, 64

GND Ground. 30, 32

GNSS global navigation satellite system. 1–4, 6, 9, 10, 14, 15, 17–19, 21, 23–25, 27, 29, 32, 35, 37, 40, 43, 45–48, 50, 51, 53, 55, 58, 60, 65, 66, 71, 73, 74, 78, 83, 84

GPIO general purpose input output. 7

GPS Global Positioning System. 27

GTSAM Georgia Tech Smoothing and Mapping. 3, 4, 25, 45, 84

IC Input Capture. 30

IMU inertial measurement unit. 1, 3, 4, 10–12, 15, 16, 19, 23, 24, 26–29, 32, 33, 35, 40, 44, 45, 53, 55, 66, 83

ISR interrupt service routine. 7

MAE mean absolute error. 52, 60, 78

MAP Maximum a posteriori. 21, 22

ME mean error. 52, 55, 60, 78

NED North East Down. 11, 12, 14, 18, 43

NTNU Norwegian University of Science and Technology. 35

PDF probability density function. 19, 21

PPK Post processed kinematic. 17, 18

RMSE root mean square error. 52, 78

ROS Robot Operating System. 29, 37, 39–42, 44, 51, 53–55, 58, 66, 71, 83

RTK Real-time kinematic positioning. 4, 17, 18, 31, 32, 37, 43, 51, 58, 60, 65, 66, 73

SB SentiBoard. 55, 58, 66, 71

SLAM simultaneous localization and mapping. 1–4, 9, 19, 20, 45, 82, 83

SPI serial peripheral interface. 30

STD standard deviation. 52, 60, 78

TOA time of arrival. 30

TOT time of transport. 30

TOV time of validity. 6, 30, 32, 41, 42, 53

UART universal asynchronous receiver-transmitter. 6, 30, 32, 43

UDP User Datagram Protocol. 39, 40, 42

USB Universal Serial Bus. 30

UTC Coordinated Universal Time. 6, 58

Chapter 1

Introduction

The purpose of this chapter is to provide the motivation behind the research conducted in this thesis and introduce the underlying hypothesis. The thesis makes several significant contributions to the field of applied sensor fusion, which will be outlined in detail.

1.1 Motivation

Advancements in autonomous systems have led to a growing demand for robots that can navigate and solve complex tasks in unknown environments. To achieve this, robots need the ability to construct maps of their surroundings and estimate their position relative to them. This critical task is known as simultaneous localization and mapping (SLAM) and has a rich history of development (Aulinas et al., 2008; Leonard and Durrant-Whyte, 1991). The first SLAM algorithms were proposed in the 1980s and relied on simple sensors such as odometry and range finders. However, these algorithms were limited in their ability to handle complex environments and often suffered from drift and accumulation of errors. Today, SLAM has become increasingly important in various applications such as autonomous driving, search and rescue operations, and exploration of extraterrestrial environments. As society continues to rely more on autonomous systems, improving the accuracy and reliability of SLAM algorithms becomes crucial for their successful implementation in real-world scenarios.

Over the years, SLAM algorithms have evolved to incorporate more advanced sensors such as LiDAR (Raj et al., 2020), stereo cameras (O'Riordan et al., 2018), inertial measurement unit (IMU) (Ahmad et al., 2013), global navigation satellite system (GNSS) (Groves, 2013), and magnetometers (Robbes, 2006) as well as more sophisticated estimation techniques such as particle filters and graph optimization (Taheri and Xia, 2021). LiDAR sensors provide precise measurements of distance and angle to objects in the environment, making them well-suited for mapping applications. Cameras, on the other hand, provide rich visual information that can be used for feature detection and localization. IMUs provide information about the robot's acceleration and angular velocity, which can be integrated

to estimate its position and orientation. GNSS can assist in correcting global drift in the map and the robot's position, as well as referencing the estimates to a global frame. Magnetometers measure the local magnetic field and can be used to estimate the robot's heading. Many SLAM systems use multiple sensors of the same type to obtain more data and make the system more robust to sensor failure. For example, using two cameras instead of one makes it possible to estimate distance to the surrounding environment and enhance the system's robustness to changes in lighting conditions. Similarly, using multiple GNSS receivers can help estimate the orientation and provide more accurate position estimates, especially in urban canyons or other environments with limited satellite visibility. However, deploying multiple sensors on a large vehicle requires compensating for relative mounting between the sensors. Software frameworks for sensor fusion often offer limited flexibility in compensating for this.

Sensor measurements capture information derived from the real world, encompassing physical quantities as well as timing details indicating the validity of the measured state. The process of obtaining such timing details is referred to as timestamping. Using multiple sensors from different manufacturers often includes the process of synchronization, such that all measurements are timestamped with respect to the same time reference. There exist multiple primitives to synchronize and timestamp sensor data, some more accurate than others (Jellum et al., 2022). What role accurate synchronization and timestamping play in applied SLAM systems is not a well-studied topic in the field of robotics. In Storli (2022), the effect of hardware vs software synchronization was studied for the tightly coupled LiDAR-inertial odometry via smoothing and mapping (LIO-SAM) algorithm (Shan et al., 2020). The result indicated that the difference between the synchronization primitives is more significant as the dynamics of the physical system increase. The test case used a handheld sensor platform that would not be considered applicable to any industry-relevant application. Therefore, the importance of accurate synchronization and timestamping in industry-relevant applications using SLAM remains an unanswered question. This thesis explores the accuracy of Liorf (an implementation of LIO-SAM) when sensor data is synchronized using hardware and software timestamping methods. The investigation focuses on a scenario where the sensors are mounted on top of a car, which is driven in urban areas.

1.2 Problem description

From Storli (2022), it is proposed that systems with fast dynamics, such as drones, race cars, fighter jets, and speed boats, may be more susceptible to estimation errors arising from synchronization inaccuracies than systems with slow dynamics, such as container ships, passenger planes, and trains. The rationale behind this is the rapid and frequent changes in the measurement magnitudes that occur in fast-moving systems. For instance, a drone may be moving at high speeds and rapidly changing directions, which can lead to abrupt changes in sensor measurements. In systems where the sensor measurement magnitudes change slowly, accurate

synchronization may not be necessary.

Based on the significant difference observed between hardware and software synchronization on the handheld platform in Storli (2022), and considering the potentially faster dynamics of a car, the following hypothesis is formulated, and considered the main hypothesis of the thesis:

A SLAM algorithm applied to a car driving in urban areas will have significantly less accurate position and orientation estimates when the sensor data is synchronized based on software methods compared to hardware methods.

This hypothesis was tested by gathering sensor data from a LiDAR, an IMU, and three GNSS receivers mounted on top of a car driving in urban areas. All sensor measurements were timestamped and synchronized based on hardware and software methods. This resulted in a dataset where all measurements were given two timestamps, one from software-based methods, and one from hardware-based methods. The sensor data was then fed into the Liorf algorithm, and the output pose and orientation estimates were compared.

Liorf was chosen due to its tight coupling of sensor data, and that it is considered state of the art in open source LiDAR-inertial SLAM algorithms (X. Xu et al., 2022b). The car platform was chosen due to its relevance to the upcoming industry challenge of making reliable self-driving cars. Such cars must be trustworthy, or else the consequences may be fatal. All of the actions a car makes are based on the information it has about the world around it, and this knowledge is mainly based on sensor data. If a car lacks realistic estimates of its surrounding environment, it becomes unlikely for it to make safe decisions.

The vehicle employed in this project is of considerable size, resulting in a substantial separation between the IMU and the GNSS antennas. Moreover, the current estimation approach employed by Liorf, which relies on the Georgia Tech Smoothing and Mapping (GTSAM) `C++` software library, does not possess the capability to compensate for this distance. To overcome this limitation, this thesis introduces a modification to Liorf that addresses the antenna separation issue. Additionally, this modification incorporates relative position measurements obtained from multiple GNSS receivers. The implementation of this modification involves the development of two new classes integrated into the GTSAM library. The hypothesis is that the proposed modification will greatly improve the estimation accuracy of Liorf when integrating GNSS measurements. To test this, the modified version of Liorf was compared to the original implementation by evaluating the resulting estimates.

1.3 Main contributions

The main contributions of this thesis are:

- Insight into how hardware and software synchronized sensor measurements affect the position and orientation estimates of a state-of-the-art SLAM algorithm. (Section 6.5)
- A synchronization module for synchronizing an Ouster LiDAR with a SentiBoard v1.30 based on the open-source LiDAR ROS driver, as well as a quantitative comparison of different LiDAR synchronization methods. (Section 4.3.3, Section 6.2.1)
- An industry-relevant dataset with hardware synchronized measurements with highly accurate timestamps (10 ns resolution), consisting of measurements from a LiDAR, an IMU, and three GNSS receivers.
- A modification of Liorf by compensation for mounting calibration between IMU and GNSS, and relative position measurements from GNSS antennas. This was done through an expansion of the GTSAM `C++` factor graph library, with two new GNSS factors, *GPSWithLeverArmFactor* and *GPSBaseLineFactor*. (Chapter 5, Section 6.6)

1.4 Organization of thesis

The rest of this thesis is organized as follows. Chapter 2 presents the most common timestamping and synchronization primitives within sensor fusion for navigation and SLAM, as well as a mathematical framework for modeling synchronization errors. In Chapter 3, background information regarding sensor fusion and SLAM is presented. This includes, coordinate system and transformations commonly used in sensor fusion algorithms, mathematical frameworks for modeling sensor measurements, and graph-based SLAM. Chapter 3 also includes a detailed description of the mechanism behind the SLAM algorithm deployed in this project, Liorf. In Chapter 4 the system used to carry out the experiment and data analysis is described. This includes sensor mounting, hardware and software integration, Real-time kinematic positioning (RTK) GNSS solution generation, and timestamp modification. Chapter 5 describes the expansion of the GTSAM library and the integration of the new factors in Liorf. In Chapter 6 the results are presented and discussed. Firstly, the results for comparing two different methods for LiDAR synchronization is presented. Then the main result of the thesis is presented, the hardware- and software-based synchronization comparison of Liorf estimates. Lastly, the different GNSS factors are compared, to establish if the modification of Liorf contributes to more accurate estimates. Finally, the conclusion and suggestions for further work are presented in Chapter 7.

Chapter 2

Synchronization and Timing Theory

Synchronization in the field of sensor fusion is the process of aligning the timestamps of measurements from multiple sensors to a common time reference. Today, there are multiple synchronization primitives capable of achieving this goal. Some rely on each sensor having an internal clock to timestamp the measurements, while others use individual pulses that are sent out each time a new measurement is sampled. There are also less sophisticated methods that timestamp the measurements with the arrival time at the host computer. However, these methods are considered less accurate since they do not take into account the transportation delay from the sensor to the host computer. Sensor measurements that contain large amounts of data, such as cameras and LiDARs, will therefore be less accurately timestamped compared to sensors such as IMUs that send small amounts of data.

Synchronizing multiple sensors boils down to timestamping all measurements with respect to the same time reference. On the other side, timestamping comes down to giving a single measurement from one sensor an estimate of when the measured value was valid in the real world. Just as for synchronization, some methods are considered more accurate than others.

2.1 Synchronization primitives

In the following, the most common synchronization primitives are presented and discussed. The information is based on the work in Jellum et al. (2022).

Network synchronization

Some sensors include advanced microcontrollers capable of running the TCP/IP stack and support IEE1588 Precision Time Protocol (Watt et al., 2015) and Network Time Protocol (Mills, 1991). These protocols allow sensors and computers

to exchange information regarding each other's clock state. Based on this information each computer can adjust its clock to match the other clocks in the system. Usually, there is a grand master clock that all other clocks in the network are trying to synchronize with. This protocol is preferred in cases when the amount of hardware cables are supposed to be at a minimum, as only an ethernet cable is necessary.

1-Pulse-Per-Second(1PPS)

This primitive is suitable for sensors containing a microcontroller which samples and timestamps sensor measurements based on its own internal clock (Mogul et al., 2000). A 1-Pulse-Per-Second (1PPS) is a 1-wire signal which expects a flank at the beginning of each second. The embedded microcontroller can then timestamp measurements relative to the beginning of a whole second. Typically the host computer is the 1PPS sender, therefore referred to as the 1PPS master, while the sensor receiving the pulse is referred to as the 1PPS slave. It is also possible for the sensor to be the 1PPS master, sending a pulse each time its internal clock is at a top of a second. Then the host computer is referred to as a 1PPS slave. To avoid unnecessary clock drift, it is recommended to use the system with the most accurate oscillator as the 1PPS master. Given that the timestamping of the 1PPS pulse is accurate, the synchronization error is given by the difference in clock drift between the two systems. Some sensors can receive and parse certain GNSS timing packets from a GNSS receiver by the use of a universal asynchronous receiver-transmitter (UART) receiver. This allows the sensor to timestamp measurements on the Coordinated Universal Time (UTC) timeline. Some systems also support multiple pulses per second, to achieve an even tighter synchronization between the systems. This method can be generalized to nPPS. If the sensor is configured to be a 10PPS-slave, it will timestamp all measurements relative to the beginning of every 10th of a second.

Time-of-validity (TOV) synchronization

Time of validity (TOV) is a commonly used synchronization primitive for sensors that do not timestamp their measurements to an internal clock. The method relies on a hardware pulse outputted by the sensor that flanks at the exact moment in time when a measurement is sampled. Since the measurement data arrives at a later time than the TOV pulse, the pulse needs to be timestamped and associated with the corresponding measurement data at the host computer.

Triggering

Trigger synchronization is based on a single wire signal from the host computer to the sensor, telling it when to sample a new measurement. This synchronization primitive is often used for sensors such as IMUs and cameras. The sensor receiving the pulse usually has a fixed delay from the arrival of the pulse to the sample

is performed. In order to obtain the exact time of sampling, this delay must be added to the timestamp of the trigger pulse. The duration of this delay is usually exposed in the sensor datasheet. Trigger synchronization is commonly used when it is important for the measurements to be taken at the exact same time. Applications such as stereo-vision greatly benefit from this type of synchronization, as the same features are identical in both frames. If the cameras sample at different times, the scene might change from one sample to the other making it harder to match features and obtain depth estimates.

2.2 Timestamping primitives

In this section, the most common timestamping primitives are covered. The information is based on the work in Jellum et al. (2022).

Hardware

The most accurate method, and the only one to achieve deterministic timestamps, is hardware timestamping. This method applies to 1PPS, TOV and Trigger synchronization. Hardware synchronization involves using dedicated timer peripherals to timestamp and generate pulses. Clock frequency and clock drift are typically the only factors that affect the accuracy of hardware timestamping.

GPIO-based timestamping

Configuring general purpose input output (GPIO) pins to generate an interrupt service routine (ISR) each time a pulse is detected, is one way to timestamp events. ISRs can also be used to generate signals used for Trigger synchronization. This is done by configuring the microcontroller to generate interrupts based on the internal clock value. When using a multi-threaded platform, the sleep function can also be utilized to generate and timestamp periodic pulses. The accuracy of GPIO-based Timestamping comes down to how the microcontroller is configured. If there are other ISRs with higher priority, the timestamp generated will not be deterministic. Additionally, when all interrupts are disabled, the timestamps will not be deterministic.

Software-based

An alternative, but less sophisticated method, is to timestamp the measurement when it arrives at the host computer. This method is referred to as software timestamping. If no synchronization primitives are available on the sensor or if the host computer has no GPIO pins available, this might be the only option. The weakness with software timestamping is the fact that it does not take into account the processing and transportation delay from the measurement is sampled to it has arrived at the host computer. If the host computer is busy with other calculations,

the measurement may be placed in a buffer before being timestamped, resulting in additional delay.

2.3 Synchronization models

As with most physical phenomena, it is beneficial to have mathematical models to lean on when studying them. The same goes for synchronization and timestamping. In this section, some basic models describing synchronization errors will be presented and discussed. The information presented is mainly based on the work in Sivrikaya and Yener (2004). Sivrikaya and Yener (2004) focuses on synchronization in sensor networks, but the theory can be generalized to most systems containing a hardware oscillator driving the internal clock.

Computing devices typically have a hardware oscillator-assisted clock that provides an approximation of real-time t, denoted as $C(t)$. The rate at which the clock runs is determined by the angular frequency of the hardware oscillator. Ideally, the rate of a perfect clock, denoted as dC/dt, would equal 1. However, all clocks are subject to clock drift, which refers to the unpredictable variation of oscillator frequency due to various physical effects. Despite the frequency of a clock changing over time, it can still be approximated with good accuracy using an oscillator with a fixed frequency. For a given sensor i in a sensor payload, its local clock can be approximated as

$$C_i(t) = a_i t + b_i, \tag{2.1}$$

where a_i represents the clock drift and b_i represents the offset of sensor i's clock. The drift denotes the rate or frequency of the clock, while the offset is the difference between the clock value and the real-time t.

By using (2.1), the local clocks of two components in a sensor payload can be compared, such as a sensor with an internal clock and a host computer. Denote the sensor as 1 and the host computer as 2, then the mathematical relationship can be expressed as

$$C_1(t) = a_{12} C_2(t) + b_{12}. \tag{2.2}$$

Here, a_{12} and b_{12} are the *relative drift* and *relative offset* between the clocks in the sensor and the host computer, respectively. When two clocks are perfectly synchronized, their relative drift is 1, indicating that they have the same rate, and their relative offset is zero, meaning they have the same value at that instant. Some studies in the literature use "skew" instead of "drift," which defines the difference, rather than the ratio, between clock rates (Elson et al., 2002). Additionally, the offset may be alternatively referred to as phase offset.

Chapter 3

Sensor Fusion and SLAM Theory

This chapter focuses on theory related to sensor fusion and SLAM, with emphasis on key aspects such as coordinate frames and transformations, sensor modeling, graph-based SLAM, and the mechanism behind Liorf. Coordinate frames and transformations are fundamental in robotics for accurate spatial representation and integration of sensor data. Understanding their relationships and application is crucial for consistent perception. Accurate sensor modeling is essential for robust sensor fusion. Various sensor models will be presented and discussed, such as LiDAR, GNSS, and IMU. Graph-based methods for solving the SLAM problem has gained prominence in recent years (X. Xu et al., 2022b). Additionally, the mechanism behind the SLAM algorithm LIO-SAM, which is the basis for Liorf, will be presented.

3.1 Coordinate systems and transformations

In the field of navigation, guidance, and control of robots and autonomous vehicles, there are different coordinate systems used for referencing physical quantities. Without a proper definition of a reference coordinate system, quantities such as orientation, position, velocity, and acceleration would give little to no information about the state of the system. This section about sensor fusion and SLAM theory is therefore based on a proper definition of coordinate systems. The information presented in this section is based on the material presented in Farrell (2008) and Cai et al. (2011).

Notation

The notation used for describing vectors in coordinate systems follows the notation of Fossen (2021). Furthermore, coordinate frames are expressed as $\{\cdot\}$, while a vector $\boldsymbol{z} \in \mathbb{R}^3$ from $\{a\}$ to $\{b\}$ is denoted $\boldsymbol{z}_{ab}^{c}$ when resolved in $\{c\}$. A rotation matrix mapping from frame $\{a\}$ to frame $\{b\}$ is denoted as $\boldsymbol{R}_{a}^{b}$. Scalars are written in lower case, while vectors and matrices are written in lower and upper case bold, respectively. Transpose is denoted $(\cdot)^\intercal$.

3.1.1 Coordinate frames

The most intensively used coordinate systems are summarized in Table 3.1. The coordinate systems $\{g\}$, $\{e\}$, $\{i\}$ are used for global referencing, while $\{u\}$ is used for referencing in a local area. The coordinate systems $\{b\}$, $\{b_i\}$ and $\{b_l\}$ refers to the body fixed frame, IMU sensor frame and LiDAR sensor frame, respectively. $\{b_{mb}\}$, $\{b_{r_1}\}$ and $\{b_{r_2}\}$ refers to three coordinates systems describing the placement of three GNSS antennas. The subscripts mb, $r1$, and $r2$ stands for Moving Base, Rover1 and Rover2, respectively. These are the names of the three GNSS antennas used in this project. The frames representing the GNSS antennas, IMU, and LiDAR are commonly referred to as "sensor frames". This section describes the individual frames in detail.

Table 3.1: Coordinate frames

Symbol	Description
g	Geodetic Coordinate System
e	Earth Centered Earth Fixed (ECEF)
i	Earth Centered Inertial (ECI)
u	East North Up (ENU)
b	Vehicle fixed body frame
b_l	LiDAR
b_i	IMU
b_{mb}	GNSS Moving Base antenna
b_{r_1}	GNSS Rover1 antenna
b_{r_2}	GNSS Rover2 antenna

Geodetic Coordinate System {g}

The Geodetic Coordinate System denoted $\{g\}$, is often used in GNSS-based positioning. This is not a usual Cartesian coordinate system, where the axes are orthogonal to each other, but a system that describes points in terms of latitude, longitude, and height, denoted λ, τ and h, respectively. Latitude is a measure of the angle between the equatorial plane and the line from the center of the reference ellipsoid passing through the point of interest. Possible values for latitude range from $-90°$ to $90°$. The longitude represents the angle between the Prime Median and the same line as Latitude. This value ranges from $-180°$ to $180°$. The Prime Median is a curved line connecting the poles through a specified point on the surface of the earth (Smith, 1976). The height is defined as the distance from the point of interest to the nearest point on the reference ellipsoid.

Earth Centered Earth Fixed {e}

The ECEF coordinate system has its origin located at the center of the earth, and rotates with it. As a consequence of this, a fixed point on the surface of the earth has a fixed set of coordinates in the ECEF coordinate system. Furthermore, the ECEF coordinate system is a right-handed system with its z-axis pointing to the north pole, and x-axis pointing in the 0° latitude and 0° longitude direction. The y-axis is orthogonal to the other axis and follows the right-hand convention.

Earth Centered Inertial {i}

The ECI frame has its origin located at the center of the earth and does not rotate with it. Being an inertial frame means that Newton's laws of motion apply. The z-axis is defined as for ECEF, and the x-axis points towards the vernal equinox. The y-axis is defined to complete the right-hand rule. At a certain point each year, the ECEF and ECI axes are identically aligned.

East North Up {u}

The ENU coordinate system is a local positioning system where the origin can be placed at any point. The x, y, and z axes point in the east, north, and up direction, respectively. It is an alternative to the North East Down (NED) coordinate system, which also has its origin placed at a local point, but with rotated axes. Typically, the origin of the ENU frame is placed at the start of a trajectory. While the ENU and NED systems provide an accurate representation of the area near the origin, as the vehicle moves farther away, the north axis no longer aligns with the north direction at the current location, leading to inaccurate representation.

Body and sensor coordinate systems {b}

The body coordinate system is a reference frame with its origin located on the vehicle. If the vehicle is a rigid body, all points on it will have constant coordinates in this frame. The origin can be placed anywhere on the vehicle, but it's usually placed at the center of mass or volume. The x-axis usually points towards the vehicle's front, while the z-axis points directly down or up when the vehicle is on horizontal ground. The y-axis follows the right-hand rule, which depends on the direction of the z-axis.

A vehicle typically has multiple sensors, each with its own coordinate system. For example, an IMU measures acceleration and angular velocity based on its own coordinate system. To convert these readings to the body frame, a clear definition of the IMU's coordinate system is required. LiDARs and cameras are often used to calculate relative motion between two scans or frames, with motion referenced to the local coordinate system of the sensor. However, because these sensors may not be placed identically on the body's origin, it is crucial to properly define their coordinate systems.

GNSS receivers can provide measurements that indicate the relative position of two antennas, typically given in ENU or NED. As with IMUs, LiDARs, and cameras, the antennas require representative coordinate systems for the measurements to be transformed into the body frame.

3.1.2 Transformations between coordinate systems

Transforming data from one coordinate frame to another is essential for applying sensor data in state estimation. An alternative viewpoint is to look at it as the data is not changed but represented alternatively. A transformation between two Cartesian frames consists of two types of actions:

1. Translation
2. Rotation

Translation involves the case when the origin of two coordinate systems is not identically at the same point. Rotation is the process when the origins are at the same point, but the axes do not point in the same direction. A transformation usually consists of a translation and a rotation.

