FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Improving Lidar Odometry and Mapping in Real-time using Inertial Measurements

Elias Maia Azevedo Dias Ferreira

DISSERTATION

Master in Electrical and Computers Engineering

Advisor: António Pedro Aguiar

Advisor: Piotr Skrzypczyński

October 30, 2021

Abstract

The design of truly autonomous mobile robots is a very complex task, starting from the navigation to the motion planning and control. The navigation is generally regarded as one of the most important issues. Without a good perception of the vehicle's surroundings and localization, the other tasks would perform poorly. In order to solve this problem, these autonomous agents are typically aided with SLAM state estimation techniques. There are many different types of algorithms, for which there is no single best solution. Recent improvements in LiDAR technology brought more powerful and accessible laser scanners than ever before. Together with its key advantage of being less insensitive to ambient lighting and optical texture in the scenes, allowed the application of LiDAR SLAM technology in a broad field of robotics. This technology is particularly useful in unstructured environments or where the use of GPS is unreliable, contributing to a better performance than purely based dead-reckoning strategies.

Motivated by the above, the goal of this thesis is to understand, implement and improve a recent 3D SLAM navigation system based on Light Detection And Ranging (LiDAR) data by adding inertial measurements to determine the position and orientation with six degrees of freedom (6-DOF), known as pose, hence creating, in the real-time a 3D map of the surroundings with application to autonomous navigation. To this end, the LiDAR Odometry and Mapping (LOAM) algorithm was chosen, since it deals with the problems stated before, it is open source and compatible with the Robot Operating System (ROS)- the chosen development platform. The LOAM system receives as an input a 3D point cloud originated from a 3D laser scanner and then divides the work load into two different algorithms. The LiDAR odometry, that runs at fast speed, in order to be able to estimate velocity and simultaneously remove motion distortion in the point clouds in real time. However, it comes with a cost- since the odometry works with fast speed and low amount of points, it has low-fidelity and cannot ensure accurate mapping. For this purpose, it is where the LiDAR mapping enters. It works at slower speed, around a tenth of the speed of the odometry. It receives the undistorted point clouds and performs fine scan matching, with a big amount of points to ensure accuracy on the map. The combination of both algorithms ensures an efficient real time performance.

This thesis starts by gathering the necessary background literature, to introduce the relevant theoretical concepts. Afterwards, the problem to be solved is exposed followed by a careful explanation of the LOAM algorithms. Subsequently, the original LOAM system is implemented, explained and tested, followed by the proposed improvements. These improvements consist in the fusion of the inertial measurements, from an IMU sensor with a complementary filter, to obtain 3D orientation. Then, the new orientation is fused with the laser odometry pose estimate, utilizing an extended Kalman filter. This improved pose estimate is fed to the mapping algorithm. To finalize, a GPS estimate is used as ground truth to validate the results.

Overall, the previously stated objectives were accomplished. The obtained results were positive and further improvements were pointed for a future work.

Resumo

A concepção de robôs autónomos é uma tarefa muito complexa, desde a navegação até ao planeamento e controlo do movimento. A navegação é geralmente considerada como uma das questões mais importantes. Sem uma boa percepção do ambiente e da localização do veículo, as outras tarefas não poderão ter um bom desempenho. Para resolver este problema, estes agentes autónomos estão tipicamente equipados com técnicas de estimação de estado SLAM. Existem muitos tipos diferentes de algoritmos, para os quais não existe uma única melhor solução. As recentes melhorias na tecnologia LiDAR trouxeram scanners laser mais potentes e acessíveis do que nunca. Juntamente com a sua principal vantagem de serem mais robustos relativamente à iluminação ambiente e à textura óptica dos cenários, permitiu a aplicação da tecnologia LiDAR SLAM num vasto campo da robótica. Esta tecnologia é particularmente útil em ambientes não estruturados ou onde a utilização de GPS não é fiável, contribuindo para um melhor desempenho do que as estratégias puramente baseadas em *dead-reckoning*.

Motivado pelo acima exposto, o objectivo desta tese é compreender, implementar e melhorar um sistema de navegação SLAM 3D actual, baseado em dados de *Light Detection And Ranging* (LiDAR), adicionando medições inerciais para determinar a posição e orientação com seis graus de liberdade (6-DOF), conhecida como *pose*. Além disso, um dos objectivos é criar, em tempo real, um mapa 3D do ambiente com aplicações à navegação autónoma. Para este fim, foi escolhido sistema *LiDAR Odometry and Mapping* (LOAM) e *Robot Operating System* (ROS) como plataforma de desenvolvimento. O LOAM recebe como entrada, uma nuvem de pontos 3D proveniente de um laser scanner 3D e depois divide o processamento em dois algoritmos diferentes. A *LiDAR odometry*, que funciona com uma velocidade rápida, de modo a poder estimar a velocidade e simultaneamente remover a distorção do movimento nas nuvens de pontos. No entanto, isto acarreta um custo, uma vez que a *LiDAR odometry* funciona com velocidade rápida e baixa quantidade de pontos, tem baixa fidelidade e não pode assegurar um mapeamento preciso. Para este fim, é introduzido o *LiDAR mapping*. Funciona com uma velocidade lenta, cerca de um décimo da velocidade da *LiDAR odometry*. Recebe as nuvens de pontos não distorcidas e efectua uma correspondência precisa, com uma grande quantidade de pontos, assegurando assim a precisão do mapa. A combinação de ambos os algoritmos assegura um desempenho eficiente em tempo real.

Esta tese começa pela análise da literatura necessária. Posteriormente, o problema a ser resolvido é exposto, seguido de uma explicação profunda do sistema LOAM. Depois, o LOAM original é implementado, explicado e testado, seguido das melhorias propostas. Estes melhorias consistem na fusão das medições inerciais, a partir de um sensor IMU com um filtro complementar, para obter uma orientação 3D. Em seguida, a nova orientação é fundida com a estimativa da pose da *LiDAR odometry*, utilizando um filtro de Kalman estendido. Esta estimativa melhorada da *pose* é alimentada ao algoritmo *LiDAR mapping*. Para finalizar, é utilizada uma estimativa GPS como *ground truth* para validar os resultados. Em geral, os objetivos anteriormente expostos foram alcançados. Os resultados obtidos foram positivos e foram apontadas melhorias para trabalho futuro.

Acknowledgements

My deepest gratitude goes to my advisor Professor Piotr Skrzypczyński for offering me the possibility of writing my Master thesis at the Poznan University of Technology. During these months he supported me continuously with his knowledge and inspiring suggestions. Also, special thanks to the PhD student Krzysztof Ćwian that was always available to help. Together, they made the work at Poznan a great and very enriching experience.

I am also very thankful to my advisor Professor António Pedro Aguiar from The Faculty of Engineering of the University of Porto. Without his assistance during the preparation of the thesis, I would not have been able to come to Poland.

I would also like to thank the amazing people at the C2SR-Lab, who always brought and provided kindness and a comfortable working environment.

I am very grateful to Dominika Skibińska for all the emotional support and for taking care of me every step of the way.

I would also like to thank my friend Francisco Neves. He is a great friend and is always available to help in any field.

Finally, I would like to thank my family, for everything: for my education, privileges, love, advice, and support.

Elias Maia Azevedo Dias Ferreira

"When something is important enough,
you do it even if the odds are not in your favor."

Elon Musk

Contents

List of Figures

List of Tables

Abbreviations and Symbols

SLAM Simultaneous localization and mapping

GPS Global Positioning System

LiDAR Light Detection And Ranging

6-DOF six degrees of freedom

ROS Robot Operating System

PUT Poznan University of Technology

C2SR Cyber-Physical Control Systems and Robotics lab

LOAM Lidar Odometry And Mapping

ROS Robot Operating System

IMU Inertial measurement unit

EKF Extended kalman Filter

Chapter 1

Introduction

In the field of robotics, the construction of truly autonomous mobile robots is a very complex task starting from the navigation to the motion planing and control. The navigation is generally regarded as one of the most important issues. Without a good perception of the vehicle's surroundings and localization the other tasks will perform poorly. One of the key enabling technologies for this purpose is the Simultaneous localization and mapping (SLAM) [79]. SLAM is the computational problem of building a map of an unknown environment by a mobile robot while navigating the environment using the map that is currently being updated. The SLAM problem is divided into two main components: localization and mapping. There are many different types of SLAM algorithms, for which there is no single best solution, The chosen method should take in consideration the requirements of the specific application such as; nature of the features in the map, resolution, time constrains, sensors being used, computation power, etc. One popular type in recent years has been the visual SLAM [78], which uses visual data from sensors such as RGB or RGB-D (Kinect) [6] cameras for its main source of information, to obtain rich 3D maps, position and orientation with six degrees of freedom (6-DOF). In opposite to previous case in certain applications there is the need to reduce the complexity of the system, where 2D SLAM[21] is used and in this case the problem is simplified to the movement in a 2D plane. Commonly 2D Laser scanners are used for this purpose.

The objective of this thesis is to addresses the SLAM problem exploiting 3D SLAM algorithms that process 3D laser scanner's data for the navigation of mobile robots. The Sequel, 1.1 highlights the motivation for investigating this problem. Section 1.2 states the main objectives and contributions. Section 2.1 describes the sensors and coordinate systems. Finally, Section 1.3 addresses the structure of the rest of this document.

1.1 Context and Motivation

The motivation for the exploration of this field comes mainly from the growing use of autonomous agents in modern day robotics, which many times aided with SLAM state estimation techniques. In parallel, the recent improvements in LiDAR technology brought more powerful and accessible laser scanners than ever before. Together with its key advantage of being insensitive to ambient lighting and optical texture in the scenes, allowed the application of LiDAR SLAM technology in a broad field of robotics. We can note this technology is particularly useful in unstructured environments or where the use of GPS is unreliable, performs better than purely based dead-reckoning strategies.

These applications can start from simple domestic vacuum cleaners robots as: shown in Figure 1.1. This model comes equipped with a 2D laser scanner and with the adding of SLAM, makes this autonomous cleaner more efficient in terms of battery life and navigation abilities. The figure 1.2 shows an example of an ground agricultural robot, developed by INESCTEC. As opposite of the previous indoor scenario, agriculture environments are much more complex and challenging. They tend to be less structured, more dynamic and sensors need to endure adverse conditions. Crop monitoring and harvesting requires robust and accurate sensing, perception and interpretation. This robot uses a GNSS-free (Global Navigation Satellite System) to navigate trough steep slope vineyards, due to its challenge conditions, a special SLAM approach was developed VineSLAM, which is a hybrid system where LiDAR SLAM is used in combination with Visual SLAM to offer extra redundancy and increase reliability.

