Master's thesis

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

Sjur Grønnevik Wroldsen

LiDAR-Inertial SLAM System for Tunnel Navigation of an Autonomous Road Roller

Master's thesis in Cybernetics and Robotics
Supervisor: Annette Stahl
Co-supervisor: Sondre Midtskogen
June 2021

NTNU
Norwegian University of
Science and Technology

Sjur Grønnevik Wroldsen

LiDAR-Inertial SLAM System for Tunnel Navigation of an Autonomous Road Roller

Master’s thesis in Cybernetics and Robotics
Supervisor: Annette Stahl
Co-supervisor: Sondre Midtskogen
June 2021

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

Abstract

Autonomous operation can reduce human exposure to hazardous environments. Additionally, it can improve time- and cost-effectiveness since several autonomous vehicles can be operated by a single operator without being physically present. This is favorable for many industries, among others the road construction industry. The problem of autonomous operation of a road roller is the subject of study in this thesis. Autonomous operation requires robust systems for navigation. Typically navigation systems rely on inertial sensors aided by measurements from Global Navigation Satellite System (GNSS) to keep their estimates globally consistent. In the case of poor satellite coverage, however, these navigation systems tend to drift quickly.

To help mitigate this drift, exteroceptive sensors such as a camera and/or a Light Detection and Ranging (LiDAR) sensors are typically used. This thesis proposes a simple feature-based LiDAR-Inertial Simultaneous Localization and Mapping (SLAM) system. The system fuses information from a LiDAR, an Inertial Measurement Unit (IMU), and a GNSS receiver for the problem of tunnel navigation.

The performance of the system is then evaluated on two simulated scenarios. To the author's knowledge, there exist no simulated data sets for this particular application. Therefore a simulation environment was constructed in the pre-project [1]. This framework is extended to include inertial sensors and GNSS in this thesis. The results show that the LiDAR-Inertial SLAM system can provide a reasonable estimate of its position, producing a Root Mean Square Error (RMSE) of 3.12m on a trajectory performing a round trip going through the tunnel before turning and traversing back through the same tunnel. However, the required standard for documentation states that position must be guaranteed within $\pm 0.2m$, so the performance is not in compliance with the standard. The results do further show that the system underestimates its covariance, giving an Average Normalized Estimation Error Squared (ANEES) of 9.85 with the 95% confidence bounds of [2.65, 3.37]. The results were achieved after smoothing when returning to the start position after re-exiting the tunnel.

Sammendrag

Autonom drift kan redusere menneskelig utsettelse mot farlige miljøer. Det kan også forbedre tids- og kostnadseffektiviteten ved at en menneskelig operator kan operere flere autonome kjøretøyer på en gang uten å være fysisk tilstede. Dette er gunstig for mange industrier, deriblant for veikonstruksjon. Denne avhandlingen ser på problemet om autonom operasjon av en veivals. Autonom operasjon krever robuste systemer for navigasjon. Tradisjonelt er slike systemer avhengige av å kombinere treghetssensorer med globale satelittsystemer for navigasjon (GNSS) for å holde estimatene globalt konsistente. I situasjoner med dårlig satelittdekning vil derimot slike systemer ha en tendens til å opparbeide drift.

For å minske denne driften blir sensorer som kamera eller laserskannere (LiDAR) ofte brukt. Denne avhandlingen foreslår et simpelt 'feature'-basert LiDAR-Inertial SLAM system som fusjonerer sensorinformasjon fra en IMU, en LiDAR og en GNSS mottaker for tunnelnavigasjonsproblemet.

Systemet blir deretter evaluert på to ulike simulerte scenarioer. Til forfatterens kjennskap eksisterer det ingen datasett til denne applikasjonen. Derfor ble det i forprosjektet [1] utviklet en simulator i MATLAB. Simulatoren blir i denne avhandlingen utvidet til å inkludere en IMU og GNSS. Resultatene viser at LiDAR-Inertial SLAM-systemet klarer å oppnå et bra estimat av posisjonen sin i planet, noe som resulterer i en Root Mean Square Error (RMSE) på 3.12m på en tur-retur bane gjennom tunnelen. Standarden for dokumentasjon av kompresjon av materiale sier at posisjonen må kunne måles med en nøyaktighet på $\pm 0.2m$. Etter disse standardene presterer ikke systemet bra nok. Resultatene viser videre at systemet undervurderer kovariansen, noe som gir en Average Normalized Estimation Error Squared (ANEES) på 9.85, der 95% konfidensintervallet tilsier at den skulle lagt innenfor intervallet [2.65, 3.37]. Disse resultatene ble tatt etter 'glattingen' har forekommet når veivalsen forlater tunnelen for siste gang.

Preface

This thesis is submitted in fulfillment of a degree as Master of science at the Department of Engineering Cybernetics, Norwegian University of Science and Technology.

The work in this thesis extends the work done in the course TTK4550 ([1]). As a result of this, sections 3.2, 3.5, and 3.6-3.10 are heavily inspired by the contents of [1].

Working with this thesis has been exciting and challenging. It has given me a broad insight on the challenges of creating simulated data, and the planning and considerations that are necessary when designing a system from scratch.

Acknowledgments

I would like to express my great appreciation to my supervisors Annette Stahl (NTNU) and Sondre Midtskogen (Semcon) for their guidance throughout the entirety of the semester. A special thanks also to Eirik Hexeberg Henriksen (Yeti Move) who also provided great insight and guidance during both the project thesis and in the creation of this thesis.

I am particularly grateful for my housemates Aksel Heggernes, Herman Jakobsen, Iver Myklebust, and Vebjørn Rognli and their support and comradery throughout the year and for the entirety of my study.

Finally, I wish to offer my special thanks to my parents and family for their continuous support and encouragement ever since I decided to pursue a master's degree in technology at NTNU.

Contents

Figures

Tables

Acronyms

ANEES Average Normalized Estimation Error Squared. iii, v, xvii, 27, 52, 61, 62, 66, 67, 71

BA Bundle Adjustment. 7

BoW Bag-of-Words. 7, 26

DoF Degrees of Freedom. xiii, 15, 16, 26, 33, 49, 50, 59

EKF Extended Kalman Filter. 8, 13, 24

ESKF Error-State Kalman Filter. 13

FOV Field of View. 3, 7

FPFH Fast Point Feature Histogram. 23, 32, 35

GN Gauss-Newton. 17, 18

GNC Guidance, Navigation and Control. xiii, 11, 12

GNSS Global Navigation Satellite System. iii, v, xiii–xv, xvii, 1–5, 9, 13, 14, 25, 29, 32, 34–38, 41, 42, 44, 45, 47–54, 56, 58, 59, 63, 65–68, 71, 72

GTSAM Georgia Tech Smoothing and Mapping. 30

HSE Health, Safety, and Environment. 1

ICP Iterative Closest Point. 3, 4, 8, 21, 22, 26, 31, 37, 38

IMU Inertial Measurement Unit. iii, v, xiv, xv, xvii, 1, 3–5, 7, 12–14, 25, 29, 30, 32–35, 37, 38, 41–45, 47–53, 63, 65, 67, 71, 72

INS Inertial Navigation System. 1, 3, 11–14

iSAM2 Incremental Smoothing and Mapping 2. 3, 9, 30

ISS Intrinsic-Shape Signatures. 22, 32, 35

KF Kalman Filter. 13, 14, 26

Lego-LLOAM Lighweight and Ground-Optimized LiDAR Odometry And Mapping. 4, 8, 9

LiDAR Light Detection and Ranging. iii, v, ix, x, xiv, xv, xvii, 2–5, 7, 8, 12, 13, 20, 26, 29, 32–36, 38–54, 63–65, 67, 68, 71–73

LIO LiDAR-Inertial Odometry. 8, 9

LM Levenberg-Marquardt. 8, 17, 18

LOAM LiDAR Odometry And Mapping. 4, 8, 9, 67, 72

MAP Maximum A Posteriori. 15, 24, 25

NDT Normal Distribution Transformation. 21, 22

NEES Normalized Estimation Error Squared. xv, xvi, 4, 26, 27, 42, 48–51, 59, 61, 65–68

PCL Point Cloud Library. 31, 32

pdf probability density function. xiii, 14–16

PFH Point Feature Histogram. 23

RANSAC Random Sample Consensus. 63

RMSE Root Mean Square Error. iii, v, xvii, 4, 26, 52, 61, 62, 65, 69, 71

ROS Robotics Operating System. xiv, 30, 32, 35

SE(3) Special Euclidean Group. 19, 20

SLAM Simultaneous Localization and Mapping. iii, v, ix, x, 2–5, 7, 8, 11, 24–27, 29, 34–38, 40, 54, 65, 67, 69, 71, 72

SO(3) Special Orthogonal Group. 19

ToF Time-of-Flight. 13

VO Visual Odometry. 7

Chapter 1

Introduction

This master thesis is written in collaboration with Semcon Norge AS, who recently started a collaboration project with SINTEF where the goal is to build a completely autonomous road roller [2]. The road construction industry exposes its employees to environments with a lot of hazardous noise and vibrations. Therefore, remote operation/surveillance yields a high Health, Safety, and Environment (HSE) benefit for the employees. Additionally, the operation would be more cost- and time-efficient, as a single operator could operate several vehicles at once from a control room.

Autonomous vehicles require significant testing to ensure the safety of people and the environment. In order to properly test a system, it is crucial to have some indication of what the position of the system *actually* is, popularly deemed the ground truth in estimation theory.

1.1 Motivation

Norway's vast mountains and hills provide a challenge for the road construction and autonomous vehicle industry, namely tunnels. The main requirement for a good autonomous navigation system is a good method for localization of the vehicle. In general, these methods rely heavily on the availability of high-precision globally fixed measurements such as from a Global Navigation Satellite System (GNSS). Systems such as Inertial Navigation System (INS) use the GNSS measurements to correct for drift from the Inertial Measurement Unit (IMU). The issue arises when an autonomous vehicle drives into an area where the GNSS measurements are not available. This is the case when a vehicle drives through a tunnel, as illustrated in Figure 1.1.

The vehicle in focus is a road roller. A road roller's main objective is to compact the concrete, soil, or gravel located in its operation area. The Norwegian standard for road construction states that the soil compression has to cover the compression area a certain amount of times, with a few dependencies that can be found in the manual [3, p. 185-200]. The manual also states that these *passings* has to be well-documented. For the problem in focus, the road roller has to cover an area six times. Through a tunnel this would require three round-trips.

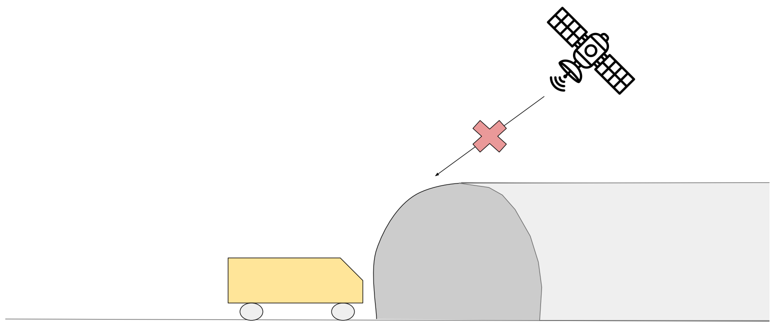


Figure 1.1: No GNSS Coverage: The problem with relying on GNSS correction for autonomous navigation arises when driving from an area with GNSS coverage to an area with no GNSS coverage. The figure shows a vehicle driving into a tunnel, therefore losing its GNSS coverage.

GNSS-denied navigation is a nontrivial problem because no measurements are rooting the system to the real world. Without relying on costly inertial sensors, it is generally hard to ensure that the vehicle provides accurate localization inside this unknown environment. This inaccuracy is not acceptable for a safe autonomous system. Modern methods mitigate errors that accumulate by creating a map while traversing the unknown environment. This is known as the Simultaneous Localization and Mapping (SLAM) problem.

SLAM is a problem addressing the complexity involved in estimating structure and pose inside an unknown environment simultaneously. Solving these problems requires extensive computational power and exteroceptive sensors capable of mapping the environment, such as a camera or a Light Detection and Ranging (LiDAR) sensor.

All exteroceptive sensors have varied performances under different environmental conditions. For example, cameras struggle under variable illumination conditions, and LiDARs cannot guarantee proper measurements, for example, when it rains. Tunnels are generally deemed dark environments, and therefore a camera is not the optimal choice for tunnel navigation. LiDARs, on the other hand, do not suffer under poor illumination conditions, making it a natural choice for this type of problem.

A weakness of LiDAR motion estimation techniques is that the LiDAR records many points at every scan. To utilize the full scan requires heavy computational effort.

Tunnel walls are typically clad with rocky surfaces, making it a feature-rich environment. Consequently, the computational complexity of the SLAM problem could be mitigated by letting feature points represent each observed point cloud. This approach is studied in [1].

Most autonomous navigation system already consists of an INS. Therefore, a framework is needed for an effective and optimal fusion of the sensor information. Modern SLAM systems are typically graph-based, and Incremental Smoothing and Mapping 2 (iSAM2) [4] provides a factor graph framework utilizing other graph structures such as the Bayes' tree for optimal ordering and incremental smoothing.

Recording ground truth is particularly challenging for any indoor environment, since there are no global indications of vehicle position, such as GNSS measurements. GNSS measurements are often interpolated to acquire ground truth, but this is not possible for indoor applications. Creating indoor ground truth data sets can be done by manual setup, however this requires a lot of work for each designed scenario and cannot guarantee exact results. Consequently, effective and realistic simulators are often designed instead. In these simulators the designer is in complete control of the environment at all times.

1.2 Aim of Study

The goal of this study is to analyze the performance of a LiDAR-inertial SLAM system in a simulated environment. The simulated environment is inspired by the autonomous road-roller project described in [2]. The problem is simply illustrated in Figure 1.1. There exist no validated data sets for this particular problem of tunnel navigation using ground vehicles to the author's knowledge. In order to have access to ground truth, a secondary goal of this master thesis is to develop a realistic simulator that can easily simulate several scenarios where the system can be tested.

For the problem of navigating an autonomous road roller through a tunnel, a system based on the fusion of LiDAR, IMU, and GNSS sensors is proposed. LiDARs provide a more robust solution to the problem compared to other mapping sensors because of the various illumination conditions that may arise inside tunnels. Another benefit of using LiDARs is that they may provide a higher situational awareness compared to, for example, cameras since they have a higher Field of View (FOV). This is desirable for several reasons. It can be used by path-planning and collision-avoidance modules of the system, which are essential to all autonomous systems. By using effective feature extraction and matching methods, the added complexity can be kept manageable. I suggest a feature-based point cloud registration system based on [1], with a map alignment based on a custom ICP algorithm and loop closure detection as suggested in [5]. Further, the optimization of the system is implemented using iSAM2, where the LiDAR, IMU, and GNSS estimations are fused and incrementally smoothed. The data sets used to validate the system are created in the simulation environment detailed in [1], which is also extended to simulate IMU and GNSS measurements. The performance of this system will be assessed using Root Mean Square Error (RMSE) and a goodness-of-fit measure based on the Normalized Estimation Error Squared (NEES), as suggested in [6].

1.3 Contributions

The development started by implementing a similar system as suggested in the pre-project [1] in C++. The Python system from the pre-project encountered severe performance issues in terms of processing speed. Additionally, the libraries used were uncompleted bindings from C++, which significantly reduced the options during development.

After converting the system to C++, a custom ICP algorithm based on [7] for aligning the currently viewed point cloud to the map was implemented. A custom algorithm was chosen over existing solutions to include whitening based on uncertainties and to include prior information from sensors like the IMU. This map alignment was used following the ideas of systems such as Lego-LLOAM [5] and LOAM [8], where they argue that this step reduces drift in the system.

After that, the MATLAB simulator was extended to include support for IMU and GNSS measurements. This involved creating a more realistic vehicle model, which was done by simulating a simple bicycle model as a vehicle. Additionally, a simple path planner and controllers for the velocity and heading were set up to make the vehicle model follow a specified route. After that, IMU simulations were easily implemented by using MATLABs toolboxes. GNSS measurements were simulated as normally distributed values around the positions outside the tunnel. Finally, datasets were created and exported to a rosbag.

Afterward, the development of the SLAM system itself started. It started as a literature review of sensor fusion in a graph-SLAM system to provide an implementation that was as effective as possible. Accordingly, the fusion of IMU and GNSS information was added to the graph-SLAM system.

The final step of the project was to improve the global consistency of the system. Inspired by the literature review from [1], the loop closure system from [5] was implemented. This method was chosen because of its simplicity, in addition to its effectiveness. However, a challenge was encountered when viewing the same scene from two different angles.

My main contributions can be summarized as follows

- An extensive theory review surrounding the SLAM problem.
- An extended, highly modifiable, simulation framework for creating data sets for tunnel navigation. The simulation includes data from LiDARs, IMUs, and

GNSS’s.
- A SLAM system implemented in C++ based on the fusion of LiDAR, IMU, and GNSS sensory information.
- A study of the consistency and the performance of the mentioned SLAM system with respect to uncertainty and estimation error.

1.4 Outline

The remainder of the thesis is organized as follows. Chapter 2 gives an overview of related work and state-of-the-art methods. Chapter 3 gives a review of the preliminaries required to follow the methods mentioned in this thesis. Chapter 4 gives an overview of the software, the data generation, and the architecture and processes of the SLAM system. Chapter 5 presents the results achieved by the system. In Chapter 6, the results are discussed. Lastly, Chapter 7 gives a conclusion of the thesis and provides pointers for further development.

Chapter 2

Related Work

This chapter provides a summary of literature and systems relevant to this thesis. Section 2.1 provides a review of modern SLAM systems. Section 2.2 gives a review of state-of-the-art LiDAR SLAM systems. Finally, Section 2.3 gives a review of a modern IMU preintegration scheme.

2.1 Review of modern SLAM systems

Modern navigation systems typically estimate motion using some visual sensors, such as a camera. These are called Visual Odometry (VO) systems, and motion is estimated by tracking features between consecutive camera images. VO has shown promise in terms of drift compared to classical odometry sensors, such as Inertial Measurement Unit (IMU) and wheel encoders [9]. VO systems typically solve the Bundle Adjustment (BA) problem to estimate structure and/or motion.

Visual SLAM differs from VO by including loop closure to achieve *global consistency*. There exist many different algorithms for managing loop closures. A common way to solve the loop closure problem is to use a Bag-of-Words (BoW) model. Modern visual-SLAM that implements loop closure includes ORB-SLAM [10] and its predecessors, LSD-SLAM [11] and PTAM [12].

Cameras do, however, rely on specific environmental conditions, such as illumination. Additionally, cameras are typically restricted to a low Field of View (FOV). As autonomous vehicles need to have a high situational awareness to plan action optimally, the low FOV of the cameras has motivated the usage of other sensors for the SLAM problem. A technology that has shown promise for solving the SLAM problem is LiDAR sensors. LiDARs offer real-time scanning of the environment with up to 360° FOV.

LiDAR SLAM methods are, as opposed to visual SLAM methods, unaffected by illumination conditions [13]. The LiDAR gives the opportunity to very accurately map the surrounding environments.

The motion estimation techniques in LiDAR SLAM are typically divided into two categories; scan-matching and feature-based methods. Scan-matching methods typically use the Iterative Closest Point (ICP) method, which aims at minimizing the distance between two point clouds. As the LiDAR sensor provides points at a high resolution, the challenge of LiDAR SLAM methods is the significant computational complexity associated with point cloud registration. To battle this, modern LiDAR SLAM systems are typically based on the feature-based ideology. This feature extraction involves computing robust keypoints and feature descriptors and utilizing KD-trees to significantly reduce the computations needed in the registration process. This approach is studied in [1]. Further, [7] suggests an ICP algorithm using Levenberg-Marquardt (LM) optimization to speed up the algorithm.

Relying on a single modality is generally not a good idea for an autonomous navigation system. Sensors have different operating conditions in which they operate well, so the fusion of sensors that complement each other could prove beneficial for a versatile SLAM system. Additional sensors may also massively improve the navigation module. Therefore, modern LiDAR navigation systems often fuse information from other sensors. Systems that fuse LiDAR and inertial sensors to produce an estimate of motion are termed LiDAR-Inertial Odometry (LIO) systems.

The original formulations of the SLAM problem suffered from maintaining large state spaces. The first solutions to the SLAM problem were based on the classical navigation paradigm, using Extended Kalman Filter (EKF) filters. However, this quickly turned out to be an inefficient solution because it involved keeping track and inverting large covariance matrices. This was improved using a Rao-Blackwellized particle filter in FAST-SLAM [14]. Although the particle filter solution had its strengths, the inclusion of loop closure proved difficult due to the absence of an explicit covariance matrix [15].

To overcome the computational complexity, but still keep an explicit covariance matrix, modern SLAM methods typically formulate the navigation problem using graphs, like the factor graph as suggested in [16]. This representation allows exploitation of the sparsity of the SLAM problem while providing a simple framework for adding any sensor information.

2.2 State-of-the-art LiDAR SLAM systems

The related work in this section is based on the related work written for the pre-project [1]. Most state-of-the-art LiDAR SLAM systems are based on LOAM [8], which proposed an odometry estimation using lines in range-images by dividing the estimation into a fast part providing rough estimates and a slow part providing fine estimates. One system that built on this formulation was Lego-LLOAM

[5], which is specialized for ground vehicles. Lego-LLOAM was also extended to include loop closures. LIO-SAM [17] suggested a system based on the feature extraction from LOAM including inertial sensor fusion and inclusion of GNSS measurements. iSAM2 [4] provides a suitable framework for fusing information from many different sensors in a factor graph while providing optimal ordering and smoothing incrementally. For more details, the reader is strongly suggested to read [4, 5, 8, 17].

2.3 IMU Preintegration On Manifold

Typically IMUs provide measurements at a much higher rate compared to other sensors. Including a factor for each of these measurements would require increasing the number of variables in the factor graph by an order of magnitude. This is clearly not desirable, as it would slow down the optimization operation in the factor graph. As a solution to this problem, [18] suggests a method for preintegrating IMU measurements on the manifold to use in the factor graph. Section 2.3.1 derives the IMU state kinematics and section 2.3.2 explains how this model is utilized to achieve preintegration of IMU factors on the manifold.

2.3.1 IMU Model

An IMU measures linear acceleration and the angular rates of the body frame, b, referenced to some inertial frame. These measurements can be modeled as the true acceleration and rates corrupted by white noise and a slowly developing bias.

$$\tilde{\boldsymbol{\omega}}^b = \boldsymbol{\omega} + \mathbf{b}_g + \boldsymbol{\eta}_g \tag{2.1}$$

$$\tilde{\mathbf{a}}^b = \mathbf{R}^T(\mathbf{a}^w - \mathbf{g}^w) + \mathbf{b}_a + \boldsymbol{\eta}_a \tag{2.2}$$

where the superscript denotes the coordinate frame, $\mathbf{R}$ is the rotation matrix transforming vectors in the body frame to the inertial frame, $\mathbf{g}^w$ is the gravity given in world coordinates, $\mathbf{b}_g$, $\mathbf{b}_a$, and $\boldsymbol{\eta}_g$, $\boldsymbol{\eta}_a$ are the biases and noises on the gyroscope and the accelerometer, respectively. Effects from the earth's rotation are neglected by assuming that the world coordinates are inertial. Further, following [18], the IMU states behave according to the discretized kinematic model

$$\mathbf{R}(t+\Delta t) = \mathbf{R}(t)\mathrm{Exp}((\tilde{\boldsymbol{\omega}}^b - \mathbf{b}_g(t) - \boldsymbol{\eta}_g(t))\Delta t) \tag{2.3}$$

$$\mathbf{v}^w(t+\Delta t) = \mathbf{v}^w(t) + \mathbf{g}^w\Delta t + \mathbf{R}(t)(\tilde{\mathbf{a}}^b(t) - \mathbf{b}_a(t) - \boldsymbol{\eta}_a(t))\Delta t \tag{2.4}$$

$$\mathbf{p}^w(t+\Delta t) = \mathbf{p}^w(t) + \mathbf{v}^w(t)\Delta t + \frac{1}{2}\mathbf{g}^w\Delta t^2 + \frac{1}{2}\mathbf{R}(t)(\tilde{\mathbf{a}}^b(t) - \mathbf{b}_a(t) - \boldsymbol{\eta}_a(t))\Delta t^2 \tag{2.5}$$

where the Exp(·)-operator is defined as in Section 3.6.