There are multiple ways of representing the orientation of one coordinate frame relative to another (Fossen, 2021). A common approach is the use of rotation matrices. A rotation matrix $\boldsymbol{R}$ is an element in the Lie group $SO(3)$ and is defined as

$$\boldsymbol{R}\boldsymbol{R}^\intercal = \boldsymbol{R}^\intercal\boldsymbol{R} = \boldsymbol{I}_3, \quad \det(\boldsymbol{R}) = 1, \tag{3.1}$$

where $\boldsymbol{I}_3$ represents the 3×3 identity matrix, and $\det(\cdot)$ refers to the determinant. Rotation a vector $\boldsymbol{p}$ expressed in frame $\{a\}$ to a frame $\{b\}$ by the use of a rotation matrix is expressed as:

$$\boldsymbol{p}^b = \boldsymbol{R}_a^b \boldsymbol{p}^a \tag{3.2}$$

The rotation from one Cartesian frame into another can always be represented by the use of three successive rotations. This representation of rotation is referred to as Euler rotations (Stevens et al., 2015). The Euler angles for roll, pitch, and yaw are denoted as ϕ, θ, and ψ. The conversion between Euler angles and a rotation matrix is expressed as:

$$\boldsymbol{R}_a^b(\boldsymbol{\Theta}) = \begin{bmatrix} c\theta c\psi & -c\phi s\psi + s\phi s\theta c\psi & s\phi s\psi + c\phi s\theta c\psi \\ c\theta s\psi & c\phi c\phi + s\phi s\theta s\psi & -s\phi c\psi + c\phi s\theta s\psi \\ -s\theta & s\phi c\theta & c\phi c\theta \end{bmatrix}, \tag{3.3}$$

where $c\cdot$ and $s\cdot$ denotes $\cos(\cdot)$ and $\sin(\cdot)$, respectively, and $\boldsymbol{\Theta} = (\phi, \theta, \psi)^\intercal$

Since the Euler angles representation of orientation involves non-unique representations, called singularities, an alternative representation is the use of quaternions (Brekke, 2021, Ch. 10). The unit quaternion is defined as

$$\boldsymbol{q} = \eta + \epsilon_1 \boldsymbol{i} + \epsilon_2 \boldsymbol{j} + \epsilon_3 \boldsymbol{k}, \quad \boldsymbol{q}^\intercal \boldsymbol{q} = 1 \tag{3.4}$$

where η, ϵ_1, ϵ_2 and ϵ_3 are real numbers and $\boldsymbol{k}$, $\boldsymbol{j}$ and $\boldsymbol{i}$ are unit-vectors pointing along the three spatial axes. The conversion from quaternion to Euler angles is given by

$$\phi = \text{atan2}\left(2(\epsilon_3\epsilon_2 + \eta\epsilon_1), \eta^2 - \epsilon_2^2 - \epsilon_3^2 + \epsilon_1^2\right), \tag{3.5a}$$
$$\theta = \text{asin}\left(2(\eta\epsilon_2 - \epsilon_1\epsilon_3)\right), \tag{3.5b}$$
$$\psi = \text{atan2}\left(2(\epsilon_1\epsilon_2 + \eta\epsilon_3), \eta^2 + \epsilon_2^2 - \epsilon_3^2 - \epsilon_1^2\right). \tag{3.5c}$$

The reverse process, conversion from Euler angles to quaternions is given as

$$\eta = \cos(\phi/2)\cos(\theta/2)\cos(\psi/2) + \sin(\phi/2)\sin(\theta/2)\sin(\psi/2) \tag{3.6a}$$
$$\epsilon_1 = \sin(\phi/2)\cos(\theta/2)\cos(\psi/2) - \cos(\phi/2)\sin(\theta/2)\sin(\psi/2) \tag{3.6b}$$
$$\epsilon_2 = \cos(\phi/2)\sin(\theta/2)\cos(\psi/2) + \sin(\phi/2)\cos(\theta/2)\sin(\psi/2) \tag{3.6c}$$
$$\epsilon_3 = \cos(\phi/2)\cos(\theta/2)\sin(\psi/2) - \sin(\phi/2)\sin(\theta/2)\cos(\psi/2) \tag{3.6d}$$

Given a quaternion describing the orientation of $\{a\}$ relative to $\{b\}$, denoted $\boldsymbol{q}_b^a$, a rotation matrix can be obtained by

$$\boldsymbol{R}_b^a(\boldsymbol{q}_b^a) = \begin{bmatrix} 1-2(\epsilon_2^2+\epsilon_3^2) & 2(\epsilon_1\epsilon_2 - \epsilon_3\eta) & 2(\epsilon_1\epsilon_3 + \epsilon_2\eta) \\ 2(\epsilon_1\epsilon_2 + \epsilon_3\eta) & 1-2(\epsilon_1^2+\epsilon_3^2) & 2(\epsilon_2\epsilon_3 - \epsilon_1\eta) \\ 2(\epsilon_1\epsilon_3 - \epsilon_2\eta) & 2(\epsilon_2\epsilon_3 + \epsilon_1\eta) & 1-2(\epsilon_1^2+\epsilon_2^2) \end{bmatrix}, \tag{3.7}$$

In some cases, both translation and rotation are involved in a transformation. Then the expression for transforming a vector $\boldsymbol{p}$ from a frame $\{a\}$ to a frame $\{b\}$ becomes

$$\boldsymbol{p}^b = \boldsymbol{R}_a^b \boldsymbol{p}^a + \boldsymbol{t}_{\boldsymbol{ba}}^{\boldsymbol{b}}, \tag{3.8}$$

where $\boldsymbol{R}_a^b$ is the same as in (3.2) and $\boldsymbol{t}_{\boldsymbol{ba}}^{\boldsymbol{b}}$ is the vector from the origin of $\{b\}$ to the origin of $\{a\}$, resolved in $\{b\}$.

The transformation in (3.8) can be expressed as a matrix equation by the use of homogenous coordinates (Bloomenthal and Rokne, 1994). The homogenous coordinates, denoted $\widetilde{(\cdot)}$, of a point $\boldsymbol{p}$ is defined as:

$$\widetilde{\boldsymbol{p}} := \begin{bmatrix} \boldsymbol{p} \\ 1 \end{bmatrix}. \tag{3.9}$$

Furthermore, the transformation of a point in frame $\{a\}$ to a frame $\{b\}$ can be expressed as:

$$\widetilde{\boldsymbol{p}}^b = \boldsymbol{T}_a^b \widetilde{\boldsymbol{p}}^a, \tag{3.10}$$

where $\boldsymbol{T}_a^b$ is defined as

$$\boldsymbol{T}_a^b := \begin{bmatrix} \boldsymbol{R}_a^b & \boldsymbol{t}_{\boldsymbol{ba}}^{\boldsymbol{b}} \\ \boldsymbol{0}_{1\times 3} & 1 \end{bmatrix}, \tag{3.11}$$

where $\boldsymbol{0}_{1\times 3}$ is a zero row vector.

An often used matrix transformation in sensor fusion is the *Skrew Symmetric Matrix* $\boldsymbol{S}$, which is a part of the Lio group $SS(3)$ (Fossen, 2021, p.24). Given a vector $\boldsymbol{z} = (z_1, z_2, z_3)^\intercal$, then $\boldsymbol{S}(\boldsymbol{z})$ is defined as:

$$\boldsymbol{S}(\boldsymbol{z}) := \begin{bmatrix} 0 & -z_3 & z_2 \\ z_3 & 0 & -z_1 \\ -z_2 & z_1 & 0 \end{bmatrix}. \tag{3.12}$$

From Geodetic to ECEF

Measurements from GNSS are often received in the Geodetic coordinate system but are usually wanted in the local ENU or NED frame. In order for this transformation to take place, an intermediate step is required. This step is the transformation from the Geodetic coordinate system, represented by latitude, longitude, and height, to the ECEF coordinate system. In order for such a transformation to take place, some parameters related to the Geodetic coordinate system need to be defined.

1. The semi-major axis a
2. The first eccentricity e
3. The prime vertical radius of curvature R_N

Their numerical values are given as follows:

$$a = 6378137, \tag{3.13a}$$
$$e = 0.08181919, \tag{3.13b}$$
$$R_N = \frac{a}{\sqrt{1 - e^2 \sin(\lambda)^2}}. \tag{3.13c}$$

Given the origin coordinates of a local frame $\{u\}$ in the Geodetic coordinate system, in terms of λ, τ, and h, the resulting coordinates in the ECEF frame is given by:

$$\boldsymbol{p}^e_{eu} = \begin{pmatrix} (R_N + h)\cos(\lambda)\cos(\tau) \\ (R_N + h)\cos(\lambda)\sin(\tau) \\ (R_N(1 - e^2) + h)\sin(\lambda) \end{pmatrix}, \tag{3.14}$$

where e and R_N are given by (3.13).

From ECEF to ENU

Given a set of coordinates represented in ECEF, a transformation following (3.8) is needed to obtain the local ENU representation. This is done by first defining the origin of the local ENU frame in ECEF coordinates, given a set of geodetic coordinates. This point is denoted as $\boldsymbol{p}^e_{eu}$, and obtained by feeding the geodetic

coordinates into (3.14). Furthermore, the rotation matrix from (3.8) is defined as

$$\boldsymbol{R}_e^u = \begin{bmatrix} -\sin(\lambda_0) & \cos(\lambda_0) & 0 \\ -\sin(\tau_0)\cos(\lambda_0) & -\sin(\tau_0)\sin(\lambda_0) & \cos(\tau_0) \\ \cos(\tau_0)\cos(\lambda_0) & \cos(\tau_0)\sin(\lambda_0) & \sin(\tau_0) \end{bmatrix}, \tag{3.15}$$

where τ_0 and λ_0 are latitude and longitude of the local ENU origin, respectively. Moreover, the position of the vehicle body in local ENU coordinates can be expressed as

$$\boldsymbol{p}_{ub}^u = \boldsymbol{R}_e^u(\boldsymbol{p}_{eb}^e - \boldsymbol{p}_{eu}^e) \tag{3.16}$$

ENU to Body

Given a rotation matrix describing the orientation of the body coordinate system relative to the local ENU frame, and the vector between the two origins, the transformation can be expressed as in (3.8). Given the position of a point a in the ENU frame $\boldsymbol{p}_{ua}^u$, the position in the body frame is given by

$$\boldsymbol{p}_{ba}^b = \boldsymbol{R}_u^b \boldsymbol{p}_{ua}^u + \boldsymbol{p}_{bu}^b. \tag{3.17}$$

3.2 Sensor modeling

In order to integrate sensor data into a SLAM algorithm, it is necessary to have a mathematical model of how the sensor behaves. Some sensor does not measure relevant physical quantities directly, but a relation to them. This section includes mathematical models for IMU, LiDAR, and GNSS. The material is based on the work in Fossen (2021, Ch. 14) and Storli (2022). Measured quantities are denoted $\hat{(\cdot)}$.

3.2.1 Inertial Measurement Unit (IMU)

IMU is a sensor that measures multiple quantities (Fossen, 2021, Ch. 14). Most IMUs include three accelerometers, three gyroscopes, and a temperature sensor. Some IMUs also include other sensors such as inclinometers and magnetometers. In this section, the focus will be on the mathematical models relating the output of accelerometers and gyroscopes with interesting physical quantities. It should be noted that these relations are simplifications, and actual sensor behavior may under certain circumstances behave very differently from the presented models.

Accelerometer

Accelerometers measure *specific force* $\boldsymbol{f}$, which can be related to the true acceleration $\boldsymbol{a}$ through

$$\hat{\boldsymbol{f}}_{ub_i}^{b_i} = \boldsymbol{R}_u^{b_i}(\boldsymbol{a}_{ub_i}^u - \boldsymbol{g}^u) + \boldsymbol{b}_{acc}^{b_i} + \boldsymbol{w}_{acc}^{b_i}, \tag{3.18}$$

where $\boldsymbol{R}_u^{b_i} \in SO(3)$ is the rotation matrix from the local ENU frame to the IMU frame, $\boldsymbol{a}_{ub_i}^u$ is acceleration of the IMU frame relative to the local ENU frame, $\boldsymbol{g}^u$ is the gravitational vector, $\boldsymbol{b}_{acc}^{b_i}$ is accelerometer bias, and $\boldsymbol{w}_{acc}^{b_i}$ is measurement noise.

Gyroscope

A gyroscope has a closer relationship to a meaningful physical quantity, which is angular velocity. The mathematical relationship between what the gyroscope measures $\hat{\boldsymbol{\omega}}$ and the true angular velocity $\boldsymbol{\omega}$ is given as

$$\hat{\boldsymbol{\omega}}_{ub_i}^{b_i} = \boldsymbol{\omega}_{ub_i}^{b_i} + \boldsymbol{u}_{gyr}^{b_i} + \boldsymbol{w}_{gyr}^{b_i}, \tag{3.19}$$

where $\hat{\boldsymbol{\omega}}_{nb_i}^{b_i}$ is the measured angular velocity received from the gyroscope, $\boldsymbol{b}_{gyr}^{b_i}$ is the gyroscope bias, and $\boldsymbol{w}_{gyr}^{b_i}$ is the measurement noise. It should be noted that gyroscopes and accelerometers measure quantities relative to the inertial frame $\{i\}$, and the relation to $\{u\}$ is a simplification.

3.2.2 LiDAR

LiDAR is a sensor that measures the distance to nearby objects (Raj et al., 2020). This is done by emitting light rays which are reflected off the surroundings and returned to the sensor. Based on the time it takes for the light ray to return, the distance can be calculated. Since the speed of light is constant, the distance d can be expressed trough

$$d = \frac{c\Delta t}{2}, \tag{3.20}$$

where $c = 299792458\frac{m}{s}$ is the speed of light and Δt is the duration for the light ray emitted until it is returned (Storli, 2022). Based on the direction the light ray is sent out, a vector can express the position of the object causing the reflection. In the LiDAR coordinate frame, this position of a point o can be defined as

$$\boldsymbol{p}_{b_l o}^{b_l} = \begin{pmatrix} x_o \\ y_o \\ z_o \end{pmatrix}. \tag{3.21}$$

Alternatively, the point can be expressed in polar coordinates. Represented by the three following parameters

$$\boldsymbol{p}_{b_l o}^{polar} = \begin{pmatrix} \theta_x \\ \theta_z \\ d \end{pmatrix}, \tag{3.22}$$

where d is the distance given by (3.20), and θ_x and θ_z are the angles from the x- and z-axis, respectively.

By emitting multiple light rays in various directions, a collection of points that describe the surrounding environment can be obtained, commonly known as a *pointcloud* (X. Huang et al., 2021). Since these light rays are not emitted simultaneously, there is a time difference between the resulting points. A LiDAR typically comprises multiple channels, representing the vertical resolution of the resulting pointcloud. Each channel consists of a laser and a light detector. All lasers usually emit light simultaneously. In most LiDARs, these lasers are mounted on a rotating platform, enabling the mapping of a full 360° view of the surroundings. The horizontal resolution, which determines the number of laser shots during a complete rotation, can often be adjusted in a LiDAR. The time difference between two points in the horizontal direction can be calculated as

$$\delta t = \frac{1}{r_h f}, \tag{3.23}$$

where r_h represents the horizontal resolution, and f denotes the rotation rate of the sensor in rounds per second (Storli, 2022).

3.2.3 Global Navigation Satellite Systems (GNSS)

GNSS refers to a group of navigation systems that enable users to obtain a three-dimensional position through passive ranging using radio signals emitted by satellites in orbit (Groves, 2013)(Misra and Enge, 2012). GNSS serves as a self-positioning system, wherein the user equipment calculates the position solution without transmitting any signals for positioning purposes. Accurate time information can also be obtained by the processing of these signals. This section presents concepts and mathematical models related to GNSS relevant to the work presented in this thesis. The concepts of RTK and Post processed kinematic (PPK) is presented, which involves a more accurate method of obtaining position estimates by utilizing antennas with known positions. Furthermore, the most basic measurement, which provides 3D-position coordinates and time information is presented. By utilizing multiple receivers with a fixed position on a vehicle, information about the vehicle's orientation can be obtained. The information presented in this section, and for a more comprehensive take on GNSS, the reader is referred to Teunissen and Montenbruck (2017).

Real Time Kinematic and Post Process Kinematic

By utilizing a *base station*, which is a receiver with its exact position known, the user can remove a lot of inaccuracies that are usually present when using GNSS. This is achieved by providing the user antenna with data from the base station. Based on data from the satellites and the base station, a more accurate position estimate can be provided. RTK is the name for this type of positioning when done in real-time. This process can also be done in post-process by collecting data from satellites and then downloading baste station data from a database. When done

post-process, the method is referred to as PPK. In the rest of the thesis, both concepts will be referred to as RTK.

Single position measurement

The single-position measurement from a GNSS antenna is often given in geodetic coordinates in terms of latitude, longitude, and height, and can be converted to ECEF coordinates by (3.14). A mathematical model for the position of the moving base antenna can be expressed as

$$\hat{\boldsymbol{p}}^{e}_{eb_{mb}} = \boldsymbol{p}^{e}_{eb_{mb}} + \boldsymbol{b}^{e}_{gnss_{mb}} + \boldsymbol{w}^{e}_{gnss_{mb}}, \tag{3.24}$$

where $\hat{\boldsymbol{p}}^{e}_{eb_{mb}}$ is the measured position, $\boldsymbol{p}^{e}_{eb_{mb}}$ is the true position vector, $\boldsymbol{b}^{e}_{gnss_{mb}}$ and $\boldsymbol{w}^{e}_{gnss_{mb}}$ is bias and measurement noise related to the Moving Base receiver, respectively.

Relative position measurement

Utilizing two GNSS antennas, where one of them functions as a base station sending signals to the other, the relative position between the two antennas can be estimated. This is usually done by referencing the position of one antenna to the other, by defining a local ENU or NED frame on one of them and then expressing the position of the other antenna in these local coordinates. Denoting the receiver giving relative position measurement as *Rover*, with frame $\{b_r\}$, and the receiver defining the measurement reference frame, Moving Base, with frame $\{b_{mb}\}$. Then, the relation can be modeled as

$$\hat{\boldsymbol{p}}^{u}_{b_{mb}b_r} = \boldsymbol{p}^{u}_{b_{mb}b_r} + \boldsymbol{b}^{u}_{gnss_r} + \boldsymbol{w}^{u}_{gnss_r}, \tag{3.25}$$

where $\hat{\boldsymbol{p}}^{u}_{b_{mb}b_r}$ is the relative position measurement between the Moving Base and the Rover antennas, $\boldsymbol{p}^{u}_{b_{mb}b_r}$ is the true relative position, $\boldsymbol{b}^{u}_{gnss_r}$ and $\boldsymbol{w}^{u}_{gnss_r}$ is bias and measurement noise related to the Rover receiver, respectively.

Full pose based on three GNSS antennas

By mounting three antennas on a rigid-body vehicle, full orientation between the body and the local ENU frames can be estimated. A prerequisite is that the vectors describing the relative positions between the GNSS antennas are not parallel and that their position in the body frame is known. Denote the three antennas as *Moving base*, *Rover1* and *Rover2*, where Rover1 and Rover2 give position measurements relative to Moving base, as modeled in (3.25). If the body frame of the vehicle is assumed rigid, the relative position of the antennas is constant in the body frame and can be measured manually.

The method for calculating the full orientation of the vehicle based on the relative position vectors is called *QUEST* and is described in detail in Shuster and Oh (1981). Since the method is quite comprehensive, and the details are not relevant

to the work presented in this thesis, the reader is referred to the paper for a full understanding. A MATLAB implementation is available on GitHub[‡].

Lever Arm compensation

In most of the cases when using GNSS receivers in combination with other sensors, the origins of the sensor frames are not identically at the same location, resulting in a lever arm between the sensors. When estimating the position of a vehicle, the interesting quantity is usually $\boldsymbol{p}^u_{ub}$. The *body* coordinate system may not be located at the same point as the GNSS frame in (3.24). To compensate for this, the lever arm between the body frame origin and the GNSS antenna has to be taken into consideration. Given the lever arm expressed in body frame $\boldsymbol{l}^b_{bb_{mb}}$, the relation between the body position and the GNSS measurement is given as

$$\hat{\boldsymbol{p}}^u_{ub_{mb}} = \boldsymbol{p}^u_{ub} + \boldsymbol{R}^u_b \boldsymbol{l}^b_{bb_{mb}}, \tag{3.26}$$

where $\boldsymbol{R}^u_b$ is the rotation matrix from the body frame to the local ENU frame. As long as the vehicle has a rigid body, the lever arm is constant in the body frame.

3.3 Graph based SLAM

The SLAM problem involves estimating the trajectory of the robot and creating a map of the environment as the robot moves within it (Grisetti et al., 2010; X. Xu et al., 2022b). To handle the inherent sensor measurement noise, presented in Section 3.2, probabilistic tools are commonly used in SLAM. The robot's movement in an unknown environment is represented by a sequence of random variables $\boldsymbol{x}_{1:T} = (\boldsymbol{x}_1, ..., \boldsymbol{x}_T)$, where the subscript denotes at what time the position was valid. For simplicity, the position $\boldsymbol{x}$ is refered to the body coordinate frame expressed in ENU, $\boldsymbol{x} := \boldsymbol{x}^u_{ub}$. Along its path, the robot collects a series of measurements $\boldsymbol{z}_{1:T} = (\boldsymbol{z}_1, ..., \boldsymbol{z}_T)$, this can be from IMU, LiDARs and cameras, as well as other sensors. These measurements are also referenced to the body frame, $\boldsymbol{z} := \boldsymbol{z}^b$. The goal of solving the SLAM problem is to estimate the posterior probability density function (PDF) p of the robot's trajectory $\boldsymbol{x}_{1:T}$ and the map of the environment $\boldsymbol{m}$, given all the measurements and an initial position $\boldsymbol{x}_0$. This can be expressed as the following PDF:

$$p(\boldsymbol{x}_{1:T}, \boldsymbol{m} | \boldsymbol{z}_{1:T}, \boldsymbol{x}_0) \tag{3.27}$$

The initial position $\boldsymbol{x}_0$ determines the map's reference point and can be chosen arbitrarily. In the case of using global coordinates, such as ECEF, $\boldsymbol{x}_0$ can be based on GNSS measurements (Grisetti et al., 2010). The robot's poses $\boldsymbol{x}_{1:T}$ are commonly expressed as 2D or 3D transformations in $SE(2)$ or $SE(3)$, respectively (Dellaert and Kaess, 2017). On the other hand, the map can be represented in various forms. It can be represented by a set of landmarks, a set of points where each point have a corresponding feature descriptor, or a pointcloud (Castellanos et

[‡]Quest MATLAB implementation: https://github.com/cybergalactic/MSS/blob/master/GNC/quest.m

al., 2007). Both Cartesian and polar coordinates can be used for this. Usually, the map is described in an earth-fixed frame. The benefit of this is that the coordinates stay constant, as long as the elements in the map do not move. However, there are benefits to keeping the map in the body frame, concerning observability in state estimation (G.P. Huang et al., 2010). Alternatively, raw sensor measurements can be used to construct the map. The choice of map representation depends on factors such as the sensor types employed, the environment's characteristics, and the estimation algorithm being utilized.

Estimating the posterior in (3.27) involves working with high-dimensional state spaces, which would be intractable without a well-defined structure in the SLAM problem (Grisetti et al., 2010). This structure emerges from certain commonly made assumptions, namely the static world assumption and the Markov assumption. The static world assumption states that the coordinates of the map are constant in the local ENU frame. In other words, $\boldsymbol{m}^u$ is constant. The Markov assumption assumes that the current state of the system contains all the information needed to predict future states. In other words, it assumes that the future states are conditionally independent of the past states given the current state.

One representation of the SLAM problem is through the "graph-based" formulation, which emphasizes the structure between sensor measurements and states to be estimated (Grisetti et al., 2010). In graph-based SLAM, the robot poses are modeled as nodes in a graph, labeled with their positions in the environment (Lu and Milios, 1997; Konolige et al., 2010). Spatial constraints between poses arising from measurements $\boldsymbol{z}_t$ which are encoded as edges between the nodes. A constraint is represented by a probability distribution over the relative transformations between the poses.

The process of solving the SLAM problem using graph-based methods encompasses two primary tasks. The first task involves constructing the graph by defining the states to be estimated and establishing the relationships between these states and the incoming measurements. This step is commonly known as the "front-end" task. Once the graph is constructed, the subsequent task is to determine the pose estimates that best represent the likely configuration of poses based on the graph edges. This task is often referred to as the "back-end" SLAM and is independent of the particular sensors employed.

3.3.1 Factor graphs

Factor graphs are a popular framework for structuring and solving the SLAM problem (Brekke, 2021, Ch. 12). They are a special type of graph consisting of two kinds of nodes. These two types of nodes are known as variable and factor nodes. All edges go between a variable node and a factor node, which is the property of being a bipartite graph. The variable nodes represent the states of the system, which should be estimated. Factor nodes represent probabilistic relationships between the variable nodes.

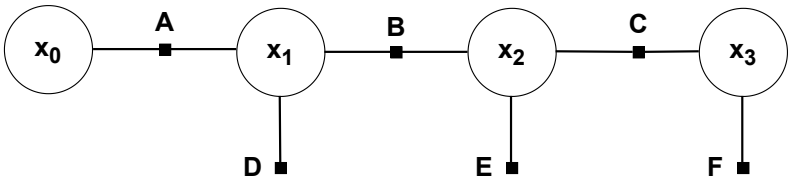


Figure 3.1: A factor graph example, with four variable nodes denoted $\boldsymbol{x}_i$, and six factor nodes, denoted with capital letters.

In the realm of graphical models, factors can be categorized into two primary types: unary factors and binary factors. Unary factors are linked to a single variable node, whereas binary factors connect two variable nodes. Unary factors are commonly used to incorporate absolute measurements, such as those obtained from GNSS. On the other hand, binary factors are often employed to integrate odometry measurements, which provide information about the relative motion between two poses.

Figure 3.1 represents a factor graph with three variable nodes $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3\}$, and five factors nodes $\{A, B, C, D, E\}$. Let $f_x(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})$ be the state prediction function for variable $\boldsymbol{x}_t$ given $\boldsymbol{x}_{t-1}$, and $f_z(\boldsymbol{z}_t|\boldsymbol{x}_t)$ the measurement prediction function for $\boldsymbol{z}_t$ given $\boldsymbol{x}_t$. The factors in the graph can then be represented as:

$$\xi_A(\boldsymbol{x}_0, \boldsymbol{x}_1) = f_x(\boldsymbol{x}_1|\boldsymbol{x}_0)p(\boldsymbol{x}_0) \tag{3.28a}$$
$$\xi_B(\boldsymbol{x}_1, \boldsymbol{x}_2) = f_x(\boldsymbol{x}_2|\boldsymbol{x}_1) \tag{3.28b}$$
$$\xi_C(\boldsymbol{x}_2, \boldsymbol{x}_3) = f_x(\boldsymbol{x}_3|\boldsymbol{x}_2) \tag{3.28c}$$
$$\xi_D(\boldsymbol{x}_1) = f_z(\boldsymbol{z}_1|\boldsymbol{x}_1) \tag{3.28d}$$
$$\xi_E(\boldsymbol{x}_2) = f_z(\boldsymbol{z}_2|\boldsymbol{x}_2) \tag{3.28e}$$
$$\xi_F(\boldsymbol{x}_3) = f_z(\boldsymbol{z}_3|\boldsymbol{x}_3), \tag{3.28f}$$

where $p(\boldsymbol{x}_0)$ is the prior PDF of $\boldsymbol{x}_0$.