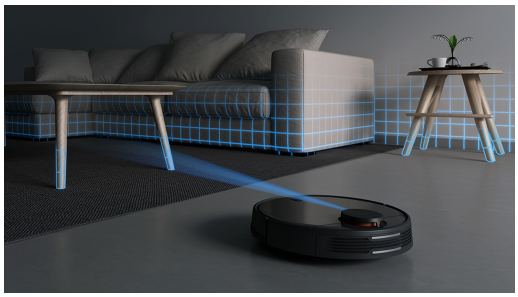

Figure 1.1: Mi Robot Vacuum Cleaner Mop Pro. [4].

Figure 1.2: Agricultural robotic platform from INESC TEC for steep slopes vineyards. [50].

The previous example gave us the understanding that in more complex environments the use of hybrid sensor systems can be more advantageous, by producing more robust results. This is where the second part of this work comes in. It consists in the exploration of sensor fusion, where we take in account that in different conditions, one sensor can perform better than other and consequently we can give it a bigger trust. The fused result achieves a better overall result than the use of both sensors alone. In this work, it was decided to introduce inertial measurements to one of the works that will be described in the Chapter 4. This motivation comes from the fact that the latest developments in Micro-Electro-Mechanical Systems, or MEMS [44] made this kind

of sensors cheaper and compacter than ever. Creating the interest to evaluate if a loose coupling of this measurements with a system that wasn't designed originally with them could be upgraded to achieve a better performance at low cost and without a complex implementation. Finally for the implementation of this work, the ROS framework was chosen, due to its open-source set of software libraries and tools. It is specifically designed for robotics's applications with its ever growing community.

1.2 Objectives and Contributions

The final goal of this thesis is to understand, exploit and improve recent 3D SLAM navigation systems based on Light Detection And Ranging (LiDAR) data by adding inertial measurements to determine the position and orientation (Pose) with six degrees of freedom (6DOF), hence creating, in real-time, a 3D map of the surroundings with application to autonomous navigation. For this purpose a selection of algorithms will be chosen and implemented in C++/Python using the Robot Operating System (ROS) framework to be further tested on data recorded from a real wheeled vehicle 5.4.

These algorithms can be divided in different parts, starting by;

- The 3D LiDAR SLAM- necessary to deal with point cloud registration, odometry and mapping. The LOAM algorithm 4.2 was chosen for this purpose since it deals with the problems stated before, it's open source and compatible with ROS. The theory and the details can be found in the 4 paragraph.

- State estimation- that will be used to perform the sensor fusion between IMU odometry and LiDAR odometry. An extended Kalman filter (EKF) was chosen for this purpose, since it meets the requirements of our application and also its availability on the ROS platform. The theory can be found in the 3 section and implementations details on 5 section.

- IMU filtering and fusion, inertial measurements are typically noisy, hence there's the need to combine the accelerometer and gyroscope data to obtain the angular position with minimized noise. For this purpose a Complementary filter was chosen due to its simplicity, good performance and also its availability on the ROS platform. The theory and details can be found in the 5 section.

- GPS odometry- in this work GPS will be used as ground truth to validate the estimated trajectory. To accomplish this, the raw messages from the GPS need to be converted from their original standard form to the local ROS frame, to be matched using the evaluation algorithm. The theory and implementation details can be found in the 2.1.1 and 5 sections.

- Evaluation- finally, in order to properly evaluate the results, the Absolute trajectory error (ATE) metrics was chosen. The ATE is well-suited for measuring the performance of visual/LiDAR SLAM systems. The theory and details can be found in the 5 section.

The work for this dissertation was conducted in two different places starting at FEUP in the first semester and at Poznan University of Technology (PUT) on the second semester. The work load was partitioned in two distinct phases:

- **Literature review and field tests at FEUP:** For the first semester, the goal was to become familiar with the fundamentals of SLAM (in this case 2D LiDAR SLAM) and with the software to be used throughout this dissertation, namely ROS. To accomplish this, some field tests were conducted using a real world robot Details about this tests can be found in the PDI report [39].
- **Practical implementation and evaluation at Poznan:** On the second semester, most of the work presented on this thesis was accomplished. Consisted of the study and implementation of 3D LiDAR SLAM, namely Lidar Odometry And Mapping (LOAM) 4.2. And the proposed sensor fusion algorithms: EKF 3 and Complementary filter 5.3.1, necessary to address the inertial measurements. These previously assessed techniques were evaluated with field recorded data at Poznan campus. Finally, a critical analysis of the obtained solutions to solve the studied and implemented navigation problem were conducted. The last step was the writing of this document.

1.3 Organization of the Dissertation

The remainder of this dissertation is organized as follows:

Chapter 2: Presents the theory behind the sensors, coordinate systems and necessary coordinate transformations to perform between them.

Chapter 3: Introduces the chosen state estimation method, giving us the necessary theoretical knowledge to understand and implement the Extended Kalman Filter;

Chapter 4: Introduces the 3D SLAM problem , describes the theory behind the LOAM algorithm;

Chapter 5: Dives into the practical side of this dissertation exposing the original system overview and developed solution. Describes its implementation using the ROS framework and the required hardware and software components. As next step, exposes the implementation of the sensor fusion component's. Namely the Extended Kalman Filter and Complementary filter; Finalizing by resenting the chosen Evaluation metrics and results;

Chapter 6: Provides the overall conclusions and possible future research directions.

Chapter 2

Sensors and coordinate systems

Autonomous vehicles can be equipped with a broad variety of sensor systems. Each particular sensor can have its own specific norms and coordinate systems. This must be carefully taken into account, to make sure all the sensors are "speaking" the same language and conversion steps should be added if necessary. We can mention the infamous example of the Mars Climate Orbiter that burned up in the Martian atmosphere in 1999 [30] because of the use of the wrong unit system in the acceleration data. The following section gives an overview of the sensors chosen for this work and its systems relevant for navigation.

2.1 Coordinate systems

2.1.1 Poses and Coordinate Frames

A key requirement to any robotics application, is the efficient track of the positions and velocities of objects in space. For this purpose the concept of coordinate frames was introduced. Its function is to allow the efficient exchange of information between interfacing systems. This chapter will give an introduction to the systems relevant to this work.

As first step conventions need to be defined, for describing locations and orientations in three dimensions. This need emerges from the fact that depending on how the axes are initially arranged, two possible incompatible coordinate systems can be created. This two systems are know as the left-handed coordinate system and the right-handed coordinate system, figure 2.1. Both conventions can be accepted, although right-handed systems are more common in the world of robotics, including on ROS- making the right-handed coordinate system the default of this work. The direction of the z axis can be determined on a right handed coordinate system, by pointing the index finger of the right hand along the positive x axis, the curved palm toward the positive y axis. Finally, the thumb will point to the direction of positive z. The same procedure is valid for the left-handed coordinate system, using the left hand.

After this, the robot *pose* can be address , that is the common abbreviation for position and orientation of an object in space. This description must always be made in relation to a coordinate frame. Typically in robotics application this track is needed to be done through multiple coordinate

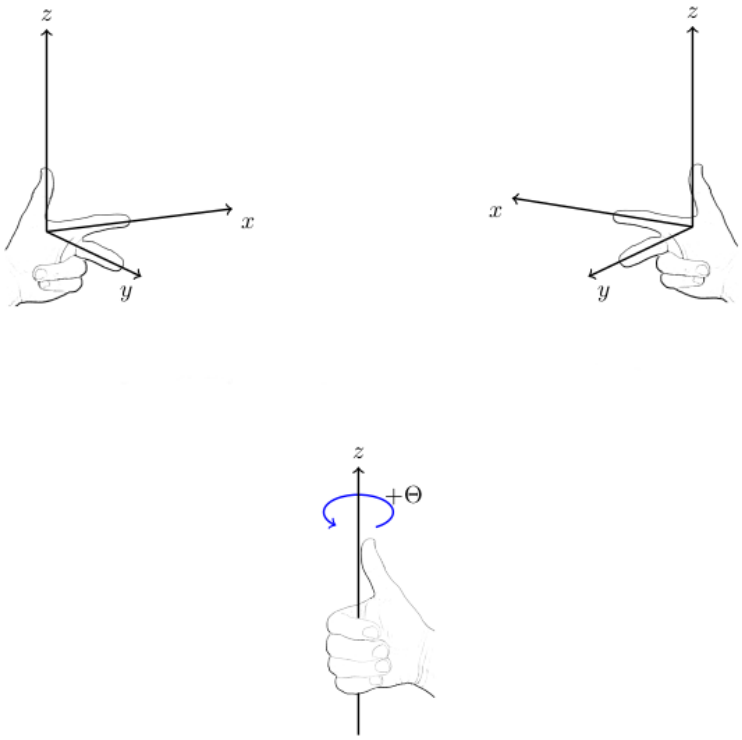

Figure 2.1: Right-hand rule for determining the direction of positive rotations around an axis. [16].

frames. The frames are typically organized in a tree, starting by the "World" frame. This frame is know as the Inertial Reference Frame. An inertial reference frame is a frame in which Newton's laws of motion are valid. It is a well known fact of classical mechanics that any frame moving at constant velocity and without rotation with respect to an inertial frame is also inertial [72].

There are different inertial coordinate systems that can be used depending on the type of the application. Figure 2.2 gives the example of the Geodetic, ECEF and Scene ENU coordinate systems.

In case of the global absolute position, there is the Geodetic (Lat/Long) Coordinate System, used for example on the GPS systems. The "geodetic" coordinate system, divides the planet in an imaginary grid, constituted of parallel East/West lines of latitude and North/South lines of longitude that intersect at the poles. The Latitude and longitude lines are sorted by the angle they form with respect to a reference. The Latitude reference is the Equator and longitude reference is the Prime Meridian, where both start at zero. Due to the spherical shape of the planet the longitude lines are not parallel, the horizontal distance for a degree of longitude depends on the location.

For local frames we have the Scene East-North-Up (Scene ENU) Coordinate System. This is the most basic coordinate system. That defines a simple plan, that uses linear X, Y and Z coordinates to locate elements with respect to the origin. This coordinate system doesn't have a direct attention to the curvature of the earth, being better used for smaller areas (less than 4 km).