2.3.2 Preintegration on Manifold

Preintegration of IMU measurements involves accumulating several measurements into one total factor between so-called 'key frames'. Let the frames at time $k = i$ and $k = j$ be denoted by the subscripts i and j. For example, the orientation at time $k = i$ is denoted $\mathbf{R}_i$ and the relative orientation between time $k = i$ and $k = j$ is denoted $\Delta\mathbf{R}_{ij} := \mathbf{R}_i^T\mathbf{R}_j$. To avoid having to repeat the integration every time a relinearization happens, all IMU preintegration factors are defined through a relative *preintegrated measurement model* derived in [18], given by

$$\Delta\tilde{\mathbf{R}}_{ij} = \mathbf{R}_i^T\mathbf{R}_j\mathrm{Exp}(\delta\boldsymbol{\phi}_{ij}) \tag{2.6}$$

$$\Delta\tilde{\mathbf{v}}_{ij} = \mathbf{R}_i^T(\mathbf{v}_j - \mathbf{v}_i - \mathbf{g}\Delta t_{ij}) + \delta\mathbf{v}_{ij} \tag{2.7}$$

$$\Delta\tilde{\mathbf{p}}_{ij} = \mathbf{R}_i^T(\mathbf{p}_j - \mathbf{p}_i - \mathbf{v}\Delta t_{ij} - \mathbf{g}\Delta t_{ij}^2) + \delta\mathbf{p}_{ij} \tag{2.8}$$

where the superscripts from (2.3) have been dropped for readability. The model is derived under the assumption of constant bias for simplicity and can be extended to include slowly varying biases, as shown in [18], using first-order expansions. The model describes the measurements as a function of the state to be estimated plus some random noise. The noise can be defined by

$$[\delta\boldsymbol{\phi}_{ij}^T, \delta\mathbf{v}_{ij}^T, \delta\mathbf{p}_{ij}^T]^T \sim \mathcal{N}(\mathbf{0}, \Sigma_{ij}) \tag{2.9}$$

under the assumption that $\delta\boldsymbol{\phi}_{ij}$ is small.

Finally the residuals, which are used as factors in the factor graph, can be written as

$$\mathbf{r}_{\Delta\mathbf{R}_{ij}} := \mathrm{Log}(\mathbf{R}_i^T\mathbf{R}_j) \tag{2.10}$$

$$\mathbf{r}_{\Delta\mathbf{v}_{ij}} := \mathbf{R}_i^T(\mathbf{v}_j - \mathbf{v}_i - \mathbf{g}\Delta t_{ij}) \tag{2.11}$$

$$\mathbf{r}_{\Delta\mathbf{p}_{ij}} := \mathbf{R}_i^T(\mathbf{p}_j - \mathbf{p}_i - \mathbf{v}\Delta t_{ij} - \mathbf{g}\Delta t_{ij}^2) \tag{2.12}$$

Again, [18] explains how to include non-constant biases into the factors and add factors between biases in the factor graph. The main benefit of this method is that the integration does not commit to a linearization point, making it faster in the case of relinearization. Also, the uncertainty is propagated on the manifold, improving the correctness.

Chapter 3

Preliminaries

This chapter describes all the preliminary theory and information needed to understand the rest of the thesis. The chapter is organized as follows. Section 3.1 sets the scope of the theory and defines the usage. Section 3.2 gives an introduction to the sensors relevant to the system. Section 3.3 gives a brief explanation of Inertial Navigation System (INS). Sections 3.4-3.6 provides the necessary tools to perform optimization for the problem. Sections 3.7-3.9 explains the elements necessary for odometry estimation. Sections 3.10-3.12 walks through the modern framework for representing the SLAM problem. Finally, Section 3.13 goes through the metrics used to analyze the system performance.

3.1 Autonomous Vehicles

Autonomous operation of vehicles is typically divided into several subsystems. These subsystems are part of a Guidance, Navigation and Control (GNC) system. The GNC system is shown in Figure 3.1. The colored boxes symbolize that the systems are fully up to design by the developer. However, there is no way to know how the true state of the vehicle develops after the commands from the actuators, except through internal or external sensor information.

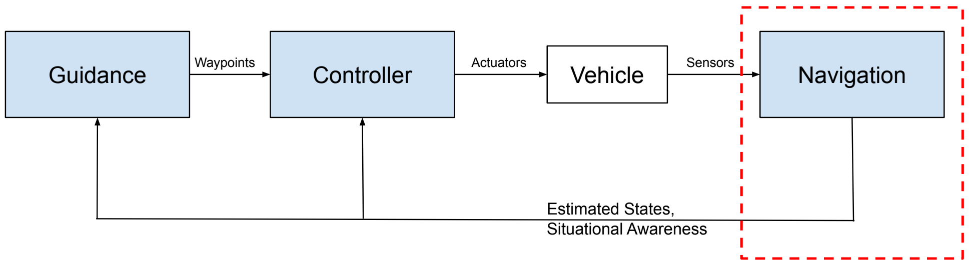


Figure 3.1: Guidance, Navigation and Control: The navigation system is essential for any GNC system. It estimates the states and provides situational awareness.
Inspired by: [19]

At the top level of the system, one finds the guidance block, which provides high-level commands to the controller, such as waypoints for the vehicle to follow. The controller block is responsible for acting on commands, which is done by providing commands to the different actuators on the vehicle. There may be many levels of controllers on a single vehicle, spanning from a path-following controller for heading to a low-level PID controller on the motor.

As mentioned, one cannot know exactly what the vehicle states are at a given time while it is operating. Additionally, sensor information may be unreliable over time and under different conditions. Therefore it is necessary to include a navigation system to estimate the states of the system optimally. Autonomy also involves situational awareness, which is typically also a responsibility of the navigation subsystem. These two responsibilities typically include estimation optimization and fusion of information from many different sensors. For this thesis, the main focus is on the navigation subsystem, marked with a red box in Figure 3.1.

3.2 Sensors

Typical for modern navigation systems is the use of satellites, inertial sensors, and *exteroceptive* sensors. Exteroceptive sensors are sensors that provide information about the environment outside the system itself. Typical exteroceptive sensors include LiDARs and cameras.

3.2.1 Inertial Measurement Unit

Inertial Measurement Unit (IMU) is perhaps the most commonly used sensor in Inertial Navigation System (INS). An IMU consists of at least an accelerometer and a gyroscope. Depending on the case, it may or may not contain other sensors, such as magnetometers.

The accelerometer measure linear accelerations along the axes of the IMU, and the gyroscope measures the angular rates. Compasses measure the heading. Combining these measurements provides the foundation to estimate the current position and orientation of the vehicle in the world. IMUs carry much information at a high frequency. However, IMUs tend to drift over time and have variable performance under different conditions. Therefore it is undesirable to use IMUs on their own for long-time operation, which is why it is typically paired with satellite systems or other sensors for correction.

3.2.2 Global Navigation Satellite System

Global Navigation Satellite System (GNSS) is a common term for satellite positioning systems all around the world, such as GPS in the USA, Galileo in Europe, and other satellite systems.

The GNSS calculates the position of a receiver by triangulating the distance measure from three different satellites simultaneously. The distance is calculated by measuring the time it takes to transfer a radio signal from the satellite to the receiver on Earth. An atomic clock then calculates the transfer time inside of the satellite. Using these clocks gives a highly accurate measure of position, which is why the GNSS measurements are often used as bona-fide measurements, or in other words, treated as ground truth.

3.2.3 Light Detection and Ranging

Light Detection and Ranging (LiDAR) sensors are Time-of-Flight (ToF) based sensors. They measure the time it takes for an emitted light to reflect from an object and return to a detector in the sensor. By assuming a constant light speed in the medium it gives an estimate of the distance to the given object.

Scanning LiDARs are typically created by employing many such lasers and keeping track of the azimuth angle that they were emitted at. This allows for creating 3D depth images, also known as point clouds.

3.3 Inertial Navigation Systems

An Inertial Navigation System (INS) typically consists of an IMU and a GNSS receiver. The goal of an INS is to estimate the full 3D position of the vehicle, as well as its attitude. This is popularly termed the *pose* of the vehicle. There are well-established solutions to this problem, such as the Kalman Filter (KF) and its predecessors such as the Extended Kalman Filter (EKF) and the more modern Error-State Kalman Filter (ESKF). The general idea is to estimate the pose by integrating the acceleration and angular velocity measured by the IMU. However, the IMU tends to drift and therefore needs to be aided by the GNSS in order to maintain globally consistent results. The problem of *dead-reckoning* arises when there are no GNSS measurements available over a long period of time, and the estimation runs solely on measurements from the IMU.

3.4 Statistics

Statistics describes ways to model *random variables*. This typically involves calculating and estimating the first- and second-order moments, namely the *mean* and *covariance* of the system. For an INS built using the KF, the goal is to estimate the likelihood distribution of the state vector after a measurement has arrived, also referred to as the *posterior* distribution of the system.

3.4.1 Gaussian Distribution

The Gaussian distribution, popularly called the normal distribution, is one of the most widely used models for random variables. A Gaussian random variable in one dimension is distributed with a mean, μ, and variance σ^2. If the random variable is a vector, then μ becomes a vector, and σ^2 becomes a square covariance matrix typically denoted P containing the variance of each random variable with itself in addition to its covariance with the other variables in the vector. A randomly distributed variable is typically denoted

$$\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, P) \tag{3.1}$$

Figure 3.2 shows how the probability density function (pdf) of the distribution looks in one dimension. The different x-σ intervals are sketched in the figure. The total probability that the value is contained within each x-σ perturbation away from the mean, μ, can then be found by accumulating the probabilities that the value is inside each interval on both sides.

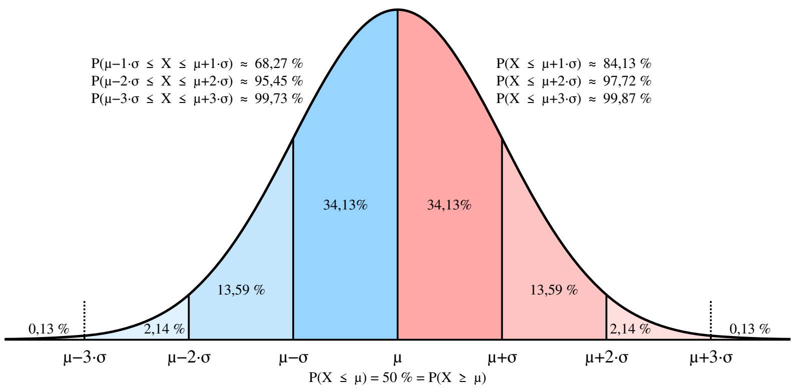


Figure 3.2: Gaussian Distribution: The figure shows the curve of a normal distribution in one dimension with x-σ intervals. The percentages show the likelihood of perturbations from the mean, μ, lying within the different σ-intervals.

Source: [20]

In the navigation problem the state vector typically represents the mean, and the measurements are assumed normally distributed around the 'true' state vector. There are many reasons why this is convenient. Among other things, it makes all the calculations much easier since any linear combination of normally distributed values remains normally distributed. Furthermore, due to the central limit theorem, physical quantities such as measurement errors tend to converge towards the normal distribution as the number of samples increases.

3.4.2 χ^2 Distribution

The χ^2 distribution is a commonly used distribution for hypothesis testing. The distribution is a special case of the gamma distribution, and it occurs when performing a summation of squared standard normally distributed values. The pdf of the distribution with different values for the k DoFs is shown in Figure 3.3.

Because the summation of squared normally standard normally distributed values yields a χ^2-distributed value distribution, the χ^2 distribution is often used in navigation to validate the consistency of estimations.

3.4.3 Maximum A Posteriori Estimation

Maximum A Posteriori (MAP) estimation concerns estimating the value of a random variable that maximizes the posterior of the variable. The MAP estimator is given by the equation

$$\hat{x} = \underset{x}{\operatorname{argmax}}\, p_{x|z}(x|z) \tag{3.2}$$

where x is the random variable to be estimated and z is the observation. $p(\cdot)$ denotes the pdf and the subscript denotes what random variable the distribution

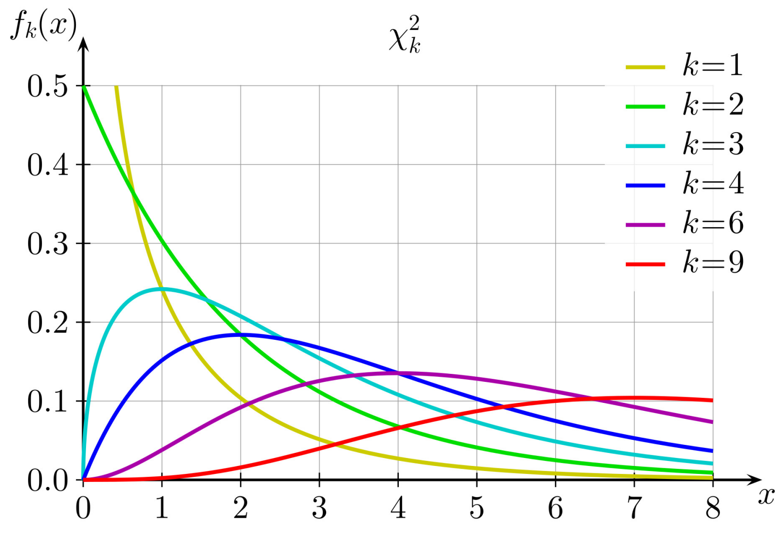


Figure 3.3: χ^2 Distribution: The figure shows the pdfs of a χ^2 distribution in one dimension. Each curve is plotted with a different amount of DoFs, denoted by k in the legend in the upper right.

Source: [20]

belongs to. Following Bayes' rule, we can write

$$\begin{aligned}\hat{x} &= \operatorname*{argmax}_x p_{x|z}(x|z) \\ &= \operatorname*{argmax}_x \frac{p_{z|x}(z|x)p_x(x)}{p_z(z)} \\ &= \operatorname*{argmax}_x p_{z|x}(z|x)p_x(x)\end{aligned} \tag{3.3}$$

where the final step could be made because the denominator is independent of the variable we wish to estimate. Therefore it does not matter for the maximization.

3.5 Nonlinear Solvers

The navigation problem is generally nonlinear, raising the need for nonlinear solvers. Most nonlinear solvers are typically iterative and update a linearized solution until some convergence criteria are met. The main difference between the nonlinear solvers described in this section is the way they choose the step in the function-minimizing direction.

3.5.1 Gradient descent

The gradient descent method is a first-order iterative optimization algorithm, meaning that it uses the gradient of the objective function. The minimization is done by moving in the opposite direction of the gradient, with the length defined by some learning rate. The algorithm is shown in Algorithm 1.

Algorithm 1: Gradient descent method

```
In: Objective function $f(x)$, learning rate $\alpha$
Out: Minimization parameters $x$
while not converged do
    $x \leftarrow x - \alpha \nabla f(x)$
end
```

3.5.2 Gauss-Newton Method

The Gauss-Newton (GN) method is a modification of Newton's method, which uses the curvature of the problem instead of the gradient. The curvature is given by the Hessian of the objective function. However, the calculation of the Hessian is not possible in many applications and requires to do inversions of large matrices, which is undesirable. For a linear least-squares problem on the form

$$f(x) = ||Ax - b||^2 \tag{3.4}$$

The Gauss-Newton method would approximate the Hessian matrix by

$$\nabla^2 f(x) \approx A^T A \tag{3.5}$$

This leaves us with the algorithm as shown in Algorithm 2.

Algorithm 2: Gauss-Newton Method

```
In: Objective function $f(x)$, initial estimate $x^0$
Out: Minimization parameters $x$
for $t \leftarrow 0$ to $t_{max}$ do
    $A, b \leftarrow$ Linearize $f(x)$ at current estimate $x^t$
    $\tau \leftarrow$ Solve the linearized problem $A^T A \tau = A^T b$
    $x^{t+1} \leftarrow x^t + \tau$
    if $f(x^{t+1})$ is very small or $x^{t+1} \approx x^t$ then
        $x = x^{t+1}$
        return
    end
end
```

3.5.3 Levenberg-Marquardt Method

Levenberg-Marquardt (LM) is a trust-region method that combines the gradient descent method with the Gauss-Newton method to minimize an objective function. Much like the Gauss-Newton method, Levenberg-Marquardt repeatedly linearizes the objective function and approximates the Hessian using the system matrix, A. However, unlike Gauss-Newton, Levenberg-Marquardt cannot achieve an objective function value higher than the previous iteration. This is done by smoothing the approximation of the Hessian from the Gauss-Newton. The approach given here is similar to Tikhonov regularization. The smoothing parameter, λ, is updated based on whether or not the iteration produced a lower objective function value. This makes the Levenberg-Marquardt method more stable than the Gauss-Newton method. For the linear least-squares problem given in (3.4) the smoothed Hessian matrix would be approximated by

$$\nabla^2 f(x) \approx A^T A + \lambda\,\mathrm{diag}(A^T A) \tag{3.6}$$

Notice that that if λ is small, the equation gives the GN method. On the other hand, if λ is large, the smoothing term becomes dominant.

The algorithm is given in Algorithm 3.

Algorithm 3: Levenberg-Marquardt Method

```
In: Objective function $f(x)$, initial estimate $x^0$
Out: Minimization parameters $x$

$\lambda \leftarrow 10^{-4}$
for $t \leftarrow 0$ to $t_{max}$ do
    $A, b \leftarrow$ Linearize $f(x)$ at current estimate $x^t$
    $\tau \leftarrow$ Solve the linearized problem $(A^T A + \lambda\,\mathrm{diag}(A^T A))\tau = A^T b$
    if $f(x^t + \tau) < f(x^t)$ then
        Accept update, increase trust region
        $x^{t+1} \leftarrow x^t + \tau$
        $\lambda \leftarrow \lambda/10$
    else
        Reject update, reduce trust region
        $x^{t+1} \leftarrow x^t$
        $\lambda \leftarrow \lambda \times 10$
    end
    if $f(x^{t+1})$ is very small or $x^{t+1} \approx x^t$ then
        $x = x^{t+1}$
        return
    end
end
```

3.6 Lie Theory

The state vector in any navigation problems includes position and attitude, which is typically parametrized into a vector of six quantities often referred to as the *pose* of the vehicle. Poses are typically defined as $[x, y, z, \phi, \theta, \psi]^T$ where x, y, z describe the position of the vehicle and ϕ, θ, ψ describe the orientation of the vehicle relative to some global coordinate system. However, the relationship between these state variables is nonlinear, making them non-additive. As a result of this, it is not straightforward to define perturbations and uncertainty using regular algebra.

This section aims to describing the Lie theory necessary to understand the theory behind the thesis. In particular, Lie theory is helpful to provide a framework for describing uncertainty and perturbations on strict vector spaces, referred to as *manifolds*. The theory in this section is paraphrased from [1].

3.6.1 Lie Groups

A Lie group is defined as smooth and differentiable manifold consisting of a set $\mathcal{G}$ and a composition operation, $\circ$ that satisfies the group axioms

$$\begin{aligned} \text{Closure under } \circ : & \; \mathcal{X} \circ \mathcal{Y} \in \mathcal{G} \\ \text{Identity } \mathcal{E} : & \; \mathcal{E} \circ \mathcal{X} = \mathcal{X} \circ \mathcal{E} = \mathcal{X} \\ \text{Inverse } \mathcal{X}^{-1} : & \; \mathcal{X}^{-1} \circ \mathcal{X} = \mathcal{X} \circ \mathcal{X}^{-1} = \mathcal{E} \\ \text{Associativity} : & \; (\mathcal{X} \circ \mathcal{Y}) \circ \mathcal{Z} = \mathcal{X} \circ (\mathcal{Y} \circ \mathcal{Z}) \end{aligned} \tag{3.7}$$

for elements $\mathcal{X}, \mathcal{Y}, \mathcal{Z} \in \mathcal{G}$. Lie groups can also transform elements of other sets through what is called a group action. A group action must fullfill the axioms

$$\begin{aligned} \text{Identity} : & \; \mathcal{E} \cdot v = v \\ \text{Compatibility} : & \; (\mathcal{X} \circ \mathcal{Y}) \cdot v = \mathcal{X} \circ (\mathcal{Y} \cdot v) \end{aligned} \tag{3.8}$$

The Lie groups to be considered in this thesis are the Special Orthogonal Group (SO(3)) and the Special Euclidean Group (SE(3)).

Special Orthogonal Group

The Special Orthogonal Group (SO(3)) describes the set of 3D rotation matrices. More formally, it is defined by

$$SO(3) := \{\mathbf{R} \in \mathbb{R}^{3x3} | \mathbf{R}^T \mathbf{R} = \mathbf{I}, \; \det(\mathbf{R}) = 1\} \tag{3.9}$$

The group action of the SO(3) on vectors is given by

$$\mathbf{R} \cdot \mathbf{v} = \mathbf{R}\mathbf{v} \tag{3.10}$$

Special Euclidean Group

The Special Euclidean Group (SE(3)) describes rigid motion in three dimensions and is more formally defined by

$$SE(3) := \{\mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \in \mathbb{R}^{4\times4} | \mathbf{R} \in SO(3), \; \mathbf{t} \in \mathbb{R}^3\} \tag{3.11}$$

The group action of the SE(3) on vectors is given by

$$\mathbf{T} \cdot \mathbf{v} = \mathbf{R}\mathbf{v} + \mathbf{t} \tag{3.12}$$

3.6.2 Lie Algebra

Lie algebra gives us the ability to move between the manifold $\mathcal{M}$, the tangent manifold $\mathcal{TM}$, denoted $\mathfrak{m}$, and the vector space $\mathcal{R}^m$. The conversion between vectors in $\mathcal{R}^m$ and Lie algebra elements in $\mathcal{TM}$ is done through the operators $(\cdot)^\vee$ and $(\cdot)^\wedge$. They are defined by

$$\text{Hat: } (\cdot)^\wedge : \mathcal{R}^m \to \mathfrak{m}; \quad \boldsymbol{\tau}^\wedge = \sum_{i=1}^{m} \tau_i \mathbf{E}_i \tag{3.13}$$

$$\text{Vee: } (\cdot)^\vee : \mathfrak{m} \to \mathcal{R}^m; \quad (\boldsymbol{\tau}^\wedge)^\vee = \sum_{i=1}^{m} \tau_i \mathbf{e}_i \tag{3.14}$$

where $\mathbf{E}_i$ describes the generators of $\mathfrak{m}$ and $\mathbf{e}_i$ describes the generators of $\mathcal{R}^m$, also known as the basis vectors. Further, the conversion between the Lie algebra elements and the manifold is given by the exponential and logarithmic maps. Additionally, the Exp operator and the Log operator have been defined as composite operations converting between the vector space, $\mathcal{R}^m$, and the manifold $\mathcal{M}$. These operations are given by

$$\exp : \mathfrak{m} \to \mathcal{M}; \quad \mathcal{X} = \exp(\boldsymbol{\tau}^\wedge) \tag{3.15}$$

$$\log : \mathcal{M} \to \mathfrak{m}; \quad \boldsymbol{\tau}^\wedge = \log(\mathcal{X}) \tag{3.16}$$

$$\text{Exp} : \mathcal{R}^m \to \mathcal{M}; \quad \mathcal{X} = \text{Exp}(\boldsymbol{\tau}) := \exp(\boldsymbol{\tau}^\wedge) \tag{3.17}$$

$$\text{Log} : \mathcal{M} \to \mathcal{R}^m; \quad \boldsymbol{\tau} = \text{Log}(\mathcal{X}) := \log(\mathcal{X})^\vee \tag{3.18}$$

In turn, these operators allow us to express perturbations on the manifold in terms of linear algebra, which is crucial to optimally fuse information from different sources, such as priors, sensors, etc.

3.7 Odometry

Odometry describes the use of sensors to estimate the position and orientation of the vehicle over time and is essential for many robotics applications. Odometry estimation thus gives a relative estimate of how much a vehicle has moved between two locations in a particular time window. Earlier forms of odometry were based on using wheel encoders, but in modern systems it is more common to use visual sensors such as cameras and LiDARs.

LiDAR odometry aims at estimating the odometry of a vehicle using LiDARs to calculate the motion between point clouds. Typically it is divided into two categories;

feature-based point cloud registration and *direct* point cloud registration. Direct registration methods typically use algorithms such as Iterative Closest Point (ICP) or Normal Distribution Transformation (NDT) directly on the full point cloud. Feature-based registration, on the other hand, extracts features from the full point cloud to decrease the problem's dimensionality. The feature-based formulation has shown to provide more accurate and faster solutions than the direct formulation [15].

3.8 Point Cloud Alignment

Point cloud registration methods typically involve rigid alignment of an incoming point cloud to one or several historic point clouds. Rigid alignment means that the scale of the clouds is assumed constant and equal. The alignment problem for rigid bodies consists of both translation and rotation, making it nonlinear. Therefore it is important to have effective iterative solvers. Modern methods include the aforementioned ICP and NDT methods. This section will give a brief explanation of the inner workings of the ICP method.

Classic ICP aims at finding the transformation matrix, T, that minimizes the alignment error. Let $\mathcal{M}$ be the set of point correspondences. Then the problem can be described mathematically as

$$\mathbf{T}^* = \underset{\mathbf{T}}{\operatorname{argmin}} \sum_{(p,q)\in\mathcal{M}} ||\mathbf{T}p - q||^2_{\Sigma_i} \tag{3.19}$$

Where the $||\cdot||_{\Sigma_i}$ denotes the Mahalanobis norm.