The joint PDF of the state variables in Figure 3.1 is given by the sum of all factors:

$$p(\boldsymbol{x}_{1:3}) = \frac{1}{Z}\xi_A(\boldsymbol{x}_0, \boldsymbol{x}_1)\xi_B(\boldsymbol{x}_1, \boldsymbol{x}_2)\xi_C(\boldsymbol{x}_2, \boldsymbol{x}_3)\xi_D(\boldsymbol{x}_1)\xi_E(\boldsymbol{x}_2)\xi_D(\boldsymbol{x}_3), \tag{3.29}$$

here Z is a normalization constant. More generally, the joint PDF of a factor graph is represented as:

$$p(\boldsymbol{x}) = \frac{1}{Z}\prod_{s \in S}\xi_s(\boldsymbol{x}_s), \tag{3.30}$$

where $\boldsymbol{x}$ is all state variables, and $\boldsymbol{x}_s$ is all state variables that are corresponding to the factor s in the set of all factors S.

MAP inference for nonlinear factor graphs

The goal of constructing a factor graph is to obtain estimates for the state variables (Dellaert and Kaess, 2017). This can be done by finding the Maximum a posteriori

(MAP) estimate of (3.30), formulated as

$$\boldsymbol{x}^{MAP} = \arg\max_{\boldsymbol{x}} p(\boldsymbol{x}) \tag{3.31a}$$

$$= \arg\max_{\boldsymbol{x}} \prod_{s \in S} \xi_s(\boldsymbol{x}_s) \tag{3.31b}$$

Assuming a Gaussian noise model for the measurement prediction function, with covariance matrix $\boldsymbol{\Sigma}_z$ then the unary factors can be represented as

$$\xi_s(\boldsymbol{x}_s) \propto \exp\{-\frac{1}{2}\|f_z(\boldsymbol{x}_s) - \boldsymbol{z}_s\|^2_{\boldsymbol{\Sigma}_{z,s}}\}, \tag{3.32}$$

where $\|\cdot\|^2_\Sigma$ denotes the squared Mahalanobis distance. Further, assuming a Gaussian state prediction function with covariance matrix $\boldsymbol{\Sigma}_x$, the binary factors can be represented as

$$\xi_s(\boldsymbol{x}_s) \propto \exp\{-\frac{1}{2}\|\boldsymbol{x}_s - f_x(\boldsymbol{x}_{s-1})\|_{\boldsymbol{\Sigma}^2_{x,s}}\}. \tag{3.33}$$

Taking the negative log of (3.31) and removing the $\frac{1}{2}$ factor, results in the following nonlinear lest-squares problem:

$$\boldsymbol{x}^{MAP} = \arg\min_{\boldsymbol{x}} \sum_{s \in B} \|\boldsymbol{x}_s - f_x(\boldsymbol{x}_{s-1})\|^2_{\boldsymbol{\Sigma}_{x,s}} + \sum_{s \in U} \|f_z(\boldsymbol{x}_s) - \boldsymbol{z}_s\|^2_{\boldsymbol{\Sigma}_{z,s}}, \tag{3.34}$$

where U and B are the set of all unary and binary factors, respectively.

3.3.2 Optimization and smoothing

*The benefit of the factor graph formulation is how the posterior function can be expressed as a product of independent likelihood and probability functions, as in (3.30). In order to find the MAP estimate of a factor graph, an optimization algorithm can be utilized. When optimizing a factor graph, the choice of optimization algorithm often comes down to the complexity of the graph and runtime requirements. Steepest descent, Gauss-Newton, and Levenberg-Marquardt are some of the most used optimization methods for finding the MAP estimate of a factor graph (Dellaert and Kaess, 2017). These methods are based on gradients and involve computing the derivatives of the cost function. All methods require evaluating the Jacobian of the cost function. Moreover, Gauss-Newton necessitates the computation of the Hessian, whereas Levenberg-Marquardt only requires an approximation of the Hessian.

As a system operates, the factor graph representing all measurements and states will grow rapidly (Brekke, 2021, Ch. 10). The process of smoothing, which involves optimizing with respect to older states, is often computationally expensive. An approach to apply smoothing optimization in real-time is by use of *fixed-lag smoothing*. This involves optimizing for a set of states stretching n timesteps

*This and subsequent paragraphs are taken from the specialization project, Storli (2022).

backward in time. The optimization variable can be expressed as $\boldsymbol{x}_{k-n:k}$, where k is the current timestep. In this optimization, the states prior to timestep $k-n$ are considered fixed. Fixed-lag smoothing is also referred to as sliding window optimization. There exists other types of smoothing principles such as fixed-point and fixed interval. Fixed-point smoothing considers the optimization of the state variables at a certain fixed point in time. Fixed-interval smoothing includes optimizing for state variables in a fixed interval independent of the current timestep.

Today, there exist multiple approaches to smoothing. What most of them have in common, is that they exploit probabilistic graphical models. Some of these methods include the *Rauch-Tung-Striebel (RTS) smoother* (Särkkä, 2008), which is a fix-lag smoother for Gaussian-linear systems, and *incremental smoothing and mapping 2* (iSAM2), which utilizes a Bayes tree. iSAM2 is possibly the most used method for real-time applications (Kaess et al., 2012).

3.3.3 Loop closure

Loop closure is the ability of an estimation algorithm to recognize that it has visited the same location in the past. This recognition results in a new constraint for the state estimation process. Such constraints can be added as a factor to the factor graph, resulting in more information that can be utilized in the optimization. However, if an algorithm needs to evaluate if a new location has been visited earlier, this will occupy computational resources.

There exist multiple algorithms which do not incorporate loop closure functionality, and these are often referred to as odometry methods. A strong argument for incorporating loop closure functionality in an estimation method is due to its ability to correct for drift. Using only IMU and LiDAR will often result in a drift since no absolute measurements, such as GNSS, are provided.[†]

[†]End of reference from Storli (2022).

3.4 Liorf - LiDAR-inertial SLAM

Liorf is an extension of LIO-SAM, which is short for *Tightly-coupled Lidar Inertial Odometry via Smoothing and Mapping*. LIO-SAM is an open-source[‡] algorithm documented in a conference paper (Shan et al., 2020). Liorf is also an open source algorithm[§], but is not documented in a scientific paper. Both methods are based on factor graph optimization for state estimation. This section describes the functionality of LIO-SAM, which the Liorf algorithm is built upon. The main difference between the methods is the code implementation, while the mathematics and mechanisms still follow the same principles.

3.4.1 System overview

LIO-SAM supports sensor inputs from IMU, LiDAR, and GNSS. IMU measurements from a gyroscope and an accelerometer are required, while heading measurements from magnetometers are also supported. 3D pointcloud data from LiDAR, with timestamps corresponding to each point, is required. Single position measurements from GNSS are an optional input but are recommended to avoid large drift in pose estimates.

Two coordinate frames are used in the estimation process. This is the world frame $\{u\}$ and the robot body frame $\{b\}$. Additionally, the sensors have their own coordinate systems, as described in Table 3.1, but the measurements are transformed to the body and world frame by the use of (3.2). For convenience, the body frame is chosen to coincide with the IMU frame $\{b_i\}$. The state to be estimated is defined as follows:

$$\boldsymbol{x} = [\boldsymbol{R}^u_b, \boldsymbol{p}^u_{ub}, \boldsymbol{v}^u_{ub}, \boldsymbol{b}^{b_i}_{IMU}], \tag{3.35}$$

where $\boldsymbol{R}^u_b \in SO(3)$ is a rotation matrix from body to world frame, $\boldsymbol{p}^u_{ub}$ is the position, $\boldsymbol{v}^u_{ub}$ is the velocity, and $\boldsymbol{b}^{b_i}_{IMU}$ is the IMU gyroscope and accelerometer bias (Shan et al., 2020). When not using GNSS, an initial orientation has to be provided to achieve a correct reference to a local ENU frame.

3.4.2 Factor graph

The mechanism behind the state estimation in LIO-SAM is factor graph optimization, as described in Section 3.3. All sensor measurements are processed to a suitable format to fit in the factor graph, depicted in Figure 3.2. The variable nodes consist of the states presented in (3.35), and the subscript denotes the timestep the states are estimated at. The red factor represents the factor generated by the IMU measurements, called *IMU preintegration factor*. The green factor is the relative movement estimate between two LiDAR scans, named *LiDAR odometry factor*. The yellow factor represents the GNSS factor and is obtained by single position

[‡]LIO-SAM implementation: https://github.com/TixiaoShan/LIO-SAM

[§]Liorf implementation: https://github.com/YJZLuckyBoy/liorf

measurements, named *GPS factor*. The name "GPS" is used since the GTSAM library, which the algorithm is built upon, uses it. However, the name "GNSS" is more correct. The GNSS measurement is represented by the orange circle. The black factor is the *Loop closure factor* and is a constraint generated by the robot recognizing it has been at the same location earlier. This factor is used to correct for drift in the map and trajectory and ensures that the map is consistent globally. In the following sections, the factors are explained in greater detail.

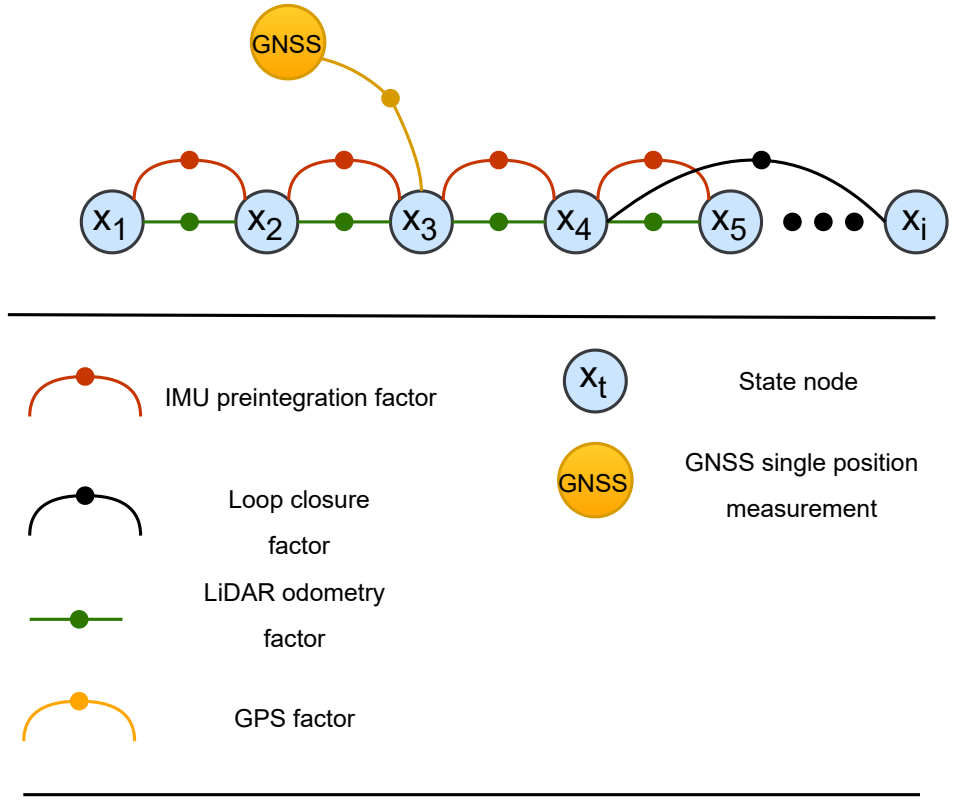


Figure 3.2: The factor graph utilized in Liorf and LIO-SAM is depicted in the upper part of the figure. The lower part of the figure provides a detailed explanation of the constituent elements comprising the graph. Inspired from Shan et al. (2020)

IMU preintegration factor

*IMU preintegration utilizes a mathematical framework derived from (3.18) and (3.19). These equations are used to express the future pose and velocity of the system, expressed as

$$\boldsymbol{v}_{k+h} = \boldsymbol{v}_k + \boldsymbol{g}h + \boldsymbol{R}_k(\hat{\boldsymbol{f}}_k - \boldsymbol{b}_{acc,k} - \boldsymbol{w}_{acc,k})h \tag{3.36a}$$

$$\boldsymbol{p}_{k+h} = \boldsymbol{p}_k + \boldsymbol{v}_k h + \frac{1}{2}\boldsymbol{g}h^2 + \frac{1}{2}\boldsymbol{R}_k(\hat{\boldsymbol{f}}_k - \boldsymbol{b}_{acc,k} - \boldsymbol{w}_{acc,k})h^2 \tag{3.36b}$$

$$\boldsymbol{R}_{k+h} = \boldsymbol{R}_k \exp((\hat{\boldsymbol{\omega}}_k - \boldsymbol{b}_{gyr,k} - \boldsymbol{w}_{gyr,k})h) \tag{3.36c}$$

where $\boldsymbol{p} := \boldsymbol{p}^u_{ub_i}$, $\boldsymbol{v} := \boldsymbol{v}^u_{ub_i}$, $\boldsymbol{R} := \boldsymbol{R}^u_{b_i}$, and $\boldsymbol{b}$ and $\boldsymbol{w}$ represents the bias and measurement noise, respectively. $\boldsymbol{g} := \boldsymbol{g}^u$ is the gravity vector, and h is the length of time into the future. In order for this propagation to be accurate, it is assumed that the true acceleration and angular velocity are constant during the time interval.

The relative motion estimates between two timesteps i and j, denoted $\Delta\boldsymbol{v}_{i,j}$, $\Delta\boldsymbol{p}_{i,j}$ and $\Delta\boldsymbol{R}_{i,j}$ can be expressed as

$$\Delta\boldsymbol{v}_{i,j} = \boldsymbol{R}_i^\intercal(\boldsymbol{v}_j - \boldsymbol{v}_i - \boldsymbol{g}\Delta t_{i,j}) \tag{3.37a}$$

$$\Delta\boldsymbol{p}_{i,j} = \boldsymbol{R}_i^\intercal(\boldsymbol{p}_j - \boldsymbol{p}_i - \boldsymbol{v}\Delta t_{i,j} - \frac{1}{2}\boldsymbol{g}\Delta t_{i,j}^2) \tag{3.37b}$$

$$\Delta\boldsymbol{R}_{i,j} = \boldsymbol{R}_i^\intercal\boldsymbol{R}_j, \tag{3.37c}$$

where $\Delta t_{i,j}$ represent the lengt of time between i and j. This time interval is usually the length of time between the insertion of two state nodes. In other words, if a new state node is added each time a LiDAR scan comes in, the IMU measurements are integrated during this time interval. This method is based on the work in Forster et al. (2016). (3.37) is the basis for the relative motion estimate used in the IMU preintegration factor.

LiDAR odometry factor

LIO-SAM is an indirect SLAM method, which means that it extracts features from the pointcloud prior to solving the optimization problem. The alternative would be to use the entire pointcloud in the optimization. Two types of features are extracted, planar features and edge features. Planar features consist of smooth structures in the pointcloud, while edge features are rougher areas. Prior to the feature extraction, the pointclouds are de-skewed based on the preintegrated motion estimates from the IMU in (3.37). This de-skewing is necessary when the LiDAR undergoes motion during a single scan. (3.23) describes the time it is necessary to preintegrate the IMU measurements to de-skew a single point in the LiDAR scan.

The extracted features in the incoming pointcloud are used to generate a LiDAR odometry factor by estimating a transformation between the current LiDAR

*This and subsequent paragraphs are taken from the specialization project, Storli (2022).

frame and the world frame. This is done by comparing the features in the new scan with the features in the current map representation. A sliding window approach is used to generate a local map of the environment, consisting of extracted features from the n latest pointclouds. The feature extraction and scan matching methods are described in Zhang and Singh (2017). The last step involves solving an optimization problem to minimize predefined distance metrics between the corresponding features in the map and the new pointcloud. These error metrics are functions of the relative transformation between the map and the new pointcloud, which is the variable to be optimized with respect to. The reader is referred to Shan et al. (2020) for a detailed description of these error metrics. Gauss-Newton is used to solve this optimization problem. †

GPS position factor

The Global Positioning System (GPS) position factor is the simplest factor, due to the small amount of preprocessing required before being added to the factor graph. The GNSS measurement comes in as latitude, longitude, and height. A transformation to the local ENU frame is done based on the equations presented in Section 3.1.2. This measurement is then compared to the current position estimate and added as a constraint to the factor graph. A weakness of this factor is the fact that it does not compensate for the potential lever arm between the IMU frame and the GNSS antenna position.

Loop closure factor

*LIO-SAM has the ability to perform loop closure and add loop closure factors to the factor graph. When a new state node is added, state nodes with position estimates close to the new node are investigated. These nodes and their corresponding LiDAR scans are used to construct a feature map. This feature map is used to find the relative transformation between the new node and the node with a similar position estimate. This relative transformation is the basis for the loop closure factor. The process of finding this transformation is similar to what is done for the LiDAR odometry factor (Shan et al., 2020).

3.4.3 Related work

In the literature, there are multiple methods utilizing LiDAR and IMU to solve the SLAM Problem. Some of these include the classical framework, EKF-SLAM (Bailey et al., 2006). This is a popular method and is considered as loosely-coupled (Shan et al., 2020). When it comes to the tight coupling of LiDAR and IMU measurements, there are methods such as LIO-Mapping (LIOM), which jointly optimizes LiDAR and IMU-based motion (Ye et al., 2019). As different from LIO-SAM this method is not applicable for real-time systems.

†End of reference from Storli (2022).

*This and subsequent paragraphs are taken from the specialization project, Storli (2022).

Another method, Lidar odometry and mapping (LOAM), uses IMU and LiDAR measurements in a loosely coupled way (Zhang and Singh, 2014). The preintegrated IMU measurements are used to de-skew the pointcloud data. What differentiates LOAM from LIO-SAM is that the estimated motion from IMU preintegration is not used in the optimization part for estimating LiDAR odometry. A two-step Levenberg-Marquardt optimization method is used in estimating LiDAR odometry. This is in contrast to Gauss-Newton for LIO-SAM. Lightweight and ground-optimized LiDAR odometry and mapping (LeGO-LOAM) is a method proposed for ground vehicle mapping and has the same loose coupling between LiDAR and IMU as LOAM. [†]

[†]End of reference from Storli (2022).

Chapter 4

System Integration

This chapter focuses on the construction of the data collection system for gathering real-world data from a sensor payload. The sensor payload comprises:

- an IMU
- a LiDAR
- three GNSS receivers

In order to have all measurements timestamped to the same time reference, a SentiBoard was used. All three GNSS receivers and the IMU were directly connected to the SentiBoard. The LiDAR, which sends data over ethernet, was connected to the host computer. This resulted in a need for a synchronization mechanism for the LiDAR, such that the sensor data from the LiDAR was timestamped based on the SentiBoard clock. In order to parse the data from the SentiBoard, dedicated software from SentiSystem was used. Most of the software used is based on the Robot Operating System (ROS) platform, which was a necessity since Liorf is implemented in ROS. The car platform used for collecting data is described, and the relative mounting between the sensors is presented. A car platform was chosen due to its relevancy for applied sensor fusion in the industry. GNSS data was collected for ground truth generation, as well as being used in the Liorf GNSS factor comparison. To assess the performance of Liorf with different synchronization primitives for sensor measurements, a custom script was developed and deployed to modify the dataset timestamps.

4.1 Hardware components and integration

The hardware setup used to gather the data is compromised of multiple components ranging from sensors, circuit boards, batteries, DC-AC converter, cables, and a computation platform. In this section, the focus lies on how the different sensors were connected to the host computer. This is what allows for data to flow from the sensors to the host computer software, where the sensor fusion takes place. The key hardware components used in this project are presented in Table 4.1.

Table 4.1: Hardware components used for data gathering.

Type	Model	Firmware version
IMU	STIM300	SWD12404 REV 0
LiDAR	Ouster os1-16-A2	v2.4
GNSS receivers	u-blox f9p	v1.30
GNSS antennas	Survey antenna gps1000	No firmware
SentiBoard	version 1.3	v1.3
Host Computer	Intel NUC 11 TNH	0064/Ubunut 20.04

4.1.1 SentiBoard - synchronization module

The SentiBoard is a circuit board used for synchronizing and timestamping measurements from different sensors to one clock reference (S. M. Albrektsen, 2018). Figure 4.1 shows a SentiBoard v.1.30, which was used in this project. It contains a microprocessor equipped with dedicated hardware peripherals used for timestamping. The hardware oscillator driving the SentiBoard clock oscillates with a frequency of 10^8 Hz. This implies that all events timestamped by the SentiBoard have an accuracy of 10 ns. The SentiBoard is capable of communicating with sensors on different interfaces. It is supplied with three UART, two serial peripheral interface (SPI), two RS232 and one RS422 communication interfaces. Each communication interface can be configured to fit the communication protocol used by the sensor. Interfacing the Sentiboard from a computer is done through Universal Serial Bus (USB).

Each communication interface has a TOV pin and a trigger pin. This allows for TOV- and Trigger-synchronization. Additionally, it is equipped with pure Input Capture (IC) ports, which only consist of two pins, Ground (GND) and TOV. Each sensor packet received by the SentiBoard is tagged with three timestamps, TOV, time of transport (TOT), and time of arrival (TOA). TOV is the time when the dedicated TOV signal flanks. TOA is the point in time when the first byte of the sensor packet has arrived, and TOA is the time when the entire packet has been received.

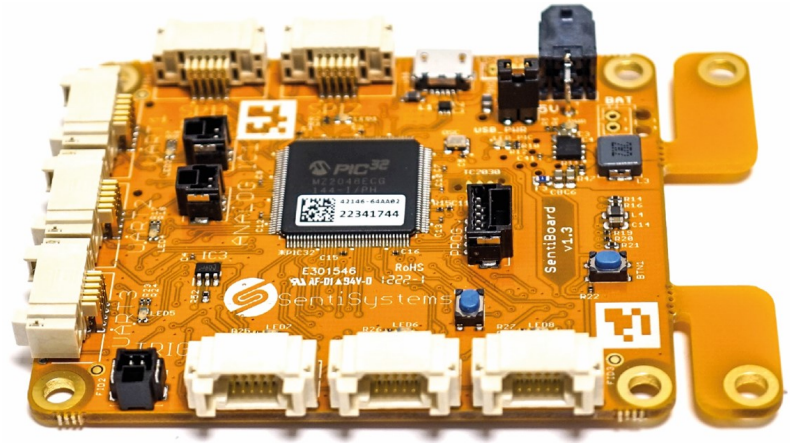


Figure 4.1: Photograph of a SentiBoard, the synchronization and timestamping module used for hardware synchronization. Courtesy of SentiSystems (SentiSystems, 2023)

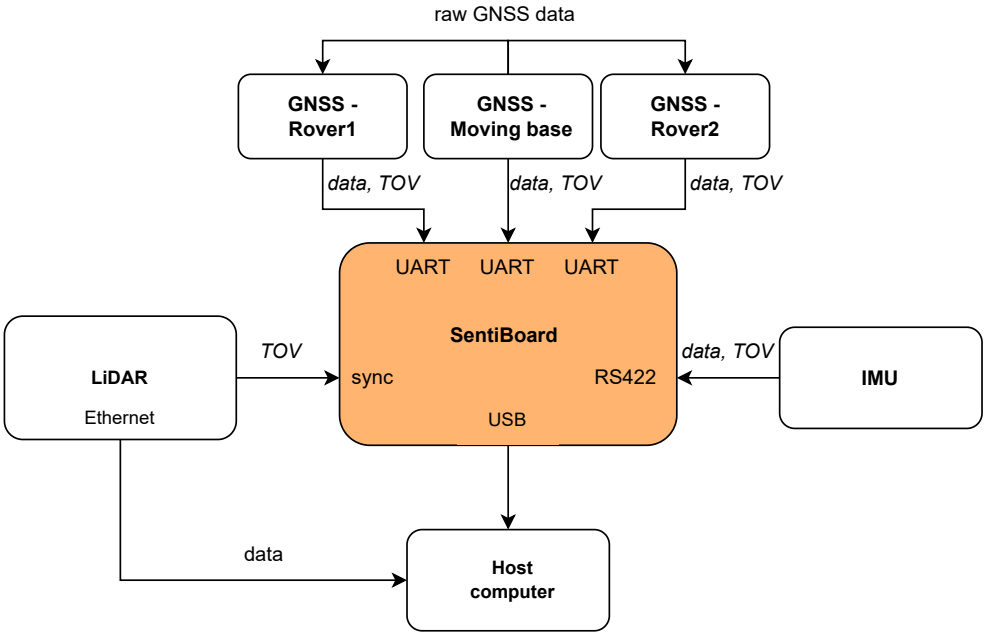


Figure 4.2: The sensor integration diagram showcases the SentiBoard (highlighted in orange) serving as the central interface for all sensors. In the upper part of the diagram, the three GNSS receivers are depicted, with the 'raw GNSS data' signal utilized for RTK generation. The LiDAR interfaces with the SentiBoard solely through a synchronization pin, while the pointcloud data is directly transmitted to the host computer.

4.1.2 Sensor integration

The sensors and hardware components listed in Table 4.1 are illustrated in Figure 4.2, as well as how they are integrated into one system. As the figure shows, all sensors are connected to the SentiBoard. The only sensor that has a direct connection to the host computer is the LiDAR. This is because the SentiBoard has no Ethernet interface, which the LiDAR uses for data transfer. However, in order to synchronize the LiDAR with the SentiBoard, a two-wire connection is present. These wires are GND and a TOV. The TOV signal can be configured to give a pulse each time the internal LiDAR clock wraps a whole second, allowing for 1PPS synchronization. Additionally, it can be configured to give a pulse each time the LiDAR starts a new scan, functioning as a TOV-synchronization for the first set of points in the LiDAR scan. This pulse is based on an encoder inside the LiDAR. The LiDAR is configured to give a full 360° scan 10 times per second. It has a horizontal and vertical resolution of 2048 and 16 points, respectively. All points are timestamped with respect to the internal LiDAR clock when sent to the host computer over Ethernet.

As shown in Figure 4.2, there are three GNSS receivers connected to the SentiBoard. The different receivers are called *Moving Base*, *Rover1* and *Rover2*. Each of these is connected to a UART port on the SentiBoard. Additionally, there is a connection between the three GNSS receivers. This is a single-wire signal, which transmits data from the Moving Base to the two rovers. This is the data that allows for RTK measurements from the two rovers. Each receiver is configured to give one measurement per second. The measurements are synchronized with the SentiBoard through the TOV pin.