This frames and the remaining non inertial frames, follow the REP-105 [56] (Coordinate Frame Conventions). This convention defines four principal coordinate frames: base_link, odom,

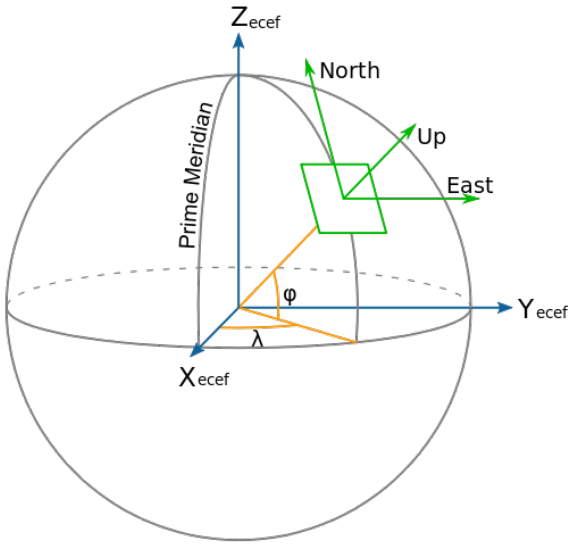

Figure 2.2: Representation of the Geodetic and the ECEF coordinate systems. [18].

map, and earth. As can be seen in the 2.3 figure. This tree allows the connection of multiple robots to the same system on the earth frame. The base_link frame is rigidly affixed to the robot. The map and odom frames are world-fixed frames whose origins are typically aligned with the robot's start position. The map frame can have discontinuities, but it's the frame with the most accurate position estimate for the robot and should not suffer from drift. The odom frame drifts over time, but is guaranteed to be continuous and is accurate enough for local planning and navigation. Any other secondary frames created by sensors mounted rigidly on the robot should be connected to the base link frame. Ideally, this tree should mirror close as possible the real physical connections between the objects involved. Its important also to refer the REP-103[55] convention (Standard Units of Measure and Coordinate Conventions), as it defines remaining details and the drawing convention, the x-axis in red, the y-axis in green and the z-axis in blue. This definitions will be followed carefully for the implementation of this work.

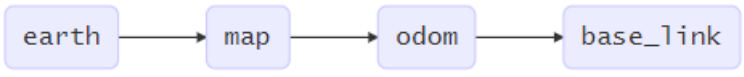

Figure 2.3: Relationship between Frames, followed by the REP-105 convention. [18].

2.1.2 Coordinate Transformation

Consider now that a tree can be created and connected with the previous standards. The mechanism to translate a point between any two coordinate frames needs to be created. To accomplish this, the representation of the pose needs to be defined together with the mechanism for converting from parent to child coordinate frames and vice-versa. This is where the coordinate transformations

comes in. The pose of an object, robot or simply a coordinate frame, needs to include both position and orientation relative to its parent coordinate frame. The position representation is simple, three coordinates can be used to represent a translation of the object referent to the axes of the parent coordinate frame. On the other side, the orientation representation is more complex, the following norms and conventions will describe this problem.

2.1.2.1 Euler Angles

There are several approaches to represent orientation, each with its own advantages and disadvantages. Euler angles are the most intuitive form of representation. Figure 2.4 illustrates the basic idea. Orientation is expressed as a sequence of three rotations around the three coordinate axes. These three rotations are traditionally referred to as roll, pitch and yaw. Working with Euler angles, requires special attention, since the order that the rotations are applied affects the final result. According to this it is necessary to define the order that the rotations are executed, together with the reference frame, if it is the relative to the parent coordinate frame or performed around the axes aligned with the previous rotations. After combining all the possibilities, there are twenty four possible conventions for specifying Euler angles. Although being intuitive and easy to visualize, Euler angles are inconvenient to implement from a mathematical point of view. The mapping from spatial orientations to Euler angles is discontinuous. In certain cases small changes in orientation can cause big jumps. This can be a problem if a smoothly update the orientation is needed.

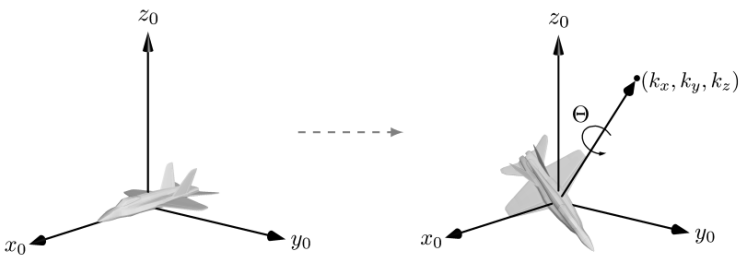

Figure 2.4: Axis-angle representation of orientation. [16].

2.1.2.2 Axis Angle

The axis angle approach encodes the orientation as a single rotation θ around a vector unit vector $[kx, ky, kz]^T$. Simplifying the three rotations system presented by Euler angles. This representation can be better seen in the figure 2.4. This system will be useful in order to better understand the Quaternions system.

2.1.2.3 Quaternions

Quaternions are the most widely used method for encoding orientation. They aren't as intuitive as Euler angles, but they support efficient computation and a spatial mapping without discontinuities.

Similar to the Axis Angle representation a quaternion can be presented as an four element array $q = (x, y, z, w)$ where we can obtain the individual components by the following equations;

$$x = k_x sin(\theta/2) \tag{2.1}$$

$$y = k_y sin(\theta/2) \tag{2.2}$$

$$z = k_z sin(\theta/2) \tag{2.3}$$

$$w = cos(\theta/2) \tag{2.4}$$

This type of representation obtained by equations (2.1)-(2.4) is also know as unit-quaternion since the $|q|$ is equal to 1.

2.1.2.4 Rotation Matrices

For a more efficient implementation this rotations can be accomplished through matrix operations [70]. This can be done by encoding the desired rotation around an axis as a 3×3 rotation matrix. Then the multiplication of this matrix by a point will create the desired rotation around the origin.

The following three matrices correspond to rotations around the x axis or roll (ϕ), y axis or pitch (θ), z axis or yaw (ψ) axes respectively:

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\phi) & -sin(\phi) \\ 0 & sin(\phi) & cos(\phi) \end{bmatrix} \tag{2.5}$$

$$R_x(\theta) = \begin{bmatrix} cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{bmatrix} \tag{2.6}$$

$$R_x(\psi) = \begin{bmatrix} cos(\psi) & -sin(\psi) & 0 \\ sin(\psi) & cos(\psi) & 0) \\ 0 & 0 & 1 \end{bmatrix} \tag{2.7}$$

The product of this three R_x, R_y, R_z matrices, results in the R_{xyz} matrix. Along with it, it is possible to represent any desired orientation. This matrix will be responsible for the rotations,

from the body frame of the robot to the world frame, in the implementation of this work.

$$R_{xyz}(\phi,\theta,\psi) = \begin{bmatrix} cos(\psi)cos(\theta) & sin(\psi)sin(\theta) - cos(\psi)cos(\theta) & sin(\psi)sin(\theta)cos(\theta) + cos(\psi)sin(\theta) \\ sin(\psi)cos(\theta) & sin(\psi)sin(\theta) + cos(\psi)cos(\theta) & sin(\psi)sin(\theta)cos(\theta) - cos(\psi)sin(\theta) \\ -sin(\theta) & cos(\theta)sin(\phi) & cos(\theta)cos(\phi) \end{bmatrix} \tag{2.8}$$

In this work the body-frame angular rates also need to be converted to the world frame. This can be solved by expressing the angular rates, represented by (p,q,r) in terms of the derivatives of the Euler angles, this rotational transformations are carried out as follows:

$$\begin{aligned} \begin{bmatrix} p \\ q \\ r \end{bmatrix} &= \begin{bmatrix} \phi' \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\phi) & -sin(\phi) \\ 0 & sin(\phi) & cos(\phi) \end{bmatrix} \begin{bmatrix} 0 \\ \theta' \\ 0 \end{bmatrix} + \\ &+ \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\phi) & -sin(\phi) \\ 0 & sin(\phi) & cos(\phi) \end{bmatrix} \begin{bmatrix} cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ \psi' \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 & -sin(\theta) \\ 0 & cos(\phi) & sin(\phi)cos(\theta) \\ 0 & -sin(\phi) & cos(\phi)cos(\theta) \end{bmatrix} \begin{bmatrix} \phi' \\ \theta' \\ \psi' \end{bmatrix} \end{aligned} \tag{2.9}$$

After this, a inversion is performed on the matrix 2.9 and results in the final matrix 2.10. This matrix expresses the derivatives of the three angular position states in terms of the angular positions ϕ and θ and the body rates p, q and r. This matrix allows the computation of the necessary rotations on the angular components. As well as the previous matrix, it will be used on a practical work to allow the conversion from the body frame of the robot to the world frame.

$$\begin{bmatrix} \phi' \\ \theta' \\ \psi' \end{bmatrix} = \begin{bmatrix} 1 & sin(\phi)sin(\theta)/cos(\theta) & cos(\phi)sin(\theta)/cos(\theta) \\ 0 & cos(\phi) & -sin(\phi) \\ 0 & sin(\phi)/cos(\theta) & cos(\phi)/cos(\theta) \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{2.10}$$

In the implementation of this work, the geodetic coordinate system needs to be converted to the local frame, to the ENU standard. This conversion is performed in two steps [25], first the geodetic coordinates latitude ϕ, longitude λ and , height h are converted to Earth Centred Earth Fixed (ECEF) using the following formulas:

$$X = (N(\phi) + h)cos(\phi)cos(\lambda) \tag{2.11}$$

$$Y = (N(\phi) + h)cos(\phi)sin(\lambda) \tag{2.12}$$

$$Z = (\frac{b^2}{a^2}N(\phi) + h)sin(\phi) \tag{2.13}$$

$$N(\phi) = \frac{a}{\sqrt{1 - e^2 sin^2\phi}} \quad e^2 = 1 - \frac{b^2}{a^2} \tag{2.14}$$

Where, a is the the equatorial radius (semi-major axis) and b is the polar radius.