The problem is solved iteratively, yielding in the algorithm

Algorithm 4: ICP registration

```
In: Initial estimate $\mathbf{T}^0$, source point cloud $P$, target point cloud $Q$
Out: Optimal transformation $\mathbf{T}^*$

for $k \leftarrow 0$ to $k_{max}$ do
    $\mathcal{M} \leftarrow$ Calculate point correspondences between $P$ and $Q$
    $\mathbf{T}^* \leftarrow$ Minimize (3.19) using for example LM
    Apply transformation $\mathbf{T}^*$ to $P$
    if converged then
        return
    end
end
```

However, it is not straightforward to calculate (3.19) without using the Lie framework. To be expressed in Lie algebra, the system must be linearized around the current estimate, which gives

$$\xi^* = \underset{\xi}{\operatorname{argmin}} \sum_{(p,q)\in\mathcal{M}} ||\hat{\mathbf{T}}p + \mathbf{J}_{\hat{\mathbf{T}}}^{\mathbf{T}p}\xi - q||^2_{\Sigma_i} \tag{3.20}$$

where ξ denotes the update step of $\mathbf{T}$ expressed in $\mathfrak{m}$ and $\mathbf{J}_{\hat{\mathbf{T}}}^{\mathbf{T}p}$ is the Jacobian of the action of $\mathbf{T}$ on p with respect to the estimate $\hat{\mathbf{T}}$.

Exploiting this allows us to express the problem as a linear least squares, which can be expressed as

$$\begin{aligned}\xi^* &= \underset{\xi}{\operatorname{argmin}} \sum_i ||\mathbf{A}_i\xi - \mathbf{b}_i||^2 \\ &= \underset{\xi}{\operatorname{argmin}} ||\mathbf{A}\xi - \mathbf{b}||^2\end{aligned} \tag{3.21}$$

where

$$\mathbf{A}_i = \Sigma_i^{-1/2}\mathbf{J}_{\hat{\mathbf{T}}}^{\mathbf{T}p_i} \tag{3.22}$$

$$\mathbf{b}_i = \Sigma_i^{-1/2}(q_i - \hat{\mathbf{T}}p_i) \tag{3.23}$$

where the subscript i is added to clarify $\mathbf{A}$ contains a row for each correspondence, and $\Sigma_i^{-1/2}$ is given by

$$\Sigma_i^{-1/2} = \mathbf{J}_{\hat{\mathbf{T}}}^{\mathbf{T}p}\Sigma_\sigma\mathbf{J}_{\hat{\mathbf{T}}}^{\mathbf{T}p\,T} \tag{3.24}$$

and Σ_σ is a covariance matrix.

3.9 Feature Extraction

Feature-based point cloud registration requires robust, repeatable point detection and description. These points are often referred to as *keypoints*. In order to encapsulate the information of a point as uniquely as possible, a *descriptor* is often used. The process of extracting keypoints and descriptors incur a fixed-time cost on each incoming point cloud. However, the goal is to decrease the number of points needed, and the time it takes to match them in the registration process. It has been shown that using features and keypoints provides a more stable and faster solution to the registration process compared to solving the full cloud registration using classical methods such as ICP or NDT.

3.9.1 Intrinsic-Shape Signatures

Intrinsic-Shape Signatures (ISS)[21] keypoints are based on a view-invariant 3D shape descriptor. ISS keypoints are calculated through a *saliency measure*, given by the magnitude of the smallest eigenvalue of a point. According to [22], ISS has proven to be a fast and reliable detector for recognition purposes. For more details, the reader is referred to [1].

3.9.2 Fast Point Feature Histogram

Fast Point Feature Histogram (FPFH)[23] is a modification of Point Feature Histogram (PFH) for problems requiring lower computational complexity. FPFH is a pose-invariant local feature descriptor, described through an array consisting of 33 bytes for each point. The goal of the descriptor is to capture the description of the surface around each keypoint as uniquely as possible for robust matching. Its low dimensionality and complexity make it especially suitable for time-critical applications such as navigation. For more details, the reader is referred to [1].

3.10 Factor Graphs

The classical way of representing navigational smoothing problems is through matrices and matrix operations. However, newer formulations using graphical models such as factor graphs have gained popularity due to their intuitive representation. Furthermore, the factor graph can be shown to provide the same performance as sparse matrix operations while still providing the insight to effectively manipulate the nodes of the graph optimally when new variables arrive or relinearization is needed. An example factor graph for a navigation system is shown in Figure 3.4.

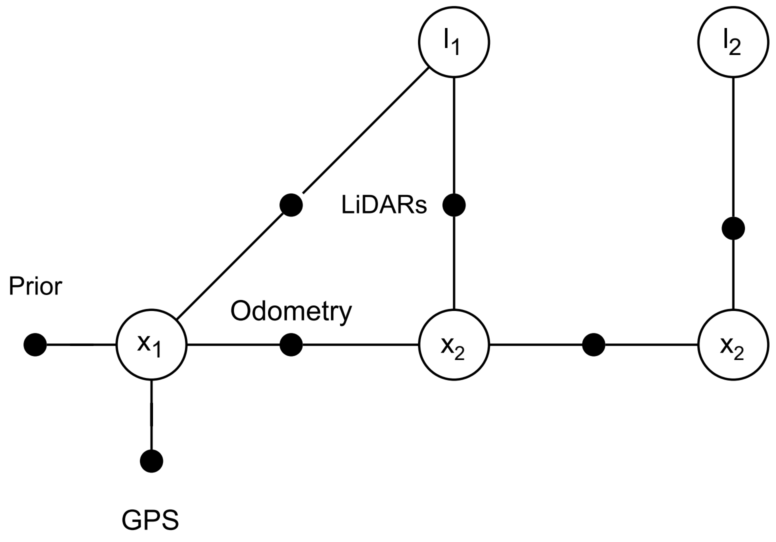


Figure 3.4: Factor Graphs: The figure shows a factor graph displaying an example navigation problem. The factors between the xs symbolize odometry measurements. The landmarks are measured using a LiDAR. The first x also has a prior and a GPS factor connected to it.

Source: [1]

A factor graph is a bipartite graph $\mathcal{G} = (\mathcal{F}, \mathcal{X}, \mathcal{E})$ with two different types of nodes:

factor nodes $f_i \in \mathcal{F}$ and *variable nodes* $x_j \in \mathcal{X}$. Edges $e_{ij} \in \mathcal{E}$ can exist only between factor nodes and variable nodes and are present if and only if the factor f_i involves a variable x_j. Thus we can express each factor f_i as a function of the variables x_j and a parametrization of the graph can be described by

$$f(\mathcal{X}) = \prod_i f_i(\mathcal{X}_i) \tag{3.25}$$

Each factor f_i can be expressed as

$$f_i(\mathcal{X}_i) = d[h_i(\mathcal{X}_i) - z_i] \tag{3.26}$$

where d denotes a certain cost function, h_i denotes a prediction and z_i denotes the real measurement. This framework can incorporate measurement models as well as process models in a similar manner. Ultimately the factor graph framework boils down to an optimization problem over all the factors, resulting in the solution [24]

$$\hat{\mathcal{X}} = \underset{\mathcal{X}}{\operatorname{argmin}} f(\mathcal{X}) \tag{3.27}$$

which under the right assumptions can be shown to be the MAP estimate. For more details, the reader is referred to [1].

3.11 Simultaneous Localization and Mapping

Simultaneous Localization and Mapping (SLAM) is a problem handling the joint estimation of both pose and landmarks. More accurately, it is the problem of mapping an unknown environment while keeping track of a vehicle's position within it. The first formulations of the SLAM problem were expressed as extensions of the EKF framework. However, this involved keeping track of and inverting large matrices that could grow infinitely large, making it ineffective for most real-world applications. Later, FastSLAM proposed a Rao-Blackwellized particle filter solution to the problem. This solution was much more scalable compared to the EKF approach, as it started exploiting the structural independence of the SLAM problem. The details of this can be found in [1].

3.12 Graph-Based SLAM

Inspired by the independence exploitation introduced in FastSLAM, modern SLAM systems, deemed graph-based SLAM systems, represent the SLAM problem using a factor graph. However, the factor graphs require all factors to be in the form of residuals. Therefore it is common to divide the system into two parts; the front-end and the back-end, as depicted in figure 3.5.

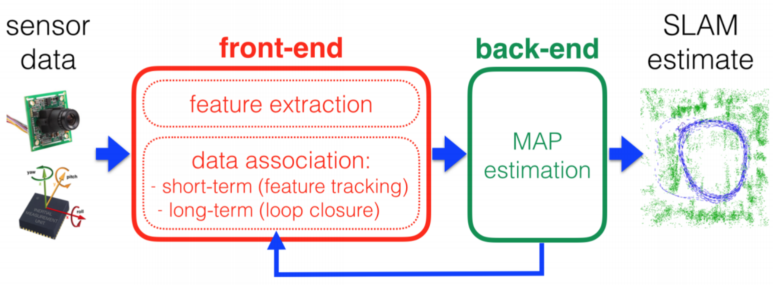


Figure 3.5: Modern SLAM Architecture: The figure shows how the different responsibilities of a modern SLAM system are split between a front-end and a back-end.

Source: [25]

Front-end

The front-end of the SLAM system is responsible for performing feature extraction and data association. Data association includes both tracking of features between frames in the short term and detecting loop closures. After the feature extraction and data association are performed, the data is delivered to the back-end as residuals inserted into the factor graph.

Back-end

The back-end of the SLAM system is responsible for performing the estimation itself. In the case of graph-based SLAM, this is implemented through a factor graph. As mentioned, the problem typically boils down to MAP estimation and involves sparse matrix calculations. Furthermore, modern back-end optimizations also handle relinearization and reordering of variables if that is needed.

3.12.1 Loop Closure

The goal of any SLAM system is to estimate the position of a vehicle in an unknown environment. However, if an estimate is given on some unknown scale or represented in an unknown coordinate system, it is not particularly useful for missions that span large environments. Loop closures were introduced to cope with drift in estimates and maintain global consistency. A loop closure means that a SLAM system recognizes a scene it has visited before. Effectively, this allows correcting for drift in estimates since it last visited this particular scene. This correction can be compared to what is done in inertial navigation when a GNSS measurement is used to correct for drift in IMU-estimations.

There are many techniques to detecting loop closures. For 3D LiDAR SLAM systems, the techniques can be split into two different categories; *segmentation*-based methods and alignment-based methods. The segmentation-based methods scan the point cloud for objects, extract them, and add them to a Bag-of-Words (BoW) model for later use. The alignment-based methods typically use ICP to fit the current point cloud to a historic point cloud. The latter method is used in this thesis.

3.13 Metrics

3.13.1 Root Mean Square Error

The Root Mean Square Error is a popular metric for describing the error between an estimated value and a truth value. The reason for its popularity is its close relation to statistical moments, which is essential in estimation theory. In fact, the RMSE metric describes the sample standard deviation of the prediction errors. The RMSE metric is defined by the equation

$$rmse = \sqrt{\frac{1}{N}\sum e^2} \tag{3.28}$$

where e is the estimation error and N is the sample size.

3.13.2 Normalized Estimation Error Squared

Normalized Estimation Error Squared (NEES) is a popular metric for analyzing consistency, most often used in combination with tuning the KF. The NEES metric gives an idea of whether or not the covariance justifies the estimates of the system. The details are taken from [6, p. 238-245]. The metric is defined by the equation

$$\epsilon_k = (\hat{x}_k - x_k)^T P_k^{-1} (\hat{x}_k - x_k) \tag{3.29}$$

where $\hat{x}_k$ denotes the estimated state, x_k denotes the true state, and P_k is the covariance at time step k. If the states are modeled correctly, then ϵ_k is a χ^2 distributed random variable with Degrees of Freedom (DoF) equal to the dimension of the state. As a result of this, the NEES can be compared with a 95% confidence interval. The higher the number of NEESes lie within this interval, the more consistent the state model is with the true state model. A NEES higher than the confidence bounds indicates an overconfident system, whereas a NEES lower than the confidence bounds indicates an underconfident system. The lower and upper bounds can be defined according to

$$lower = \texttt{chi2inv}(0.025, d) \tag{3.30}$$

$$upper = \texttt{chi2inv}(0.975, d) \tag{3.31}$$

where chi2inv describes the inverse cumulative distribution function of the χ^2 distribution and d is the dimension of the state vector.

The NEES can be used to tune the process covariances of the system. An underconfident system suggests that the system constantly overestimates the covariances. This is generally easy to adjust for by decreasing the process noise covariances. On the other hand, an overconfident system suggests that the system underestimates the covariance, which is clearly undesirable. This suggests increasing the process noise covariances of the system. However, this may make the estimations worse, yielding an increased NEES. This fact makes the problem of overconfidence generally more challenging to adjust for than underconfidence.

Another way to analyze the NEES is to investigate how it develops for a reduced state space. This can give insight into how the different process noise covariances affect the different parts of the system. However, this disregards the covariances between the removed states and may lead to unforeseen developments when going back to analyzing the entire state space. Often it is beneficial to study the development of the NEES for a reduced state space, and tune it for the most 'important' states.

The NEES can also be analyzed in terms of its average. The Average Normalized Estimation Error Squared (ANEES) can be used to measure the consistency of the system on the entirety of the samples. When studying the ANEES, the limits defined in (3.32) are changed. The lower and upper bounds are given by

$$lower = \texttt{chi2inv}(0.025, Nd)/N \tag{3.32}$$

$$upper = \texttt{chi2inv}(0.975, Nd)/N \tag{3.33}$$

where N is the number of samples, d is the dimensionality of the state vector and chi2inv is the inverse cumulative distribution function of the χ^2 distribution.

In this thesis, the NEES metric will be used to analyze the performance of a graph-based SLAM system.

Chapter 4

Proposed Approach

This chapter presents the proposed approach for simulating the data, as well as the proposed system for solving the tunnel navigation problem. Section 4.1 goes through the relevant software and how it is used. Section 4.2 explains how the simulator is extended from the pre-project [1] to include IMU and GNSS data. Finally, Section 4.3 describes the front-end and the back-end of the LiDAR-Inertial SLAM system.

4.1 Software

This section goes through the essential software and dependencies of the system, and how they are used. The code itself can, at the moment of writing this thesis, be found fully available at [26].

4.1.1 MATLAB and Unreal Engine

MATLAB is a well-known proprietary programming platform popularly used for robotics purposes. It comes equipped with many toolboxes for many different applications. Among those toolboxes are `Automated Driving Toolbox` and `Automated Driving Toolbox Interface for Unreal Engine 4 Projects`, which allows MATLAB, more specifically, Simulink, places actors inside a simulation environment created in Unreal Engine 4[27]. Unreal Engine is an open-source game engine developed by Epic Games written in C++. It comes equipped with high-quality graphics and tools for quickly creating simulated environments.

The `Automated Driving Toolbox` and `Automated Driving Toolbox Interface for Unreal Engine 4 Projects` toolboxes allow placing a vehicle with several exteroceptive and interoceptive sensors equipped inside the simulated environment created in Unreal Engine. The data can then be extracted in real-time. Additionally, MATLAB has extra functionality for simulating vehicle dynamics, performing path planning, and much more, which is used in this thesis to create the simulated data to evaluate the proposed system. The final Simulink diagram is shown in Appendix C.

4.1.2 Robotics Operating System

Robotics Operating System (ROS)[28] is a development framework for creating scalable, standardized robotics software across many different development platforms. This scalability is achieved by its network-like structure, where it is possible to create 'nodes' that run independently of each other, and communication between them is achieved by using standardized messages. Additionally, ROS comes equipped with a lot of tools and functionality for visualizing large streams of data, as well as manipulating them to the desired format. Due to its considerable popularity, it also has a large user community and is typically compatible with most existing professional robotics systems and tools.

In this thesis, ROS handles all communication between the different modules of the system. Because it is built on ROS it is easy to define a modular structure of the system, where each part has its own responsibilities. This can be done because ROS implements a vast variety of message types that are typically used in robotics applications, as for example `PointCloud2`, `Odometry`, and `Pose` messages with time stamps. ROS is also used to visualize the performance of the system. This visualization is done via the `rviz` package, which provides tools for visualizing navigation problems in three dimensions with uncertainty estimates. Figure 4.1 shows a snippet of this visualization software.

The entire ROS communication setup can be seen in Appendix B.

4.1.3 Georgia Tech Smoothing and Mapping

Georgia Tech Smoothing and Mapping (GTSAM)[29] is an optimization library for navigation purposes written in C++. The optimization problem is represented using a factor graph. Among other things, GTSAM 4.0 implements the state-of-the-art incremental inference algorithm iSAM2, which utilizes the Bayes-tree structure to minimize the extent of changes coming into the factor graph. The main advantage of this is that it maintains real-time performance even when the graph grows very large. GTSAM also comes equipped with tools for preintegrating measurements from IMU sensors and conversion between the Lie framework and the normal matrix representations. All of these are features that are used in this thesis.

The GTSAM framework is used to perform preintegration of IMU measurements, as well as performing the optimization itself. Therefore, when compared to Figure 3.5, GTSAM performs actions for the back-end and the front-end. The preintegration is done in the front-end, while the iSAM2 framework is pure optimization

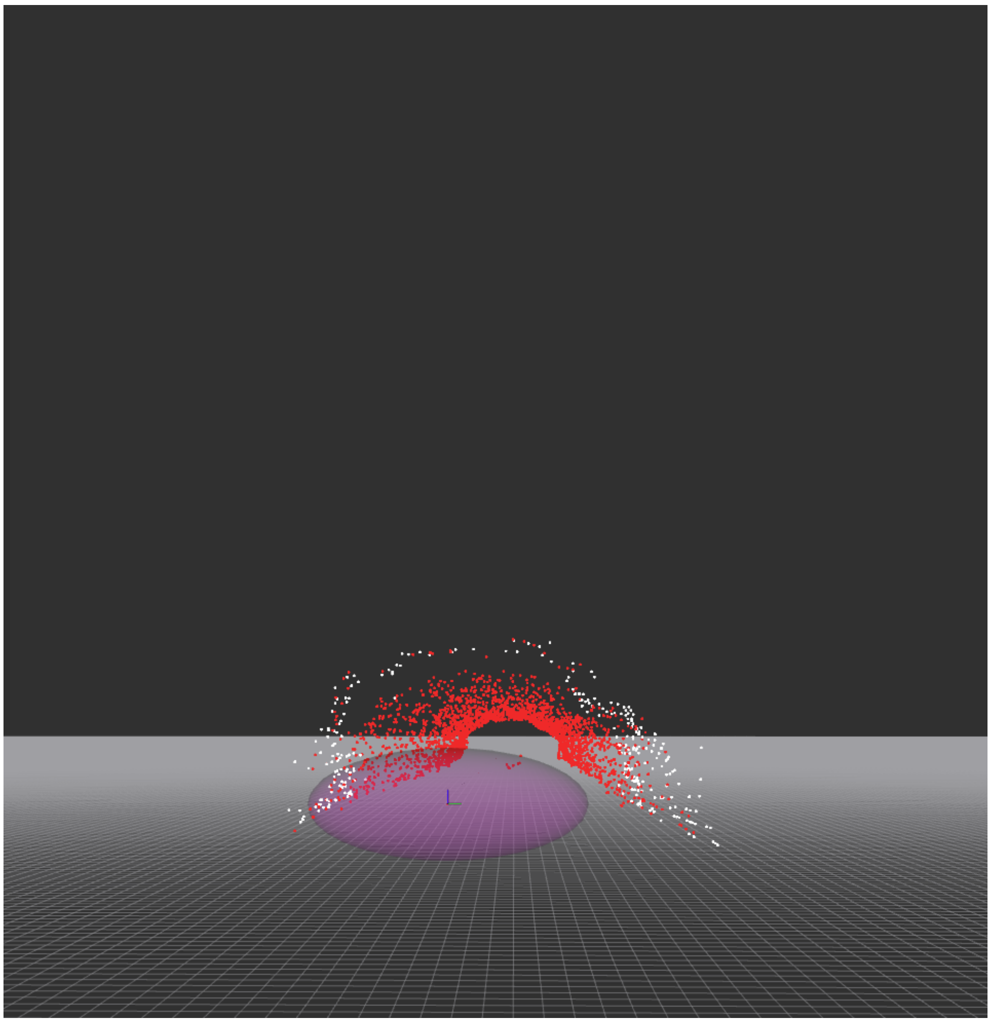


Figure 4.1: Rviz Visualization: The figure shows how the rviz module of ROS is used to visualize the performance of the system. The white dots are the currently viewed points, the red dots are the points contained in the map and the axes symbolize the sensor. The surrounding purple 'bubble' is the one-sigma ellipsoid in three dimensions.

and belongs in the back-end.

4.1.4 Point Cloud Library

Point Cloud Library (PCL)[30] is an open-source C++ library for manipulating point clouds. It comes with much inbuilt functionality, such as effective algorithms for feature extraction, feature matching, and match rejections for single clouds, in addition to calculating transformations based on matched points. This includes algorithms such as the ICP and its variants, the RANSAC algorithms in many different forms and calculation of point cloud normals, keypoints such as the ISS, and descriptors like the FPFH. These features and many others are utilized in this thesis.

PCL is very useful for manipulating point clouds. Comparing this to the architecture shown in Figure 3.5, PCL is responsible for most of the 'visual' part of the front-end. This includes feature tracking in the short term as well as long term.

4.2 Data Generation

In order to properly evaluate the proposed system, simulated data is generated. The data is generated through MATLAB, using Unreal Engine to simulate objects measured by the LiDAR sensor. The simulation environment is similar to the one used in [1]. However, it is extended to include an IMU sensor and GNSS measurements. The entire flow of the data generation scheme is shown in figure 4.2. The simulation environment created in [1] can be seen in Appendix D.

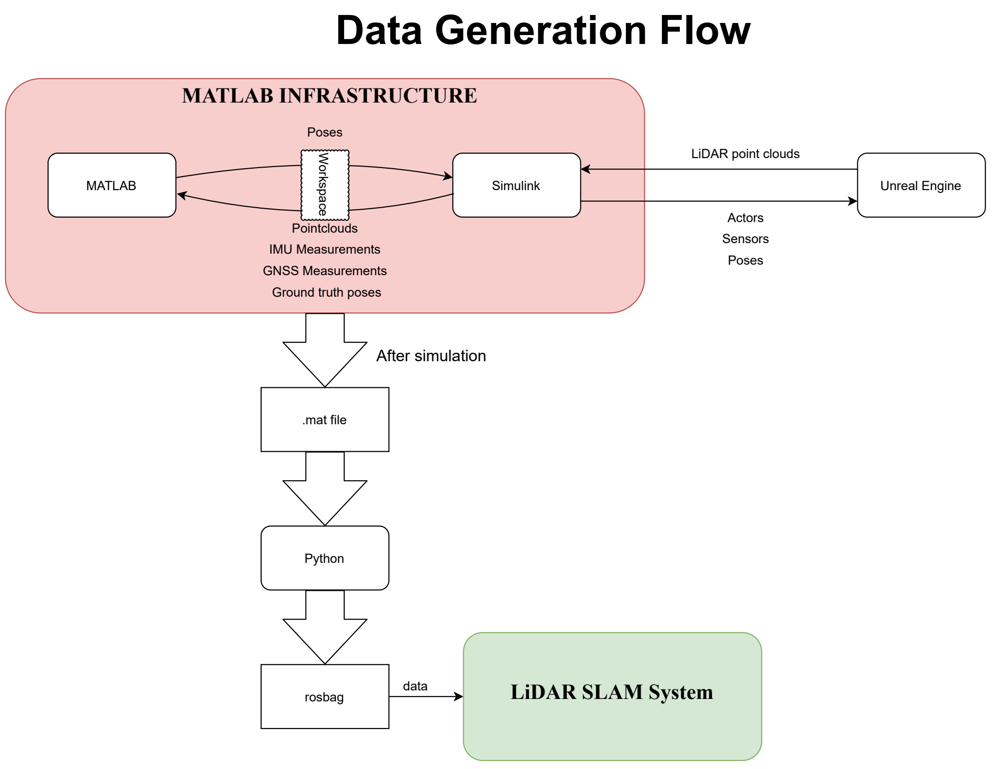


Figure 4.2: Data Generation: The figure shows how the data is created before being written to a format readable in ROS. The data is generated in MATLAB using a link between Simulink and Unreal Engine. Thereafter the produced .mat file is read in Python and written to a rosbag.

Inspired by: [1]

4.2.1 IMU

In order to include the IMU sensor properly into the system, a more realistic vehicle model was implemented. The vehicle model is implemented as a rigid body with three DoF following a kinematic bicycle model. The body accelerations and angular velocities are thereafter extracted from this model and sent to the IMU sensor. The IMU measurements arrive in the NED frame and are after that rotated to follow the same coordinate system as the LiDAR. Further, it is assumed in the simulations that the IMU is perfectly aligned with the body frame of the vehicle. The results of the IMU simulations are shown in figure 4.3.