The IMU used in this project is interfaced through the RS422 communication protocol, as shown in Figure 4.2. It is synchronized with the SentiBoard through the TOV pin. The sampling rate is set to 500 samples per second for accelerometer and gyroscope. The Host machine used for data storage is an Intel NUC running Ubuntu 20.04. This platform was chosen due to its capabilities for writing data to memory, and its rich set of interface ports. Its size of $10x11x11\ cm^3$ makes it handy for field experiments.

4.2 Car platform and sensor mounting

This section describes the car platform used in the data gathering, and how the sensors were mounted relative to each other.

4.2.1 LiDAR-IMU platform

The LiDAR and IMU used in this project were mounted rigidly on a plastic plate, as shown in Figure 4.3. This is the same platform as the one used in Storli (2022) to gather data. The coordinate systems for LiDAR and IMU are defined as in Figure 4.4. Data received from the two sensors are resolved in these frames. The IMU frame serves as the reference body frame, requiring all measurements to be transformed and expressed relative to that frame. The transformation between the LiDAR and IMU frame follows (3.8), resulting in the following transformation rotation matrix.

$$\boldsymbol{R}_{b_i}^{b_l} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{4.1}$$

and the following translation vector

$$\boldsymbol{t}_{b_l b_i}^{b_l} = [-0.0946, -0.0224, -0.0204]^\intercal \text{m}. \tag{4.2}$$

These numbers were measured by hand, and verified by collecting a dataset and running the LiDAR-IMU-based calibration presented in Lv et al. (2020).

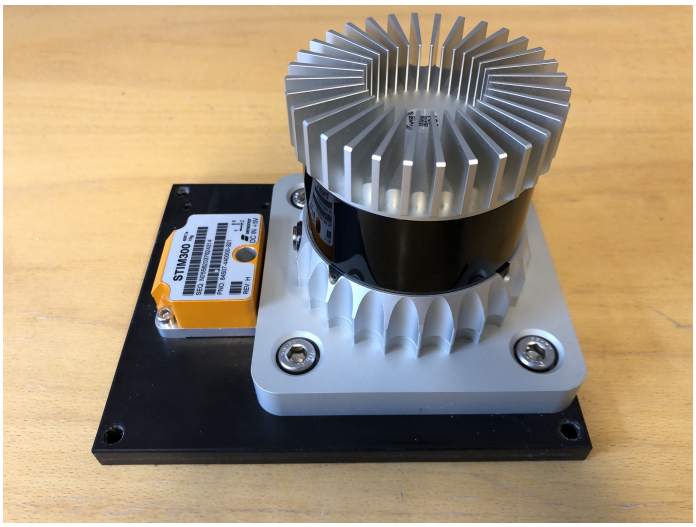


Figure 4.3: Photograph of the LiDAR-IMU platform. The LiDAR is positioned on the right side, characterized by its taller structure, while the IMU is located on the left side, represented by the small partially orange block. Both sensors are securely fastened to a black plastic plate to ensure a rigid configuration.

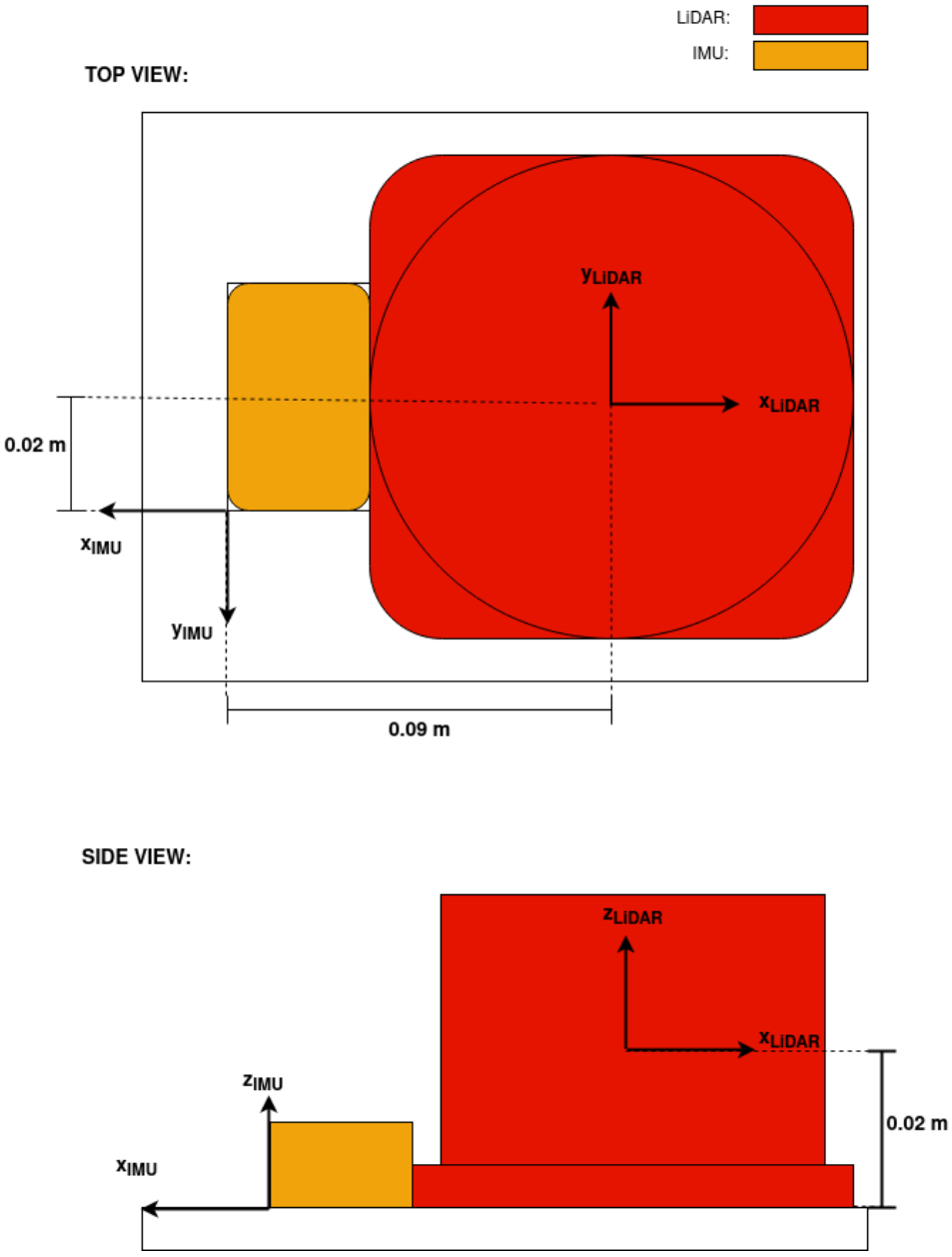


Figure 4.4: Illustration of the LiDAR-IMU relative mounting, with the LiDAR represented in red and the IMU in orange. The top view is depicted in the upper figure, while the side view is shown in the lower figure. Both the LiDAR and IMU coordinate systems are illustrated. For reference, the actual platform can be seen in Figure 4.3.

4.2.2 Car platform

The sensors were mounted on a platform that was placed on the roof of a car, as shown in Figure 4.5. The Car model is a Mercedes Vito, and the platform was custom-made by a workshop at Norwegian University of Science and Technology (NTNU). The car roof exclusively houses the sensors, while all other components are positioned inside the car. To make sure that most of the light rays from the LiDAR hit the surrounding environment, and not the car, the LiDAR and IMU platform was elevated.

Figure 4.6 shows how the sensors are located relative to each other. The upper part illustrates how it looks from a top view and the lower part from a side view. The distance between the three GNSS antennas and the LiDAR frame is presented. The LiDAR is illustrated in red, IMU in orange, and GNSS antennas in green. The antenna at the front is Moving Base, while the right and left antennas at the back of the car are Rover1 and Rover2, respectively. Left and right with respect to looking in the front direction. All GNSS measurements are referred to the center of the antenna, indicated by the dotted line. Since the LiDAR frame functions as the body frame, the distances between the IMU and GNSS antennas are not presented in the figure but can be interpreted by combining the measures in Figure 4.4 and Figure 4.6.

Figure 4.5: Photograph of the Mercedes Vito car with sensors installed on the roof. The sensors are mounted on a sturdy platform, with two GNSS antennas positioned at the rear, one at the front, and the LiDAR-IMU platform elevated in the center.

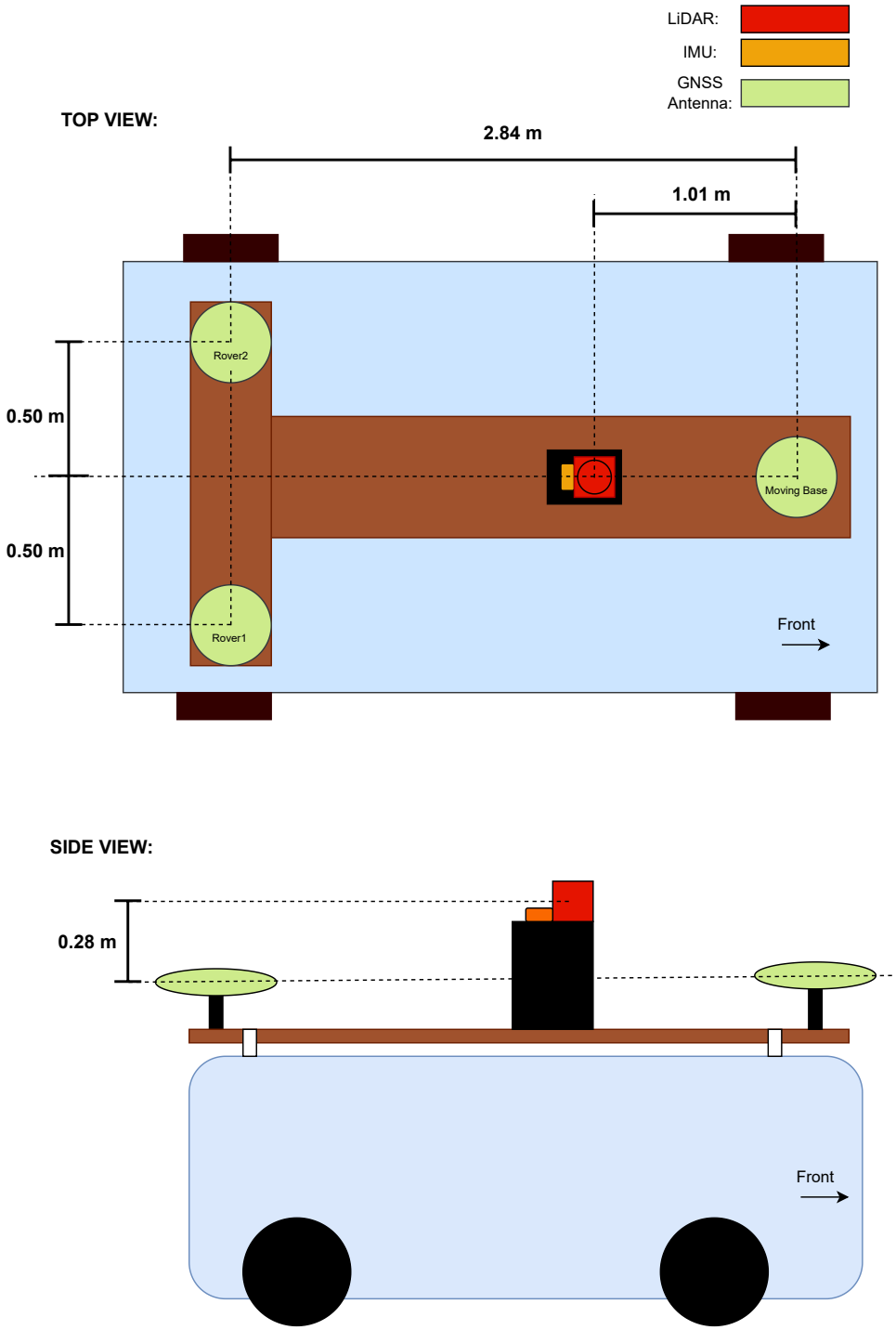


Figure 4.6: Illustration of the car platform with distance metrics. The top part showcases the top view of the car, while the bottom figure presents the side view. GNSS antennas are depicted in green, LiDAR in red, and IMU in orange. This figure represents the same setup as in Figure 4.5

4.3 Software integration and data collection

Data from sensors generally do not come in the format that the sensor fusion algorithm requires, which requires data conversion in software. This issue can be quite cumbersome when working with sensor fusion algorithms. Sensor data coming in on a serial port must be parsed and converted to the respective format used by the sensor fusion algorithm. In this project, the algorithm used was developed in ROS – a software platform specifically designed for robotic applications. ROS has a flexible data storage format called *rosbag*, in which the collected data was stored for post-processing. The sensor data coming from the sentiBoard was parsed and forwarded to the ROS framework, which was done through dedicated SentiBoard parsers, called *SentiReader* and *SentiCom*. Figure 4.7 shows an abstraction of the software modules and how data flows through the system and ends up being stored in a rosbag. Raw GNSS data were stored in a dedicated SentiSytems file format, for RTK generation. The synchronization module, located within the "ouster driver", is responsible for the alignment of LiDAR data with the SentiBoard clock. This module, presented in this section, utilizes the association of point cloud data from the LiDAR with TOV messages from the SentiBoard. The TOV messages from the SentiBoard are generated based on the TOV pulses emitted by the LiDAR, as depicted in Figure 4.2 and Figure 4.7.

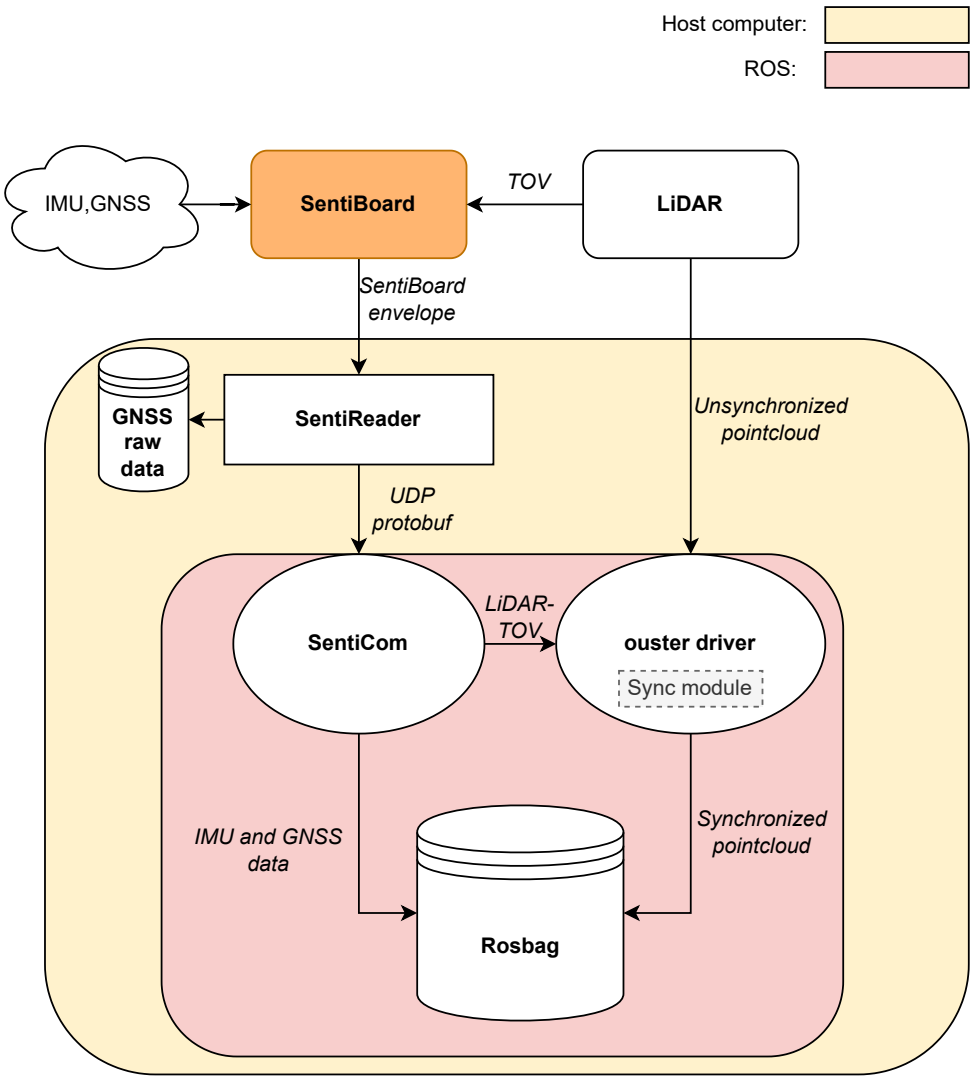


Figure 4.7: Software setup diagram with data flow. The hardware components are depicted by three blocks at the top, while the host computer is represented by the yellow area. The pink area displays the modules in the ROS framework. "GNSS raw data" and "Rosbag" represent data storage. "SentiReader", "SentiCom", and "ouster driver" represent software processes.

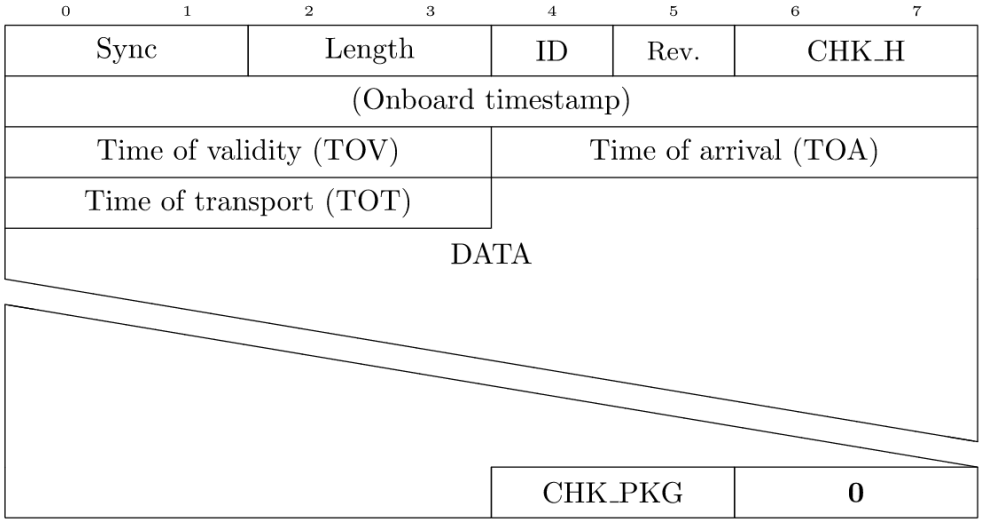


Figure 4.8: SentiBoard Envelope data format. The first 32 bytes contain timing and package information. The unaltered sensor data is contained in the *DATA*-field. A checksum and filler bytes are added to the end of the packet. Courtesy of SentiSystems (Albrektsen, 2022).

4.3.1 SentiReader - SentiBoard data parser

Data from the SentiBoard follows a protocol called *SentiBoard Envelope*, which is shown in Figure 4.8 (Albrektsen, 2022)(Storli, 2022). SentiBoard Envelope wraps the unaltered sensor data from the sensor in a header and a trailer. The header describes timing information and metadata related to the sensor packet. The trailer is used for checksum calculation. SentiReader is a custom software designed to parse data in SentiBoard Envelopes and forward it on network interfaces, utilizing the User Datagram Protocol (UDP) network protocol. Data is parsed and converted to a custom struct that is built on the Google protobuf format (Varda, 2008). The struct format is dependent on what type of sensor the data originates from.

4.3.2 Robot Operating System (ROS)

ROS is an open-source, meta-operating system designed for robots(*ROS Wiki*, 2018). It offers a comprehensive range of operating system services, including hardware abstraction, low-level device control, common functionality implementation, inter-process communication through message-passing, and package management. Additionally, ROS provides an array of tools and libraries for code acquisition, building, writing, and execution across multiple computers. Compared to other robot frameworks, ROS distinguishes itself through its peer-to-peer runtime "graph". This graph comprises loosely coupled processes, possibly distributed across machines, which leverage the ROS communication infrastructure. Although ROS is not inherently a real-time framework, it supports integration with real-time

code.

In ROS, processing capabilities have been decoupled through the utilization of message passing and timing functionalities. *Topics* in ROS function as network interfaces between parallel processes. The processes are referred to as *nodes*. A topic can only send data in one direction. Each time a node receives a message on a topic, a corresponding function is called inside the node. These functions are referred to as *callback functions*. A node can have multiple topics connected to it. Timing in ROS is based upon an internal clock that runs in the background of all processes. To access this clock, function calls like `time_now()` can be called, which would return the clock value at that time. When data is saved in a rosbag, the data is timestamped based on this clock. The data being stored can additionally have timing information in the stored data.

SentiCom - SentiBoard ROS driver

SentiCom is the ROS interface for packets coming from the SentiBoard(Storli, 2022). It is made to be compatible with SentiReader. It works by receiving UDP packets on a specified network interface, which are parsed into ROS messages. These messages are then sent out on dedicated ROS topics. SentiCom has dedicated topics for different sensor messages, like GNSS and IMU.

LiDAR ROS driver

In order to parse the data from the LiDAR and turn it into ROS messages, a dedicated driver[‡] made by Ouster was utilized. This driver binds to a specific network port and parses the incoming LiDAR data into a ROS message type for pointclouds. The message used in this project is called *sensor_msgs::PointCloud2*. By default, the user can configure if the pointcloud message should be timestamped based on the LiDAR internal clock, or the ROS internal time reference. In other words, the driver comes with no support for SentiBoard synchronization. To synchronize the pointcloud data with the SentiBoard, a software modification to the driver was performed.

[‡]Ouster ROS driver Github: https://github.com/ouster-lidar/ouster-ros

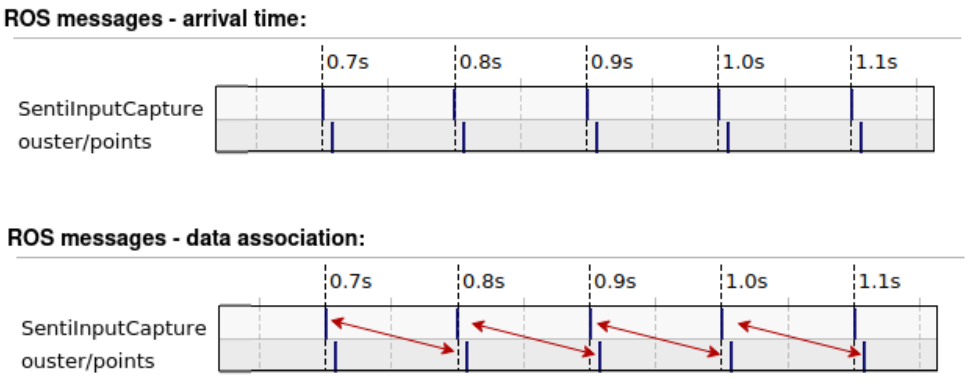


Figure 4.9: Association of LiDAR Data and TOV messages. The upper figure illustrates the arrival times of the TOV messages and pointcloud data, denoted as "SentiInputCapture" and "ouster/points," respectively. In the lower figure, arrows are added to indicate the corresponding association between TOV messages and their respective pointcloud packages.

4.3.3 LiDAR synchronization

In Storli (2022) the synchronization primitive used for synchronizing the LiDAR with the SentiBoard was 1PPS. The conclusion was that this primitive, with only one pulse per second, gives a too-loose mapping between the LiDAR clock and the SentiBoard clock. The synchronization accuracy comes down to the relative drift of the LiDAR clock to the SentiBoard clock, which can be modeled through (2.2).

The LiDAR used in this project also has a mode for encoder-based pulses. Each time the LiDAR completes a full rotation, a pulse is generated. This pulse functions as a TOV pulse for the first measurement in the scan. With the LiDAR configured to rotate ten times per second, ten pulses are sent out each second. Compared to the one pulse per second in Storli (2022), ten pulses should give a much tighter synchronization between the LiDAR and the SentiBoard. Additionally, this method is less dependent on the accuracy of the internal LiDAR clock. However, the accuracy of the encoder will play a major role in the accuracy of this synchronization method.

As in Storli (2022), the LiDAR packets received on the Ethernet interface must be associated with the corresponding TOV pulse based on the Encoder. This TOV signal is encapsulated in a SentiBoard Envelope and forwarded to the Host computer. The TOV message carries information regarding the point in time when the TOV pulse was received by the SentiBoard. The correspondence between the TOV message and the LiDAR data is based on the ROS-based timestamp the messages are given once they arrive in ROS. Figure 4.9 shows the arrival times for LiDAR data and the TOV message in ROS, as well as how they are associated. *SentiInputCapture* and *ouster/points* represents the TOV message and the LiDAR messages, respectively. As the figure shows, the LiDAR data is not associated with the nearest

TOV message. The reason for this is that the encoder pulse is sent at the beginning of the scan, and the LiDAR data is sent out once the scan has been completed and the data has been formatted to fit the UDP network protocol. Since the duration of a scan is 0.1 s, the expected arrival time of the LiDAR data is right above 0.1 s after the TOV message arrives.

Implementation of Sync module

To implement this association in practice, a synchronization module had to be coded. In Figure 4.7, the gray module inside the ouster driver represents this functionality. A pseudocode representation of the implementation is represented in Listing 4.1 and Listing 4.2. The functionality is implemented in ROS callback functions, which are called each time a new message is received on its respective topic. In the pseudocode, the variables *tov_sbts_prev* and *tov_sbts* are global variables holding the SentiBoard timestamp of the newest and second to newest TOV message, respectively. The name "sbts" is short for SentiBoard timestamp.