The second step is to convert the ECEF coordinates to ENU, this is achieved using a a rotation matrix, where the displacements in ECEF coordinates are transformed to ENU coordinates. The orientation of ENU (de, dn, du) coordinates is determined by rotating the ECEF (dx, dy, dz) coordinates: firstly about the z axis by λ degrees and then the new y axis b ϕ represented in the following matrix:

$$\begin{bmatrix} de \\ dn \\ du \end{bmatrix} = \begin{bmatrix} -sin(\lambda) & cos(\lambda) & 0 \\ -sin(\phi)cos(\lambda) & -sin(\phi)sin(\lambda & cos(\phi)) \\ cos(\phi)cos(\lambda) & cos(\phi)sin(\lambda) & sin(\lambda) \end{bmatrix} \begin{bmatrix} dx \\ dy \\ dz \end{bmatrix}. \tag{2.15}$$

2.2 Sensors

2.2.1 LiDAR

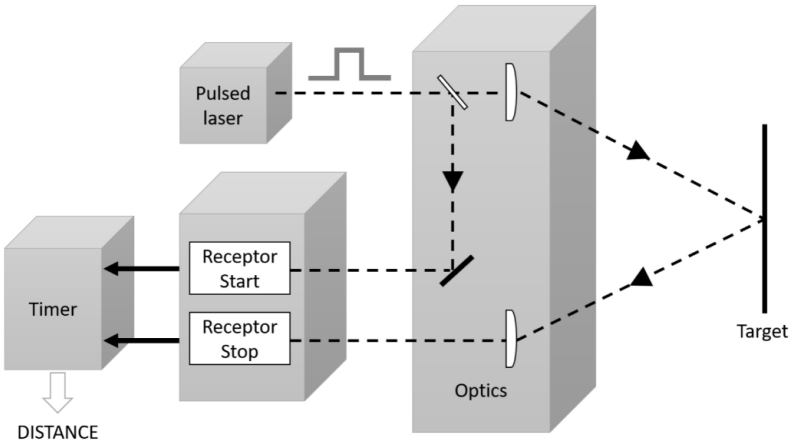

Figure 2.5: LiDAR physics principles representation. [3].

The term LiDAR (Light Detection And Ranging) appeared first in 1963. It was a combination of the words “light” and “radar". Its initial use mostly fell in military and aerospacial fields and its development fell mostly into remote sensing applications [38]. Today LiDAR technology matured much since then, especially thanks to the latest race towards the autonomous car and robotic vehicles. New tighter requirements were created, giving the example of automotive systems that require a combination of long range, high spatial resolution and real time performance with the

addition of tolerance to solar background noise. This leads to the development of different working principles for different possible use cases, including the modern rotating scanners, with higher scanning frequency and multiple stacked detectors.

The working principle continues to be the same, the reflection of light and calculation of the time-of-flight (TOF), as can be better seen on the 2.5. By projecting an optical signal onto an object/surface the reflected backscattered signal is received and processed to determine the distance. This allows the creation of 3D point clouds of the soundings. The formula is very simple, distance D to the target is measured based on the round-trip delay of light waves that travel to the target.

$$D = c \cdot \Delta T \tag{2.16}$$

where: D = The distance of the object c = Speed of light Δ = Time required by the light to travel

2.2.1.1 Modulation Methods

There are 3 main methods that can be applied [3]; modulation of the intensity, phase and frequency of the transmitted signal. The final step is the measurement of the delay for that modulation pattern to appear at the receiver.

- The modulation of the intensity consists of the emission of a short light pulse onto a target, the arrival time of the pulse's reflection at the detector determines the distance. This approach provides resolutions around the centimeter-level over a large window of ranges. This pulses can have high instantaneous peak power, but very short duration, on the nanosecond level. This enables the reach of long distances, maintaining average power under eye safety limits.

- The second approach uses the amplitude modulation of a continuous wave (AMCW). The phase modulated on the amplitude of the emitted and received waves are compared and it's offset measured to calculate the distance. The precision is similar to the previous mentioned method but only to medium ranges. The worst performance comes due to the limited range imposed by the 2Pi ambiguity [71], in frequency modulation. Also the reflected signal coming from distant objects is strongly attenuated. Since the emission is continuous, the amplitude is lowered to remain below the eye-safe limit.

- The third approach applies the technique of frequency modulated continuous wave (FMCW). Within this method the modulation and demodulation are done on the frequency domain, this allows the detection by a coherent superposition of the emitted and revived waves. This method is more robust and consequently, it achieves resolutions, between 150 µm to 1 µm at long distance.

Independent of the modulation method, LiDAR can also be classified on the basis of the output dimensions,that can be 1D,2D or 3D. The typical output data format and file extension from the

scanned LiDAR point cloud data comes in the (.LAS), derived from laser. As all physical sensing systems, LiDAR also suffers from noise, creating 2 types of erros: Systematic error: This kind of error creates permanent deviations on the measurements in a systematic and predictable way. These errors can't be eliminated, but their influence can be compensated. Statistical error: A statistical error occurs when for the exact same measurement experiment the LiDAR reads different values. This typically occurs due to the environment and physics parameters, like; refractions, diffraction, etc.

2.2.2 Inertial Measurement Unit

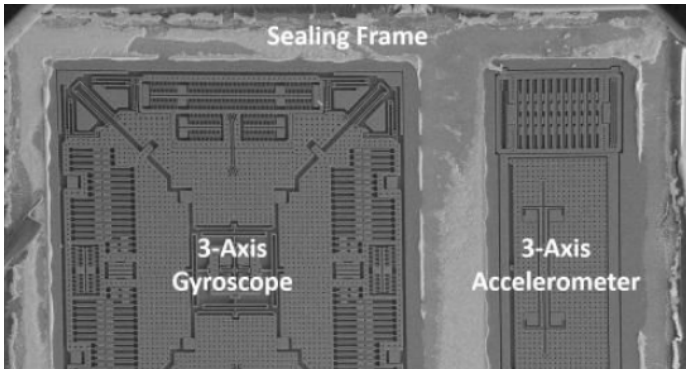

Figure 2.6: Interior of a inertial measurement unit, using MEMS technology. [31].

The inertial measurement unit, known as IMU, typically consist of two 3-axis sensors, an accelerometer and a gyroscope. Optionally, a third sensor can be present- a magnetometer. In the first form they are capable of 6 Degrees of Freedom (6-DOF), by measuring acceleration ($ms^{-}2$) and angular velocity (rad/s). In the second case by adding the magnetic orientation (μTesla) the degrees of Freedom increase to 9-DOF. A system composed of an IMU and a navigation computer can also be known as INS.

This sensing systems are very popular in robotics: being exploited from inertial-only navigation [11], attitude estimation [5], and visual-inertial navigation [19]. Also, they are used in all types of smart mobile devices in motion and orientation tracking [75],[51]. The typical sensors widely used nowadays are micro-electro-mechanical-systems more known as MEMS. Advancements in the field produced cost effective accelerometers, gyroscopes and magnetometers with the advantage of reducing cost, size and weight. A disadvantage that should be referred in the case of low cost MEMS based IMUs is that they are usually affected by non accurate scaling, sensor axis misalignments, cross-axi sensitivities, and non zero biases. This creates the need of a careful calibration in order to try to identify these quantities. Even though the sensors are typically calibrated in the factory, (possibly time-varying) errors can still remain.

2.2.2.1 Accelerometer

An accelerometer works by measuring the specific force f^b associated with the body. This force is the difference between the true acceleration in space with the acceleration caused by gravity. The most common MEMS accelerometer is constituted by a small plate attached to torsion levers. The acceleration changes, moving these plates and changing the capacitance between them. Consequently, the capacitance change is proportional to the linear acceleration.

$$f^b = r'' - G \tag{2.17}$$

where: G = Is acceleration due to gravity r = the position vector

2.2.2.2 Gyroscope

The function of the gyroscope is to obtain the orientation of the body. It achieves this by measuring the angular rate that corresponds to the rotation of body within respect to the inertial frame expressed in body frame. The typical sensing mechanism of a MEMS gyroscope is a vibrating, resonating ring or lever. When an angular rotation is applied the vibration frequency is changed due to the Coriolis force [17]. This force is described by the following equation;

$$F_c = -2m \cdot \omega \cdot v \tag{2.18}$$

where: F_c = The vector sum of the physical forces acting on the object. m = Mass of the object. ω = The angular velocity, of the rotating reference frame relative to the inertial frame. v = Velocity relative to the rotating reference frame.

2.2.2.3 MEMS IMUs errors

As sated before MEMS IMUs are vulnerable to errors, that can be partitioned as follows;

- Accelerometer Bias error; This error affects the output of the accelerometer with a constant offset. This offset can change slightly after each run.
- Misalignment and Non orthogonality; In the other words it means that the axis of the sensors can be slightly misaligned relatively to the body, producing deviations on the real measurements.
- Accelerometer Scale Factor error; In this case the error is caused by the scaling factor that is used to convert the actual reading of the sensor to the measurement unit. This error is proportional to the acceleration measured.
- Gyroscope Drift; This drift or also known by bias error, results from a constant offset from the real measurement. Like in the case of the accelarometer, this bias varies after each run.

- Gyroscope Scale Factor error; The error is caused by the scaling factor that is used to convert the actual reading of the angular rate to the measurement unit. This error is proportional to the sensed.
- Random Noise; This error is the random noise associated with all the measurements of physical quantities.
- Nonlinearity due to Temperature variations; Due to is small size, the operation of the MEMS technology can be affected by temperature variations.

2.2.3 Global Positioning System

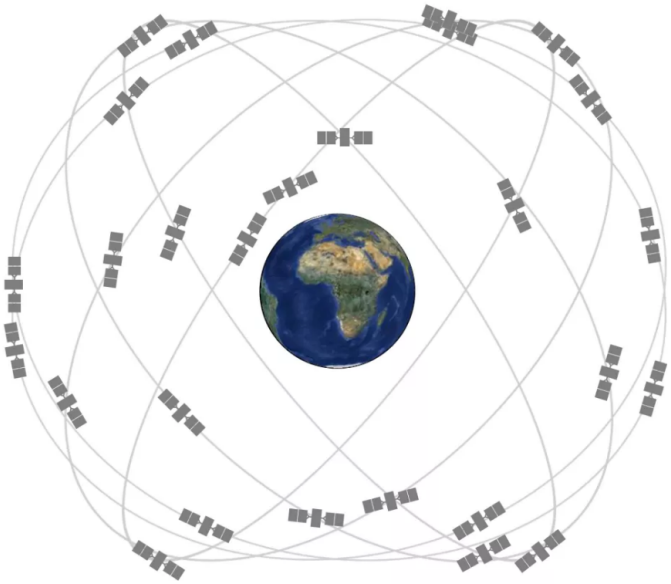

Figure 2.7: Global positioning system (GPS), satellites orbits representation. [27].