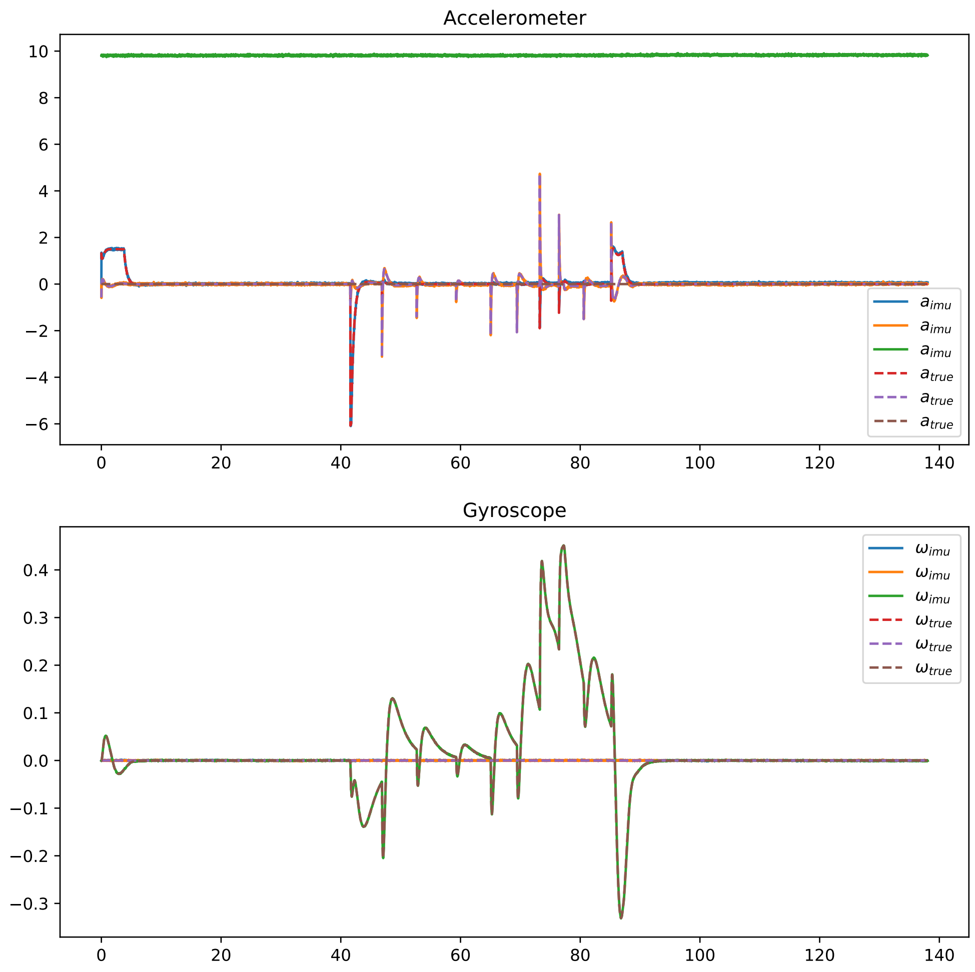


Figure 4.3: IMU Generation: The figure shows the results of the IMU measurement generation. The top figure shows the accelerometer measurements plotted against the real linear acceleration. The bottom figure shows the gyroscope measurements plotted against the true measurements.

4.2.2 GNSS

GNSS measurements are completely positional, and since the navigation module does not affect the simulation, it is trivial to incorporate them after the simulation is done. The measurements are simulated by assuming that they are normally distributed around the ground truth. This assumption lets us generate the GNSS measurements by perturbing the ground truth data by zero-mean noise. The noise is generated according to the parameters of Table 4.2. Note that the GNSS measurements are only added when the vehicle is outside the tunnel.

4.2.3 Sensor Parameters

The configured sensors are tuned according to Table 4.1, Table 4.2, and Table 4.3. In addition, the gyroscope has a cross-bias with the accelerometer of 1.78×10^{-4}. This means that high accelerations will affect the gyroscope.

Table 4.1: LiDAR Specifications

Parameter	Value
Detection range	50m
Range resolution	0.002m
Vertical FOV	40°
Vertical resolution	1.25°
Horizontal FOV	180°
Horizontal resolution	0.16°
Frequency	5Hz

Table 4.2: GNSS Specifications

Parameter	Value
σ_x^2	0.1m
σ_y^2	0.1m
σ_z^2	0.2m

Table 4.3: IMU Specifications

Parameter	Accelerometer	Gyroscope
Resolution	6.00×10^{-4}	1.33×10^{-4}
Constant Bias	0	0
Measurement Noise	1.54×10^{-3}	3.24×10^{-5}
Bias Driving Noise	3.92×10^{-4}	8.73×10^{-5}
Frequency	60Hz	60Hz

4.3 LiDAR-Inertial SLAM System

4.3.1 Front-end

The vehicle studied in this thesis consists of three sensors used for navigation; a LiDAR, an IMU, and a GNSS. The front-end is responsible for all pre-processing of sensory information before it is added to the back-end. For the proposed system, this includes the short-term feature tracking between LiDAR point clouds, performing the map alignment, and refining the map before adding points to the graph. Further, it also includes preintegrating IMU measurements, pre-processing GNSS measurements, keeping track of the timing between the different sensors, and detection of loop closures.

Short-term Feature Tracking

The vehicle is equipped with a 3D LiDAR sensor. All modern SLAM systems use some visual sensor to estimate odometry. This is because these types of visual odometry systems generally provide more robust and precise estimates compared to inertial navigation systems without having to buy expensive sensors [31]. Additionally, inertial sensors require correction to perform well, which is not available in an indoor scene.

The short-term feature tracking part of the front-end is responsible for solving the pairwise point cloud registration problem. The proposed method is based on using features and descriptors for matching and calculating the transformation between a pair of point clouds. Similar to the method explored in [1], it involves using the robust ISS keypoints and FPFH as descriptors. As opposed to the method in [1], the rejections are based on the partial overlap between the point clouds. The pipeline is shown in figure 4.4.

The first three steps in Figure 4.4 consist of removing NaN points, meaning points that never measured anything. Thereafter, noise is reduced by voxel grid filtering. This means downsampling all points within a small area to their centroid. After this, the normals of the cloud are computed.

The next step is to remove the ground plane from the cloud. This is done by fitting a plane to the points measured beneath the LiDAR corresponding to the height the sensor is mounted to. Any points that lie in that plane are consequently removed from the point cloud.

The final three steps of short-term feature tracking involve feature extraction and matching, including the calculation of the relative transformation itself. As in [1], the system uses ISS keypoints and FPFH descriptors to perform the matching. The matching is thereafter done using a KD-tree for fast computation before matches are rejected by using a partial overlap rejector. Finally, the transformation is calculated before publishing the transformation over ROS.

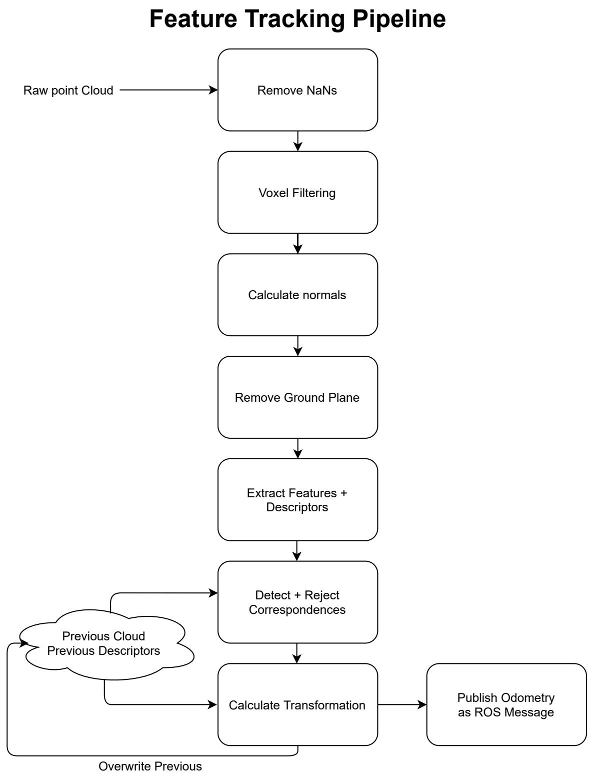


Figure 4.4: Short-term Feature Tracking Pipeline: The figure shows the pipeline of the short-term feature tracking module. The steps consist of filtering, segmentation, feature matching and transformation calculation.

Keyframe Selection

A SLAM system aims to maintain a globally consistent estimate of the vehicle pose and map at all times. This includes keeping a history of poses and points for an indefinite amount of time. In order to reduce the number of variables to maintain, a crucial part of any SLAM system is the selection of so-called keyframes. It is vital for any SLAM system that their keyframes are as descriptive as possible to avoid losing necessary information as time passes.

The proposed system fuses information from several sensors. Consequently, the keyframe selection should not be dependent on a single sensor. A GNSS measurement in an inertial navigation system is often considered to be very accurate and is therefore used to account for drift. Therefore it is natural that anytime a GNSS measurement is received, a new keyframe is recorded. Following the same logic, a keyframe is also recorded whenever a loop closure is detected. Further, a keyframe is selected whenever a fixed amount of distance has been covered since the last keyframe, as suggested in [5].

Map Alignment

Drift is a severe issue for any SLAM system. Drift occurs when there is nothing rooting the estimates to the world. Since every pose is relative to the previous, drift will gradually build over time. Essentially, the map and the estimated trajectory will not be globally consistent and cannot be used to tell where the vehicle has traveled. The proposed system uses a map alignment step to minimize the effects of drift between each pose. This map alignment is based on ICP to adjust the estimate from the short-term feature tracking.

After the short-term feature tracking is performed, the front-end fine grains the estimated odometry by aligning the currently viewed point cloud to the map. This is done by a custom ICP algorithm, in order to add the possibility of adding whitened priors from the IMU. The alignment follows algorithm 4 using LM from algorithm 3 as the kernel of the optimization between each iteration. The feature matching inside the algorithm is a naive reciprocal closest-point matching algorithm in every iteration. IMU priors are integrated into the solution by adding an extra row to the **A** and **b** matrices in (3.21). The expression to be added into the matrices is given by the difference between the predicted pose from the IMU and the last predicted pose. Expressed mathematically, this gives

$$\mathbf{A}_{imu} = \mathbf{I}_{6x6} \tag{4.1}$$

$$\mathbf{b}_{imu} = -\Sigma_{imu}^{-1/2}\mathrm{Log}(\mathrm{Exp}(\hat{\xi}_{imu})^{-1}\mathrm{Exp}(\xi_{prev})) \tag{4.2}$$

where the Log(·) and Exp(·) operators are defined in Section 3.6.

Map refinement

The map alignment process fine grains the pose with respect to the map. Consequently, a map containing a lot of 'poor' points will only make the map alignment process worse. Therefore, the proposed system uses a map refinement thread that selects 'good' points in order to complement the map alignment process as well as possible. The map refinement process of the front-end runs on a separate thread with a frequency of 1Hz. The best points are chosen according to a RANSAC matching algorithm and further by comparing the selected points to points already in the map to avoid duplicates.

Preintegration of IMU measurements

IMU sensors generally run on a very high frequency compared to the other sensors of the system. Typically the sampling frequency lies around 100Hz, but it may lie anywhere in the 60Hz-1000Hz range. Compared to sensors like GNSS, which provides measurements with a frequency of $< 1Hz$, there are a lot of IMU measurements between each GNSS measurement. Including a factor in the factor graph for each IMU measurement would mean that the number of poses in the graph would exceed 1000 for a run over 10 seconds. Clearly, this is undesirable for optimization purposes, as a single relinearization would require the recalculation of many variables. Since the inner workings of the optimization use sparse matrices and reordering algorithms, this would increase computation times by a significant amount. Therefore the proposed system follows the suggestions of [18] for preintegrating IMU measurements between keyframes.

Timing Between Sensors

The factor graph formulation of the problem has a convenient representation that makes including measurements with latency possible. This also goes for measurements that, for some reason, have been received with a significant time delay. These measurements would have to be discarded for a pure filtering system, and valuable information may have been lost. However, handling such measurements requires more overhead as one must be careful in inserting the measurement in the correct place in the factor graph. For example, if a GNSS measurement arrives late, this must not get added to the latest estimate, as this would severely ruin the estimation.

Loop Closure Detection

Loop closures are the main difference between a SLAM system and an odometry system. They are essential for maintaining global consistency. The proposed system uses an ICP-based method for detecting loop closures, as suggested in [5]. However, this type of loop closure detection requires some overhead in order to work as desired. The first part of the loop closure detection is to extract relevant historic keyframes for the ICP algorithm to compare with. Relevant historic keyframes are extracted according to the current estimated position of the robot, the time the historic keyframes were recorded, and the relative heading between the two keyframes. After that, a set amount of frames around this historic keyframe are concatenated to form a larger segment. Finally, the ICP algorithm is used to align the latest keyframe to the concatenated cloud, and an average Euclidean distance score is used as a fitness measure to decide whether or not the latest keyframe is taken from the same scene.

The difference in viewpoint is accounted for by scaling the search point according to the difference in heading between the closest historic pose and the current pose.

This new search point is calculated according to

$$x_{search} = x_{current} + s \times \frac{|\psi_{closest} - \psi_{current}|}{\pi} cos(\psi_{closest} - \psi_{current}) \tag{4.3}$$

$$y_{search} = y_{current} + s \times \frac{|\psi_{closest} - \psi_{current}|}{\pi} sin(\psi_{closest} - \psi_{current}) \tag{4.4}$$

$$z_{search} = z_{current} \tag{4.5}$$

where ψ denotes the heading and s is a scaling parameter based on the range of the LiDAR. This equation makes it so that if the viewing angles are similar, the search point is simply the current point. If the scene is viewed from two completely different angles, then it accounts for the view indifference by searching s distance away from the current point in the direction $\psi_{closest} - \psi_{current}$. In this thesis the LiDAR has a range of 50m. Since viewing a point from two completely different scenes could mean two LiDAR scans apart, 60m was found to be a good value for s.

4.3.2 Optimization Back-end

As mentioned, the back-end is responsible for doing all the optimization. This includes sparse matrix inference, smoothing, and relinearization if needed. The proposed system uses the ISAM2 algorithm [4] as optimization engine. The ISAM2 tree in the system contains all the key poses as well as selected viewed points and the estimated velocity and bias at these key poses. The interaction between the front-end and the back-end is shown in figure 4.5.

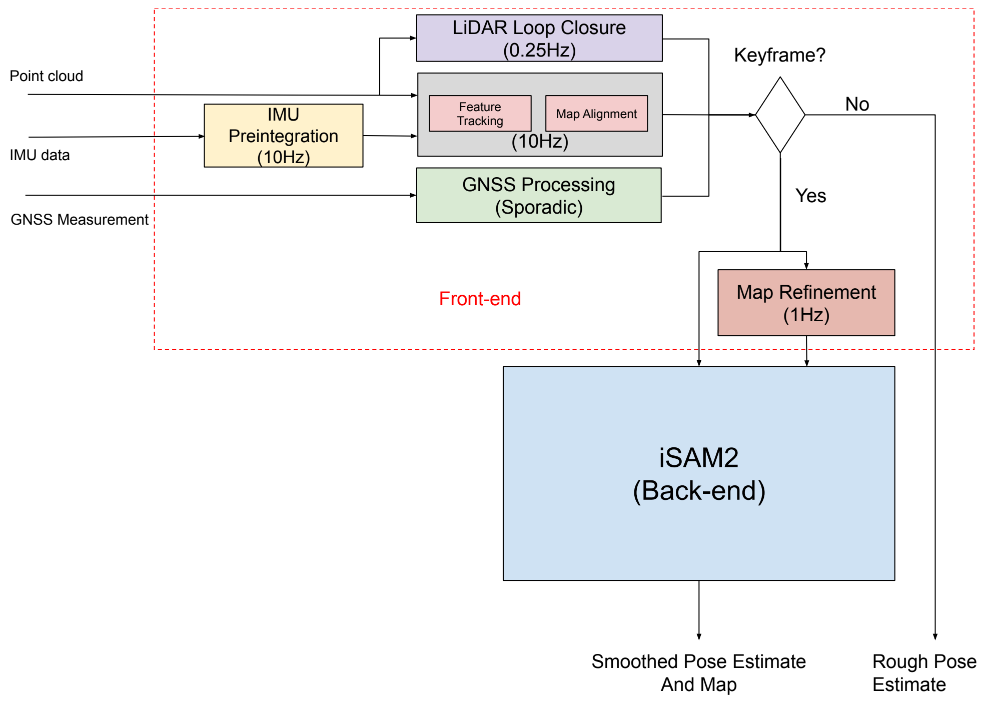


Figure 4.5: Back-end: The figure shows how the front-end prepares factors for the measurements before sending them to the back-end, and at what frequency they do so. The front-end delivers a rough pose estimate, while the back-end delivers a smoothed estimate whenever a keyframe is recorded.

Chapter 5

Results

This section explains the experiments and the results of the experiments on a superficial level. Section 5.1 shows how the system performs as more modalities are added. This addition is evaluated by inspecting the uncertainties and errors of a single passing through the tunnel. Thereafter, section 5.2 shows the results achieved by the system using the loop closure algorithm as explained in Section 4.3.1. Likewise, the analysis is based on using errors and uncertainties. However, it investigates when the vehicle travels back through the same tunnel. Each section describes the process noise covariances used in the simulations. The remaining hyperparameters can be found in Appendix A.

5.1 Sensor Comparison

5.1.1 Experiment

The experiment takes place in a simulated environment. The simulation consists of a large ground vehicle passing through an approximately 200m long tunnel and exiting the other side. The experiment is repeated three times, with an additional modality (sensor) added for each repetition. In addition to the metrics discussed in Section 3.13, the heading error is calculated, and the estimated trajectories are shown alongside the ground truth trajectory. In total, the vehicle is equipped with an IMU, a LiDAR, and a GNSS receiver. The GNSS measurements are received before entering the tunnel and after leaving the tunnel and are marked by the red crosses in figure 5.1. This experiment aims to study how the performance of the estimation develops with the addition of more modalities. The inclusion of each new modality is expected to yield a better result for the estimations.

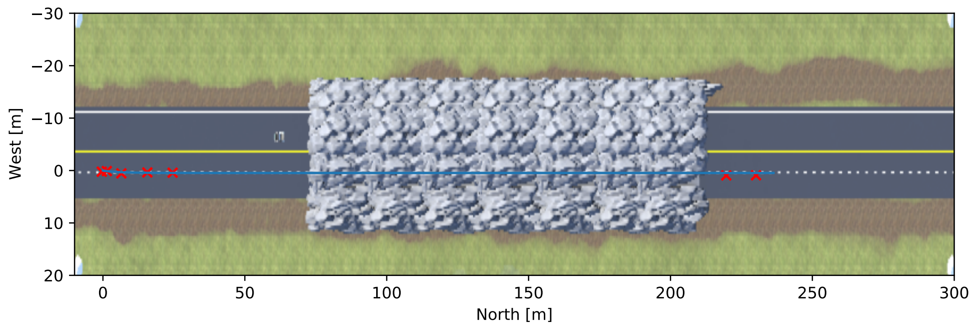


Figure 5.1: Experiment: The figure shows the trajectory of the sensor comparison experiment. It shows a straight path starting outside the tunnel on one side before exiting on the other side. The red crosses mark GNSS measurements

The experiments are run one by one. The figures are organized so that each separate experiment is divided into three subfigures. (a) is for the LiDAR-only experiment, (b) for the LiDAR+IMU experiment, and (c) for the LiDAR+IMU+GNSS experiment. After all experiments are shown, a summarizing caption is shown. The errors are calculated by comparing the ground truth value with the estimated value closest in time. The process noises are tuned to yield an approximately consistent NEES without having a too large impact on the trajectory tracking. The process noises are tuned according to Table 5.1.

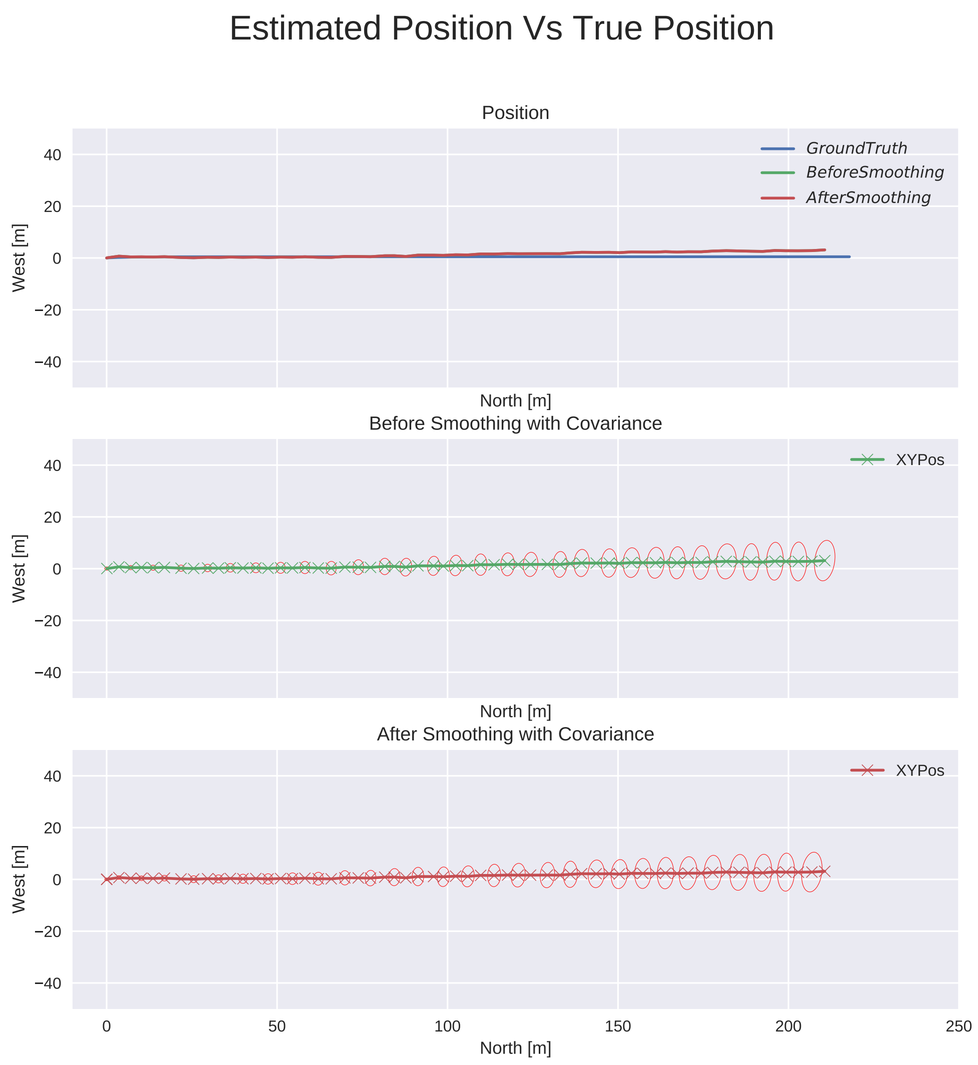
Table 5.1: Noise Specifications. All noises are to be considered as the diagonal of a square matrix. Any off-diagonal elements are assumed zero.

Parameter	Value
Structure Noise(m)	$[0.1, 0.1, 0.1]^T$
Odometry Position Noise(m)	$[0.1, 0.1, 0.3]^T$
Odometry Attitude Noise(rad)	$[5 \times 10^{-3}, 5 \times 10^{-3}, 1 \times 10^{-6}]^T$
GNSS Noise(m)	$[1.5, 1.5, 0.3]^T$
Accel. Measurement Noise$(m/s^2/\sqrt{Hz})$	$[0.01, 0.01, 0.01]^T$
Accel. Bias Driving Noise$(m/s/\sqrt{Hz})$	$[5 \times 10^{-3}, 5 \times 10^{-3}, 5 \times 10^{-3}]^T$
Gyro. Measurement Noise$(rad/s/\sqrt{Hz})$	$[1 \times 10^{-5}, 1 \times 10^{-5}, 1 \times 10^{-5}]^T$
Gyro. Bias Driving Noise$(rad/\sqrt{Hz})$	$[1 \times 10^{-4}, 1 \times 10^{-4}, 1 \times 10^{-4}]^T$

5.1.2 Results

This section goes through three repetitions of the experiment described above. Firstly, the experiment will be performed using only a LiDAR. Thereafter, a LiDAR and an IMU. Finally, the experiment will be repeated using all of the sensors, namely, a LiDAR, an IMU, and a GNSS receiver.