Listing 4.1: Pointcloud callback function

```
pointcloud_callback(pointcloud){
    pointcloud_sbts = tov_sbts_prev;

    pointcloud_synced = pointcloud
    pointcloud_synced.timestamp = pointcloud_sbts

    publish(pointcloud_synced)
}
```

Listing 4.2: TOV message callback function

```
tov_message_callback(tov_message){
    new_tov_sbts = tov_message.timestamp

    tov_sbts_prev = tov_sbts;
    tov_sbts = new_tov_sbts;
}
```

Listing 4.2 shows what happens to the TOV message when it arrives in ROS. The timestamp is stored in a global variable called *tov_sbts*, and the previous timestamp is stored in *tov_sbts_prev*. Since the association between the LiDAR data and the TOV message is as shown in Figure 4.9, the newest and previous TOV timestamps must be stored. Listing 4.1 shows that the incoming pointcloud is assigned the SentiBoard timestamp of the previous TOV packet. In the real implementation, some sanity checks were implemented to handle cases where the LiDAR or TOV message was delayed significantly. These checks are not included in Listing 4.1 and Listing 4.2 due to clarity.

4.4 GNSS messages

Three u-blox f9p receivers were used to gather data for ground truth reference, as well as data to test the implementation of the new factors in the Liorf factor graph. The placement of the antennas and their names are described in Figure 4.6. u-blox GNSS receivers have a rich configuration, where the user can choose between a lot of message types to be created from the raw satellite signals. The receivers used in this project were configured to give the messages presented in Table 4.2 (ublox, 2021).

Table 4.2: GNSS receiver messages. The Moving Base receiver was used to give position estimates in the geodetic coordinate system, as well as time information. The Rovers were used to get relative position measurements to the Moving Base antenna in NED coordinates. UBX-RXM-RAWX were used for RTK generation (ublox, 2021).

Receiver name	u-blox messages	Transmission rate
Moving Base	UBX-NAV-PVT and UBX-RXM-RAWX	1 Hz
Rover1	UBX-NAV-RELPOSNED	1 Hz
Rover2	UBX-NAV-RELPOSNED	1 Hz

The two rover receivers were used to obtain position estimates relative to the Moving Base antenna, in NED coordinates. This was achieved by configuring the Moving Base to send raw GNSS data to the rovers over UART, as shown in Figure 4.2. The rovers could then send *UBX-NAV-RELPOSNED* messages to the SentiBoard. Moving Base was configured to give single position measurements in geodetic coordinates, which was achieved through configuring the receiver to send *UBX-NAV-PVT* messages. In order to compare the Liorf estimates with RTK GNSS estimates, the Moving Base was configured to output *UBX-RXM-RAWX* messages. These messages contain the raw GNSS data received from the satellites.

4.4.1 RTK GNSS - ground truth generation

In order to obtain as accurate a ground truth estimate as possible, a RTK solution was estimated from the collected GNSS data. The process included saving raw GNSS data from the u-blox receiver listed in Table 4.1. This data was written to a file by SentiReader, as shown in Figure 4.7. The further process of obtaining RTK estimates was done offline. This included downloading raw base station GNSS data from Kartverket from the day the experimentation was performed, which was April 4th, 2023 (kartverket, 2023). Furthermore, the raw u-blox data was combined with the data from Kartverket in a software tool called "RTKlib" in order to produce the RTK solution (rtklibexplorer, 2023).

4.5 Timestamp modification for software synchronization

In order to compare the accuracy of Liorf estimates with software vs hardware synchronization, the timestamps in the collected dataset had to be swapped. Data stored in rosbags are given a timestamp that is not directly visible in the ROS message. This is the point in time when the message was stored in the rosbag and is referenced to the internal clock running in ROS. This timestamp follows a software timestamping primitive. A Python script was made to change the timestamps in the original ROS messages with the timestamp given by the rosbag.

Listing 4.3 shows the code for making a new bag that has the exact same data, but the timestamps are changed to use the once set by ROS. The name of the original rosbag is *input_bag*, while the new bag is named *output_bag*. The "for" loop goes through all messages in the original bag, and checks if the messages belong to the LiDAR or IMU topic. Moreover, the messages are extracted and the timestamps are changed. At the end, the modified messages are written to the new rosbag.

Listing 4.3: Python script for modifying timestamps in a rosbag.

```
import rosbag
import numpy as np

input_bag = 'path_to_input_bag.bag'
output_bag = 'path_to_new_bag.bag'

with rosbag.Bag(output_bag, 'w') as outbag:
    for topic, msg, t in rosbag.Bag(input_bag).read_messages():
        if topic == "/imu/data_raw" and msg.header.stamp:
            imu_msg = msg
            ros_timestamp_imu = t

            imu_msg.header.stamp = ros_timestamp
            outbag.write(topic, msg, msg.header.stamp)

        if topic == "/os_cloud_node/points" and msg.header.stamp:
            lidar_msg = msg
            ros_timestamp_lidar = t

            lidar_msg.header.stamp = ros_timestamp_lidar
            outbag.write(topic, lidar_msg, lidar_msg.header.stamp)
```

Chapter 5

GTSAM and Liorf Modifications

One of the main contributions of this thesis is the modification of the Liorf factor graph. Liorf currently offers small amount of flexibility in specifying where the GNSS antennas are placed. Liorf assumes that the GNSS antenna origin and the body frame origin coincides. This is not the case for Liorf, since the IMU frame is used as body frame, and it is impracticable to place the GNSS antenna inside the IMU. In order to implement flexibility in GNSS antenna placement, a couple of new factors were developed in the GTSAM `C++` library and integrated into the current Liorf implementation. This section describes the mathematics behind the new factors, as well as how they are integrated into the Liorf factor graph.

5.1 GTSAM factor creation

The GTSAM toolbox, known as the "Georgia Tech Smoothing and Mapping," is a freely available `C++` library centered around factor graphs (Dellaert, 2012). Developed by a collaborative team of researchers, students, and partners at the Georgia Institute of Technology, this toolbox offers cutting-edge solutions for both SLAM and SFM (Structure from Motion) problems. Its design allows for versatile modeling and solving of a wide spectrum of estimation problems, ranging from simple to intricate scenarios. Moreover, GTSAM provides a MATLAB interface that facilitates swift prototyping, visualization, and user engagement.

This project contributes to the GTSAM toolbox by designing two new factors. Both of these factors are related to GNSS measurements and are named the following

- GPSWithLeverArmFactor (GLA)
- GPSBaseLineFactor (GBL),

where GLA and GBL are short for GPSLeverArm and GPSBaseLine, respectively.

5.1.1 GPSWithLeverArmFactor

When the GNSS antenna origin does not coincide with the body frame origin, a lever arm between them is present. Using the GNSS measurement for estimating the body pose, a lever arm compensation has to be made, as expressed in (3.26). Assuming white Gaussian noise for the GNSS measurement expressed in (3.24), with covariance $\boldsymbol{\Sigma}_{mb}$, gives the following expression for the GPSWithLeverArmFactor

$$\xi_{GLA} \propto \exp\{-\frac{1}{2}\|\boldsymbol{p}^u_{ub_{mb}} - \hat{\boldsymbol{p}}^u_{ub_{mb}}\|^2_{\boldsymbol{\Sigma}_{mb}}\}, \tag{5.1}$$

where $\boldsymbol{p}^u_{ub_{mb}}$ is the estimated position of the moving base antenna, and $\hat{\boldsymbol{p}}^u_{ub_{mb}}$ is the GNSS position measurement. Expressing $\boldsymbol{p}^u_{ub_{mb}}$ as a function of the body frame pose and the lever arm between the body frame and the GNSS frame, the expression becomes

$$\xi_{GLA} \propto \exp\{-\frac{1}{2}\|\boldsymbol{p}^u_{ub} + \boldsymbol{R}^u_b \boldsymbol{l}^b_{bb_{mb}} - \hat{\boldsymbol{p}}^u_{ub_{mb}}\|^2_{\boldsymbol{\Sigma}_{mb}}\}, \tag{5.2}$$

where $\boldsymbol{R}^u_b$ is the rotation matrix from body to ENU, and $\boldsymbol{l}^b_{bb_{mb}}$ is the lever arm from body to GNSS frame. This is a unary factor, as described in Section 3.3 and (3.32), where (3.26) functions as the measurement prediction function.

A typical use case for this factor would be to measure the lever arm manually in the body frame and receive measurements from GNSS. The factor graph is then optimized with respect to the position and orientation, represented by $\boldsymbol{p}^u_{ub}$ and $\boldsymbol{R}^u_b$, respectively. The covariance $\boldsymbol{\Sigma}_{mb}$ can be obtained by configuring the GNSS receivers to send messages characterizing the uncertainty in the measurements.

Jacobian

Utilizing (5.2) in factor graph-based optimization requires the explicit expression for the Jacobian of the measurement prediction function. As stated, the measurement prediction function for the GPSWithLeverArmFactor is presented in (3.26), and can be stated as a transformation of a point in homogenous coordinates:

$$\widetilde{f}_{GLA}(\boldsymbol{R}^u_b, \boldsymbol{p}^u_{ub}) = \begin{bmatrix} \boldsymbol{R}^u_b & \boldsymbol{p}^u_{ub} \\ \boldsymbol{0}_{1\times 3} & 1 \end{bmatrix} \widetilde{\boldsymbol{l}}^b_{bb_{mb}}, \tag{5.3}$$

Based on Theorem 3. in Dellaert (2022), which describes the derivative of a point transformation, the resulting Jacobian $\boldsymbol{F}_{GLA}$ becomes:

$$\boldsymbol{F}_{GLA}(\boldsymbol{R}^u_b, \boldsymbol{p}^u_{ub}) = \begin{bmatrix} -\boldsymbol{R}^u_b \boldsymbol{S}(\boldsymbol{l}^b_{bb_{mb}}) & \boldsymbol{R}^u_b \end{bmatrix}, \tag{5.4}$$

where $\boldsymbol{S}$ is the skew-symmetric matrix operator described in (3.12). (5.4) is the derivative with respect to the pose, where the first three columns correspond to the orientation and the last three columns correspond to the position.

5.1.2 GPSBaseLineFactor

The second factor developed was the *GPSBaseLineFactor*, which is applicable when there are at least two GNSS receivers available. This factor uses (3.25) as measurement prediction function and compares it to the measured relative position. As for the GPSWithLeverArmFactor, it is assumed that the relative position estimate has white Gaussian noise with covariance $\mathbf{\Sigma}_r$. The factor is then given by the following expression

$$\xi_{GBL} \propto \exp\{-\frac{1}{2}\|\boldsymbol{R}_b^u \boldsymbol{l}_{b_{mb}b_r}^b - \hat{\boldsymbol{p}}_{b_{mb}b_r}^u\|_{\mathbf{\Sigma}_r}^2\}, \tag{5.5}$$

where $\boldsymbol{l}_{b_{mb}b_r}^b$ is the base line from the moving base antenna to the rover antenna, $\boldsymbol{R}_b^u$ is the rotation matrix from body to ENU, and $\hat{\boldsymbol{p}}_{b_{mb}b_r}^u$ is the relative position measurement between the two GNSS antennas. $\mathbf{\Sigma}_r$ can be obtained in the same way as for $\mathbf{\Sigma}_{mb}$ in (5.2).

This factor does not, on its own, help with position estimates, but will benefit the estimation of orientation. The orientation, represented by $\boldsymbol{R}_b^u$ is the optimization variable.

Jacobian

In order to incorporate the GPSBaseLineFactor into a gradient-based optimization framework, similar to the GPSWithLeverArmFactor, it is necessary to explicitly express the Jacobian. The measurement prediction function utilized in (5.5) is derived from (3.25) and can be represented as a transformation of a point in homogeneous coordinates. The transformation takes the following form:

$$\widetilde{f}_{GBL}(\boldsymbol{R}_b^u, \boldsymbol{p}_{ub}^u) = \begin{bmatrix} \boldsymbol{R}_b^u & \mathbf{0}_{3\times1} \\ \mathbf{0}_{1\times3} & 1 \end{bmatrix} \widetilde{\boldsymbol{l}}_{b_{mb}b_r}^b, \tag{5.6}$$

where $\mathbf{0}_{3\times1}$ is a zero column vector. By again, utilizing Theorem 3. in Dellaert (2022), the Jacobian $\boldsymbol{F}_{GBL}$ can be expressed as:

$$\boldsymbol{F}_{GBL}(\boldsymbol{R}_b^u, \boldsymbol{p}_{ub}^u) = \begin{bmatrix} \mathbf{0}_{3\times3} & \boldsymbol{R}_b^u \end{bmatrix}, \tag{5.7}$$

where $\mathbf{0}_{3\times3}$ is a zero matrix. (5.7) is the derivative with respect to the pose, where the first three columns correspond to the orientation and the last three columns correspond to the position.

5.2 Liorf contributions

The two factors described in the above section were developed to improve the estimates given by Liorf. This was done by adding new factors to the graph as GNSS measurements were received. Since the GNSS receivers were configured to give a new measurement every second, it would not be beneficial to add a new factor each time a measurement came in. A criterion for adding new factors to the graph was developed.

5.2.1 New factor graph

The new factor graph is shown in Figure 5.1. Compared to Figure 3.2, the changes involve two new factors. GPSWithLeverArmFactor is represented by a purple line between the GNSS single position measurement and a state node. GPSBaseLine-Factor is represented by a blue line between the GNSS relative position measurement and a state node. The GNSS relative position measurement is given as in (3.25).

The original GNSS factor is replaced by the GPSWithLeverArmFactor. They have the same structure, in the sense that both are dependent on the same GNSS measurement. However, the difference is that the GPSWithLeverArmFactor compensates for the lever arm, which the original GNSS factor does not. The mathematical expression for the GPSWithLeverArm factor is given in (5.2).

GPSBaseLineFactor is dependent on a new type of measurement that is not compatible with the original Liorf implementation. The new measurement gives the relative position between two GNSS antennas and can be used to estimate orientation. The mathematical expression for the GPSBaseLineFactor is given in (5.5).

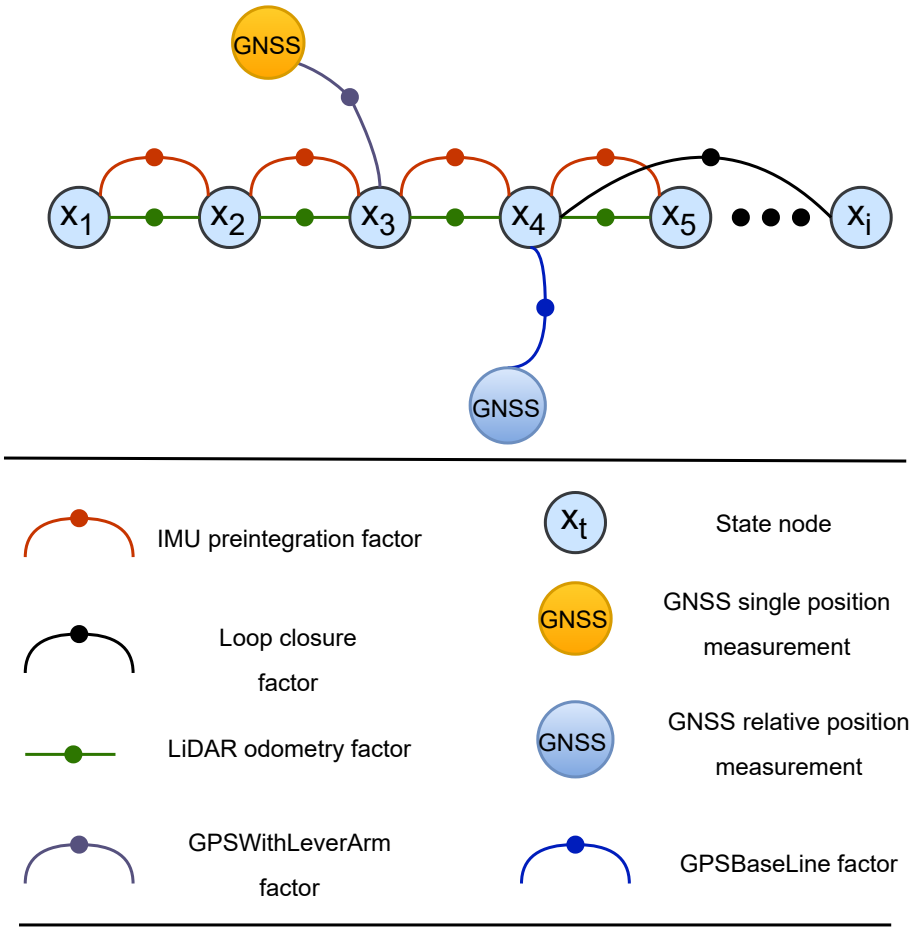


Figure 5.1: Liorf factor graph with new GNSS factors. The new factors, namely GPSWithLeverArmFactor and GPSBaseLineFactor, are depicted in purple and blue, respectively. These factors extend the factor graph introduced in Figure 3.2.

5.2.2 Factor adding criterion

Adding a new factor to a factor graph will most likely increase the computational cost, as the cost function to be optimized gets more terms that should be evaluated. Because of this, a configurable criterion for adding a new factor was developed. The original GNSS factor in Liorf also has such a criterion, which the new criteria were based on. These criteria were based on comparing the uncertainty in the current pose estimate with the uncertainty in the GNSS measurements.

The pseudo-code for adding a GPSBaseLineFactor is given in Listing 5.1. *orientationCovariance* is the orientation uncertainty estimate of the newest state node, while *orientationCovThreshold* is the configurable value given by the user. *factorGraph* is the whole factor graph, and GPSBaseLineFactor represents the factor to be added.

Listing 5.1: Pseudo-code for adding a GPSBaseLineFactor.

```
if (orientationCovariance > orientationCovThreshold) {
    factorGraph.add(GPSBaseLineFactor)
}
```

Similarly, the pseudo-code for adding a GPSWithLeverArmFactor is given in Listing 5.2. *positionCovariance* is the position uncertainty estimate of the newest state node, while *positionCovThreshold* is the configurable value given by the user. *factorGraph* is the same factor graph as in Listing 5.1, and GPSWithLeverArmFactor represents the factor to be added.

Listing 5.2: Pseudo-code for adding a GPSWithLeverArmFactor.

```
if (positionCovariance > positionCovThreshold) {
    factorGraph.add(GPSWithLeverArmFactor)
}
```

The covariance threshold can be chosen based on the accuracy of the GNSS receivers used. In the case of very precise GNSS receivers, the threshold should be set low, and vice versa. The computational recourses should also be considered when choosing a threshold.

Chapter 6

Results and Discussion

In order to evaluate how different synchronization methods, and how the use of different GNSS factors affect the output of Liorf, multiple plots are presented. These plots compare the different pose estimates. Since pose consists of position and orientation, both of these are compared. In order to evaluate these estimates, they are compared to the pose estimates obtained from RTK GNSS. The dataset gathered in this project is gathered from a car driving in urban areas, and is referred to as the *Jonsvatnet dataset*. It was gathered during good weather with a clear sky. The car route consisted of multiple turns and roundabouts, as well as ending up on the highway with speeds up to 80 km/h. In order to say something about the accuracy of the sensor timestamps and the estimates from Liorf, compared to reference values, error metrics are used. These are defined at the beginning of the chapter and used consequently throughout it. The timestamp difference between two consecutive measurements is plotted to get an intuition of what the difference between software and hardware timestamping means in practice. This is done for all sensors. Furthermore, the timestamp difference for encoder-based synchronization of LiDAR and SentiBoard is compared to timestamps based on 1PPS and ROS synchronization. This chapter ends with a broader discussion focusing on what can be stated based on the conducted experiment and analysis.

6.1 Error metrics

*Statistical measures are important when analyzing data. In this report statistical measures such as mean error (ME), root mean square error (RMSE), mean absolute error (MAE), standard deviation (STD), and Max are used. So no confusion occurs, the formulas used in this project are presented in the following subsections. Given a dataset $\boldsymbol{d}$, that should be compared to a dataset $\boldsymbol{f}$, where both have length N, the following error metrics are defined. The notation $(\cdot)_k$ and $\overline{(\cdot)}$ denotes element k in the dataset and the mean value, respectively.

Mean error (ME)

$$ME = \frac{1}{N}\sum_{k=1}^{N}(d_k - f_k) \tag{6.1}$$

Mean absolute error (MAE)

$$MAE = \frac{1}{N}\sum_{k=1}^{N}|d_k - f_k| \tag{6.2}$$

Root mean square error (RMSE)

$$RMSE = \sqrt{\frac{1}{N}\sum_{k=1}^{N}(d_k - f_k)^2} \tag{6.3}$$

Standard deviation (STD)

$$\sigma = \sqrt{\frac{1}{N-1}\sum_{k=1}^{N}(d_k - \overline{\boldsymbol{d}})^2} \tag{6.4}$$

Max error

Max error for a dataset is the largest error value, in absolute value.†

*This and subsequent paragraphs are taken from the specialization project, Storli (2022).
†End of reference from Storli (2022).

6.2 Timestamp analysis

In Storli (2022), the LiDAR was synchronized by the use of the 1PPS synchronization primitive, with the SentiBoard as a slave and the LiDAR as master. In this project, the synchronization primitive used was encoder-based pulses, which function as a TOV for the first set of points in the scan. This section compares the different timestamps produced by the two different synchronization primitives, as well as the ones set by ROS. This is done by comparing the results from Storli (2022) with the data gathered in this project.

Additionally, the timestamps set by ROS and SentiBoard are compared for IMU and GNSS measurements. This result is presented to understand how the different synchronization primitives affect the timestamp difference. It is these timestamps which sets the basis for the estimates produced by Liorf, presented later in this chapter.

6.2.1 LiDAR Synchronization comparison - encoder, 1PPS and ROS

In Table 6.1, the error metrics for the different synchronization primitives used for LiDAR-SentiBoard synchronization are shown. The values represent the time difference from one measurement to the next, compared to the sampling period given by the sensor datasheet (Ouster, 2022). The sampling rate is set to 10 Hz, which gives a sampling period of 10^5 μs. The ROS-based timestamps are also included, for comparison. The values for 1PPS and *ROS laptop* are taken from the *vehicle dataset* in Storli (2022), while the encoder and ROS based are taken from the Jonsvatnet dataset collected in this project. The data collected in Storli (2022) used a laptop running a virtual machine for data collection, which was much slower than the host computer used in this project.

The most outstanding dataset is the ROS synchronized vehicle dataset in Storli (2022), which has much larger error values, for all metrics, compared to the other datasets. This indicates that the accuracy of the software timestamps produced by ROS is highly dependent on the hardware it runs on. The ROS-based timestamps from this project, running on an Intel NUC produces as accurate timestamps as the hardware-based methods. This analysis does not evaluate the accuracy of the synchronization between the LiDAR and the SentiBoard. Even though the timestamp difference is more accurate for the ROS based method, the synchronization with the other sensors may be much worse than for the hardware-based methods. Moreover, this analysis is highly dependent on the sampling period specified by the sensor manufacturer being accurate.

Table 6.1: Error metrics for different LiDAR synchronization methods. The time difference between two consecutive measurements is compared to the sampling period given by the sensor producer, 10 Hz. Encoder and ROS is based on the dataset collected in this project, while ROS laptop and 1PPS is based on the vehicle dataset in Storli (2022).

LiDAR synchronization method – timestamp difference					
	ME [μs]	MAE [μs]	STD [μs]	RMSE [μs]	Max error [μs]
Encoder:	0.24	116.14	166.08	166.06	907.76
1PPS:	-1.51	81.48	447.48	447.45	$2.03 \cdot 10^4$
ROS:	-0.23	167.96	222.00	221.97	$1.13 \cdot 10^3$
ROS laptop:	-6.61	$5.89 \cdot 10^3$	$8.41 \cdot 10^3$	$8.41 \cdot 10^3$	$2.35 \cdot 10^3$

6.2.2 Timestamp analysis for Jonsvatnet dataset

The dataset collected in this project, Jonsvatnet dataset, consists of measurements from IMU, LiDAR, and GNSS. All measurements are given two timestamps, one by ROS and one by the SentiBoard. In this section, the timestamp difference for the two timestamping methods is presented and analyzed.

IMU data

In Figure 6.1, the timestamp difference for the timestamps set by ROS and SentiBoard is presented in blue and orange, respectively. Both the ROS and SentiBoard (SB) data seem to alter around the sampling period of $2 \cdot 106$ ns. This sampling period coincides with the sampling frequency of the IMU, 500 Hz. Although both graphs alter around the sampling period, it is evident that the ROS-based timestamps have some unwanted characteristics. First of all, the ROS data varies a lot compared to the SB data. Secondly, there are some major spikes in the ROS data, that may indicate that some of the measurements are stuck in a buffer before entering the software. The SB data seem to have timestamps that represent the exact sampling period throughout the dataset.

Table 6.2 shows the error metrics corresponding to the data presented in Figure 6.1. The most outstanding value is the ME for the ROS-based timestamps, which is less than 2 ns. This may indicate, that on average, the ROS timestamps are extremely precise. However, as the other error metrics indicate for the ROS data, the variation in the data is high. The SB data has low values for all error metrics. Even though the value for ME is greater, in absolute value than the ME for the ROS data, it is still extremely low. The SentiBoard has a sampling accuracy of 10 ns, greater than the ME value (S. M. Albrektsen, 2018).

Table 6.2: Error metrics for hardware and software synchronization of IMU timestamps. Hardware- and software-based timestamps are realized by SentiBoard and ROS, respectively. The sampling period given by the sensor manufacturer is 500 Hz.