The global positioning system [26], more known as GPS, is a satellite based navigation system originally developed for military purposes that entered in full service in 1995. As legacy from its initial development, there are two different modes of quality of service, one for military and another for civilian uses. Initially, there was the policy of "selective availability" that allowed the degradation of the quality of the signals whenever and wherever was defined by the military. Nowadays this policy was removed, making any person with a simple GPS-enabled smartphone able to reach accuracy's within 4.9 meters.

The GPS system has three distinct components; the satellites or space segment, the control centers or control segment and GPS receiver modules or user segment.

The modern GPS space component consists of 27 satellites, from 24 original, orbiting in six different orbital planes. Each plane has four satellites- as can be seen in the figure 2.7. The average orbital radius of the satellites is of 20200 km. This arrangement was planed to make sure that at least four satellites are visible in any point of earth. The GPS satellites communicate

by sending continually navigation messages at rate of 50 bit/s. The information contained on this messages starts from; the time when message was sent, precise orbital information and eventually-the general system health and rough orbits of all GPS satellites. The data format received follows the NMEA [49] message standard. The control center function is to continuously check the health of the satellites and to performs corrections when needed. Finally, the receiver calculates the transit time of each message and with the use of it, computes the distance to each satellite.

2.2.3.1 Calculation of Position

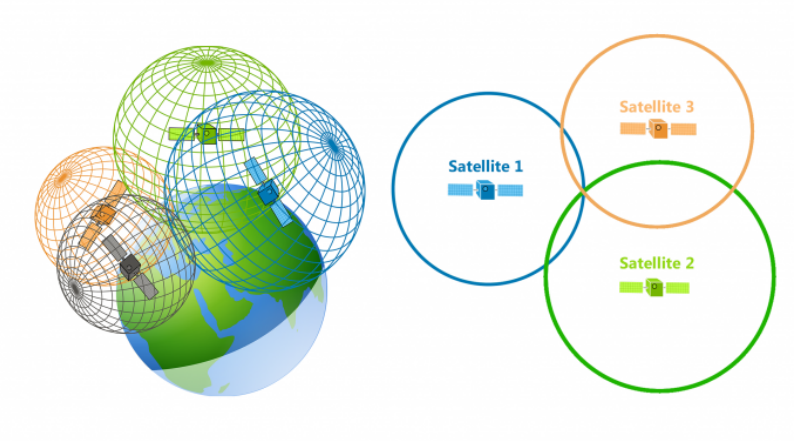

Figure 2.8: GPS trilateration technique . [29].

GPS receivers use the Trilateration technique, that can be better seen on the figure 2.8. After the calculation of the distance from one satellite, the receiver can assume that is on a surface of a sphere centered by the satellite. with the radius equal to it's distance. As can be understood, one satellite is not enough to discover the real position of the receiver. But with the addition of more spheres, the initial area of the sphere can be reduced to the area of a circle in case of the intersection of two spheres (if it intersection happens at more than one point). If three spheres intersect, the result will be two points. Finally, with the intersection of four spheres the position of the GPS receiver will only be one point, with the assumption of no errors. In reality, this is a bad assumption since the GPS system is affected from a broad range of errors.

2.2.3.2 Errors in GPS

- Clock and Calculation Errors; GPS satellites, are equipped with highly accurate atomic clocks. Although, in reality there is also a particular inaccuracy problem derived from the synchronization of all clocks in satellites. In the other end of the clock in the receiver side, there is usually a low cost quartz oscillator, which has bad accuracy. The calculations on the microprocessors in the receivers also cause round off errors.

- Atmospheric Effects; The atmosphere has a considerable influence on the accuracy of the GPS signal. The major effects occur from delay in ionosphere and troposphere. In ionosphere, the presence of electrons and other charged particles affect the propagation of the signals- these charges distribution often are affected by the solar activities. In the troposphere the water vapor causes the refraction of the signals, creating phase delay.

- Multipath; Reflections from geographical objects and buildings are also received in the GPS receiver. These received reflected signals interfere with the pure signals and add errors in real estimated position. Multipath is a major issue in navigation in cities.

- Relativity; Relativity effects have certain contribution to the GPS inaccuracies. The gravitational time dilation makes a satellite clock run slightly faster than an earth based clock. The Sagnac effect [63] also creates time discrepancies due to rotation relative to receivers on earth.

- Measurement Noise; The measurement noise is created in the stages of signal propagation and processing. Some sources of this error are the receiver noise and quantization noise.

The previous errors can be added in the worst case and accumulate up to ±15m, without considering the case of selective availability. Trying to contradict this effects, satellite based augmentation systems, as EGNOS [20], are being deployed and can improve accuracy erros up to or less than ±5 m. These systems work by compensating the ionospheric effects and also improve orbit and clock errors.

Chapter 3

Stochastic Estimation

Robotics is the science of perceiving and manipulating the physical world through computer-controlled mechanical devices. In general, this robotic systems need to face the enormous amounts of uncertainty of the physical world. These uncertainties can be divided in three major elements:1) Sensors, which are limited in range, resolution and simultaneity subject to measurement noise; 2) Actuators that are subjected to wear-and-tear, with limited precision and subject to control noise; and the model of the system, which is used for the control software of the robot, that is an abstraction of the real world, making it just an approximation due to unknown variables, correlations or computational limitations. Robots are real-time systems, which limits the amount of computation that can be carried out. The accuracy is often a question of the costs of a system. Traditionally, in the past many of this uncertainties have mostly been ignored in robotics. However, as robots complexity increases, the ability to cope with uncertainty is critical for building successful robots. This chapter gives an introduction and overview of stochastic estimation topics that are relevant to this thesis.

3.1 Basic Stochastic and Robot concepts

3.1.1 Stochastic Robot Model

In a Stochastic system, typically all the relevant characteristics of robot state are encoded in a so called state vector X. This state vector may contain (depending on the specific robotic system) static and dynamic variables such as:

- Pose (location and orientation).
- Velocities and accelerations (of the robot itself, but also of its manipulators).
- Configuration of its manipulators.
- Environmental variables such as landmarks.
- Condition of robot battery, heat, etc.

In this work the characteristics of robot state are encoded in a 15-dimensional state vector, X. That comprises the vehicle's 3D pose, 3D orientation, and their respective velocities, and linear acceleration. Rotational values are expressed as Euler angles. The state vector is represented as follows:

$$X = [x, y, z, roll, pitch, yaw, x', y', z', roll', pitch', yaw', x'', y'', z'']. \tag{3.1}$$

Typically, the robotic model is described after the discretization by a discrete-time system. In this case, it is a standard 3D kinematic model derived from Newtonian mechanics, where the following equations represent the heterogeneous system model:

$$X_{k+1} = \begin{cases} x_k + x'_k \cdot \Delta T + \frac{1}{2} \cdot x''_k \cdot \Delta T^2 \\ y_k + y'_k \cdot \Delta T + \frac{1}{2} \cdot y''_k \cdot \Delta T^2 \\ z_k + z'_k \cdot \Delta T + \frac{1}{2} \cdot z''_k \cdot \Delta T^2 \\ roll_k + roll'_k \cdot \Delta T \\ pitch_k + pitch'_k \cdot \Delta T \\ yaw_k + yaw'_k \cdot \Delta T \\ x'_k + X''_k \cdot \Delta T \\ y'_k + Y''_k \cdot \Delta T \\ z'_k + z''_k \cdot \Delta T \\ roll'_k \\ pitch'_k \\ yaw'_k \\ x''_k \\ y''_k \\ y''_z \end{cases} . \tag{3.2}$$

3.2 The Kalman Filter

The Kalman filter (KF) was invented in the 1950s by Rudolph Emil Kalman [61], as a technique for filtering and prediction in linear systems, which is based on the implementation of the Bayes filters [53]. It is an optimal estimator for linear systems, subject to zero mean Gaussian noise. The Kalman filter addresses the general problem of trying to estimate the state X of a discrete-time controlled process that is governed by the linear system state transition equation 3.3.

$$x^{-}_{k+1} = A_k x_k + B_k u_k + w_t \tag{3.3}$$

Where A is the n × n system matrix , B is the n × m input gain matrix, with m dimension equal to the input vector u_k , x_k is a n-dimensional state. w_k is a n-dimensional vector of the zero mean

Gaussian plant noise. which is added to take in consideration the uncertainty of the model and the actuators.

The measurement equation is given by

$$z_k = Hx_k + v_k \tag{3.4}$$

H is an q × n matrix, that relates the state to the measurement z_k, q has the same dimension of the measurement vector z_k. v_k is a q-dimensional zero mean Gaussian noise vector, known by measurement noise vector. This vector incorporates the uncertainty of the measurement.

The algorithm, consists of two parts: the prediction step and the update step. In the update step, the state is projected forward using the equation 3.3 However, the uncertainty in the state also needs to be propagate forward. Since the state is a Gaussian distribution, and is fully parameterized by a mean $\hat{x}_k$ and covariance P_K, we can update the covariance as in equation 3.5.

$$P_k^- = A \cdot P_k \cdot A^T + Q \tag{3.5}$$

where, A is the same matrix used to propagate the state, and Q is random Gaussian noise. Equations 3.3 and 3.5 explore the property of Gaussian's distributions: adding two Gaussians results in a Gaussian, and applying a linear transformation to a Gaussian also results in a Gaussian. After the predict phase, the original Gaussian distribution defined by by x_k and P_k becomes a new Gaussian, now characterized by x_{k+1} and P_{k+1}. This concludes the prediction step of the algorithm.

The update is also the correction step of the Kalman Filter. Where a measurement of an observable variable is made and fused with the previous estimate. First, the measurement of the system is made using a linear measurement model in the equation 3.4 After this, the Kalman Gain is calculated, depicted in Equation 3.6

$$K = P^- H^T (HP^- H^T + Q)^{-1} \tag{3.6}$$

Next , the calculation of the difference between the expected observation, and the actual observation is performed, this is known as residual or innovation equation 3.7.

$$r_k = (z_k - H\hat{x}_k^-) \tag{3.7}$$

Finally, by combining Equations 3.6 and 3.7. The Equation 3.8 corrects the mean, while equation 3.9 corrects the covariance.

$$\hat{x}_k = \hat{x}_k^- + r_k \tag{3.8}$$

$$P_k = (I - K_k H_k) P_k^- \tag{3.9}$$

The figure 3.1 summarizes the previous steps.