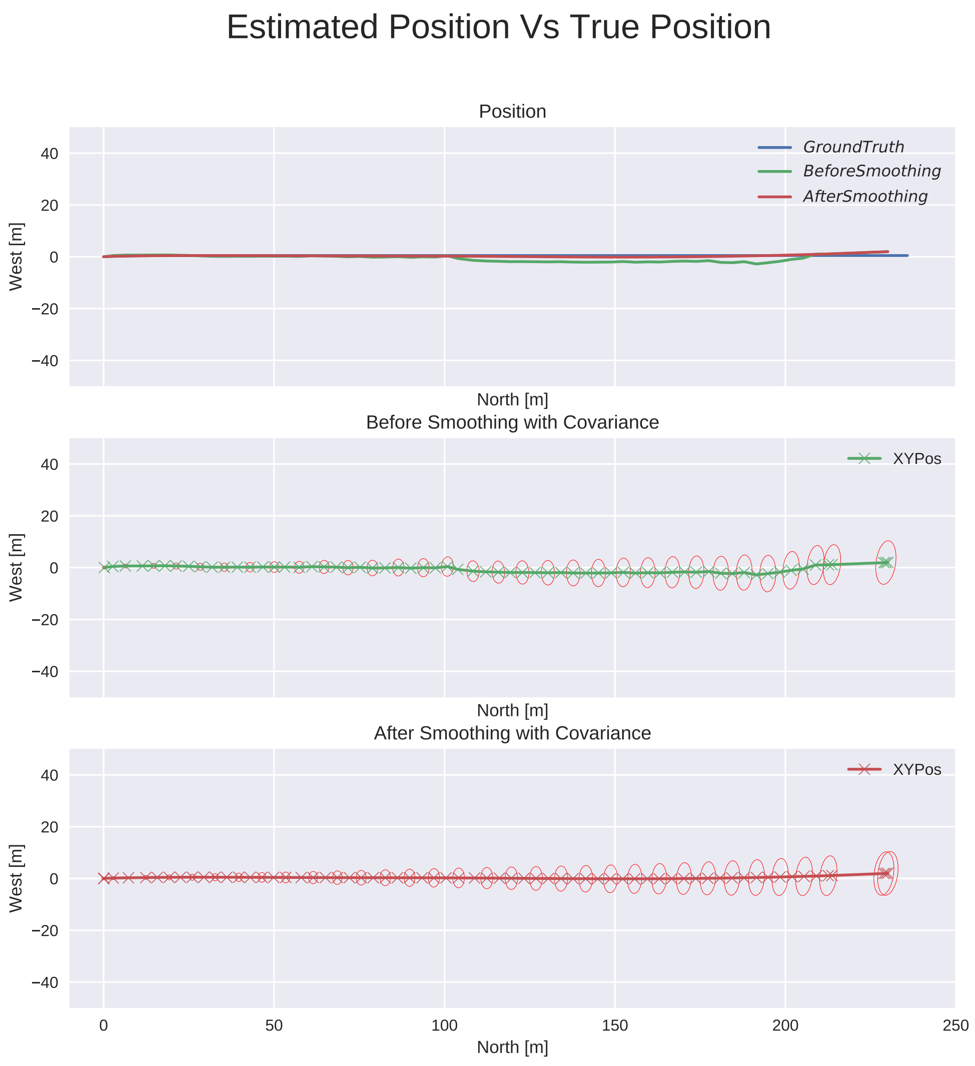
The upper plot of each of the three figures in Figure 5.2 shows the estimated trajectory before and after smoothing plotted against the ground truth trajectory for the three experiments. The other two plots show the estimated trajectory before and after smoothing, with the respective one-sigma ellipsoid around every other pose. The plots in Figure 5.2a show a shorter ground truth trajectory. The reason for this is that the trajectory has been truncated when there are no estimations, which is the case when the system runs using only the LiDAR sensor and the vehicle drives out of the tunnel. Notice that when using only the single modality, estimate before and after smoothing is exactly alike, which is valid for the entirety of the experiments using LiDAR only.



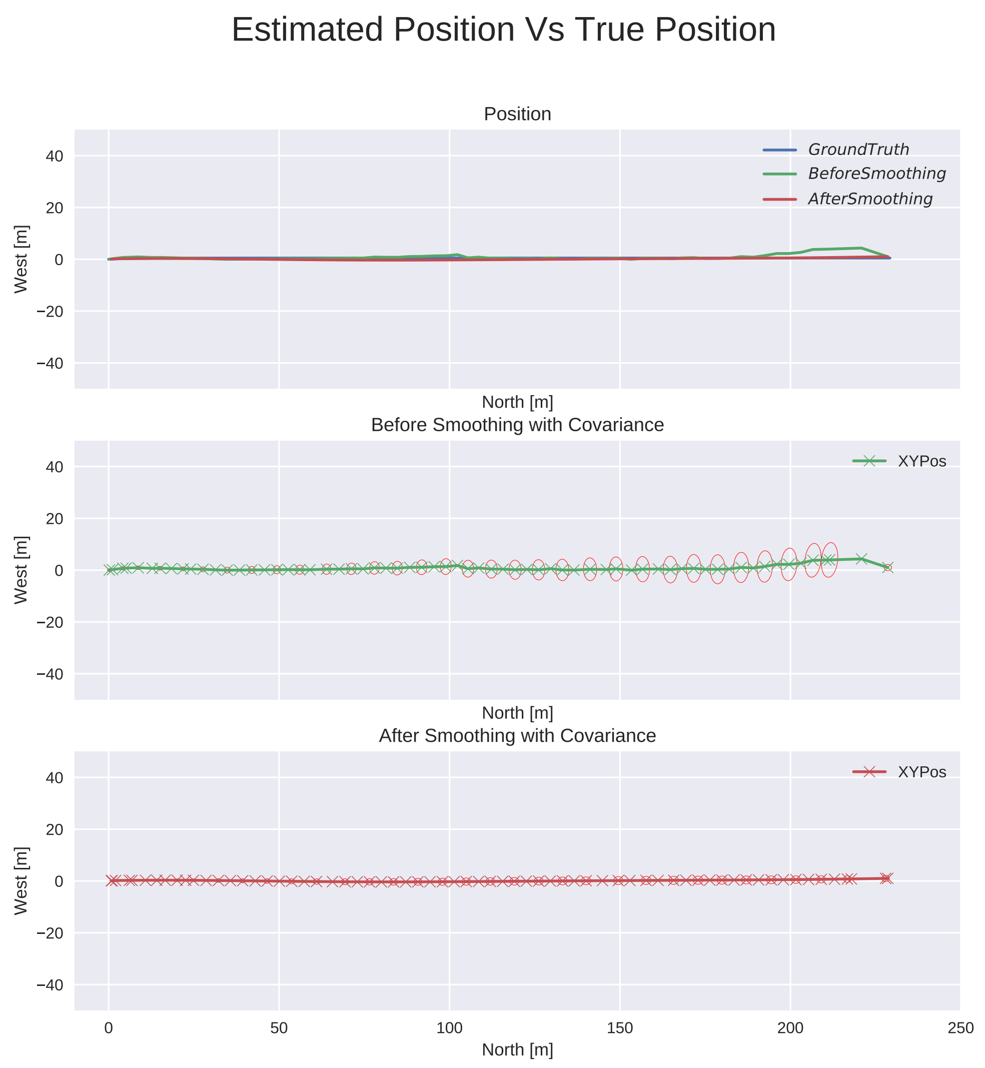
(a) LiDAR only

Figure 5.2b shows the trajectory when fusing LiDAR and IMU sensory information.

The final estimates are purely based on IMU dead-reckoning prediction, which arrives when the LiDAR times out. Figure 5.2c shows the same trajectory, but the fusion also includes GNSS measurements when the vehicle leaves the tunnel. The latter figure shows a smaller one-sigma ellipsoid in the smoothed trajectory.



(b) LiDAR+IMU

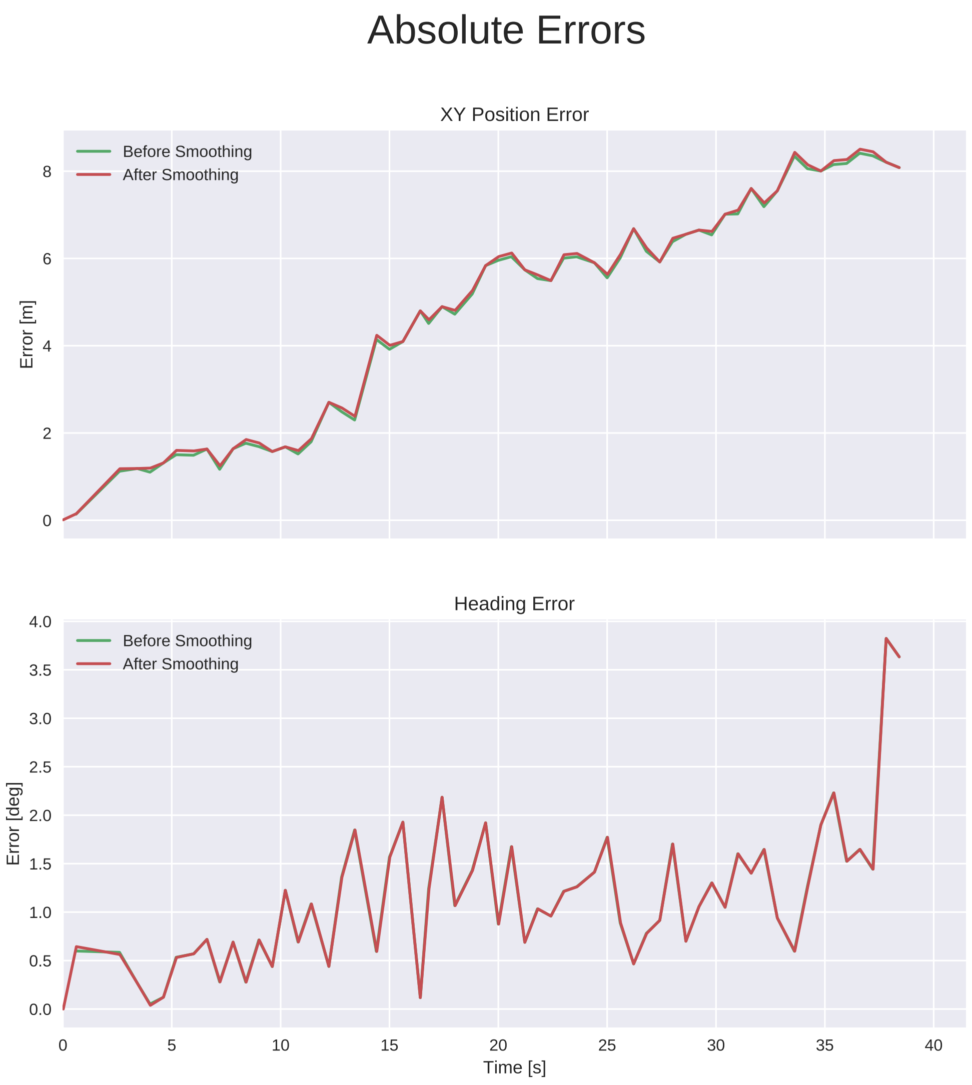


(c) LiDAR+IMU+GNSS

Figure 5.2: XY Position: The figures show the planar position for the sensor comparison experiment. (a) shows the results using only LiDAR. (b) shows the results when LiDAR odometry is fused with an IMU. (c) shows the results when GNSS information is fused when leaving the tunnel. The upper plot in each figure shows how the estimated trajectories before and after smoothing match the ground truth trajectory in the plane. The middle plot in each figure shows how the estimated trajectory with its respective one-sigma ellipses around every other pose. The lower plot in each figure shows the same as the middle plot, but after smoothing.

Figure 5.3 shows how the absolute errors of the system develop over time as the vehicle traverses the tunnel for the three experiments. The top plot in each figure shows the absolute error when comparing the position in the XY plane with their respective ground truth counterpart. The bottom plot in each figure shows the error in the vehicle's heading compared with the ground truth.

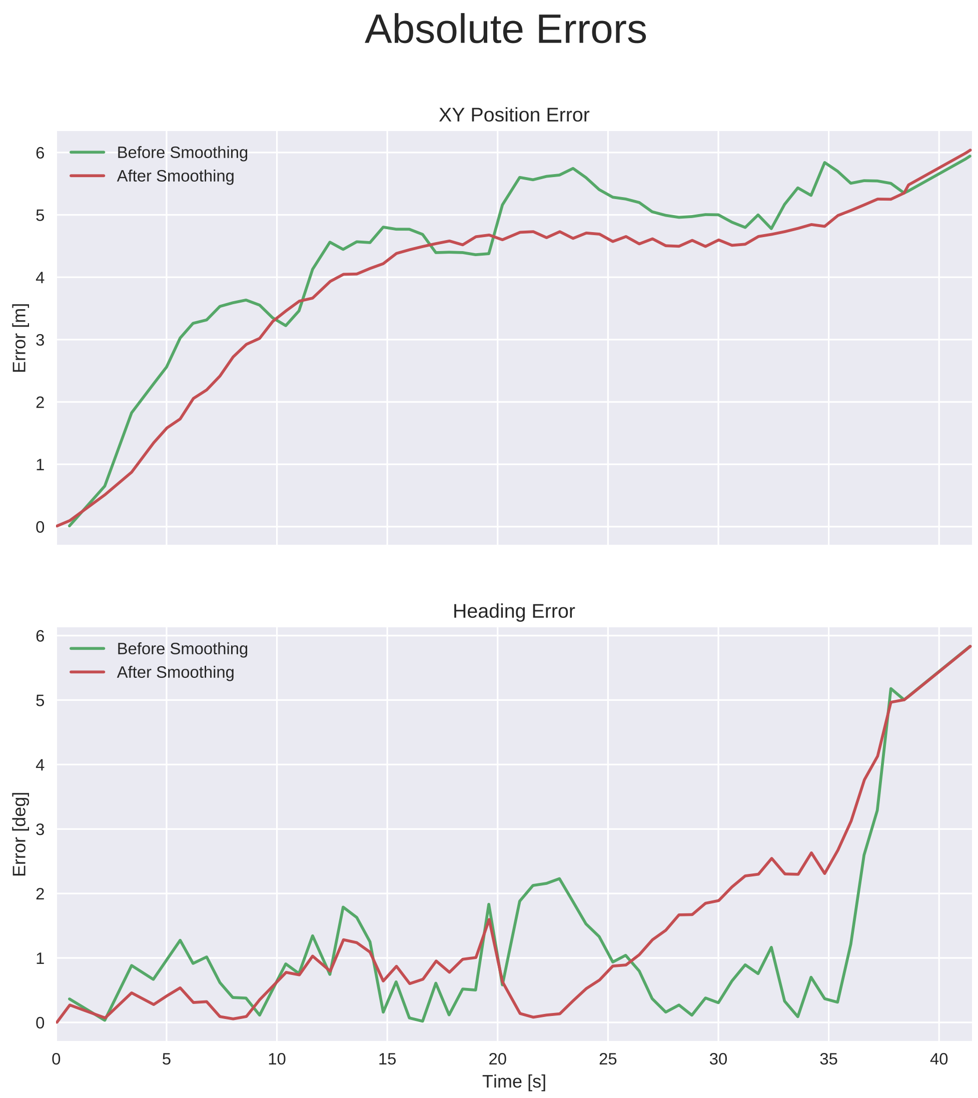
The absolute XY position error in figure 5.3a shows an approximately linear trend, reaching its peak at around 8m towards the end of the tunnel. The heading error shows no apparent trend. However, it oscillates at around 1 degree for the most part. Towards the end, it spikes and reaches its peak at around 3.6 degrees.



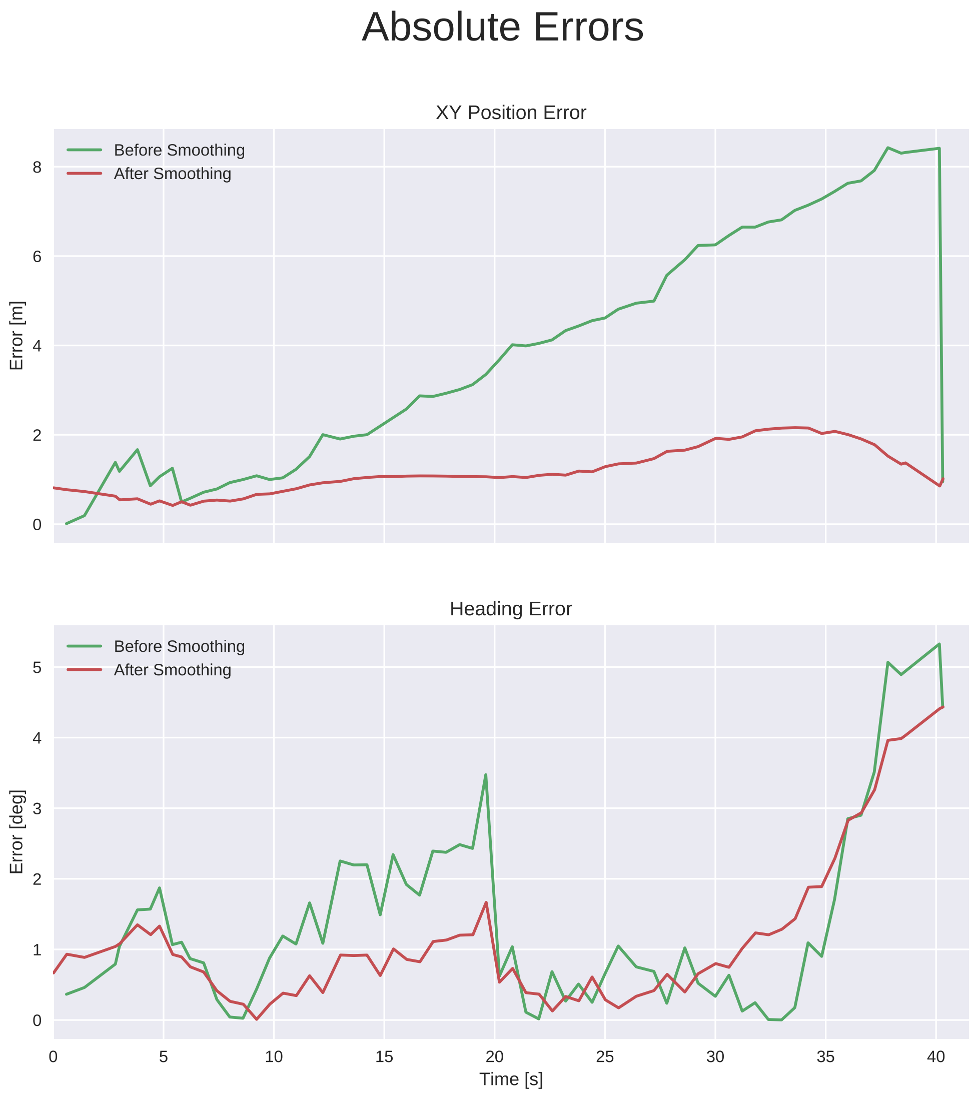
(a) LiDAR only

The absolute XY positional error in figure 5.3b shows an increasing trend that stops as the vehicle is in the middle of the tunnel before slightly increasing again and reaching the peak at around 6m when the vehicle is reaching the end. When there is information from several sensors, the difference between the smoothed and non-smoothed curves becomes apparent. The heading error shows no apparent trend. However, it increases towards the end of the tunnel, reaching its peak at around 6 degrees.

Figure 5.3c shows an increasing trend before smoothing, but an approximately constant error after smoothing. Before smoothing, it reaches its peak just before a GNSS measurement arrives at approximately 8m. Like the previous experiments, the heading error shows no apparent trend; however, it is generally smaller after smoothing. Both curves reach their peak towards the end of the tunnel. The trajectory before smoothing peaks at approximately 5.5 degrees right before the GNSS measurement. The trajectory after smoothing peaks at approximately 4.5 after the GNSS measurement.



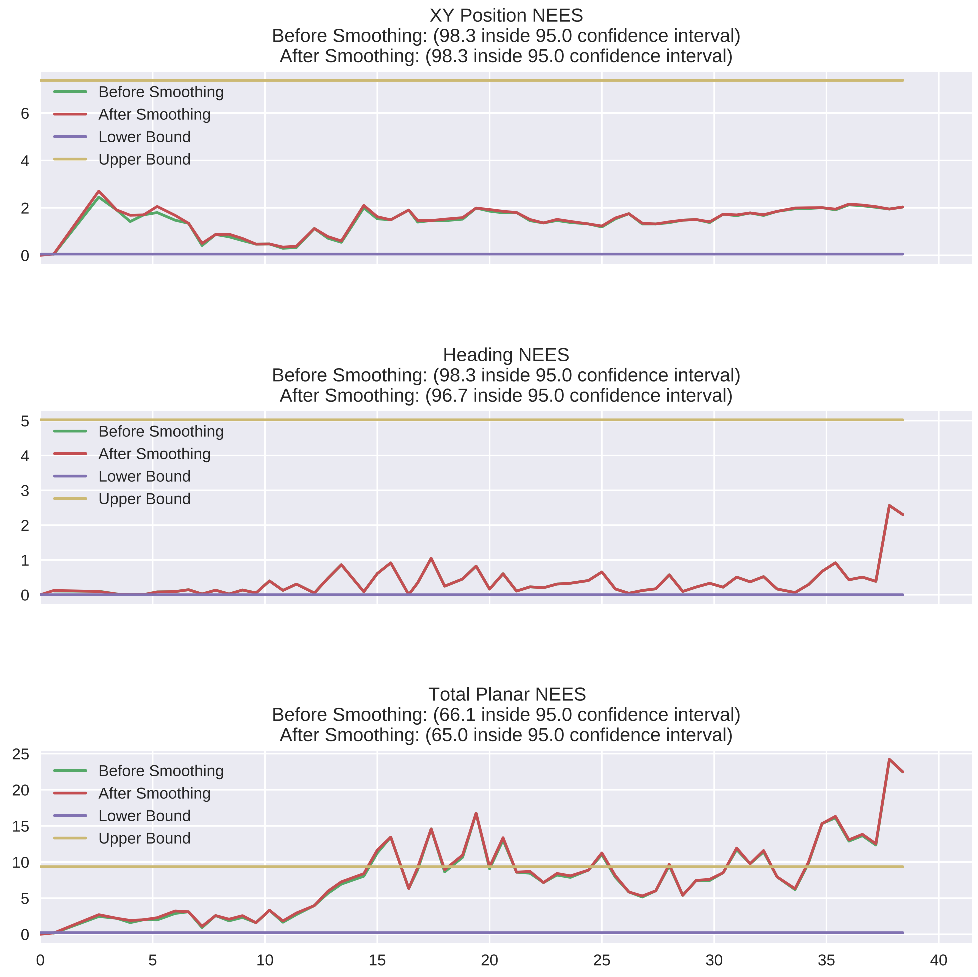
(b) LiDAR+IMU



(c) LiDAR+IMU+GNSS

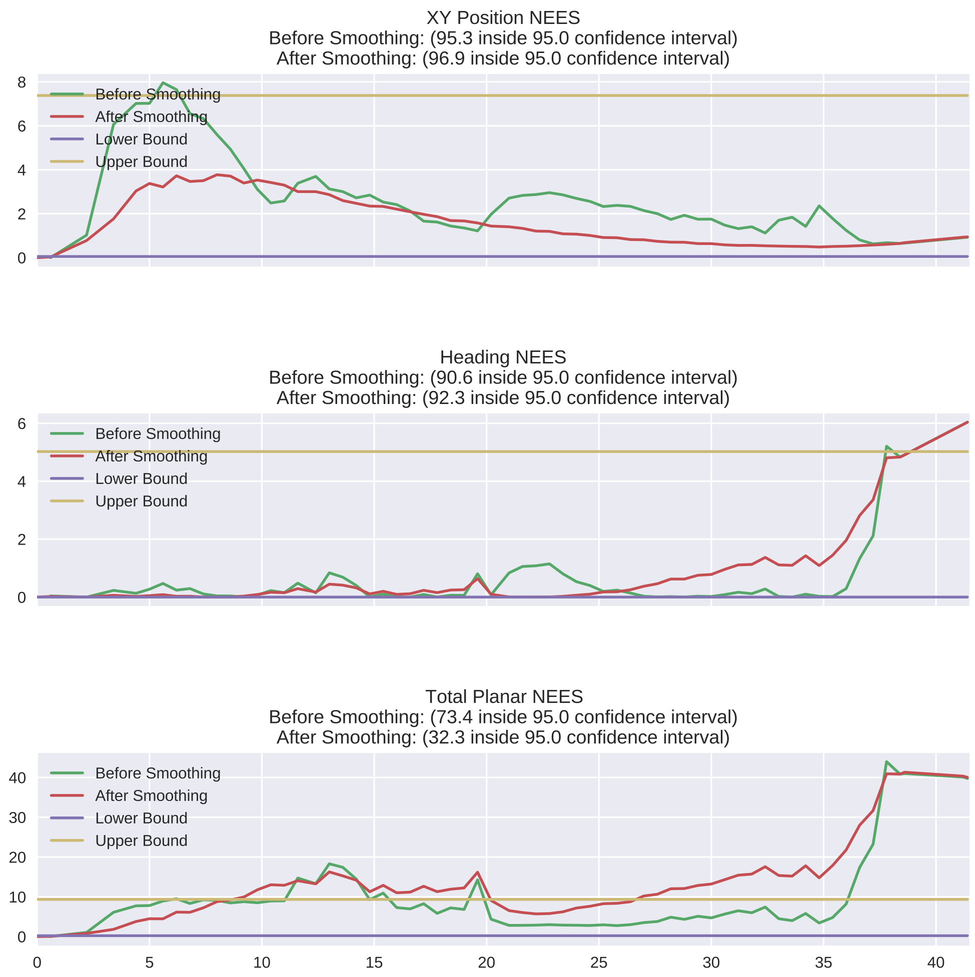
Figure 5.3: Absolute Errors: The figures show how the planar errors develop over time for the sensor comparison experiment. (a) shows the results when estimations are solely based on the LiDAR sensor. (b) shows the results when fusing LiDAR and IMU sensory information is fused. (c) shows the results when LiDAR, IMU, and GNSS information is fused. The top plot of each figure shows how the absolute error in the XY position develops before and after smoothing. The bottom plot of each figure shows how the absolute error in heading develops before and after smoothing as the vehicle traverses the tunnel.