IMU timestamps - error metrics					
	ME [μs]	MAE [μs]	STD [μs]	RMSE [μs]	Max error [μs]
SB:	0.01	0.05	0.08	0.08	0.26
ROS:	-0.001	85.38	108.77	108.77	$6.63 \cdot 10^3$

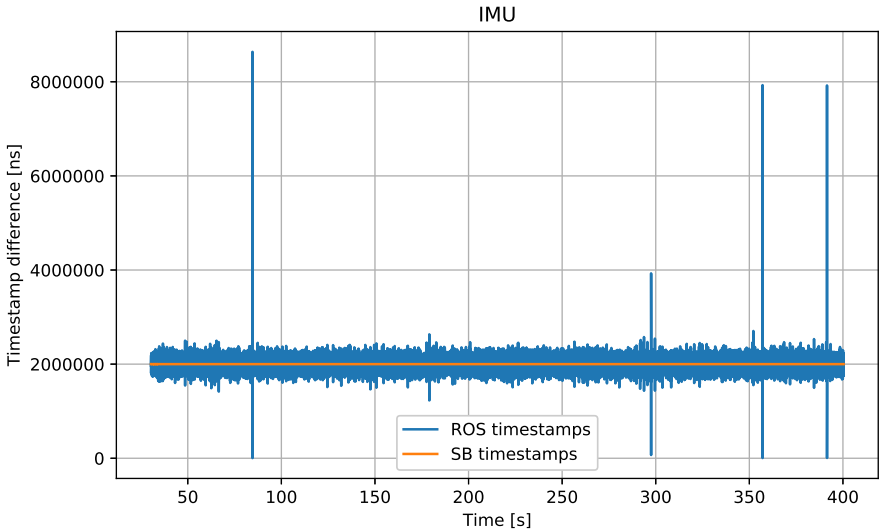


Figure 6.1: Timestamp difference for hardware and software synchronization of IMU measurements. Hardware and software based timestamps are realized by SentiBoard and ROS, respectively. ROS-based timestamps are shown in blue, while SentiBoard are shown in orange. The sampling period, based on the manufacturer's specifications, is $2 \cdot 10^6$ ns.

LiDAR data

Figure 6.2 shows the timestamp difference obtained by the encoder pulse synchronization method, as well as the ROS-based timestamps. The SB- and ROS-based timestamps are shown in orange and blue, respectively. Both graphs are altering the sampling period of 10^8 ns, which is precisely what the LiDAR datasheet specifies. The error metrics are presented earlier in Table 6.1, where the encoder-based values resemble the orange graph, and the ROS synchronization values resemble the blue graph.

There seems to be a periodic variation in the timestamps set by the SB, lasting from 75 s to the end of the dataset. Additionally, the first 50 seconds of the SB data seems to have much lower variation than the rest of the dataset. This could be a result of the dynamics of the rotating part inside the LiDAR. If the LiDAR rotates or accelerates, this will affect the torque of the rotating part inside the LiDAR, and consequently affect at what time the encoder-based pulse is sent out.

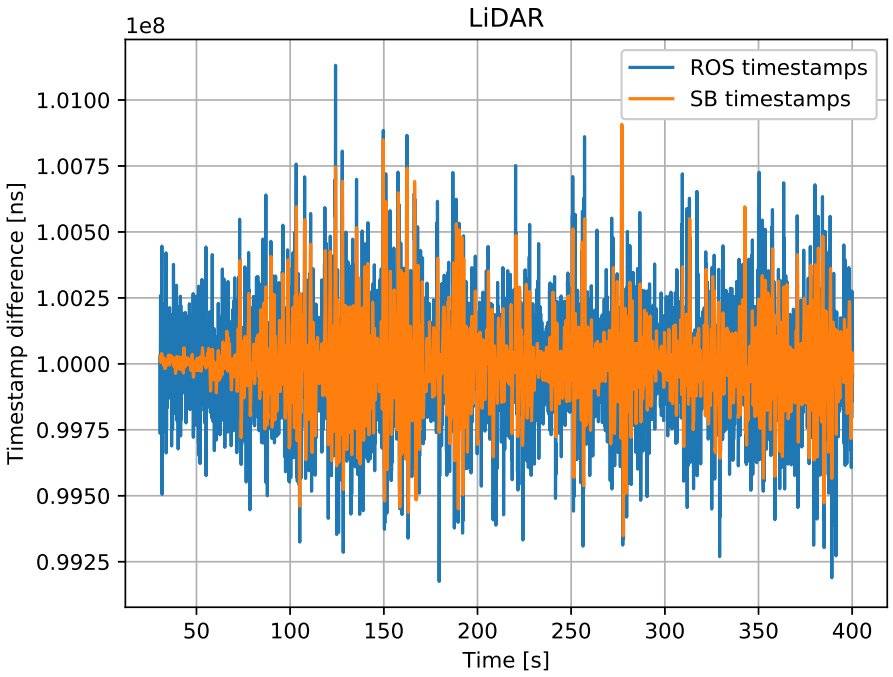


Figure 6.2: Timestamp difference for hardware and software synchronization of LiDAR measurements. Hardware- and software-based timestamps are realized by SentiBoard and ROS, respectively. ROS-based timestamps are shown in blue, while SentiBoard-based are shown in orange. The sampling period, based on the manufacturer's specifications, is 10^8 ns.

GNSS data

In the Jonsvatnet dataset, there are three different types of GNSS measurements. The three types are named *Moving Base*, *Rover1* and *Rover2*. The GNSS messages received are described in Section 4.4. All three measurement types are individually timestamped with a SB and ROS timestamp.

Figure 6.3 shows the SB and ROS timestamp difference for the three types of GNSS measurements. The Moving Base data is presented in the top plot, while the data for Rover1 and Rover2 is presented in the middle and bottom plots, respectively. ROS and SB timestamps are represented by the blue and orange graphs, respectively. From all three plots it is evident that the ROS timestamps vary much more than the ones set by the SB. This is most noticeable in the Moving Base plot, but is also visible in the two rover plots. The two spikes in the rover plots are most likely caused by packet loss. The rover messages are based on finding the RTK solution based on the data from the Moving base. In some cases, the signals may not be of such quality that the RTK solution can be found, resulting in the rover packages not being sent. In contrast, the delivery of Moving Base data is consistent throughout the whole dataset.

Table 6.3 shows the error metrics corresponding to the data presented in Figure 6.3. All SB values for all message types are almost identical. This is expected as the three receivers base their pulses on the UTC, and sends a pulse a top of each second. The values in this table do not include the large peaks in the Rover1 and Rover2 plots in Figure 6.3. These were considered outliers and were removed. The ROS values are significantly less accurate than the SB ones and vary a lot between the three receivers.

Table 6.3: Error metrics for hardware and software synchronization of GNSS timestamps. Hardware- and software-based timestamps are realized by SentiBoard and ROS, respectively. Hardware synchronization involved timestamping with the SentiBoard, while software synchronization relied on timestamping via ROS.

GNSS timestamps - error metrics					
	ME [μs]	MAE [μs]	STD [μs]	RMSE [μs]	Max error [μs]
Moving Base					
SB:	-0.63	0.63	0.01	0.63	0.66
ROS:	-22.17	$1.11 \cdot 10^4$	$1.23 \cdot 10^4$	$1.23 \cdot 10^4$	$2.62 \cdot 10^4$
Rover1					
SB:	-0.62	0.63	0.01	0.63	0.65
ROS:	34.93	$8.99 \cdot 10^3$	$1.04 \cdot 10^4$	$1.04 \cdot 10^4$	$2.47 \cdot 10^4$
Rover2					
SB:	-0.63	0.63	0.01	0.63	0.65
ROS:	57.53	$8.11 \cdot 10^3$	$9.49 \cdot 10^3$	$9.47 \cdot 10^3$	$2.38 \cdot 10^3$

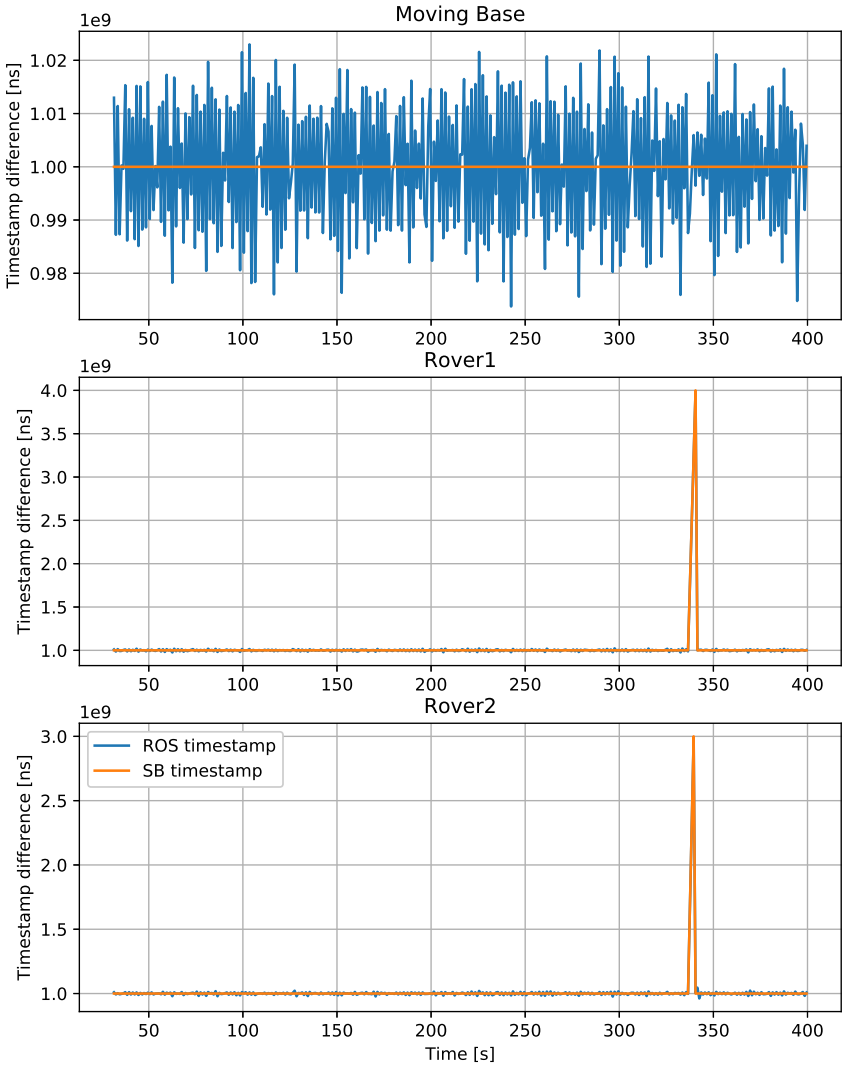


Figure 6.3: Timestamp difference for hardware and software synchronization of GNSS measurements. Hardware- and software-based timestamps are realized by SentiBoard and ROS, respectively. ROS-based timestamps are shown in blue, while SentiBoard are shown in orange. The sampling period, based on the manufacturer's specifications, is 10^9 ns.

6.3 GNSS estimates and RTK solution

In order to compare the output of Liorf with high-precision absolute measurements, the RTK solution was calculated from the Moving Base receiver raw GNSS data. However, when running Liorf with GNSS as input, the position measurements from Moving Base are used as input and not the RTK solution. An interesting result is the comparison of the Moving Base position estimates with the RTK position estimates. In this section, the north, east, and height estimates are compared. The estimates are compared in a local ENU frame defined by the first RTK estimate. By the use of the QUEST method, presented in Section 3.2.3, full orientation estimates from the two Rover antennas could be estimated and used for comparison to the Liorf orientation estimates. These orientation estimates are also presented in this section.

6.3.1 North, east and height estimates

In Figure 6.4 the North-East trajectory from the Moving Base messages and the RTK solution are plotted. The RTK solution is plotted in blue, and the Moving Base is plotted in dotted orange. From this figure, the two trajectories seem to be completely identical. It should be noted that the axis scale is large, making it hard to see small deviations. Further in this report, the RTK trajectory will be used as ground truth.

The height estimates from Moving Base and RTK are plotted against time in Figure 6.5. The RTK solution is plotted in blue, and the Moving Base is plotted in dotted orange. This plot shows no significant difference between the two graphs. The most notable difference may be that the Moving Base estimates are above the RTK during the entire trajectory. Another notable difference is that the RTK estimate is not as smooth as the Moving Base. This can be seen by the 260 s and 340 s marks.

To investigate the difference between the RTK and Moving Base estimates without being concerned about the large-scale axis hiding the difference, error plots are presented. Figure 6.6 shows the numerical difference between the east, north, and height estimates from top to bottom, respectively. For comparison, all plots have the same scale on the vertical axis. Table 6.4 contains error metrics for the same data. The east error is on average half of the north error, as seen by the ME and MAE. However, the variation seems to be the same, as seen by the values for STD. The height error varies more than the east and north estimates. However, the error is not as large as for the north estimates. At the 340 s mark the difference has a large spike, as also seen in Figure 6.5. This is most likely caused by the RTK solution producing inaccurate estimates. A car driving in urban areas is not expected to drop height as rapidly as the RTK trajectory presents. This may indicate that the calculated RTK solution has some inaccuracies.

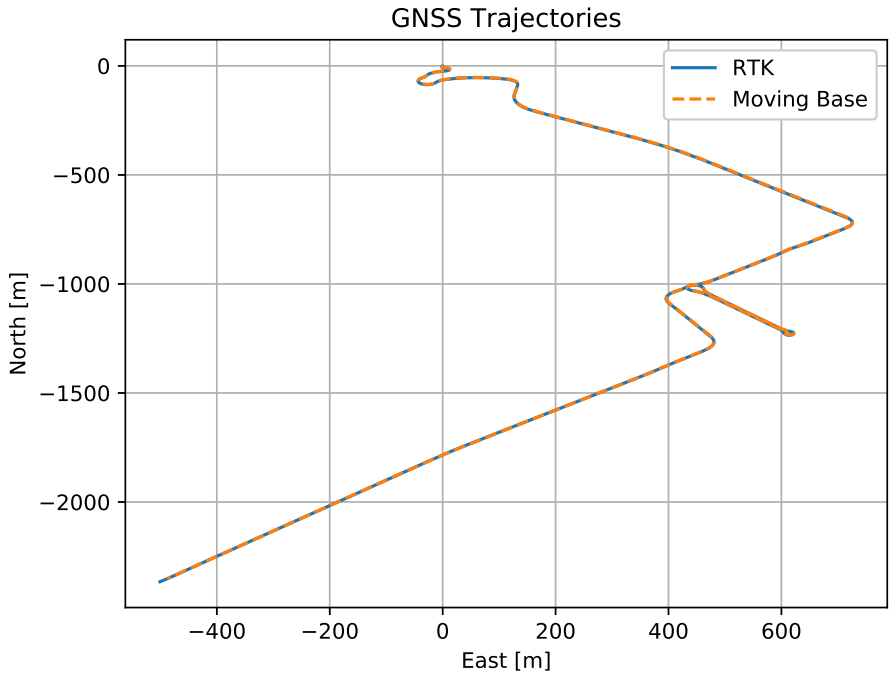


Figure 6.4: Comparison of North-East estimates between RTK and Moving Base GNSS solutions. The RTK solution is depicted in blue, while the Moving Base trajectory is illustrated as a dotted orange line.

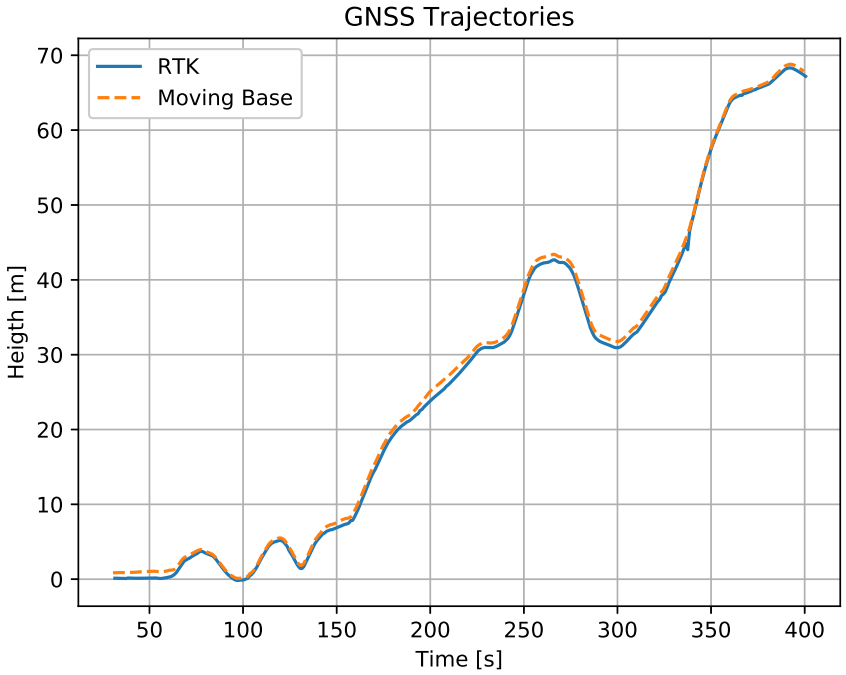


Figure 6.5: Comparison of height estimates between RTK and Moving Base GNSS solutions. The RTK solution is depicted in blue, while the Moving Base trajectory is illustrated as a dotted orange line.

Table 6.4: Error metrics for RTK and Moving Base GNSS solutions. The error metrics for RTK and Moving Base GNSS solutions are calculated by subtracting the Moving Base estimates from the RTK estimates. Zero is used as the reference value for the error metrics.

RTK vs Moving Base – error metrics					
	ME	MAE	STD	RMSE	Max error
North [m]:	-1.044	1.044	0.094	1.048	1.366
East [m]:	-0.493	0.493	0.099	0.503	0.810
Height [m]:	-0.627	0.627	0.279	0.686	2.118

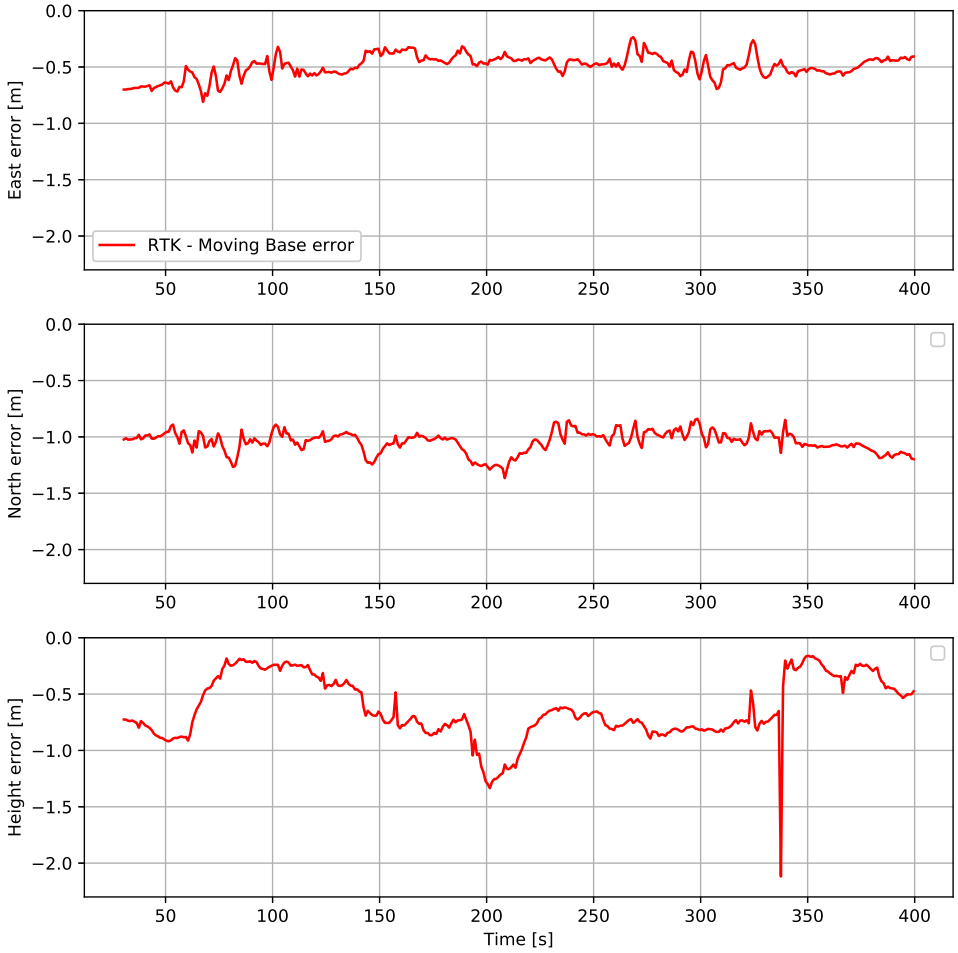


Figure 6.6: The plots show the East-North-Height errors of the RTK and Moving Base GNSS solutions. The error is computed by subtracting the Moving Base estimates from the RTK estimates. All axes are scaled equally to ensure a fair comparison of the three directions.

6.3.2 Orientation estimates

The orientation estimates for roll, pitch, and yaw are presented in Figure 6.7. Roll and pitch are plotted in the upper plot, in blue and green, respectively. Yaw, which is another term for heading, is plotted in the lower plot. This separation is done because the yaw values span a much greater range than the roll and pitch values, which is logical for a car application. These angles follow the right-hand rule for the ENU frame. The yaw angle is referenced to the east axis, not the north axis, which is common in many applications. The accuracy of these values is hard to indicate and must be considered when compared to the Liorf orientation estimates.

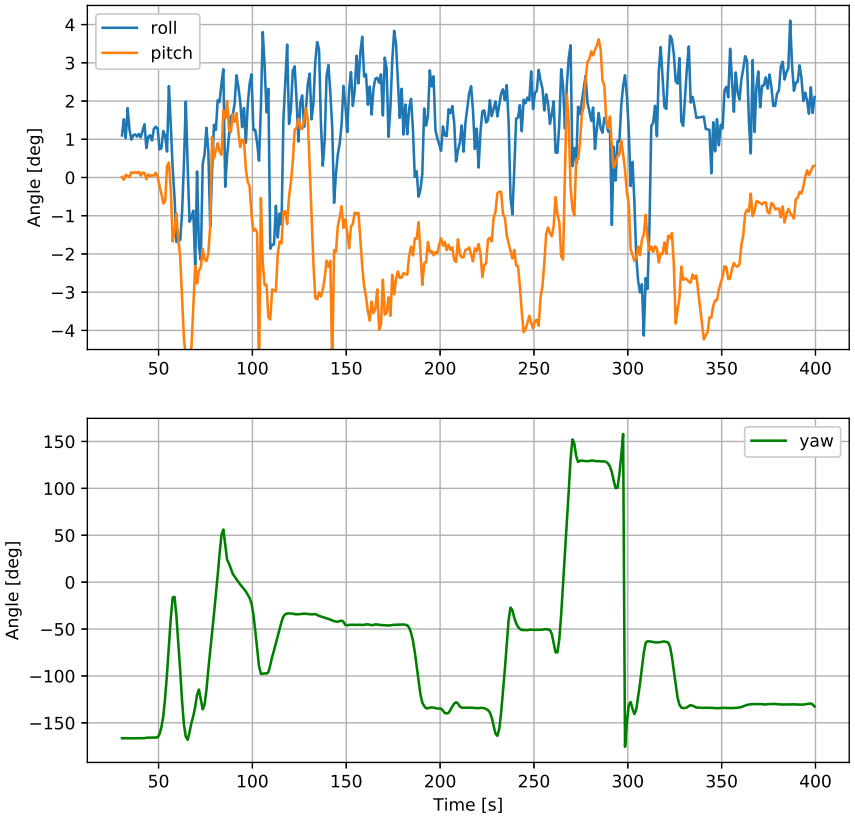


Figure 6.7: The plot displays the roll, pitch, and yaw estimates obtained from three GNSS antennas. In the upper plot, the roll is depicted in blue, while the pitch is shown in orange. The lower plot illustrates the yaw angle in green. The yaw angle is calculated with respect to the east axis as a reference.

6.4 Liorf initialization

Before comparing the pure Liorf-based estimates to the RTK generated, a note on initialization should be made. Originally, the GNSS obtained orientation estimates were used to initialize the body frame. Due to inaccuracies in the GNSS measurements, the initial heading caused such a large drift in the estimated trajectory that the comparison with the RTK path was meaningless. This huge drift can be seen in Figure 6.8 where the orange trajectory resembles the pure IMU and LiDAR base Liorf North-East estimate with GNSS based orientation initialization. The blue line is the RTK path presented earlier.

This resulted in a need for a more precise orientation initialization. The method used was to run Liorf with IMU, LiDAR, and GNSS as input, which then outputs a complete factor graph with all the state nodes. Due to the nature of optimizing a factor graph, where earlier states can be re-optimized to achieve more accurate estimates, the first state node can be considered extremely accurate. The orientation estimate of this state node was extracted and used for initialization. The resulting trajectories, with this initial orientation, are presented in the following sections.

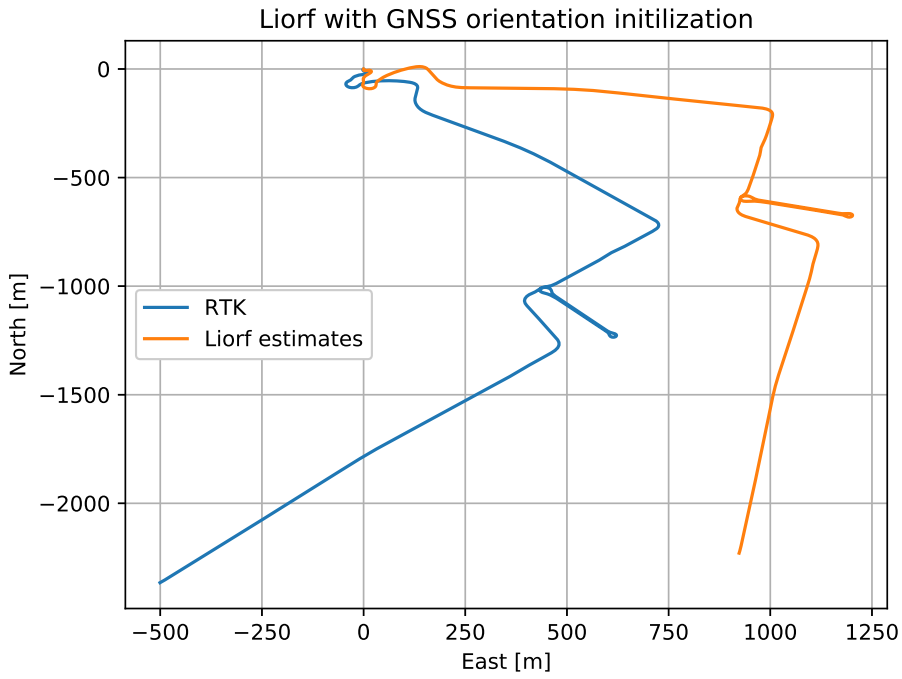


Figure 6.8: Liorf North-East trajectory initialized with orientation estimates from GNSS compared to RTK GNSS solution. The RTK solution is plotted in blue, while the Liorf estimates are depicted in orange.