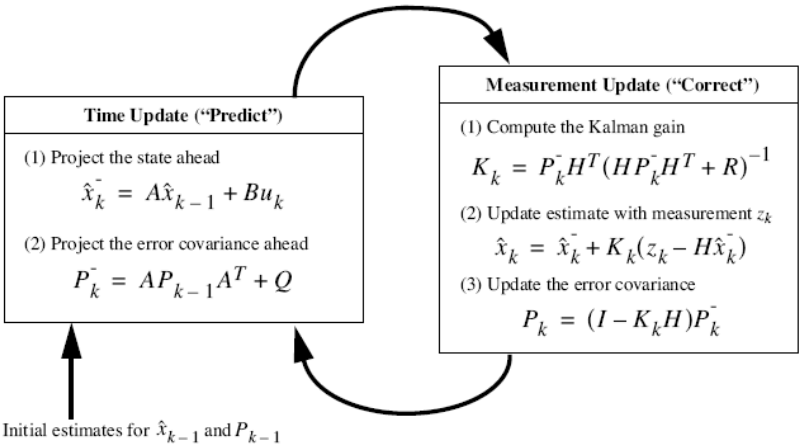

Figure 3.1: The Kalman Filter algorithm.[37].

3.3 The Extended Kalman Filter

The extended Kalman filter (EKF) is also a Gaussian filter and consists basically in the applying a linearization around the current value of the state estimate when the process is nonlinear. In this case, it is considered a nonlinear dynamic system, described by the nonlinear state transition function 3.10.

$$x_{k+1} = f(x_k, u_k) + w_k \tag{3.10}$$

The x_{k+1} term represents the robots system state at time k, f is a nonlinear state transition function and w_k is the process noise, which is assumed to be normally distributed. The u_k represents the control inputs. The measurements can be described as in equation 3.11

$$z_k = h(x_k) + v_k \tag{3.11}$$

z_k is the measurement at time k, h is a nonlinear sensor model that maps the state into measurement space, and v_k is the normally distributed measurement noise.

The first stage in the algorithm, is the prediction step, that projects the current state estimate and error covariance forward in time, shown as equations, 3.12 and 3.13.

$$\hat{x}_k^- = f(\hat{x}_{k-1}, 0) \tag{3.12}$$

$$P_k^- = A_k P_{k-1} A_K^T + W_K Q_{k-1} W_k^T \tag{3.13}$$

f can be better described as a standard 3D forward kinematic model derived from Newtonian mechanics. The estimate error covariance, P, W and A are the process jacobians at step k, and perturbed by Q, the process noise covariance.

The next step is to calculate the Kalman gain, K, this is achieved by using H and V, that are the measurement Jacobians at step k, the measurement covariance, R, and $\hat{P}_K$, equation 3.14.

$$K_k = P_k^- H_k^T (H_k P_K^- H_k^T + V_k R_k V_k^T)^{-1}. \tag{3.14}$$

On the correction step the gain is used to update the state vector and covariance matrix, equation 3.15.

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - h(\hat{x}_K^-, 0)) \tag{3.15}$$

As final step the Joseph form covariance update equation is used to promote filter stability by ensuring that P_K remains positive semi-definite, equation 3.16.

$$P_k = (I - K_k H_K) P_K^-. \tag{3.16}$$

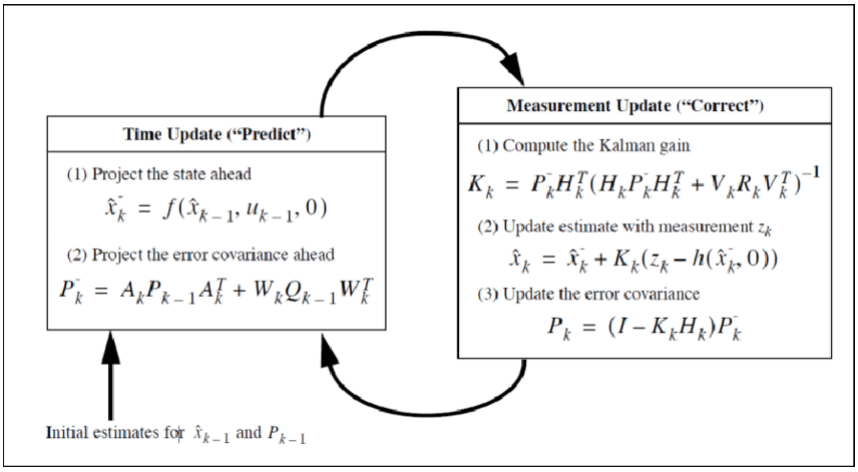

Figure 3.2: The Extended Kalman Filter algorithm.[12].

The figure 3.2 summarizes the previous steps.

Chapter 4

3D Simultaneous Localization and Mapping

4.1 Problem

Simultaneous localization and mapping (SLAM), is the computational problem of building a map of an unknown environment by a mobile robot while navigating the environment using the map that is currently being updated. In practical terms this is a process that fuses sensor observations of features, landmarks and dead-reckoning information over time. Simultaneously the location of the robot in an unknown area is estimated and a map is built that includes the detected feature locations. This process is affected from both noisy controls and observations. The figure 4.1 illustrates this problem, where a robot moving through an environment is taking relative observations of a number of unknown landmarks, that are used for navigation reference.

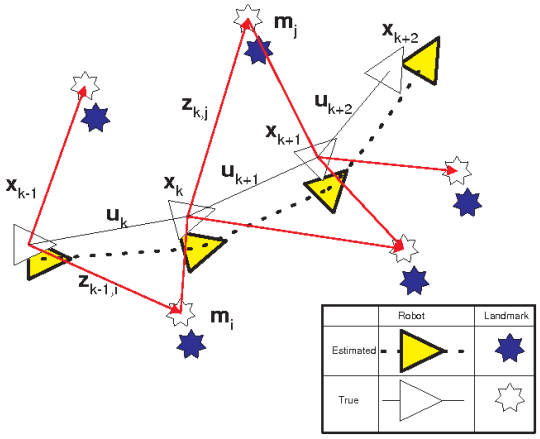

Figure 4.1: The of essential the SLAM problem.[68].

This process can be described with a probabilistic approach, as can be seen in the equation 4.1. where; m is the probability distribution describing the map, x_k the robot's pose, $Z_{0:k}$ the set of previously observed measurements, $U_{0:k}$ the set of control inputs already applied to the robot,

x_0 it's initial pose. The objective is to compute an estimate of x and the map m over the discrete time steps k. For this purpose the simultaneous estimate of both robot and landmark locations is required, but in reality the true locations are never known or measured directly.

$$P(x_k, m|z_{0:k}, U_{0:K}, X_0) \tag{4.1}$$

As the example used before shows, there are many proposed systems, which use different kind of sensors and algorithms, for which there is no single best solution. In this work the 3D LiDAR SLAM is explored. LiDAR was the preferred technology, because of the advantages of the laser sensors, that have high measurement accuracy, strong anti-interference ability as well as wide sensing range. It allows laser-based SLAM to have higher positioning accuracy and better robustness. In this work, no reference landmarks are previously given to the system and the position estimation relies mainly in online extracted features. The pose update is calculated by the matching points and consequently features extracted between consecutive frames of the point cloud provided by the laser scanner. For this purpose the LOAM algorithm is introduced. In this chapter, the inner working and main features of Loam algorithm that are considered important for this work will be analysed and explained. For more detailed explanation, the original LOAM [80] article should be consulted.

4.2 LOAM

The LiDAR Odometry and Mapping in Real-time (LOAM) [80] algorithm, was initial proposed to solve the problem of real-time odometry and mapping using range measurements from a 3D LiDAR system. This problem is complex since the range measurements are received at different times, together with motion estimation errors. This can cause miss-registrations on the resulting point cloud. Previously this problem was only solved efficiently with off-line batch methods, often using loop closure to correct for drift over time [9][15]. The LOAM algorithm is able to achieve both low-drift and low-computational complexity, without the requirement of high accuracy ranging or inertial measurements. This level of performance comes from the division of the complex problem of simultaneous localization and mapping in two algorithms. Instead of optimizing a large number of variables simultaneously. One algorithm performs odometry at a high frequency, with less accuracy, having the objective of estimating the velocity of the LiDAR. The other algorithm runs at a lower rate performing fine matching and registration of the point cloud. The combination of both algorithms allows the mapping in real-time. The problem addressed by LOAM is commonly known as ego-motion estimation- that is the motion estimation using visual data. In this case, this data is a point cloud perceived by a 3D LiDAR, where simultaneously a map for the traversed environment is built. It starts by assuming that the LiDAR is pre-calibrated, the angular and linear velocities of the LiDAR are smooth, continuous over time and without abrupt changes. The authors defined as convention, a sweep as being one complete LiDAR scan, Where

k indicates the individual sweeps, and P_k is the point cloud perceived during a sweep k. Two coordinate systems are defined as follows;

- LiDAR coordinate system L : This is a 3D coordinate system with it's origin at the geometric center of the LiDAR. The x-axis is pointing to the left, the y-axis is pointing upward, and the z-axis is pointing forward. The coordinates of a point $i, i \in P_k$, in L_k are denoted as $X^L_{(k,i)}$.
- World coordinate system W: It is also a 3D coordinate system coinciding with L at the initial position. The coordinates of a point $i, i \in P_k$, in W_k are $X^W_{k,i}$.

After this, the LiDAR odometry and mapping problem can be better defined as: given a sequence of LiDAR clouds P_k , compute the ego-motion of the LiDAR during each sweep k, and build a map with P_k for the traversed environment.

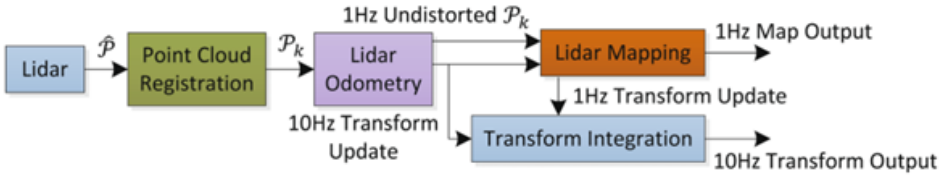

Figure 4.2: Block diagram represents the different algorithms which constitutes the LOAM system. [7].