Figure 5.4 displays how different NEESes (as described in section 3.13) develop over time before and after smoothing for each experiment. The top plot shows the NEES of the positional states in the XY plane. This state vector is two-dimensional, and is therefore compared to a χ^2-distribution with two DoF. For the first experiment, shown in Figure 5.4a, the results after smoothing differ a negligible amount from the results before smoothing because there is a prior on the first pose. Figure 5.4a shows that the XY position lies within the 95% confidence interval 98% of the time both before and after smoothing for the experiment using a LiDAR only. Figure 5.4b shows that the XY position manages to lie within the bounds 95% confidence interval 95.3% of the time before smoothing and 96.9% of the time after smoothing when fusing LiDAR and IMU sensor information. Figure 5.4c shows that the fusing LiDAR, IMU, and GNSS sensor information results in an XY position NEES of 98.5% of the time before smoothing and 100% of the time after smoothing.



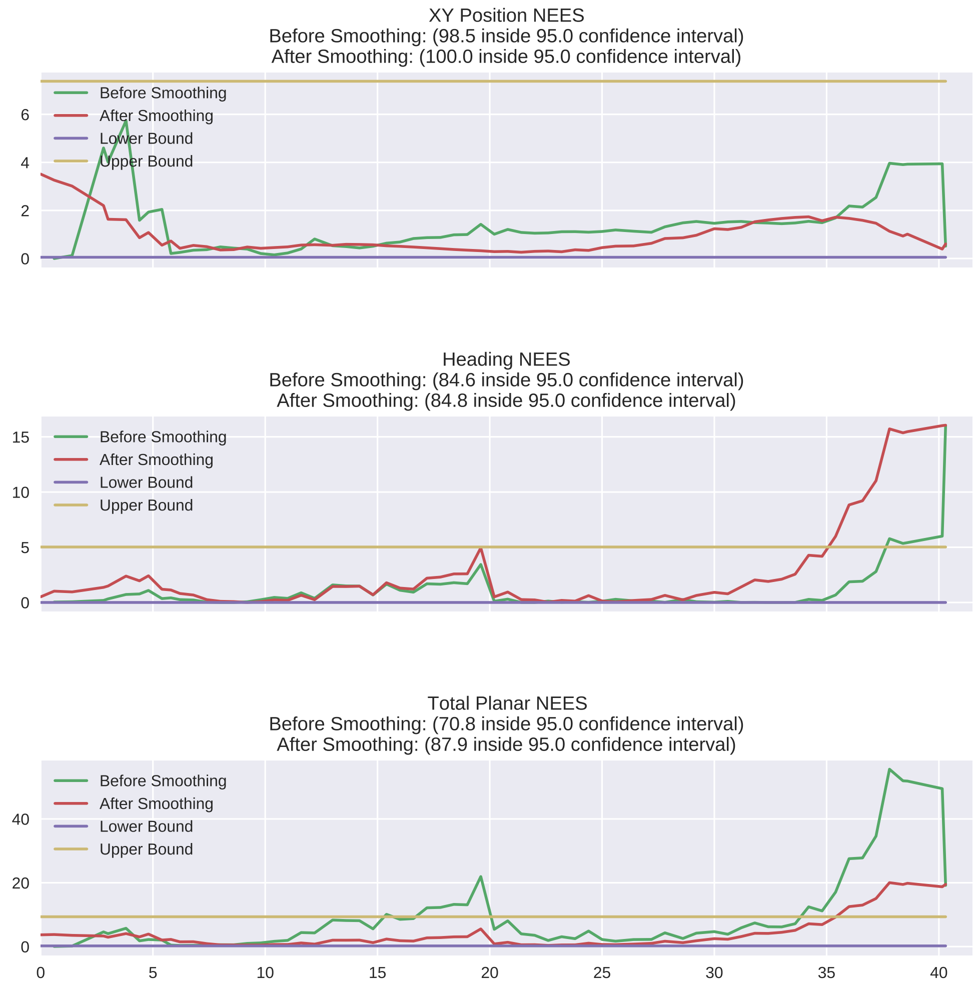
(a) LiDAR only

The middle plot shows the NEES of the heading state. Following the same method, this is compared to a χ^2-distribution with one degree of freedom. When using only the LiDAR sensor, Figure 5.4a shows that the system lies within the 95% confidence interval 98.3% of the time both before and after smoothing. Figure 5.4b shows that including IMU information results in 90.6% of the heading NEESes lying inside the 95% confidence interval before smoothing, while 92.3% of the NEESes lie inside the interval after smoothing. The final experiment including GNSS information before the vehicle enters the tunnel and after the vehicle leaves results in the system achieving 84.6% of the heading NEESes inside the interval before smoothing and 84.8% of the NEESes after the smoothing.



(b) LiDAR+IMU

The bottom plot shows the total planar NEES. As this state vector is three-dimensional, the confidence interval is taken from a χ^2-distribution with three DoF. According to figure 5.4a, the LiDAR-only experiment achieves its NEESes inside the 95% confidence interval 66.1% of the time. Figure 5.4b shows that fusing IMU and LiDAR information gives 73.4% of the total planar NEESes inside the 95% confidence interval before smoothing, while 32.3% of the NEESes are inside the interval after smoothing. When including all the sensory information available, the system achieves its total planar NEESes inside the 95% confidence interval 70.8% of the time before smoothing and 87.9% of the time after smoothing.



(c) LiDAR+IMU+GNSS

Figure 5.4: Planar NEESes: The figures show the NEESes over time for the sensor comparison experiment. (a) shows the results when estimations are solely based on the LiDAR sensor. (b) shows the results when fusing LiDAR and IMU sensory information is fused. (c) shows the results when LiDAR, IMU, and GNSS information is fused. The top plot of each figure shows the XY-plane positional NEES before and after smoothing along with the 95% confidence interval bounds. The middle plot of each figure shows the heading NEES before and after smoothing along with the 95% confidence interval bounds. The lower plot of each figure shows the total NEES of the planar states before and after smoothing along with their respective 95% confidence interval bounds.

The RMSEs and the ANEESes summarize the results on the entirety of the trajectory. The RMSEs and ANEESes for each included modality before and after smoothing are shown in Table 5.2 and Table 5.3.

Table 5.2: RMSEs for the Sensor Comparison Experiment

State	**Experiment**	**Before Smoothing**	**After Smoothing**
North	LiDAR	4.66m	4.66m
	LiDAR + IMU	4.25m	4.13m
	LiDAR + IMU + GNSS	4.33m	1.03m
West	LiDAR	1.32m	1.32m
	LiDAR + IMU	1.61m	0.45m
	LiDAR + IMU + GNSS	1.03m	0.46m
Planar Position	LiDAR	4.85m	4.85m
	LiDAR + IMU	4.55m	4.15m
	LiDAR + IMU + GNSS	4.46m	1.13m
ψ	LiDAR	1.35°	1.35°
	LiDAR + IMU	1.97°	2.17°
	LiDAR + IMU + GNSS	1.94°	1.58°

Table 5.3: ANEESes for the Sensor Comparison Experiment

State	**Experiment**	**Before Smoothing**	**After Smoothing**	**Confidence Interval**
XY Position	LiDAR	1.50	1.50	[1.52, 2.54]
	LiDAR + IMU	2.56	1.49	[1.54, 2.52]
	LiDAR + IMU + GNSS	1.43	0.92	[2.43, 3.62]
ψ	LiDAR	0.38	0.38	[0.67, 1.39]
	LiDAR + IMU	0.78	0.93	[0.68, 1.38]
	LiDAR + IMU + GNSS	1.14	2.74	[0.69, 1.37]
Total	LiDAR	8.1	7.9	[2.41, 3.66]
	LiDAR + IMU	12.88	13.67	[2.43, 3.63]
	LiDAR + IMU + GNSS	10.79	4.14	[2.43, 3.62]

5.2 Loop Closure

This section shows the results achieved with a simple and naive loop closure detection algorithm.

5.2.1 Experiment

The experiment takes place in the same simulated environment as Section 5.1. The road roller is equipped with three sensors; a LiDAR, an IMU, and a GNSS receiver. Additionally, a simple loop closure algorithm is implemented to provide a higher global consistency. As mentioned in 1.1, the road roller is at minimum required to do a set number of *passings* over the material to comply with the road construction standards. The purpose of the experiment is therefore to investigate how the road rollers performance would change when driving through an already visited scene, and how it changes from the first time it drove through. The loop closure detection algorithm is expected to yield a better result for the estimations when passing through an already visited area.

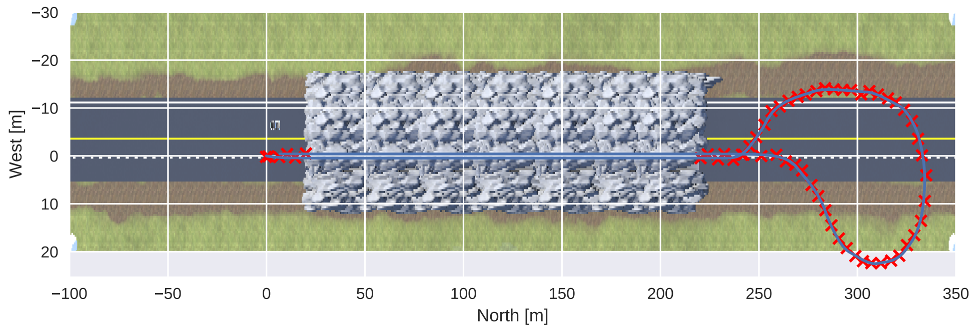


Figure 5.5: Experiment: The figure shows the trajectory of the loop closure experiment. It is a straight trajectory, starting at the outside of the tunnel on one side before performing a loop and driving back through the same tunnel. The red crosses mark GNSS measurements.

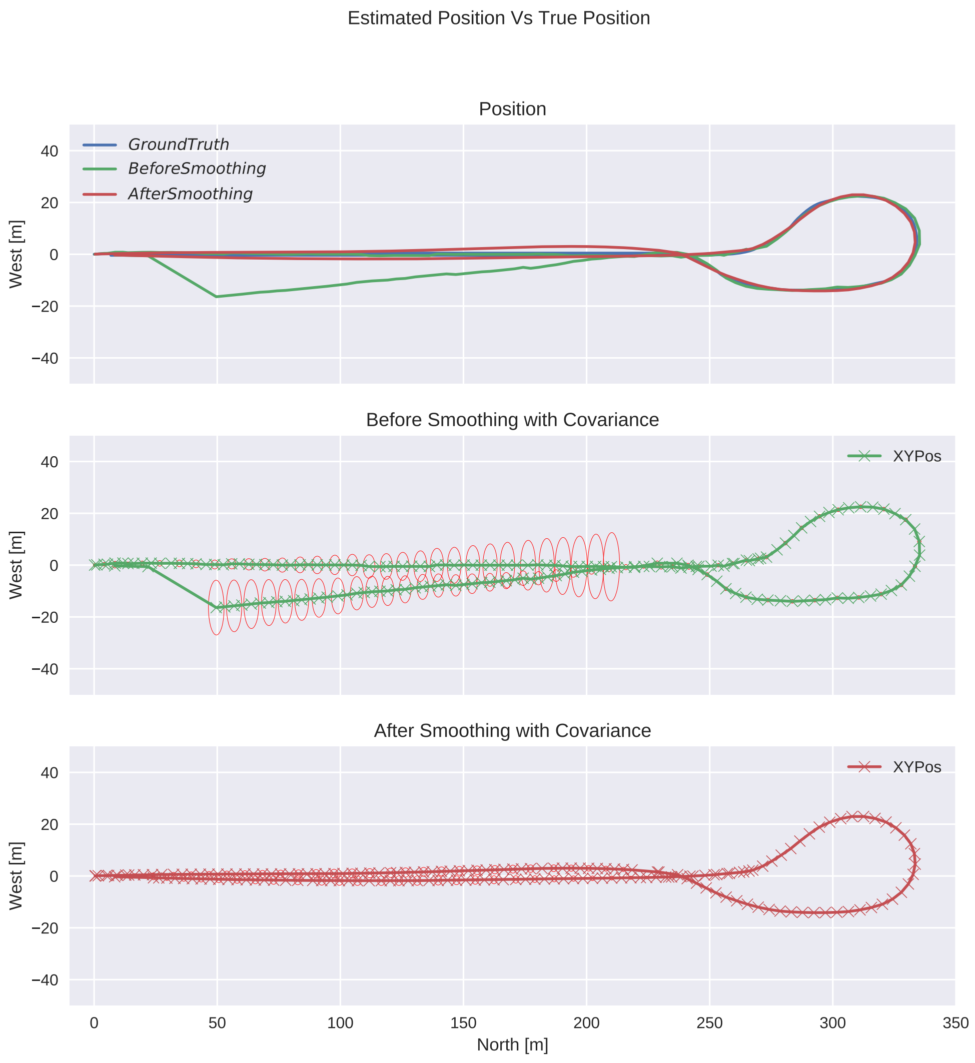
The figures are organized similarly as in the previous section. Each figure is divided into two subfigures. (a) shows the results without loop closure activated, and (b) shows the results with loop closure activated. After both subfigures are displayed, a summarizing caption is displayed. The errors are calculated by comparing the ground truth value with the estimated value closest in time. The process noises are tuned according to Table 5.4

Table 5.4: Noise Specifications. All noises are to be considered as the diagonal of a square matrix. Any off-diagonal elements are assumed zero.

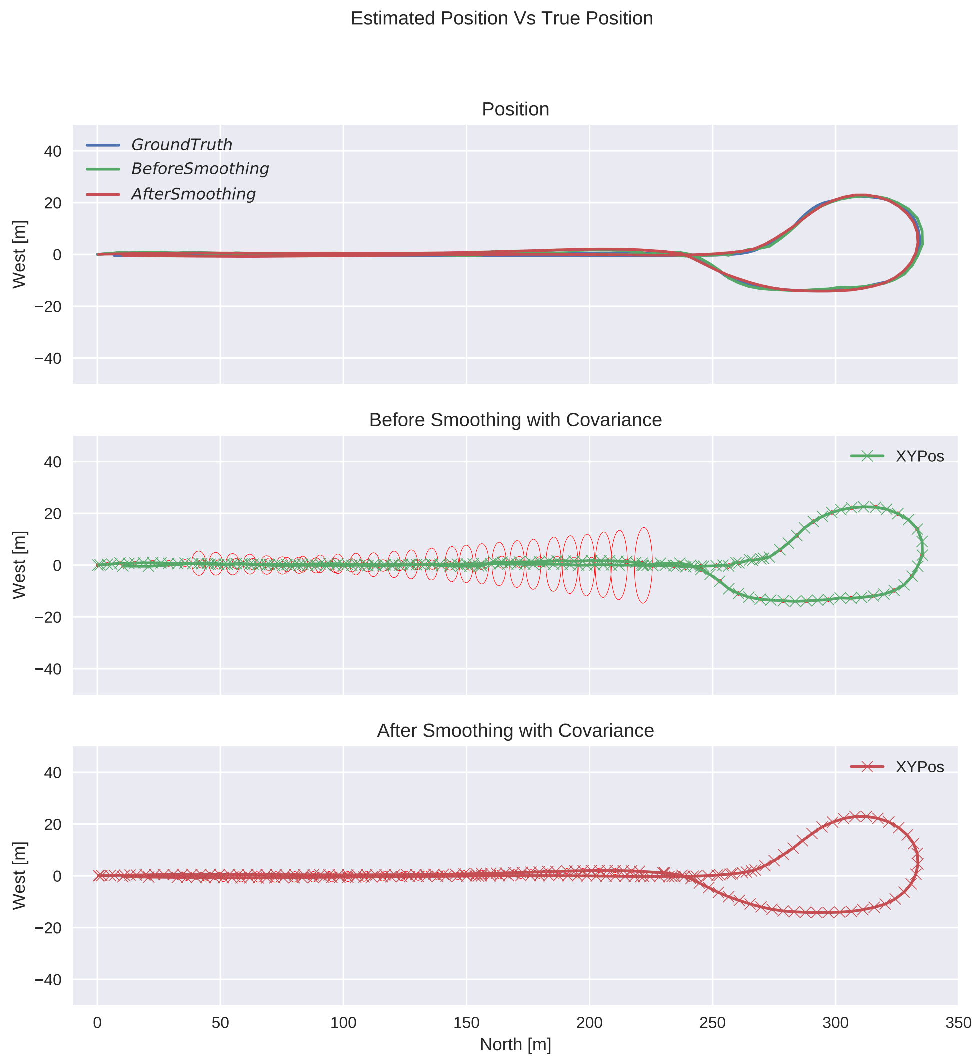
Parameter	**Value**
Structure Noise(m)	$[0.1, 0.1, 0.1]^T$
Odometry Position Noise(m)	$[0.1, 0.1, 0.3]^T$
Odometry Attitude Noise(rad)	$[5 \times 10^{-3}, 5 \times 10^{-3}, 1 \times 10^{-6}]^T$
GNSS Noise(m)	$[1.5, 1.5, 0.3]^T$
Accel. Measurement Noise($m/s^2/\sqrt{Hz}$)	$[0.01, 0.01, 0.01]^T$
Accel. Bias Driving Noise($m/s/\sqrt{Hz}$)	$[5 \times 10^{-3}, 5 \times 10^{-3}, 5 \times 10^{-3}]^T$
Gyro. Measurement Noise($rad/s/\sqrt{Hz}$)	$[1 \times 10^{-5}, 1 \times 10^{-5}, 1 \times 10^{-5}]^T$
Gyro. Bias Driving Noise($rad/\sqrt{Hz}$)	$[1 \times 10^{-4}, 1 \times 10^{-4}, 1 \times 10^{-4}]^T$

5.2.2 Results

This section shows the results achieved by the complete LiDAR SLAM system on the round trip shown in Figure 5.5. Figure 5.6 shows the position in the XY plane. Figure 5.6a shows the trajectories without loop closure enabled and Figure 5.6b shows them with loop closure enabled. The top plots show the estimated position before and after smoothing plotted against the ground truth. The middle plots show the estimate before smoothing, where every other pose is marked with its respective one-σ ellipsoid. Poses are marked with x. The bottom plots show the same as the middle plot, but the estimate is taken after smoothing when the trajectory is finished.



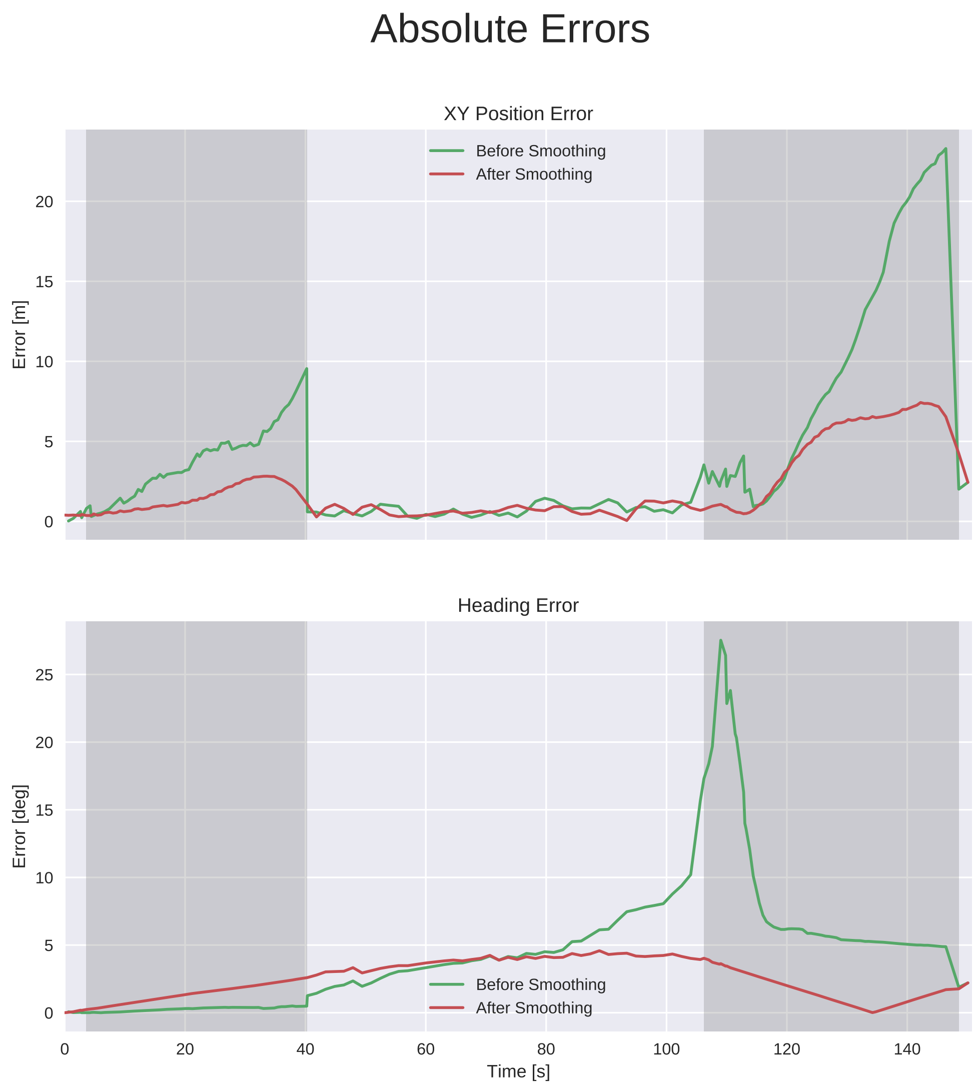
(a) Without Loop Closure



(b) With Loop Closure

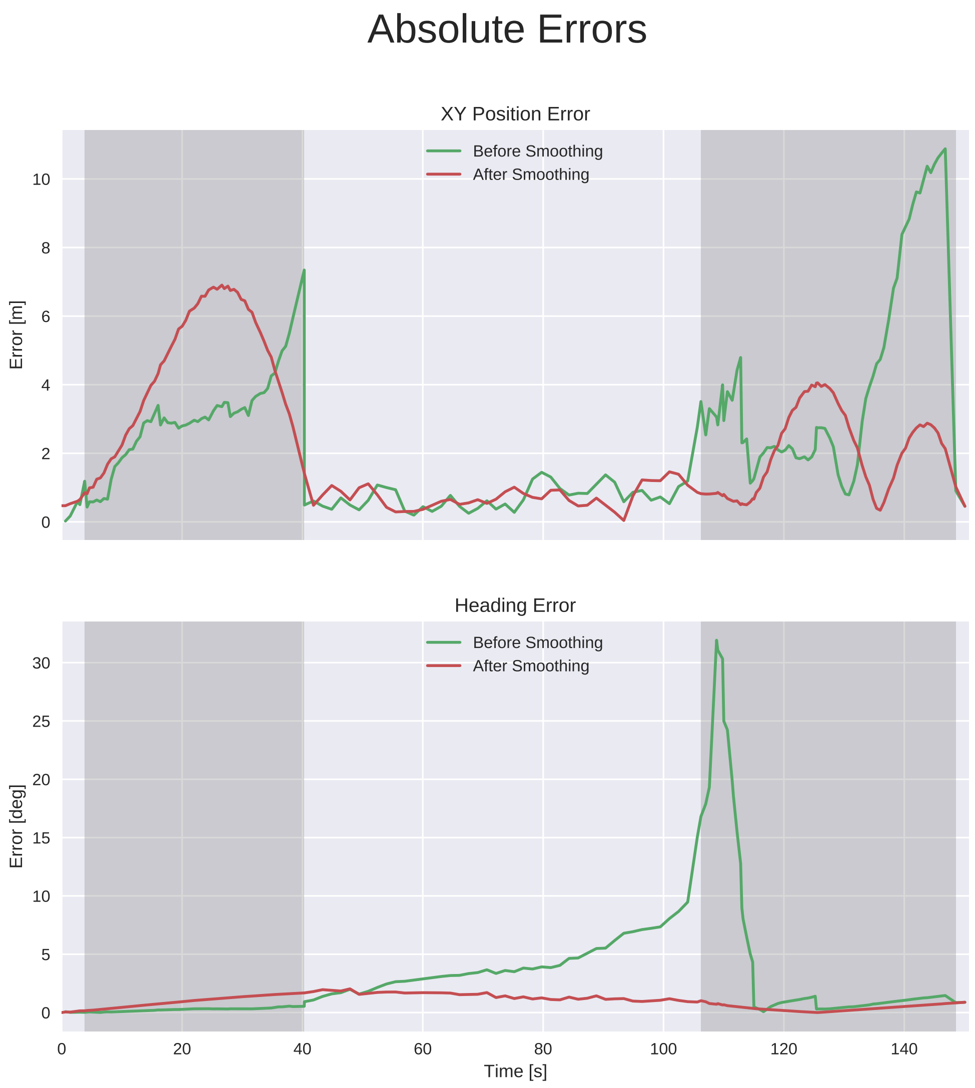
Figure 5.6: XY Position: Estimate vs Ground Truth Before Smoothing for the loop closure experiment. (a) shows the results without the loop closure module activated, while (b) shows the results with loop closures enabled. The top plot in each figure shows the estimated trajectory before and after smoothing plotted against ground truth. The middle plot shows the estimated trajectory with the corresponding one-σ ellipsoid plotted at every other pose. The bottom plot shows the same as the middle, but after smoothing.