6.5 Liorf software and hardware synchronization comparison

This section presents the main result of the report, which is the comparison of Liorf estimates when using software vs hardware timestamps for sensor synchronization. Only LiDAR and IMU measurements are used as input to Liorf. This is done to investigate the effect of accurate time synchronization between the two sensors. In many cases, the GNSS signals may fall out, which means that a system must be able to function properly with only LiDAR and IMU.

6.5.1 Position estimate comparison

Figure 6.9 shows the North-East trajectory produced by Liorf when using timestamps obtained by ROS and SB. The dotted green trajectory is the result of using ROS timestamps, and the orange is the result of using SB timestamps. For comparison, the RTK path is plotted in blue. Based on this plot, the effect of using hardware timestamps, compared to software timestamps, is minimal. Both the ROS and SB based trajectories seem to have the same drift compared to the RTK path. The ROS based path seems to lie further away from the RTK, but this is minimal compared to the total drift for both paths.

Figure 6.10 shows the height estimates plotted against time for the same trajectory as in Figure 6.9. This plot also indicates that the difference between the two timestamping methods has minimal effect on the Liorf output. In contrast to the North-East plot, the SB based height is now further away from the RTK compared to the ROS based. This major drift in height is expected when using no GNSS, as stated in the original LIO-SAM paper (Shan et al., 2020).

In Figure 6.11 the east, north, and height errors with respect to the RTK estimates are plotted against time. The SB and ROS based estimates are plotted in orange and dotted green, respectively. The upper plot, which shows east error, indicates that the SB based trajectory is more accurate, but not by a huge margin. Both trajectories drift significantly, and the end difference between the SB and ROS based trajectories is about 5 m. The middle plot shows the north error and indicates the same as the east plot. The SB based trajectory is somewhat more similar to the RTK trajectory compared to the ROS based. At the end of the trajectory, the difference seems to be about 3 m, in favor of the SB trajectory. The height error, shown in the lower plot, is of such a large scale that the difference between the SB- and ROS-based trajectory is not visible.

Table 6.5 shows the error metrics for the data presented in Figure 6.11. The SB based estimates have lower values for North and East errors compared to the ROS based estimates. This goes for all error metrics. It is completely opposite for the height values, where the ROS-based values are larger than the SB-based.

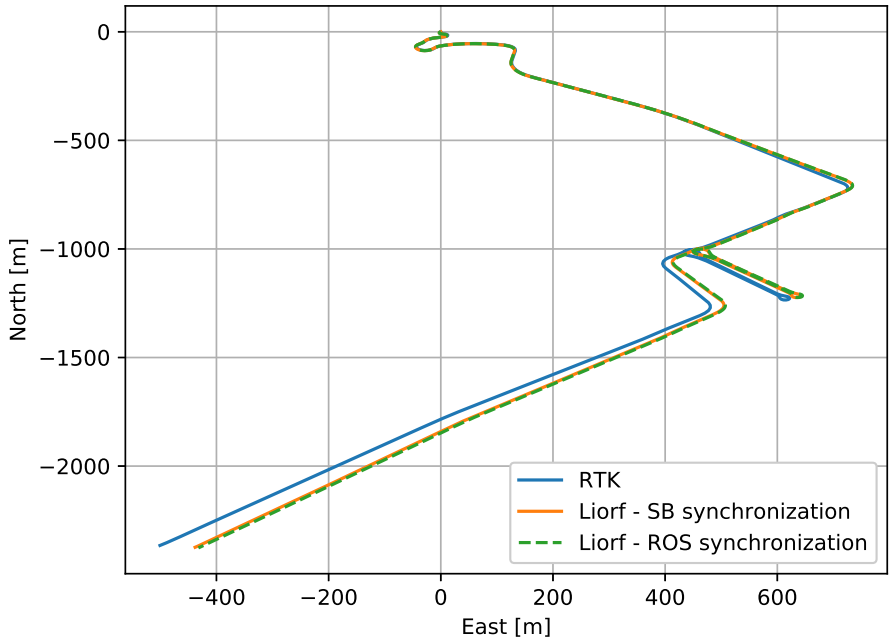


Figure 6.9: North-East trajectory of LiDAR-IMU-based Liorf when using hardware and software synchronization. The trajectory obtained through hardware synchronization is represented by a dotted green line, achieved by timestamping with the SentiBoard. The trajectory obtained through software synchronization, achieved by timestamping with ROS, is depicted in orange. For comparison, the RTK GNSS solution is plotted in blue.

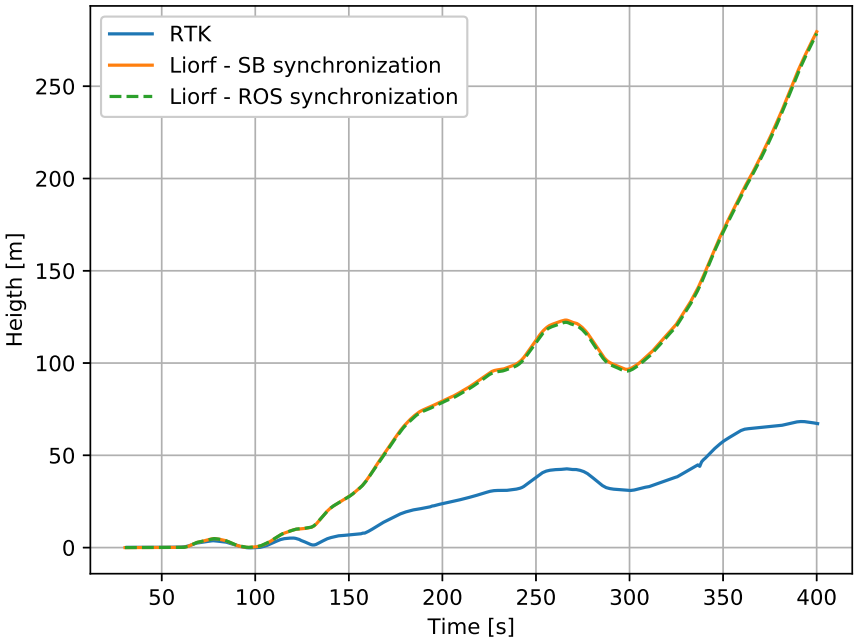


Figure 6.10: Height estimates of LiDAR-IMU-based Liorf when using hardware and software synchronization. The trajectory obtained through hardware synchronization is represented by a dotted green line, achieved by timestamping with the SentiBoard. The trajectory obtained through software synchronization, achieved by timestamping with ROS, is depicted in orange. For comparison, the RTK GNSS solution is plotted in blue.

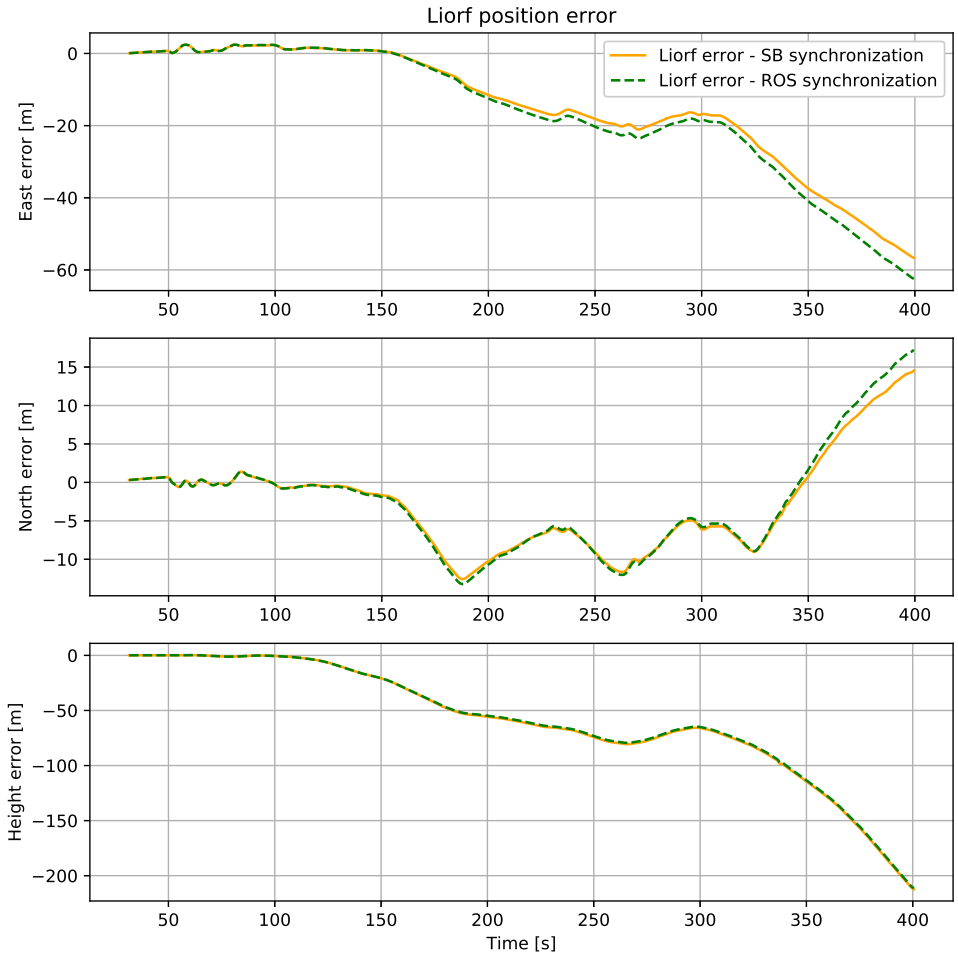


Figure 6.11: East-North-Height error of hardware and software synchronized LiDAR-IMU based Liorf estimates compared to RTK GNSS. The error based on hardware synchronization is represented by a orange graph, and achieved by timestamping with the SentiBoard. The error based on software synchronization, achieved by timestamping with ROS, is depicted in dotted green.

Table 6.5: Error metrics for hardware and software synchronized LiDAR-IMU-based Liorf East-North-Height estimates compared to RTK GNSS. Hardware synchronization involved timestamping with the SentiBoard, while software synchronization relied on timestamping via ROS.

Liorf position – error metrics					
	ME	MAE	STD	RMSE	Max error
SentiBoard synchronization					
North [m]:	-2.792	5.353	6.218	6.809	14.533
East [m]:	-14.772	15.567	16.538	22.158	62.637
Height [m]:	-59.031	59.058	52.568	78.997	212.351
ROS synchronization					
North [m]:	-2.581	5.664	6.827	7.290	17.235
East [m]:	-16.333	17.132	18.181	24.422	68.422
Height [m]:	-58.399	58.426	52.130	78.235	211.392

6.5.2 Orientation estimate comparison

Figure 6.12 shows the orientation error for the SB and ROS based estimates in orange and dotted green respectively. Roll, pitch, and yaw errors are plotted from top to bottom, respectively. Based on these plots, there seems to be no significant difference in the SB and ROS based estimates. The only notable deviation is in the yaw error plot at the 300 s mark. At this point, the ROS based estimate has a large spike, while the SB estimate seems unaffected. The cause of this large error is unknown but may be a cause of interpolating values from the GNSS based orientation estimates to the Liorf-based estimates. This seems likely as the yaw angle wraps from $180°$ to $-180°$ at this point, seen in the lower plot in Figure 6.7. This might be a result of the ROS based trajectory producing estimates which are shifted in time. In the case where these estimates should have been used in feedback control, this error spike may cause unwanted behavior.

The error metrics corresponding to the data in Figure 6.12 are shown in Table 6.8. Roll and pitch values are all higher for the SB based estimates compared to the ROS based values. Height values have the opposite trend, where all values are large for the ROS based estimate, compare to the SB based estimates.

Table 6.6: Error metrics for hardware and software synchronized LiDAR-IMU-based Liorf orientation estimates compared to GNSS obtained orientation. Hardware synchronization involved timestamping with the SentiBoard, while software synchronization relied on timestamping via ROS.

Liorf orientation – error metrics					
	ME	MAE	STD	RMSE	Max error
SentiBoard synchronization					
Roll [deg]:	0.293	1.784	2.070	2.088	5.500
Pitch [deg]:	1.748	2.449	2.351	2.927	6.444
Yaw [deg]:	-1.588	1.604	1.378	2.101	10.078
ROS synchronization					
Roll [deg]:	0.286	1.745	2.029	2.046	5.340
Pitch [deg]:	1.730	2.418	2.331	2.900	6.618
Yaw [deg]:	-1.783	1.793	2.012	2.686	29.521

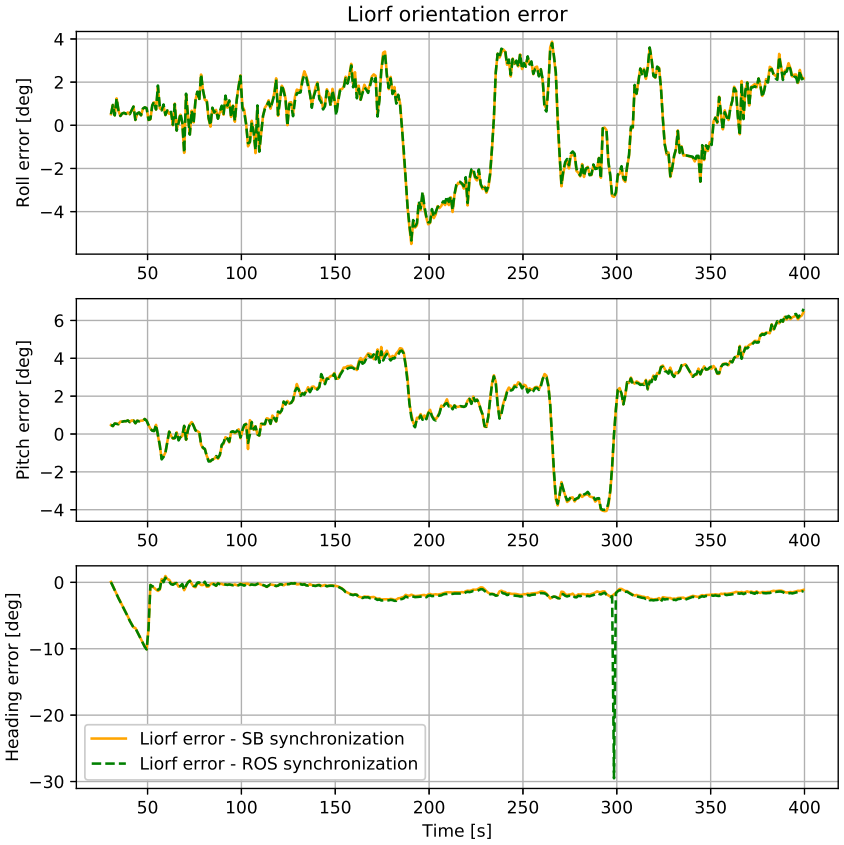


Figure 6.12: Roll, pitch, and yaw errors of hardware and software synchronized LiDAR-IMU based Liorf estimates compared to RTK GNSS. The error based on hardware synchronization is represented by a red line, and achieved by timestamping with the SentiBoard. The error based on software synchronization, achieved by timestamping with ROS, is depicted in dotted blue.

6.6 GNSS factor comparison

Comparing the output of Liorf with the GNSS factors presented is another important result of this thesis. In the case of significantly more accurate estimates, these factors can be considered an extension of Liorf. In this section the position and orientation estimates from using the different factors are compared to the GNSS based pose estimates. The three different trajectories presented are based on the use of the following GNSS factors:

- GPSWithLeverArmFactor and GPSBaseLineFactor
- GPSBaseLineFactor
- Liorf original GNSS factor

The first trajectory uses both new factors presented in this thesis, GPSWithLeverArmFactor, and GPSBaseLineFactor. The second trajectory is based on only using the GPSBaseLineFactor, and the third trajectory is based on the factor used in the original Liorf implementation. This factor is called *GPSFactor* and is a part of the GTSAM library (Dellaert, 2023). First, the position estimates are presented and compared to the RTK estimates. Secondly, the same is done for the orientation estimates.

6.6.1 Position estimate comparison

Figure 6.13 shows the North-East trajectory when using the different GNSS factors. All trajectories lie almost completely on top of each other, and it is hard to tell any difference based on this plot. The height estimates are plotted against time in Figure 6.14. In this plot, the difference between the trajectories is more visible. Using only the GPSBasaLineFactor causes the largest deviation from the RTK path. This is an expected result, as GPSBasaLineFactor gives only information regarding orientation and not position. For the other trajectories, it seems like the one based on the GPSWithLeverArmFactor and GPSBaseLineFactor lie closest to the RTK solution. However, the trajectory based on the original GNSS factor lies almost on top of the RTK path as well.

In Figure 6.15 the position errors are plotted. East, north, and height are plotted from top to bottom respectively. It is clear that the trajectory based on only the GPSBaseLineFactor behaves much worse than the other two. This does not come through in the Figure 6.13 plot, due to the large scale on the axis. The trajectory based on the GPSBaselineFactor and the GPSLeverArmFactor, and the one based on the original GNSS factor are both relatively close to the RTK solution. The north and east error plots reveal that there is a larger deviation in the trajectory.

In Table 6.7 the error metrics for the GNSS factor comparison is presented. It is evident that the purely GPSBaseLineFactor trajectory has the worst error metrics, as indicated by the earlier plots. Comparing the metrics for the original factor and the two new factors shows that north and height estimates are worse for the trajectory based on the original factor. However, it is the opposite for the east

estimates, where the trajectory based on the original GNSS factor has a lower error.

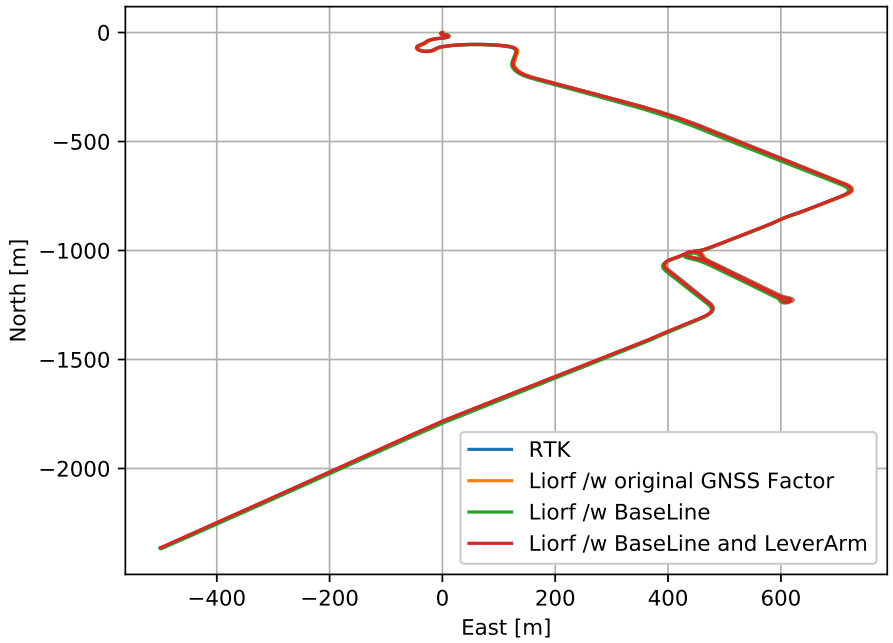


Figure 6.13: Liorf North-East trajectory comparison based on different GNSS factors. The trajectory obtained using the two newly developed factors is depicted in red, while the trajectory with only the BaseLine factor is shown in green. The trajectory using the original GNSS factor is represented in orange. For comparison, the RTK GNSS solution is plotted in blue.

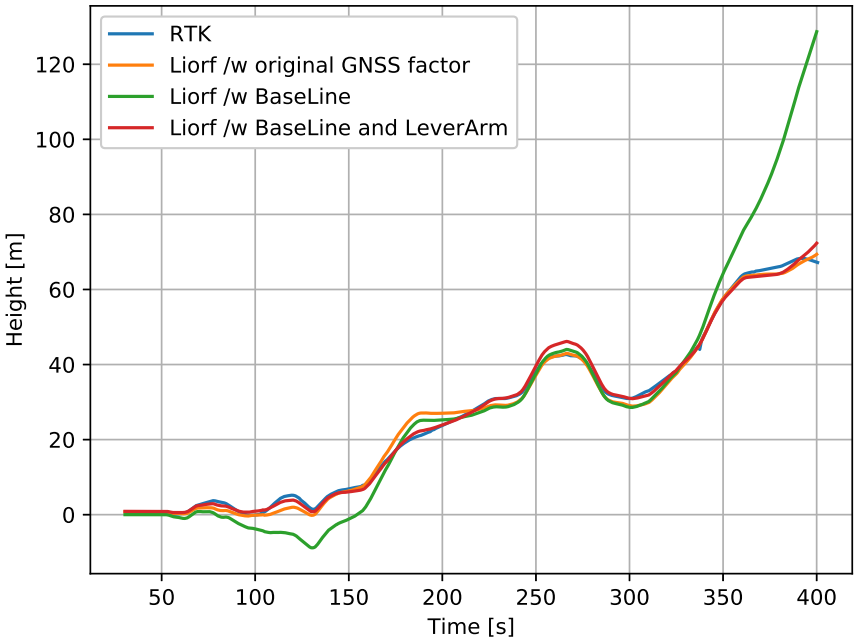


Figure 6.14: Liorf height estimates comparison based on different GNSS factors. The trajectory obtained using the two newly developed factors is depicted in red, while the trajectory with only the BaseLine factor is shown in green. The trajectory using the original GNSS factor is represented in orange. For comparison, the RTK GNSS solution is plotted in blue.

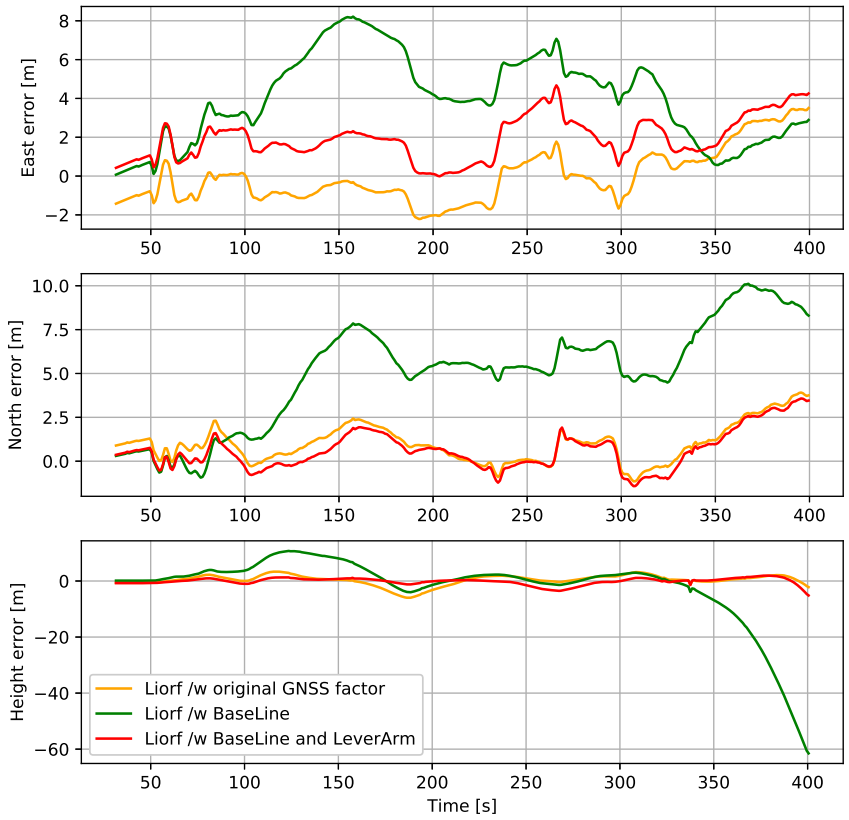


Figure 6.15: East-North-Height error of different approaches to GNSS integration in Liorf factor graph. The errors are calculated using RTK GNSS solution as reference trajectory. The error obtained using the two newly developed factors is depicted in red, while the error with only the BaseLine factor is shown in green. The trajectory using the original GNSS factor is represented in orange.

Table 6.7: Error metrics for Liorf's position estimates using different GNSS factors. The first trajectory corresponds to the original GNSS factor employed by Liorf, the second trajectory utilizes the two newly developed factors, and the third trajectory relies solely on the BaseLine factor. The GNSS RTK solution is employed as the reference trajectory for error calculation.

Liorf GNSS factor comparison - position error					
	ME	MAE	STD	RMSE	Max error
Original GNSS factor					
North [m]:	0.976	1.128	1.089	1.462	3.913
East [m]:	-0.033	1.122	1.393	1.392	3.504
Height [m]:	0.354	1.412	1.862	1.893	5.988
GPSBaseLineFactor & GPSWithLeverArmFactor					
North [m]:	0.611	0.943	1.106	1.262	3.575
East [m]:	1.923	1.931	1.103	2.216	4.674
Height [m]:	-0.103	0.858	1.186	1.189	5.148
GPSBaseLineFactor					
North [m]:	5.033	5.093	2.907	5.811	10.113
East [m]:	4.010	4.026	2.237	4.590	8.216
Height [m]:	-2.111	6.563	12.705	12.862	61.499

6.6.2 Orientation estimate comparison

Figure 6.16 shows the angle error for the different trajectories using the presented GNSS factors. The roll error, shown in the upper plot, indicates no huge difference between the trajectories. An interesting observation is that all three error trajectories seem to get closer and closer to each other as time goes by. This is not the case for the pitch error, where the GPSBaseLineFactor trajectory seems to have much worse performance compared to the two others. For the yaw error, it is difficult to separate the three trajectories due to the axis scale in the plot. The most noticeable feature in this plot is at the 300 s mark, where the yaw error drops down to $-180°$. This is most likely due to the fact that the quest obtained orientation estimates wrap from $180°$ to $-180°$ at that point. This behavior is only present for the original GNSS factor, and may cause a problem if the estimates should have been used for feedback control.