The figure 4.2 presents a diagram of the LOAM system. Where first, the $\hat{\mathrm{P}}$ points received in a laser scan are registered during each sweep in L. This combined point Laser cloud, acquired during the sweeps k forms P_k. At this point the further processing of P_k is divided in two algorithms. First, the LiDAR odometry receives the point cloud and estimates the motion of the LiDAR between two successive sweeps. With this motion, the distortion present in P_k is corrected. This algorithm performs the previous steps at a frequency around 10Hz. After this, the LiDAR mapping receives the final output from the odometry and performs the further processing. It consist in the fine matching and registering of the undistorted point cloud onto a map, this process runs at a frequency of 1Hz. The last step consist in the pose transforms integration, where the published pose from both algorithms is integrated, generating a transform output with the same frequency as the odometry, regarding the LiDAR pose with respect to the map.

4.2.1 LiDAR Odometry

The odometry algorithm can be divided in three distinct steps; the first is the feature point extraction, followed by feature Point Correspondence and motion estimation step.

4.2.1.1 Feature Point Extraction

This step consist in the extraction of feature points present on the LiDAR cloud, P_k. The feature points are extracted from individual scans P_k and selected depending on it's co-planar geometric

relationships. The desired points to be selected, are the ones present on sharp edges and planar surface patches. This process starts by analysing the smoothness of the local surface, this is accomplished by sorting the points present in a scan, using a defined threshold called C values. The feature points inside the defined maximum threshold, are labeled as edge points, in opposition the points within the minimum threshold, correspond to the planar points. The default threshold for C defined by the authors is $5 \cdot 10^{-3}$.

$$c = \frac{1}{|S| \cdot ||X^L_{(k,i)}||} \sum_{j \in S. j \neq i} ||(X^L_{(k,i)} - X^L_{(k,j)})|| \tag{4.2}$$

The equation 4.2 represents the calculation of the C values, where;

i is a point in P_k.

S is the set of consecutive points i returned by the laser scanner in the same scan.

$X^L_{k,i}$ The coordinates of a point.

The individual scans are divided into four identical sub-regions. Where each one only can provide in maximum two edge points and four planar points. The selection of these points needs to obey the following restrictions to ensure an even distribution of the feature points in the environment;

- The selected edge or planar points cannot exceed the maximum quantity defined for the equivalent sub-region.
- The surrounding points were not selected yet.
- Cannot make part of a surface patch perpendicular within 10° to the laser beam, or on the boundary of an obstructed region.

The figure 4.3 shows an example of the extracted feature points from a corridor, using this method. The edge points can be seen in yellow and planar points in red color.

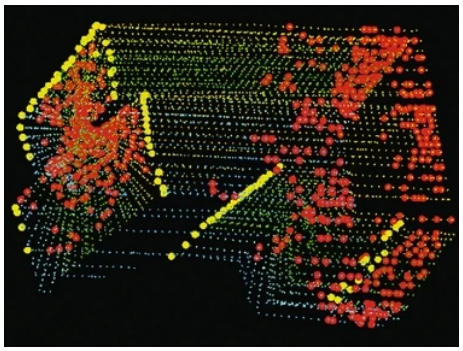

Figure 4.3: An example of extracted edge points (yellow) and planar points (red) from LiDAR cloud taken in a corridor. The LiDAR is moving towards a wall on the left side of the figure, resulting in motion distortion on the wall. [22].

4.2.1.2 Feature Point Correspondence

This step starts by estimating the motion of the LiDAR within a sweep. This process can be seen on the figure 4.4. The current time stamp of a sweep k is represented by t_k. After receiving a sweep, the point cloud P_k is projected to the future, to t_{k+1}. Then, this projected point cloud represented by $\bar{P}_k$, is compared during the next sweep $k+1$ with the newly received point cloud, P_{k+1} and the motion of the LiDAR is estimated. The motion estimation is a recursive process where the 6 DOF motion is determined using the points received on the on going sweep, where at the beginning of sweep $k+1$, the cloud P_k+1 is empty and grows as more points are received. At each iteration, the detected edges ε_{k+1} and planar points H_{k+1} are projected to the beginning of the sweep using the currently estimated transform. For this purpose the projected edge's $\tilde{\varepsilon}_{k+1}$ and planar points $\tilde{H}_{k+1}$ are stored on $\bar{P}_k$ in a 3D KD-tree [13]. This allows higher efficiency in the calculation of the closest neighbour positions of the featuring points in $\bar{P}_k$.

To be considered an edge line, two edge feature points need to be found. For this purpose, the correspondent close neighbour of a point i, present in the projected point cloud $\tilde{\varepsilon}_{k+1}$ needs to be compared with its closest neighbor in $\bar{P}_k$ represented by j, and l that is the closest neighbor in the two consecutive scans. This two points (j,l) form the correspondence of i. To confirm the correct relation, the local surface is verified using the equation 4.2.

To be considered a planar patch, three feature points need to be found. Similar to the previous method, this points need to be verified. Starting by the closest neighbor of i in $\bar{P}_k$, referred as j. Then, another two points, l as the closest neighbors of i, in the same scan of j and m as the other, in the two consecutive scans to the scan of j. This ensures that the three points are non-collinear. To confirm this relation, the local surface is verified again using the equation 4.2.

After finding the correspondences of the feature points, the following equations are used to compute the feature points distance to its respective correspondences;

For a point $i \in \bar{\varepsilon}_{k+1}$, if (j,l) is the corresponding edge line, j, $l \in \bar{P}_k$, the point to line distance can be computed as:

$$d\varepsilon = \frac{|(\tilde{X}^L_{(k+1,i)} - \bar{X}^L_{(k,j)}) \cdot (\tilde{X}^L_{(k+1,i)} - \bar{X}^L_{(k,j)})|}{|\bar{X}^L_{(k,i)} - \bar{X}^L_{(k,j)}|} \tag{4.3}$$

where $\tilde{X}^L_{(k+1,i)}$, $\tilde{X}^L_{(k,j)})$, and $\tilde{X}^L_{(k,l)})$ are the coordinates of points i, j and l in L.

Then, for a point $i \in \tilde{H}_{k+1}$, assuming (j,l,m) is the corresponding planar patch, $j,l,m \in \tilde{P}_k$,the point to plane distance can be:

$$dH = \frac{|(\tilde{X}^L_{(k+1,i)} - \bar{X}^L_{(k,j)}) \cdot ((\bar{X}^L_{(k,j)} - \bar{X}^L_{(k,l)}) \cdot (\bar{X}^L_{(k,j)} - \bar{X}^L_{(k,m)}))|}{|(\bar{X}^L_{(k,i)} - \bar{X}^L_{(k,l)}) \cdot ((\bar{X}^L_{(k,j)} - \bar{X}^L_{(k,m)})|} \tag{4.4}$$

where $\tilde{X}^L_{(k,m)})$ is the coordinates of point m in L.

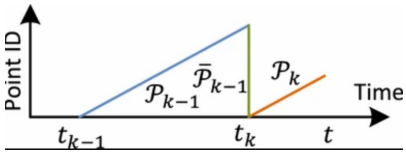

Figure 4.4: Projected point cloud. The blue segment represents the point cloud perceived during sweep k, P_k . At the end of sweep, this cloud is projected to the time stamp t_{k+1} to obtain $\bar{P}_k$ (green segment). Then, during sweep $k+1$, $\bar{P}_k$ is compared with the newly perceived point cloud t_{k+1} (orange segment) and the LiDAR motion is estimated.[54].

4.2.1.3 Motion Estimation

To reduce the complexity of the problem, the motion during a sweep is modeled with constant angular and linear velocities. This allows the use of linear interpolation to calculate the pose transform, for points that are being received at different times. The LiDAR 6-DOF pose transform is stored in a state vector, that contains the translations and rotations. This vector also encondes the motion of the LiDAR and is represented by $T_k^L(t) = [\tau_k^L(t), \theta_k^L(t)]$, where t is the current time stamp, and as stated before, t_k represents the starting time of the current sweep k. The translations are represented by $\tau_k^L(t) = [t_x, t_y, t_z]^T$ and the rotation by $\theta_k^L(t) = [\theta_x, \theta_y, \theta_z]^T$ in L_k. The rotations $\theta_k^L(t)$ can be enconded in a rotation matrix, defined with the help of the Rodrigues formula [41]. This results in following formula

$$R_K^L(t) = e^{\hat{\theta}_k^L(t)} = I + \frac{\hat{\theta}_k^L(t)}{||\theta_k^L(t)||} \cdot sin||\theta_k^L(t)|| + (\frac{\hat{\theta}_k^L(t)}{||\theta_k^L(t)||})^2 \cdot (1 - cos||\theta_k^L(t)||) \tag{4.5}$$

where, $\hat{\theta}_k^L(t))$ is the skew symmetric (or anti symmetric) matrix of $\theta_k^L(t)$ and I the identity matrix . The pose transform $T_{k(k,i)}(t)$ between $[t_k, t_{(k,i)}]$, can be found using linear interpolation of $T_k(t)$, using the following equation;

$$T_{(k,i)}^L = \frac{t_{(k,i)} - t_k}{t - t_k} \cdot T_k^L(t) \tag{4.6}$$

where, i is a point belonging to P_k and $t_{(i,k)}$ the equivalent time stamp. Since $T_k^L(t)$ is not static and changes over time, the interpolation method uses the transform of current time t.

Defined the rotations and translations, the previous edges ε_k and planar H_K sets of points, can be projected to the beginning of the sweep ($\tilde{\varepsilon}_k$ and $\tilde{H}_K$) , using the following equation

$$\tilde{X}_{(k,i)}^L = R_{(k,i)}^L \cdot X_{(k,i)}^L + \tau_k^L \tag{4.7}$$

where, $X_{(k,i)}^L$ is a point in ε_k or H_K and $\tilde{X}_{(k,i)}^L$ is the correspondent point $\tilde{\varepsilon}_k$ and $\tilde{H}_K$. τ_k^L represents the translation vector corresponding to $T_{(k,i)}^L$ and $R_{(k,i)}^L$ the equivalent rotation matrix.

The geometric relationship between an edge point in ε_k and the corresponding edge line, is stated by

$$f_\varepsilon(X_{(k,i)}^L, T_{(k,i)}^L) = d_\varepsilon \tag{4.8}$$

Where, i is a point belonging to ε_k. This relation results from the combination of the equation presented in 4.3 and 4.7.