Figures 5.7a and 5.7b show the errors in the XY plane without and with loop closure enabled, respectively. The shaded areas are sketched to show where GNSS coverage is available and not. GNSS coverage is not available where the graph inside the grey sketched area. It is available in the lighter parts of the graph, namely the start, middle, and end of the graph. The top plot shows the absolute error in the XY position, while the bottom plot shows the absolute error in the heading. They are both compared to the ground truth. The top plots of both Figure 5.7a and Figure 5.7b shows that the error before smoothing increases linearly before reaching its peak just before leaving the tunnel. After smoothing, however, they differ, and neither of them shows a linearly increasing trend. Outside the tunnel, the system maintains a low error. During the return through the tunnel, Figure 5.7a shows that the system errors increase faster. Figure 5.7b shows that the system with loop closure contains the error for a little while before an increasing trend is re-initiated during the return through the tunnel.



(a) Without Loop Closure

The bottom plots show that the heading error maintains a low value for the entirety of the run after smoothing. Before smoothing it increases approximately linearly outside of the tunnel. A significant peak appears just after re-entering the tunnel. The plot shows that the trajectory after smoothing is kept almost constant or is slowly increasing.



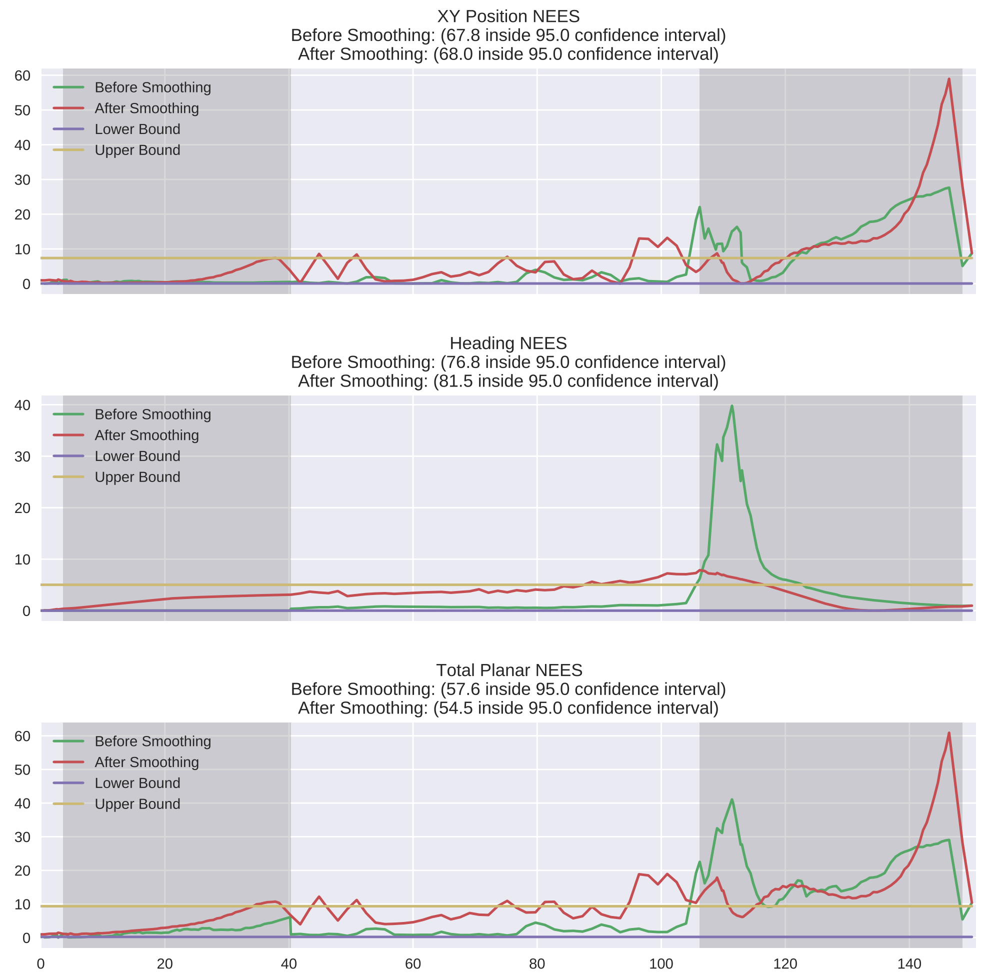
(b) With Loop Closure

Figure 5.7: Absolute Errors: The figures show the absolute planar error over time for the loop closure experiment. (a) Shows the errors without loop closures enabled, while (b) shows the errors with loop closures enabled. The top plot in each figure shows the absolute error in the XY plane before and after smoothing. The bottom plot in each figure shows the heading error before and after smoothing. The shaded areas show where the GNSS coverage isn't available.

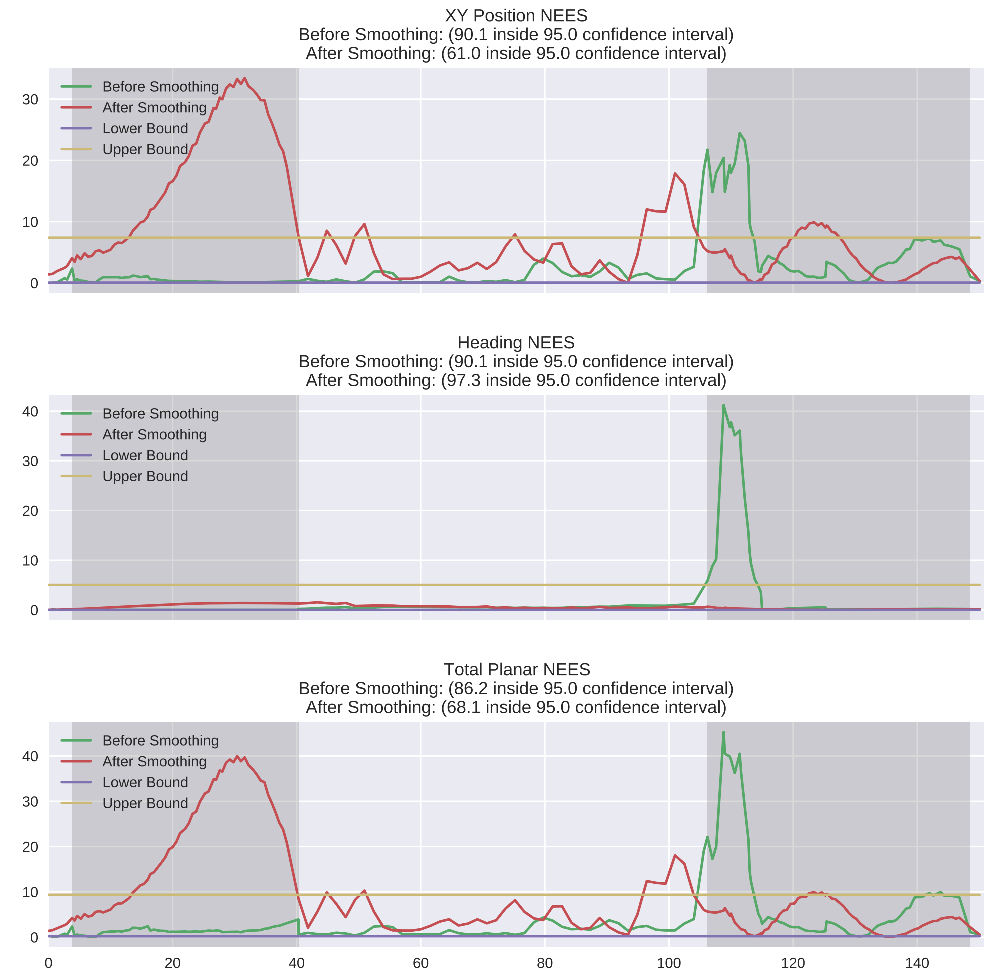
Figure 5.8 shows how the planar NEES develops throughout the trajectory with and without loop closure. It is shown without loop closure enabled in Figure 5.8a, and with loop closure in Figure 5.8b. Similar to the previous figure, the shaded areas and light areas indicate where there is and is not GNSS coverage. The top plot shows the XY position NEES, which is compared to a χ^2 distribution with two DoFs. It shows that the estimation before smoothing lies within the 95% confidence bounds 67.8% of the time without loop closure and 90.1% of the time with loop closure. After smoothing, it lies within the confidence interval 68% of the time without loop closure, while it lies within the confidence bounds 61% of the time with loop closure.

The middle plots show the NEES of the heading state. Consequently, this is compared to a χ^2 distribution with one DoF. It shows that the system achieves a heading NEES that lies within the 95% confidence bound 76.8% of the time before smoothing and 81.5% after smoothing without loop closure. With loop closure, it lies within the same confidence bounds 90.1% of the time before smoothing and 97.3% of the time after smoothing.

The bottom plots show the total planar NEES. This state vector is three-dimensional. Hence it is compared to a χ^2 distribution with three DoFs. It shows that the system achieves a total planar NEES that lies within the 95% confidence bounds 76.8% before smoothing and 81.5% after smoothing without loop closure. With loop closure enabled, the same tuning yields a system that lies within the confidence bounds 86.2% of the time before smoothing and 68.1% of the time after smoothing.



(a) Without Loop Closure



(b) With Loop Closure

Figure 5.8: Planar NEESes: The figures show how the planar NEESes develop over time before and after smoothing. (a) shows how the NEES develops without loop closure enabled, while (b) shows how it develops with loop closure enabled. The top plot of each figure shows the planar position NEES. The middle plot shows the heading NEES. The bottom plot shows the total planar NEES. Every plot includes its corresponding two-sided 95% confidence interval.

The overall results on the trajectory are summarized by the RMSEs and ANEESes. The RMSEs are shown in Table 5.5 and the ANEESes are shown in Table 5.6. The tables show the results with and without the loop closure module enabled before and after smoothing.

Table 5.5: RMSEs for the Loop Closure Experiment

State	Experiment	Before Smoothing	After Smoothing
North	Without Loop Closure	5.89m	3.08m
	With Loop Closure	3.57m	3.07m
West	Without Loop Closure	5.00m	1.03m
	With Loop Closure	0.69m	0.55m
Planar Position	Without Loop Closure	7.73	3.25m
	With Loop Closure	3.64m	3.12m
ψ	Without Loop Closure	7.26°	2.47°
	With Loop Closure	6.48°	0.99°

Table 5.6: ANEESes for the Loop Closure Experiment

State	Experiment	Before Smoothing	After Smoothing	Confidence Interval
XY Position	Without Loop Closure	5.84	7.04	[1.72, 2.31]
	With Loop Closure	2.97	8.59	[1.72, 2.31]
ψ	Without Loop Closure	3.64	2.84	[0.80, 1.22]
	With Loop Closure	2.20	0.54	[0.80, 1,22]
Total	Without Loop Closure	9.14	10.34	[2.65, 3.37]
	With Loop Closure	4.62	9.85	[2.65, 3.37]

Chapter 6

Discussion

This chapter aims at providing a detailed discussion of the results represented in Chapter 5. Section 6.1 discusses the results relevant to the sensor comparison experiment conducted in Section 5.1. Similarly, Section 6.2 discusses the results relevant to the loop closure experiment conducted in Section 5.2.

6.1 Sensor Comparison

The goal of the sensor comparisons experiments in Section 5.1 is to study how the performance develops as more modalities are added to the system. This lets us investigate how the LiDAR can be incorporated to contribute to improving the standard navigation system. This section will investigate factors that influenced the results of the different experiments and how they may be further improved.

Before discussing the results, it is important to note that there is an element of randomness in the estimation system. The map building process uses a RANSAC algorithm for outlier removal. Therefore the results may slightly vary from run to run, even though the system is tuned with the same hyperparameters. This should, however, not have a great impact but is worth keeping in mind when comparing the different runs. For example, the LiDAR+IMU run displayed as a green line in Figure 6.1 suggests that the system has better estimates inside the tunnel than the system that also includes GNSS measurements. This does not make sense since there are only IMU and LiDAR measurements available inside the tunnel. Therefore, the general trend shown by each experiment is what should be analyzed.

Figure 5.2 shows that the LiDAR-only experiment provides a shorter estimated trajectory compared to the others. The same can be seen by studying the blue line in Figure 6.1. LiDAR sensors cannot perform any odometry estimates when there is no structure available, and since the trajectory ends outside the tunnel, the final part has no structure. This result highlights a weakness of motion estimation based on LiDARs, and exteroceptive sensors in general, namely that they require structure. Relying on a single modality alone is rarely applicable in a real-world application since any sensor has different conditions in which it will not work. Consequently, the inclusion of inertial sensors is natural, seeing as most modern autonomous vehicles already have these types of systems incorporated.

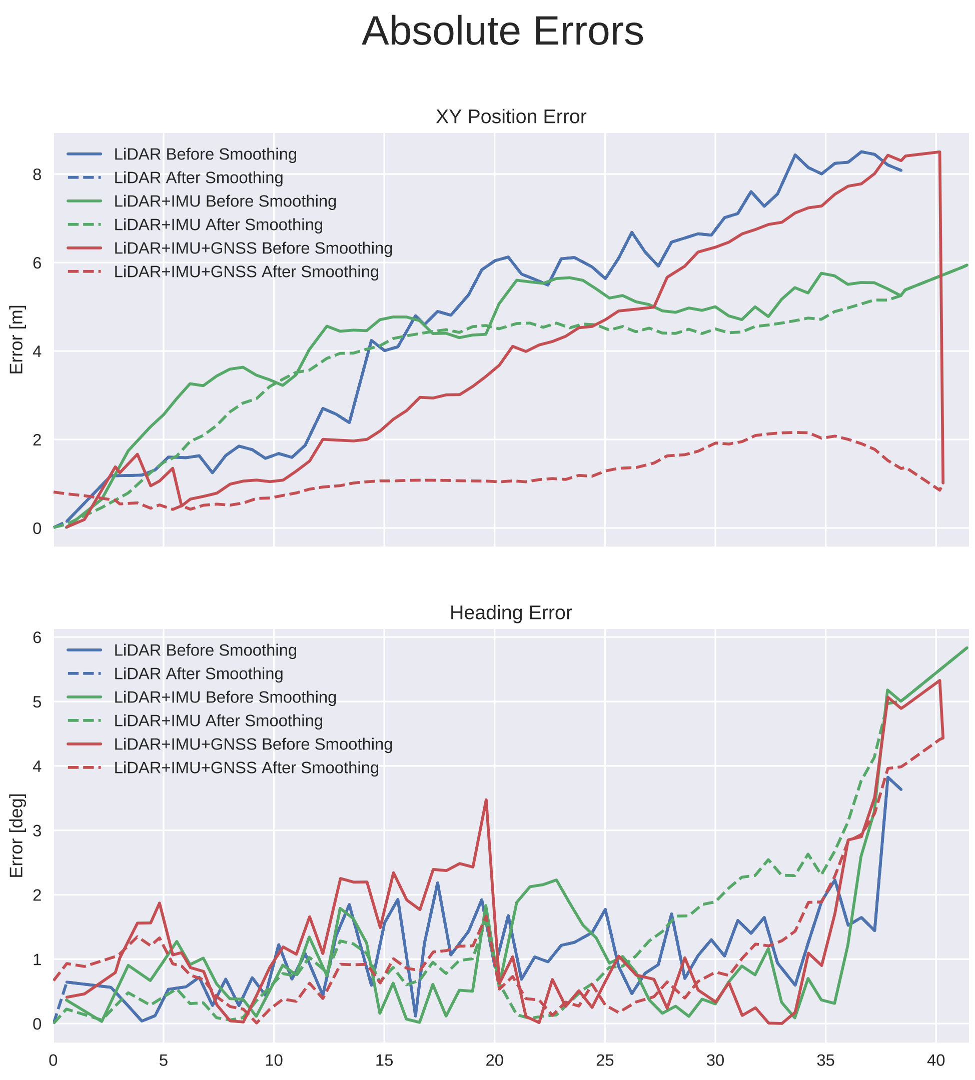


Figure 6.1: Sensor Comparison Experiment: The plot shows the absolute errors from the sensor comparisons experiments. Each experiment is plotted with the results before and after smoothing in the same color. The results before smoothing are plotted as a solid line, while the results after smoothing are plotted as a dashed line.

Notice in Figure 6.1 that the blue dotted line and the solid blue line lie on top of each other. The reason for this is that the smoothing does not have any effect on the trajectory relying only on the LiDAR sensor. Smoothing occurs due to the fusion of information from several sources. Thus, smoothing cannot occur if there is only one modality and there is no other form of estimates, such as a loop closure. That is the case for the LiDAR-only run. Furthermore, inspecting the trajectories before smoothing also shows that the trajectories do not differ significantly in estimation error, neither in XY position nor heading error. However, the smoothed trajectories appear to be less ‘spiky’ and generally provide lower error than before smoothing. The difference becomes evident when studying the trajectory with GNSS. After smoothing (the dashed line), the estimation system shows an approximately constant error at around 1m for the entire trajectory. The full system should outperform the others since the GNSS measurements would correct for any drift that accumulated through the tunnel. Table 5.2 confirms that the smoothing generally reduces the error. The largest contribution towards error reduction is shown in the LiDAR+IMU+GNSS experiment for the ‘North’ state, where the RMSE goes from 4.33m before smoothing to 1.03m after smoothing. This correction could be especially beneficial when including loop closures. Loop closures are relative, so a good estimate of the state before loop closure has been detected could also improve the estimation of the new state. Also notice that the inclusion of IMU decreases the RMSE of the ‘West’ state after smoothing, from 1.32m without IMU to 0.45m with IMU. This decrease indicates that the inclusion of the IMU and GNSS improves the estimation errors of the system, particularly after smoothing.

Although the system so far seems to be improving with the addition of more modalities, the RMSE of the heading state, ψ, gets noticeably worse. However, the comparison of the heading errors in Figure 6.1 shows that the errors behave somewhat likewise throughout the run. Towards the end, after the LiDAR-only estimation has stopped running, the LiDAR + IMU and the LiDAR + IMU + GNSS experiments further increase in heading error. A reason for this might be that the final LiDAR estimation is poor due to the map alignment being very sparse, which gives a bad initialization of the heading before leaving the tunnel. This error could be improved by adding a heading estimation based on the difference between two consecutive GNSS measurements. It could also be improved by introducing an uncertainty estimation based on ‘goodness-of-fit’ of the alignment, which then should make the poor alignments of the map more uncertain and thus trusted less in the estimation.

Figure 6.2 shows a comparison plot of all the NEESes in the plane. Notice that even though the XY-position NEES and the heading NEES lie inside the confidence bounds most of the time, the total planar NEES exceeds the bounds often in almost all the cases. This difference arises because studying the separated state spaces assumes that their covariances are decoupled. Heading and the XY position are closely linked, so this assumption is not fulfilled. The only line that does not follow this trend is the full LiDAR-Inertial SLAM system, which manages to lie inside the confidence bounds 87.9% of the time after smoothing, as shown in Figure

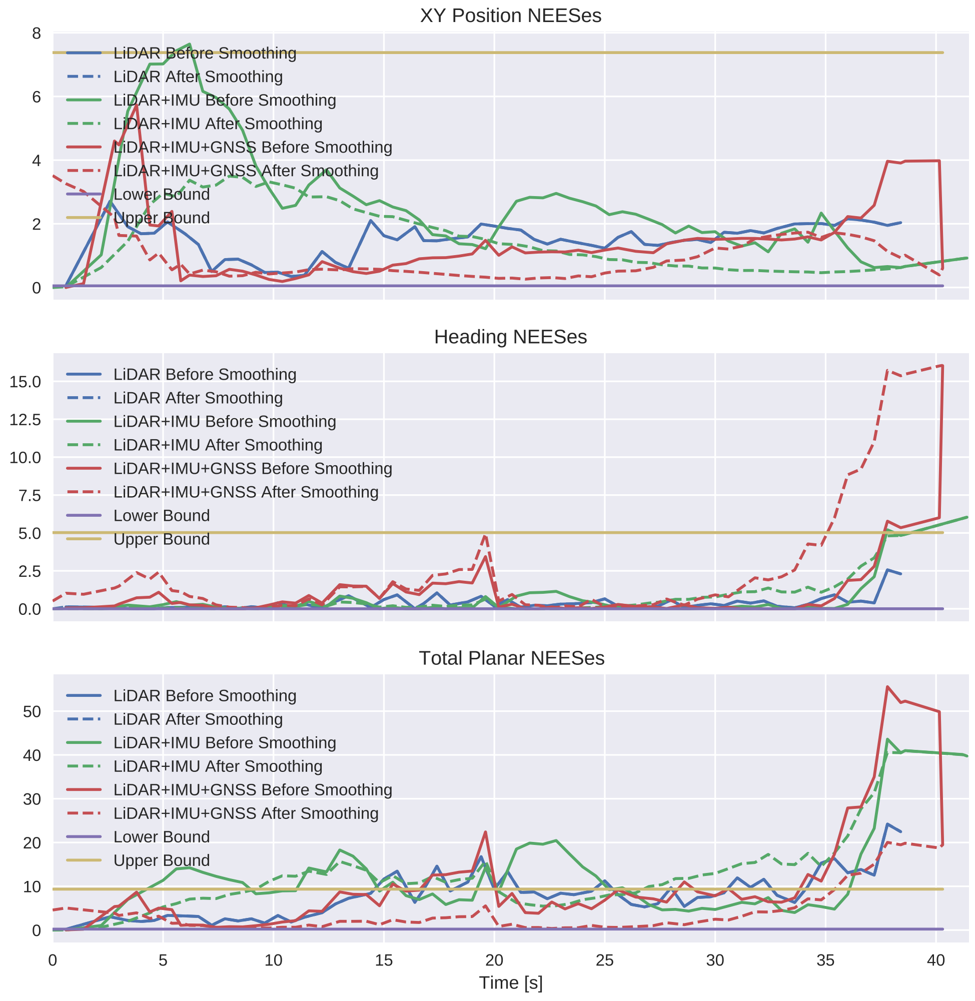


Figure 6.2: Sensor Comparison Experiment: The plot shows the development of the NEESes from the sensor comparisons experiments. Each experiment is plotted with the results before and after smoothing in the same color. The results before smoothing are plotted as a solid line, while the results after smoothing are plotted as a dashed line.

5.4c. Comparing this to Figure 5.4c shows that the trajectory after smoothing also provides a very low covariance. Thus, the NEESes are not low because the covariance explodes, and we can conclude that the estimations are close to the truth. Studying the full system error after smoothing (the red dashed line) in the upper plot in Figure 6.1 further confirms this statement. The ANEES displayed in Table 5.3 also confirms that the model is close to being consistent. It makes sense that the full system is the one that provides the 'best' results in terms of consistency since GNSS measurements are rooted directly in the global frame. However, as mentioned earlier, the system does not estimate heading based on consecutive measurements. GNSS only measures position, but in itself, it does not provide any orientation information. Therefore, the NEES of the heading does not improve after a GNSS measurement is received. This may be a reason why the total ANEES in Table 5.3 still lies outside the confidence interval.

During the experiments, a source of error was found to be the occlusion of parts the scene during the alignment. The system had a tendency to provide poor estimates in these situations, especially when relying on LiDAR-only. A reason for these types of errors probably comes from the fact that the LiDAR-odometry estimation is based on point features. Modern LiDAR odometry systems, like LOAM, use a combination of line features and planar features, which in turn could make the system more robust in general. However, for the experiments with more sensors, these errors are not as apparent. This indicates that the custom alignment algorithm incorporating IMU priors adjusts for this, making the system more robust.

6.2 Loop Closure

The intention of the loop closure experiment shown in Section 5.2 is to show that a system including loop closure will help improve the global consistency and the performance of the position estimation when the road roller has to perform continuous operation or drive in and out the tunnel. This section will further discuss the different factors that influenced the results and how to improve the system.