Table 6.8 gives a clearer comparison for the yaw estimates. The original GNSS factor has the worst performance overall, but this might be strongly affected by the large deviation at the 300 s mark. The GPSBaseLineFactor performs better than both others on ME, but worse than GPSBaseLineFactor & GPSWithLeverArmFactor on MAE. For pitch and roll, the ME and MAE are quite similar for all three trajectories, but the combined GPSBaseLineFactor & GPSWithLeverArmFactor seem to perform somewhat better. The most outstanding value in the whole table is the Max error for yaw on the original GNSS factor, with a value of $140°$. The values for STD and RMSE are presented but do not bring much insight into the performance.

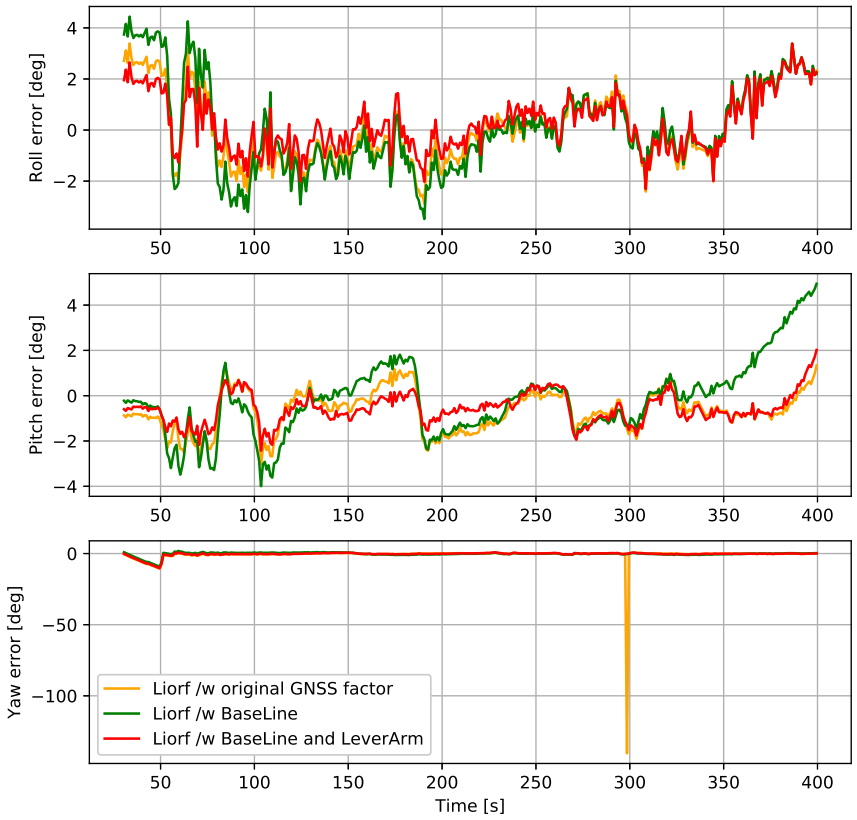


Figure 6.16: Roll, pitch, and yaw errors of different approaches to GNSS integration in Liorf factor graph. The errors are calculated using GNSS orientation estimates as a reference. The error obtained using the two newly developed factors is depicted in red, while the error with only the BaseLine factor is shown in green. The error using the original GNSS factor is represented in orange.

Table 6.8: Error metrics for Liorf's orientation estimates using different GNSS factors. The first trajectory corresponds to the original GNSS factor employed by Liorf, the second trajectory utilizes the two newly developed factors, and the third trajectory relies solely on the BaseLine factor. The GNSS orientation estimates are employed as a reference for error calculation.

Liorf GNSS factor comparison - orientation error					
	ME	MAE	STD	RMSE	Max error
Liorf original GNSS factor					
Roll [deg]:	0.050	1.171	1.416	1.415	3.392
Pitch [deg]:	-0.659	0.882	0.877	1.096	3.010
Yaw [deg]:	-0.877	1.000	7.468	7.509	140.435
GPSBaseLineFactor & GPSWithLeverArmFactor					
Roll [deg]:	0.260	0.908	1.100	1.129	3.381
Pitch [deg]:	-0.566	0.735	0.665	0.873	2.429
Yaw [deg]:	-0.429	0.596	1.487	1.545	10.295
GPSBaseLineFactor					
Roll [deg]:	0.069	1.395	1.740	1.739	4.441
Pitch [deg]:	-0.181	1.227	1.626	1.634	4.941
Yaw [deg]:	-0.272	0.685	1.359	1.384	9.194

6.7 Overall discussion

This thesis covers multiple research questions related to the chosen algorithm, Liorf. Firstly, the question of how what role different synchronization primitives play in the resulting pose estimates from Liorf. More specifically, the use of hardware timestamps is compared to the use of software timestamps. Before concluding, some points have to be discussed to get a clear view of what can be said regarding the results. These are points related to the choice of algorithm, synchronization primitive, sensor quality, host computer capacity, software performance, and the dynamics of the physical system.

6.7.1 Host computer

When performing a computational research problem, like the one related to the estimates from Liorf, the choice of hardware components may play a significant role, especially when the effect of time is studied. It is well known that different computers have different computational performance. Some might run a specific task 100 times faster than others, depending on the integrated circuit architecture and component quality. This process is in most computers not deterministic, meaning there is no guarantee of how much time a computation will take.

The host computer used in this project is considered fast and robust. At the point of choosing the host computer, the role of the different hardware components was not taken into consideration. An Intel NUC, which is the host computer, is usually used for desktop purposes and is much too big to be tightly integrated into a drone or a similar application. Compared to the host computer used in Storli (2022), which additionally used a virtual machine, it can be considered an extreme upgrade in computation power. If the experiments performed in Storli (2022) had used the same host computer as in this project, the results are believed to have been much different.

6.7.2 Generality of timestamping primitives

In this project, software timestamping was used to evaluate the performance of Liorf. When presented with the question of what quality software-timestamping and synchronization offers, there is no absolute answer. All software is man-made, which means that its structure and architecture can vary a lot. Additionally, the same software may perform differently when run on different hardware. As a consequence of this, there is no way to generalize the quality of software-timestamping and synchronization. The software quality in this project is given by the system components and setup presented in chapter 4.

Hardware timestamping can be generalized much more than its counterpart. Its accuracy boils down to much fewer factors. Since hardware synchronization often comes down to timestamping rising or falling edges on signal pulses. The accuracy is therefore dependent on the length of the wire the pulse propagates on. In a hypothetical scenario where the wire is one light-year long, the timestamp

will be off by one year. Additionally, the clock frequency in the host computer also affects the accuracy of hardware timestamping. In another hypothetical scenario where the clock frequency is 1 Hz, the accuracy of the given timestamp is 1 s. In some cases, the event to be timestamped may occur at the exact nanosecond the clock ticks, resulting in an extremely accurate timestamp. In another case, the event may happen at the exact time between two ticks, resulting in an error of 0.5 s. Due to the relatively short distances in system setups, and the high velocity of propagation speed, the effect of wire length is often neglected in accuracy evaluation.

6.7.3 Impact of system dynamics, sensors and SLAM algorithm

As presented in Storli (2022), the effect of timestamping and synchronization accuracy on the output of a particular SLAM algorithm is highly dependent on the dynamics of the physical system. The same goes for the case studied in this project. To get a full overview of the effect of hardware vs software synchronization in physical systems, a much wider range of physical systems must be covered. Only collecting a dataset from a car with relatively slow dynamics does not proved enough data to come up with a strong conclusion. Due to the LiDAR being an exteroceptive sensor, collecting data from a different environment could cause a completely different output. The Jonsvatnet dataset consists only of data from urban areas.

Liorf was chosen as SLAM algorithm to test the impact of software vs hardware synchronization. Since SLAM algorithms are such intricate systems, and the programmatic implementation may vary for the same algorithm, it is very difficult to come up whit a strong statement regarding the effect of timestamping primitive based only on testing one algorithm. No software or hardware module is free of bugs and errors, which may cause unwanted effects. These effects can mistakenly be thought to be due to the timestamping error, when in fact it may be caused by something completely else.

Sensors will also have different implementations, just like algorithms. There is no IMU that can be considered a general IMU. The same goes for GNSS and LiDAR. Different sensors may have different sources of error, depending on the internal structure. Sensors are the source of data and the first step in a long chain of data processing. There is hard to say how the different error types may propagate through the system and affect the end estimation result. Fixing all other impactful components in this project, like the algorithm, host computer, test case, and synchronization primitive, the effect of using a different sensor cannot be guaranteed to be insignificant.

Chapter 7

Conclusion and Further Work

This project investigates the effect of using hardware timestamping for synchronization compared to software timestamping, for sensor measurement in LiDAR-IMU-based SLAM. This was investigated by building a sensor payload capable of tagging measurements with both hardware and software timestamps. A car application was used to gather data, which also defines the physical system dynamics. The results indicate that there is no significant difference between the two timestamping methods. However, this conclusion can not be generalized to all LiDAR-IMU-based SLAM algorithms. Neither can it be generalized to all physical systems, nor all computational platforms. This is due to the huge variation concerning these factors. If a different computational platform, sensor manufacturer, timestamping software, or physical system was used, the results may have looked completely different for the exact same algorithm. However, the effect of software vs hardware synchronization on Liorf in this project can be concluded to have no significant effect.

A synchronization module for synchronizing a LiDAR with a SentiBoard based on encoder-based pulses was developed and used in the data collection. The timestamping accuracy was compared to the 1PPS-based timestamps used for LiDAR-SentiBoard synchronization in Storli (2022). Evaluating the results from both projects, it cannot be concluded which synchronization method is more accurate. It may come down to a trade-off between the accuracy of the encoder and the internal LiDAR clock. Moreover, the accuracy of the software-based timestamps produced by ROS is highly dependent on the hardware compromising the host machine, and the operating system installed.

Modifying the GNSS factor used in Liorf, by compensating for the lever arm between the LiDAR and the GNSS antenna, improves the accuracy of the pose estimates. Yet, the improvement is of such a small magnitude that it cannot be concluded to have any meaningful impact. Still, the results indicate that the factor is correctly implemented, and can be used further in applications running factor graphs for estimation. A factor based on the relative position of two GNSS antennas was developed and can be concluded to have a significant impact compared to only using LiDAR and IMU in Liorf. The development of these factors is a con-

tribution to the C++ factor graph library, GTSAM. Resulting in more flexibility in GNSS antenna placement on the vehicle.

The final takeaway from this project and the work in Storli (2022) is that the effect of synchronization and timestamping in LiDAR-IMU-based SLAM is highly dependent on the dynamics of the physical system. Regarding the question of whether to use software- or hardware-based timestamping in an application, the quality of the chosen sensors, host computer, and host computer software must be considered. The accuracy of LiDAR-IMU-based SLAM algorithms, based on the level of synchronization, must be seen as application-specific.

Further work

This thesis collected a multi-sensor dataset with high-precision hardware timestamps from an industry-relevant case. Since the effect of using hardware timestamps compared to software timestamps in Liorf is negligible, an interesting research question would be to inject artificial synchronization errors in the dataset and investigate at what level the estimates from the Liorf would break down completely. The synchronization models, presented in this thesis, could be used for producing artificial synchronization errors. and similar results as in Jellum et al. (2022) could be obtained.

The effect of timestamping and synchronization on other LiDAR-IMU-based estimation algorithms can be studied with the use of the collected dataset. An example of such an algorithm is FAST-LIO2, which is a state-of-the-art LiDAR-inertial odometry method (W. Xu et al., 2022a).

References

Ahmad, N., Ghazilla, R.A.R., Khairi, N.M., and Kasi, V., 2013. Reviews on various inertial measurement unit (imu) sensor applications. *International journal of signal processing systems*, 1(2), pp.256–262.

Albrektsen, S.M., 2022. *SentiSystems AS sentiboard envelope description* [Online]. Available from: `https://gitlab.senti.no/senti/senti-doc/-/blob/master/sentiboard/envelope.md` [Accessed August 6, 2023].

Aulinas, J., Petillot, Y., Salvi, J., and Lladó, X., 2008. The slam problem: a survey. *Artificial intelligence research and development*, pp.363–371.

Bailey, T., Nieto, J., Guivant, J., Stevens, M., and Nebot, E., 2006. Consistency of the ekf-slam algorithm. *2006 ieee/rsj international conference on intelligent robots and systems* [Online], pp.3562–3568. Available from: `https://doi.org/10.1109/IROS.2006.281644`.

Bloomenthal, J. and Rokne, J., 1994. Homogeneous coordinates. *The visual computer*, 11, pp.15–26.

Brekke, E.F., 2021. *Fundamentals of sensor fusion*. Edmund Brekke.

Cai, G., Chen, B.M., Lee, T.H., Cai, G., Chen, B.M., and Lee, T.H., 2011. Coordinate systems and transformations. *Unmanned rotorcraft systems*, pp.23–34.

Castellanos, J., Martinez-Cantin, R., Tardós, J., and Neira, J., 2007. Robocentric map joining: improving the consistency of ekf-slam. *Robotics and autonomous systems* [Online], 55(1). Simultaneous Localisation and Map Building, pp.21–29. Available from: `https://doi.org/https://doi.org/10.1016/j.robot.2006.06.005`.

Dellaert, F., 2012. *Factor graphs and gtsam: a hands-on introduction*. (technical report). Georgia Institute of Technology.

Dellaert, F., 2022. *Derivatives and differentials*. `https://github.com/borglab/gtsam/blob/develop/doc/math.pdf`. Accessed: 6 16, 2023.

Dellaert, F., 2023. *GTSAM factor graphs for sensor fusion in robotics.* [Online]. Available from: `https://gtsam.org/` [Accessed April 6, 2023].

Dellaert, F. and Kaess, M., 2017. Factor graphs for robot perception. *Foundations and trends® in robotics* [Online], 6(1-2), pp.1–139. Available from: https://doi.org/10.1561/2300000043.

Elson, J., Girod, L., and Estrin, D., 2002. Fine-grained network time synchronization using reference broadcasts. *Acm sigops operating systems review*, 36(SI), pp.147–163.

Farrell, J., 2008. *Aided navigation: gps with high rate sensors*. McGraw-Hill, Inc., pp.19–58.

Forster, C., Carlone, L., Dellaert, F., and Scaramuzza, D., 2016. On-manifold preintegration for real-time visual–inertial odometry. *Ieee transactions on robotics*, 33(1), pp.1–21.

Fossen, T., 2021. *Handbook of marine craft hydrodynamics and motion control*. 1st. John Wiley & Sons.

Grisetti, G., Kümmerle, R., Stachniss, C., and Burgard, W., 2010. A tutorial on graph-based slam. *Ieee intelligent transportation systems magazine* [Online], 2(4), pp.31–43. Available from: https://doi.org/10.1109/MITS.2010.939925.

Groves, P.D., 2013. *Principles of gnss, inertial, and multisensor integrated navigation systems*. 2nd. Artech House, pp.300–344.

Huang, G.P., Mourikis, A.I., and Roumeliotis, S.I., 2010. Observability-based rules for designing consistent ekf slam estimators. *The international journal of robotics research*, 29(5), pp.502–528.

Huang, X., Mei, G., Zhang, J., and Abbas, R., 2021. A comprehensive survey on point cloud registration. *Arxiv preprint arxiv:2103.02690*.

Jellum, E.R., Bryne, T.H., Johansen, T.A., and Orlandíc, M., 2022. The syncline model–analyzing the impact of time synchronization in sensor fusion. *Arxiv preprint arxiv:2209.01136*.

Kaess, M., Johannsson, H., Roberts, R., Ila, V., Leonard, J.J., and Dellaert, F., 2012. Isam2: incremental smoothing and mapping using the bayes tree. *The international journal of robotics research*, 31(2), pp.216–235.

kartverket, 2023. *Kartverket kartverket base station gnss data* [Online]. Available from: https://etpos.kartverket.no/ [Accessed April 4, 2023].

Konolige, K., Bowman, J., Chen, J., Mihelich, P., Calonder, M., Lepetit, V., and Fua, P., 2010. View-based maps. *The international journal of robotics research*, 29(8), pp.941–957.

Leonard, J.J. and Durrant-Whyte, H.F., 1991. Mobile robot localization by tracking geometric beacons. *Ieee transactions on robotics and automation*, 7(3), pp.376–382.

Lu, F. and Milios, E., 1997. Globally consistent range scan alignment for environment mapping. *Autonomous robots*, 4, pp.333–349.

Lv, J., Xu, J., Hu, K., Liu, Y., and Zuo, X., 2020. Targetless calibration of lidar-imu system based on continuous-time batch estimation. *2020 ieee/rsj international conference on intelligent robots and systems (iros)* [Online], pp.9968–9975. Available from: https://doi.org/10.1109/IROS45743.2020.9341405.

Mills, D.L., 1991. Internet time synchronization: the network time protocol. *Ieee transactions on communications*, 39(10), pp.1482–1493.

Misra, P. and Enge, P., 2012. *Global positioning system: signals, measurements, and performance*. 2nd Rev. Ganga-Jamuna Press.

Mogul, J., Mills, D., Brittenson, J., Stone, J., and Windl, U., 2000. *Pulse-per-second api for unix-like operating systems, version 1.0*. (technical report).

O'Riordan, A., Newe, T., Dooly, G., and Toal, D., 2018. Stereo vision sensing: review of existing systems. *2018 12th international conference on sensing technology (icst)*. IEEE, pp.178–184.

Ouster, 2022. *Os1 hardware user manual* [Online]. Ouster. Available from: https://ouster.com/downloads/.

Raj, T., Hanim Hashim, F., Baseri Huddin, A., Ibrahim, M.F., and Hussain, A., 2020. A survey on lidar scanning mechanisms. *Electronics*, 9(5), p.741.

Robbes, D., 2006. Highly sensitive magnetometers—a review. *Sensors and actuators a: physical*, 129(1-2), pp.86–93.

ROS Wiki, 2018. http://wiki.ros.org/ROS/Introduction. Accessed: 5 16, 2023.

rtklibexplorer, 2023. *RTKlib kartverket base station gnss data* [Online]. Available from: https://github.com/rtklibexplorer/RTKLIB/releases [Accessed April 4, 2023].

S. M. Albrektsen, T.A.J., 2018. Reconfigurable sensor timing and navigation system for uavs. *Sensors*, 18.

Särkkä, S., 2008. Unscented rauch–tung–striebel smoother. *Ieee transactions on automatic control*, 53(3), pp.845–849.

SentiSystems, 2023. *SentiSystems AS sensor fusion - empowering autonomy* [Online]. Available from: https://sentisystems.com/.

Shan, T., Englot, B., Meyers, D., Wang, W., Ratti, C., and Rus, D., 2020. Lio-sam: tightly-coupled lidar inertial odometry via smoothing and mapping. *2020 ieee/rsj international conference on intelligent robots and systems (iros)* [Online], pp.5135–5142. Available from: https://doi.org/10.1109/IROS45743.2020.9341176.

Shuster, M.D. and Oh, S.D., 1981. Three-axis attitude determination from vector observations. *Journal of guidance and control*, 4(1), pp.70–77.

Sivrikaya, F. and Yener, B., 2004. Time synchronization in sensor networks: a survey. *Ieee network* [Online], 18(4), pp.45–50. Available from: https://doi.org/10.1109/MNET.2004.1316761.

Smith, H.M., 1976. Greenwich time and the prime meridian. *Vistas in astronomy* [Online], 20, pp.219–229. Available from: https://doi.org/https://doi.org/10.1016/0083-6656(76)90039-8.

Stevens, B.L., Lewis, F.L., and Johnson, E.N., 2015. *Aircraft control and simulation: dynamics, controls design, and autonomous systems*. John Wiley & Sons.

Storli, H.S., 2022. *Timing and time synchronization within lidar- and imu-based simultaneous localization and mapping (slam)*. specialization project. Unpublished.

Taheri, H. and Xia, Z.C., 2021. Slam definition and evolution. *Engineering applications of artificial intelligence* [Online], 97, p.104032. Available from: https://doi.org/https://doi.org/10.1016/j.engappai.2020.104032.

Teunissen, P.J. and Montenbruck, O., 2017. *Springer handbook of global navigation satellite systems*. Vol. 10. Springer.

ublox, 2021. *u-blox F9 HPG 1.30 u-blox f9 high precision gnss receiver* (v.R01). ublox.

Varda, K., 2008. *Protocol buffers*. https://developers.google.com/protocol-buffers/.

Watt, S.T., Achanta, S., Abubakari, H., Sagen, E., Korkmaz, Z., and Ahmed, H., 2015. Understanding and applying precision time protocol. *2015 saudi arabia smart grid (sasg)* [Online], pp.1–7. Available from: https://doi.org/10.1109/SASG.2015.7449285.

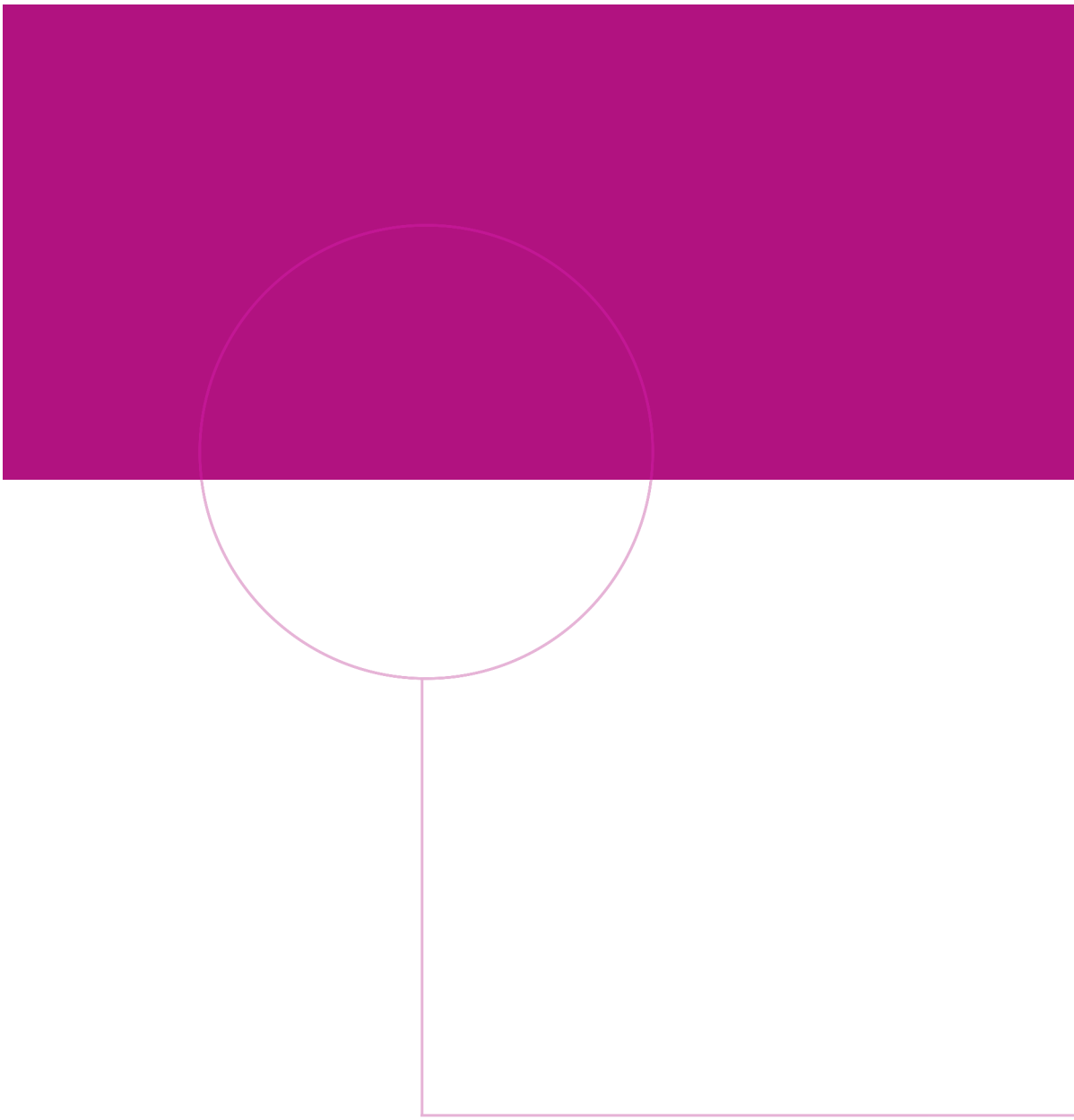
Xu, W., Cai, Y., He, D., Lin, J., and Zhang, F., 2022a. Fast-lio2: fast direct lidar-inertial odometry. *Ieee transactions on robotics* [Online], 38(4), pp.2053–2073. Available from: https://doi.org/10.1109/TRO.2022.3141876.

Xu, X., Zhang, L., Yang, J., Cao, C., Wang, W., Ran, Y., Tan, Z., and Luo, M., 2022b. A review of multi-sensor fusion slam systems based on 3d lidar. *Remote sensing*, 14(12), p.2835.

Ye, H., Chen, Y., and Liu, M., 2019. Tightly coupled 3d lidar inertial odometry and mapping. *2019 international conference on robotics and automation (icra)*. IEEE, pp.3144–3150.

Zhang, J. and Singh, S., 2014. Loam: lidar odometry and mapping in real-time. *Robotics: science and systems*. Vol. 2, 9. Berkeley, CA, pp.1–9.

Zhang, J. and Singh, S., 2017. Low-drift and real-time lidar odometry and mapping. *Autonomous robots*, 41(2), pp.401–416.



NTNU

Norwegian University of
Science and Technology