Figure 5.6 shows the estimated position without and with loop closure. The tunnel spans the area from roughly around 20m to 220m in the North direction. Comparing Figure 5.6a and Figure 5.6b clearly shows the effects of not having loop closure. In Figure 5.6a we see that the vehicle on its way back starts driving through a location it earlier mapped as a tunnel. The final correction in Figure 5.6a comes from the GNSS measurement arriving. In addition to recognizing the location as seen earlier and correcting, Figure 5.6b shows that the covariance ellipsoid towards the end of the round trip is a lot smaller before smoothing than the covariance ellipsoid when no loop closure is made. They seem to be very small for both round trips after smoothing, but this is because the GNSS measurement significantly reduces the covariances.

Comparing the observation made in Figure 5.6 to Table 5.6 shows that even though the covariances are smaller, the ANEESes are significantly smaller before smoothing. This suggests that the estimations are indeed more correct, which can be confirmed by studying Table 5.5.

Figures 5.7 and 5.8 can be studied to see where the full LiDAR SLAM system benefits from the loop closure, and where the challenges of including loop closures lie. Figure 5.8 shows that the system with loop closure drastically outperforms the system without loop closure in terms of heading consistency. The system manages a heading NEES inside the 95% confidence bounds over 90% of the time before and after smoothing. The spike in the heading error in Figure 5.7 comes from the system re-initializing the estimates because structure appears after having disappeared. These estimates take some time to stabilize. This instability causes a spike in the estimate, and thus in the NEES as well. This event shows that LiDAR motion suffers from unstable estimates when there is little structure. However, the system with loop closure then recognizes a previously viewed location and quickly goes to zero. On the other hand, the one without loop closure does not, and a steady-state error in the heading is visible throughout the second passing through the tunnel. As a result, the XY-position error will also rise much higher since the vehicle estimates its driving in a completely different direction.

The loop closure generally gives a better estimate of the heading and position for the system with a lower covariance, according to Tables 5.5 and 5.6. The same can be seen by studying the Figure 5.8. However, Figure 5.7b shows an undesirable effect that only occurs when activating the loop closure module. In the first run through the tunnel, the figure shows that the error after smoothing peaks in the middle of the tunnel and with a peak similar to the peak before smoothing. Without loop closure, the peak in error after smoothing is significantly smaller. A reason for this may be the search point scaling in the loop closure detection algorithm suggested in Section 4.3.1. The loop closures connect differences in pose between the different key poses. Since the loop closure is connected through the poses and not the scene it views, it may result in a compromise between the current pose and the historic pose. This could be the reason for the 'wave'-like behavior in the top plot of Figure 5.7b, which also reflects in the NEES plot in Figure 5.8b.

Although the loop closure seems to have a desirable effect when tested on the simulated data set, it must be mentioned that the naive loop closure algorithm implemented is based on the current estimate. Therefore it will not be able to find any loop closures if there is too much drift or for some other reason has provided a very poor estimate of its location. This case could arise, for example, in a very long tunnel where the vehicle does not exit and receive GNSS measurements. Additionally, this type of loop closure algorithm requires significant overhead and tuning of several hyperparameters, such as `history search radius`, `fitness tolerance` and `history keyframe extraction number`. The algorithm is, in fact, very susceptible to these parameters, especially the fitness tolerance, which decides how good an alignment must 'fit' before a loop closure is decided. These parameters are typically different based on the use-case and the environment it is used in. Therefore, this solution is not optimal for an autonomous system that should operate inside any environment.

The Norwegian standards for documentation of road construction work [3, p. 185] (translated to english) states that

> The extent and scope of the compaction work shall be located in the horizontal plane by means of GNSS or other forms of dynamic positioning with satisfactory accuracy. The requirement for the location

> also applies in tunnels and other areas with a lack of satellite coverage. The equipment must be able to store compression and positioning data and have a system for transferring the results to a central database. The location must have an accuracy of $\pm 0.2m$ or better.

The quote can be seen as two separate requirements. Number one is the ability to store compressed positioning data and store it in a central database. This is something any SLAM system should be able to do and is therefore fulfilled. The second part states that it has minimum requirements for the accuracy of the estimations. This requirement shows that the estimation system needs significant improvements before being realized in the real world. The results in Table 5.5 showed that the RMSE is most extensive in the X-direction, reaching $\pm 3.07m$ for the complete system. Thus the main point of improvement lies in the along-track error. Even so, the RMSE in the Y-direction was $\pm 0.55m$ for the complete system and does therefore not lie inside the requirements neither.

Chapter 7

Conclusion and Further Work

The first part of the thesis was to extend the simulator for the problem of tunnel navigation using slowly-moving ground vehicles to include an IMU and a GNSS. The simulation environment proved to yield a feature-rich environment where the navigation module could be tested, as seen in Figure 4.1.

The sensor comparison experiment in Section 5.1 shows that the LiDAR can be successfully combined with more modalities to provide a better estimation of the pose. The LiDAR-Inertial SLAM system showed that it could provide consistent estimates of its trajectory after syncing with GNSS when leaving the tunnel. However, the total planar ANEESes in Table 5.3 showed that the system underestimates the couplings between the planar states, and further work is required to model the dynamics better.

The loop closure experiment showed that loop closures helps reducing the RMSEs and the ANEESes of the system, as shown in Tables 5.5 and 5.6. This joint reduction indicates that the loop closure is beneficial and provides better estimates for the pose of the road roller when re-visiting a previously visited scene. Since the road roller is required to perform several *passings*, the inclusion of loop closure seems beneficial for an autonomous road roller.

The main goal of the thesis was to analyze the performance of a LiDAR-inertial SLAM system in a simulated environment. The LiDAR-inertial SLAM system shows improved results in terms of estimation error when adding more modalities and loop closures. It does, however, struggle to achieve an uncertainty estimate that coincides with the estimation error, especially after smoothing. The system did not comply with the documentation requirements in the standards for Norwegian road construction. However, despite the strict requirements, I believe that a LiDAR-Inertial SLAM system could provide a solution capable of complying with these requirements.

7.1 Future Work

Although the proposed system seems to perform decently on the simulated data, many improvements must be made before implementing this system in full scale. Therefore, this section will focus on the next step towards a full-scale implementation of the system.

7.1.1 Testing

Although the full LiDAR SLAM system seemed to perform well on the entire trajectory, the system needs to be qualitative testing on a real-world data set to validate that the simulation provides realistic results. When deemed sufficiently accurate, the navigation module should then be tested in unison with a validated control module to provide further insight into improvements and challenges with the current system.

Due to the lack of known datasets for autonomous ground-vehicle tunnel navigation combining IMU, GNSS and LiDARs, the simulator data has not been compared to real data sets. This comparison is required to check the validity of the simulations.

7.1.2 Line Features

Systems such as LOAM and all systems that extend it use the line features and planar features suggested in [8]. Since a source of error during the experiments was found to be poor matching when part of the scene is occluded, it may be beneficial to implement a front-end more similar to LOAM to make the system more robust.

7.1.3 Uncertainty Estimation

The suggested LiDAR odometry estimation produces volatile estimates when the system observes few points. A way to mitigate this is to increase the covariance in these situations, thus 'trusting' the IMU measurements more. For this to work, the system needs to provide an estimate of its covariance. An approach was tried using the spread of the points in the point cloud after alignment in the odometry estimation module to yield an estimate of the covariance in $\Delta x, \Delta y, \Delta z$. However, this did not have the desired effect and was removed from the final implementation. A significant improvement would be to improve the uncertainty estimation, which would make the fusion of the sensor information more optimal.

7.1.4 Improved Loop Closures

As mentioned in Section 4.3.1, the system uses a naive approach to detecting loop closures. A huge advantage of loop closures is that it is possible to correct for drift after a long operation time. Since the suggested approach is dependent on the positional estimate, it cannot guarantee to even search for loop closures if the drift is large enough. Therefore, to fully exploit the strengths that loop closures provide, the system could implement a loop closure detection algorithm similar to that of [32] which provides place recognition based on segments.

However, it is worth noting that a challenge may arise when testing such a detector in the simulated environment because simulated objects may appear alike.

7.1.5 Exploit Situational Awareness

The current navigation system builds a map of its surroundings. A natural next step of development for the system would be to use this map and the incoming LiDAR measurements to produce guidance commands for the complete system. From the map and LiDAR measurements, it is possible to produce an occupancy grid and produce an optimal plan for traversing the tunnel and compacting the concrete as efficiently as possible while avoiding obstacles.

Bibliography

[1] S. G. Wroldsen, *SLAM-Based Tunnel Navigation System for Autonomous Ground Vehicles*, 20th Dec. 2020.

[2] *Autonomous Road Roller Project - A cooperation between Sintef, Semcon and AF Gruppen*, https://www.sintef.no/prosjekter/2019/autonom-vals/, Accessed: 2021-05-15.

[3] S. Vegvesen, *Norwegian standard manual for road construction*, 2018. [Online]. Available: https://www.vegvesen.no/_attachment/2364236/binary/1269980.

[4] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. J. Leonard and F. Dellaert, 'ISAM2: Incremental Smoothing and Mapping Using the Bayes Tree,' *Int. J. Rob. Res.*, vol. 31, no. 2, pp. 216–235, Feb. 2012, ISSN: 0278-3649. DOI: 10.1177/0278364911430419. [Online]. Available: https://doi.org/10.1177/0278364911430419.

[5] T. Shan and B. Englot, 'LeGO-LOAM: Lightweight and Ground-Optimized Lidar Odometry and Mapping on Variable Terrain,' in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2018, pp. 4758–4765.

[6] 'State Estimation in Discrete-Time Linear Dynamic Systems,' in *Estimation with Applications to Tracking and Navigation*. John Wiley Sons, Ltd, ch. 5, pp. 199–266, ISBN: 9780471221272. DOI: https://doi.org/10.1002/0471221279.ch5. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/0471221279.ch5. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/0471221279.ch5.

[7] A. W. Fitzgibbon, 'Robust registration of 2d and 3d point sets,' *Image and Vision Computing*, vol. 21, no. 13, pp. 1145–1153, 2003, British Machine Vision Computing 2001, ISSN: 0262-8856. DOI: https://doi.org/10.1016/j.imavis.2003.09.004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0262885603001835.

[8] J. Zhang and S. Singh, 'LOAM: Lidar Odometry and Mapping in Real-time,' in *Robotics: Science and Systems*, vol. 2, 2014.

[9] K. Yousif, A. Bab-Hadiashar and R. Hoseinnezhad, 'An Overview to Visual Odometry and Visual SLAM: Applications to Mobile Robotics,' *Intelligent Industrial Systems*, vol. 1, Nov. 2015. DOI: `10.1007/s40903-015-0032-7`.

[10] R. Mur-Artal, J. Montiel and J. Tardos, 'ORB-SLAM: a versatile and accurate monocular SLAM system,' *IEEE Transactions on Robotics*, vol. 31, pp. 1147–1163, Oct. 2015. DOI: `10.1109/TRO.2015.2463671`.

[11] J. Engel, T. Schöps and D. Cremers, 'LSD-SLAM: Large-scale direct monocular SLAM,' in *European Conference on Computer Vision (ECCV)*, Sep. 2014.

[12] G. Klein and D. Murray, 'Parallel tracking and mapping for small AR workspaces,' in *Proc. Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'07)*, Nara, Japan, Nov. 2007.

[13] C. Debeunne and D. Vivet, 'A Review of Visual-LiDAR Fusion based Simultaneous Localization and Mapping,' *Sensors*, vol. 20, no. 7, 2020, ISSN: 1424-8220. DOI: `10.3390/s20072068`. [Online]. Available: `https://www.mdpi.com/1424-8220/20/7/2068`.

[14] S. Thrun, M. Montemerlo, D. Koller, B. Wegbreit, J. Nieto and E. Nebot, 'Fastslam: An efficient solution to the simultaneous localization and mapping problem with unknown data,' *Journal of Machine Learning Research*, vol. 4, May 2004.

[15] Z. Lu, Z. Hu and K. Uchimura, 'Slam estimation in dynamic outdoor environments: A review,' in *Intelligent Robotics and Applications*, M. Xie, Y. Xiong, C. Xiong, H. Liu and Z. Hu, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 255–267, ISBN: 978-3-642-10817-4.

[16] V. Indelman, S. Williams, M. Kaess and F. Dellaert, 'Information fusion in navigation systems via factor graph based incremental smoothing,' *Robotics and Autonomous Systems*, vol. 61, no. 8, pp. 721–738, 2013, ISSN: 0921-8890. DOI: `https://doi.org/10.1016/j.robot.2013.05.001`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S092188901300081X`.

[17] T. Shan, B. Englot, D. Meyers, W. Wang, C. Ratti and R. Daniela, 'LIO-SAM: Tightly-coupled Lidar Inertial Odometry via Smoothing and Mapping,' in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2020, pp. 5135–5142.

[18] C. Forster, L. Carlone, F. Dellaert and D. Scaramuzza, 'IMU Preintegration on Manifold for Efficient Visual-Inertial Maximum-a-Posteriori Estimation,' Jul. 2015. DOI: `10.15607/RSS.2015.XI.006`.

[19] J. González Agudelo, 'Contribution to the model and navigation control of an autonomous underwater vehicle,' Ph.D. dissertation, Jul. 2015.

[20] *Wikipedia commons*, `https://commons.wikimedia.org/wiki/`, Accessed: 2021-05-11.

[21] Y. Zhong, 'Intrinsic shape signatures: A shape descriptor for 3D object recognition,' in *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*, 2009, pp. 689–696. DOI: `10.1109/ICCVW.2009.5457637`.

[22] S. Salti, F. Tombari and L. D. Stefano, 'A performance evaluation of 3d keypoint detectors,' in *2011 International Conference on 3D Imaging, Modeling, Processing, Visualization and Transmission*, 2011, pp. 236–243. DOI: `10.1109/3DIMPVT.2011.37`.

[23] R. B. Rusu, N. Blodow and M. Beetz, 'Fast point feature histograms (FPFH) for 3D registration,' in *2009 IEEE international conference on robotics and automation*, IEEE, 2009, pp. 3212–3217.

[24] V. Indelman, S. Williams, M. Kaess and F. Dellaert, 'Factor graph based incremental smoothing in inertial navigation systems,' in *2012 15th International Conference on Information Fusion*, 2012, pp. 2154–2161.

[25] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid and J. Leonard, 'Past, Present, and Future of Simultaneous Localization And Mapping: Towards the Robust-Perception Age,' *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1309–1332, 2016.

[26] S. G. Wroldsen. (2021). 'Code: LiDAR-Inertial Slam System For Tunnel Navigation of an Autonomous Road Roller,' [Online]. Available: `https://github.com/Sjurinho/ttk4900` (visited on 04/06/2021).

[27] Epic Games, *Unreal Engine*, version 4.23, 30th Jan. 2019. [Online]. Available: `https://www.unrealengine.com/en-US/`.

[28] Stanford Artificial Intelligence Laboratory et al., *Robotic Operating System*, version ROS Melodic Morenia, 23rd May 2018. [Online]. Available: `https://www.ros.org`.

[29] F. Dellaert, 'Factor graphs and GTSAM: A hands-on introduction,' Georgia Institute of Technology, Tech. Rep., 2012.

[30] R. B. Rusu and S. Cousins, '3D is here: Point Cloud Library (PCL),' in *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 2011.

[31] M. Aqel, M. H. Marhaban, M. I. Saripan and N. Ismail, 'Review of visual odometry: types, approaches, challenges, and applications,' *SpringerPlus*, vol. 5, Dec. 2016. DOI: `10.1186/s40064-016-3573-7`.

[32] R. Dubé, D. Dugas, E. Stumm, J. Nieto, R. Siegwart and C. Cadena, 'Segmatch: Segment based place recognition in 3d point clouds,' in *International Conference on Robotics and Automation (ICRA)*, IEEE, 2017, pp. 5266–5272.

Appendix A

Hyperparameter Tuning

A.1 Feature Association Hyperparameters

Table A.1: Feature Extraction Hyperparameters

Parameter	Value
Voxelgrid Leaf Size	0.2
Normal Radius	0.5
ISS Salient Radius	1
ISS Non-Max Radius	0.6
ISS Saliency Threshold	0.8
Minimum Number Of Features	30

Table A.2: Map Alignment Hyperparameters

Parameter	Value
Max Iterations	100
Function Value Tolerance	0.05
Step Size Tolerance	10^{-8}

A.2 Keyframe Selection Hyperparameters

Table A.3: Keyframe Selection Hyperparameters

Parameter	Value
Keyframe Save Distance	3
GNSS Keyframes	True
Loop Closure Keyframes	True

A.3 Loop Closure Hyperparameters

Table A.4: Loop Closure Hyperparameters

Parameter	**Value**
History Keyframe Search Radius	100
History Keyframe Search Number	10
Minimum Time Passed	60
Minimum Keyframe Fitness Score	0.6

Appendix B

ROS Graph

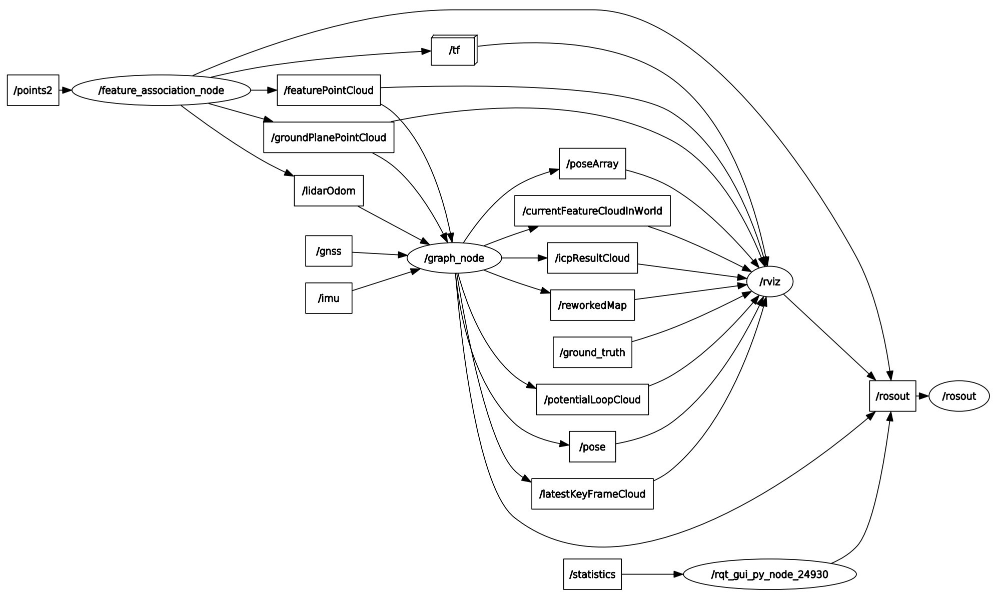


Figure B.1: Overview of ROS Communication

Appendix C

Simulink Block Diagram

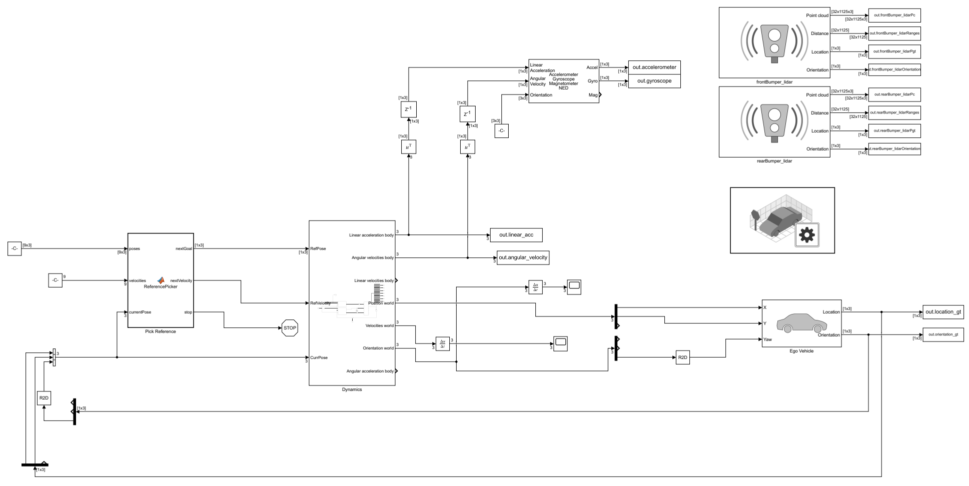


Figure C.1: The Simulink used to perform the communication between Unreal Engine. Data is then exported to the MATLAB workspace.

Appendix D

Simulation Environment

D.1 Screenshots of Simulation Environment

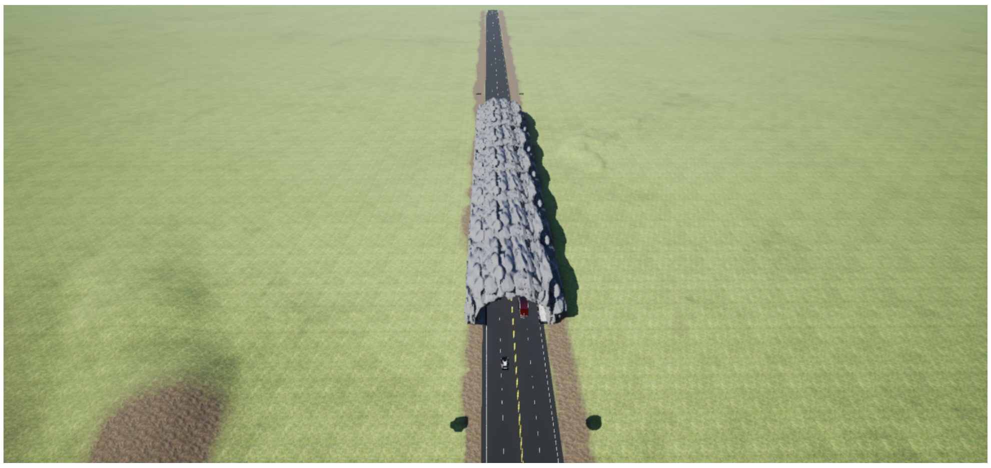


Figure D.1: Birds-eye-view of simulation environment.

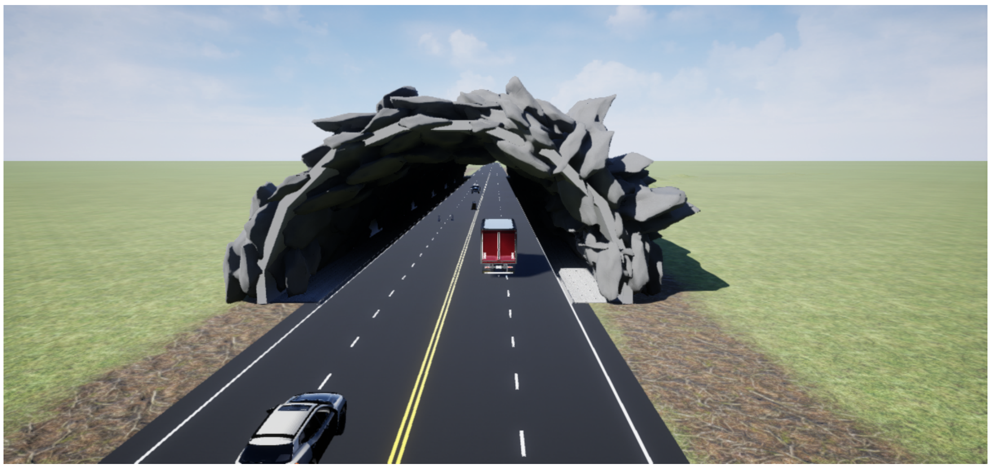


Figure D.2: The tunnel seen from the start. The red trailer is the moving actor.

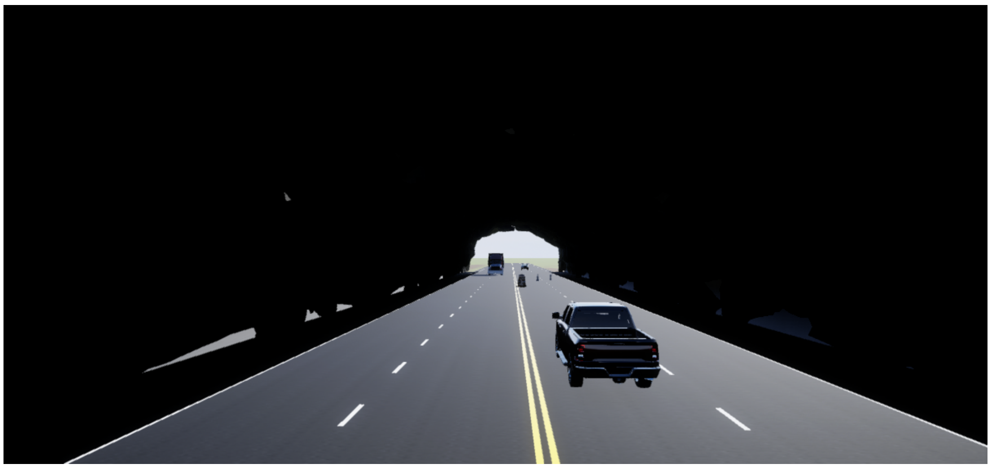


Figure D.3: Props are placed inside the tunnel.