Equivalently, the geometric relationship between a planar point in H_K and the corresponding planar patch, is given by

$$f_H(X^L_{(k,i)}, T^L_{(k,i)}) = d_H \tag{4.9}$$

Similarly to the previous case, i is a point belonging to H_k and this relation results from the combination of the equation presented in 4.4 and 4.7.

The problem of the LiDAR motion estimation is solved with the Levenberg-Marquardt method [47] using a special adaptation for robust fitting from [45]. The two previous functions are condensed and form the non linear function

$$f(T^L_k(t)) = d \tag{4.10}$$

As presented here, each row of f corresponds to a feature point, and equivalently d corresponds to its distance. This function presents a minimization problem. It is solved using nonlinear iterations by minimizing the distance d toward zero;

$$T^L_k(t) \approx f(T^L_k(t)) - (J^T \cdot J + \lambda \cdot diag(J^T \cdot J))^{-1} \cdot J^T \cdot d \tag{4.11}$$

For this purpose, the Jacobian matrix of f with respect to $T^L_k(t)$ is calculated $J = \frac{\partial f}{\partial T^L_k(t)}$. The λ is a damping parameter adjusted at each iteration and determined by the Levenberg-Marquardt method.

4.2.1.4 LiDAR odometry algorithm

Due to the complexity of the LiDAR odometry algorithm, it is presented in a format of a pseudo-code, Algorithm 1, to allow the easier understanding. An extra step is taken to filter outlier points, the algorithm assigns a bi-square weight for each feature point, described on the equation 4.12. This method attributes smaller weights w to the feature points that have larger distances to their correspondences, in the opposite side the feature points with distances larger than a threshold are assigned with zero weights.

$$w = \begin{cases} (1-\alpha^2)^2 & -1 < \alpha < 1, \\ 0 & otherwise, \end{cases} \tag{4.12}$$

$$\alpha = \frac{r}{6.9459\sigma\sqrt{1-h}} \tag{4.13}$$

Where, r is the corresponding residual from the least square problem, σ is the absolute deviation of the residuals on the median and h is the corresponding element on the diagonal of the matrix $(J \cdot (J^T \cdot J))^{-1} \cdot J^T$ where J is the Jacobian matrix, present on 4.11.

Algorithm 1: LiDAR Odometry

Data: Point cloud from the last sweep $\bar{P}_{k-1}$,
The growing point cloud of the current sweep P_k,
Pose transform from the last recursion as initial guess $T_k^L(t)$.
Result: $\bar{P}_k$ and newly computed $T_k^L(t)$
initialization;
if *new sweep is started* **then**
  $T_k^L(t) = 0$
end
Detect edge points and planar points in P_k, construct ε_K and H_k.
while *number of iterations* < *max defined iterations* **do**
  for *each edge point in* ε_K **do**
    Find an edge line as the correspondence, then compute point to line distance based on 4.8 and stack the equation to 4.10.
  end
  for *each planar point in* H_K **do**
    Find a planar patch as the correspondence, then compute point to plane distance based on 4.9 and stack the equation to 4.10.
  end
  Compute a bi-square weight for each row of 4.10
  Update $T_k^L(t)$ for a nonlinear iteration based on 4.11
  if *the nonlinear optimization converges* **then**
    break
  end
end
if *at the end of a sweep* **then**
  Project each point in P_k to t_{k+1} and form $\tilde{P}_k$,
  Return $T_k^L(t)$ and $\tilde{P}_k$.
else
  Return $T_k^L(t)$ for the next round of recursion.
end

4.2.2 LiDAR mapping

The odometry algorithm performs multiple interactions during each sweep of the LiDAR. In contrast, the mapping algorithm runs at a frequency around ten times slower, being only called once per sweep. At the end of a sweep k, the odometry generates an undistorted point cloud, $\bar{P}_k$, together with the equivalent pose transform, T_k^L. This transform contains the LiDAR motion during two consecutive sweeps, t_k to t_{k+1}. In parallel, the mapping algorithm performs the matching and registers $\bar{P}_k$ in the world coordinates, W. This process can be seen in the figure 4.5.

As standard required for the mapping algorithm, the following parameters were defined;

- The transformation of $\bar{P}_k$ to world coordinates, W, is represented with $\bar{\rho}_K$.
- A point cloud accumulated until sweep $k-1$ on the map, is denoted by ρ_{K-1}.
- The pose of the LiDAR on the map at the end of sweep $k-1, t_k$ is defined by $T_{K-1}^w(t_k)$.
- A set of surrounding points around a feature point inside ρ_{K-1} is known as S'.

After receiving the output from the odometry, the mapping algorithm starts by extending the $T_{K-1}^w(t_k)$ from the sweep t_k to t_{k+1} and gets the $T_K^w(t_{k+1})$. Afterwards, the $\bar{P}_k$ is converted to world coordinates, $\bar{\rho}_K$. Then, the LiDAR pose $T_K^w(t_{k+1})$ is optimized and the matching of $\bar{\rho}_K$ with ρ_{K-1} is performed. The feature point extraction is accomplished in identical way as in the odometry algorithm in Section 4.2.1.2, with the particularity that in this case the number of feature points used is 10 times bigger. Due to the bigger amount of points and for more efficient matching, the feature points present in the cloud ρ_{K-1} are stored in 10m cubic areas, then these points are intersected with $\bar{\rho}_K$. The matches are extracted and stored in a 3D KD-tree [13], corresponding to the W frame. This matching is performed inside ρ_{K-1} within regions of (10×10×10 cm) around the feature points. This process is based on the smoothness of the local surface, where the points are sorted based on their c values using the same threshold as in section 4.2.1.1. The points are filtered, in case of edges, by only keeping the points on edge lines in S'. The same happens in the case of the planar points- by only keeping the points on planar patches. The next step preformed by the mapping algorithm, is to compute the covariance matrix of ρ_{K-1}. This matrix is represented by M, the eigenvalues E and eigenvectors V from this matrix are also calculated. These values are used to determine the poses of the point clusters and it's equivalent distances to a point-to-line or point-to-plane. This positions are calculated in a manner that the line or the plane must pass through the center of S' and obey to the following rules;

- If S' is surrounding an edge line, V must contain one eigenvalue, considerably larger than the other two. The eigenvector in E bonded to the largest eigenvalue exposes the orientation of correspondent edge line.
- Equivalently, if S' is distributed on a planar patch, V includes two large eigenvalues, where the third one is significantly smaller than the other two. The eigenvector, present in E, connected to the smallest eigenvalue, represents the orientation of the planar patch.

The calculation of distances is based in the same principles used in the chapter 4.2.1.2. First the distance from a feature point to it's correspondence is based on the selection of two points on an edge line and three points on a planar patch. In the next step, these distances are computed using the same equations as 4.3 and 4.4, equivalently. Then, the equation correspondent for each feature point 4.8 or 4.9 is used. With the particularly that those points present in $\bar{\rho}_k$ share the same time stamp, t_{k+1} . As next step the same method based on the Levenberg-Marquardt [47] using a special adaptation for robust fitting [45] is used to solve the problem of nonlinear optimization. After this the $\bar{\rho}_k$ point cloud is finally registered on the map. The mapping algorithm performs an extra step to ensure evenly distributed points through the map. Every time a new scan is merged on the map, the map cloud is downsized by voxel-grid filters [1]. This voxel-grid filters work by making an average of all points in each voxel, and leaving just the averaged point in the voxel. The size of the voxel's varies, where the edge points, have a voxel size of 5×5×5 cm. And the size corresponding to the planar points, is 10×10×10 cm.

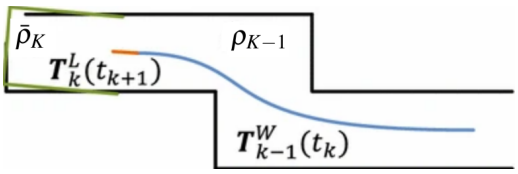

Figure 4.5: Representation of the mapping process. The pose on the map,$T_{K-1}^w(t_k)$ is pictured by the blue curve belonging to the sweep $k-1$. In the same way, the LiDAR motion estimated by the odometry algorithm is represented with the orange curve, equivalent to an entire sweep k, $T_K^L(t_{k+1})$. Having the pose on the map $T_{K-1}^w(t_k)$, the odometry motion $T_K^L(t_{k+1})$, the undistorted point cloud $\bar{P}_k$ can be published on the map, represented by $\bar{\rho}_K$ and the green segments. Finally this cloud is matched to the existing cloud from the map, ρ_{K-1} depicted by the black segments. [33].

4.2.2.1 Integration of pose transforms

The integration of the pose transforms can be visualized on the figure 4.6. The pose originating from the mapping algorithm, $T_{K-1}^W(tk)$ and created once per sweep is represented with the blue region. The LiDAR motion estimated by the odometry algorithm within the current sweep,$T_K^L(tk)$, is represented by orange colored region. The final motion estimation of the LiDAR is the fusion of the two transforms at the frequency of $T_K^L(tk)$.

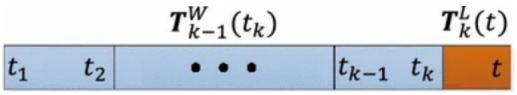

Figure 4.6: Integration of the pose transforms. The pose originating from the mapping, $T_{K-1}^W(tk)$, is represented with the blue region. The LiDAR motion estimated by the odometry algorithm, $T_K^L(tk)$, is represented by orange colored region.[36].

Table 4.1: Computation break-down for accuracy tests [80].

Algorithm	Build	Match	Others (ms)	Total (ms)
	KD-tree (ms)	Features (ms)		
Odometry	11	23	14	48
Mapping	58	134	117	309

4.2.3 Results and performance

The following results are based in a custom built 3D LiDAR, based on a 2D Hokuyo UTM-30LX laser scanner [64], that was the main tested hardware originally used by the authors. As explained before, the LOAM system divides the work load in two different algorithms. The responsibility of LiDAR odometry is to be fast, in order to be able to estimate velocity and simultaneity remove motion distortion in point clouds in real time. But this comes with a cost- since the odometry works with fast speed and low amount of points, it has low-fidelity and cannot ensure accurate mapping. In the other side, the LiDAR mapping, works at slower speed, about a tenth of the of the odometry. It receives the undistorted point clouds and performs fine scan matching, with a big amount of points to ensure accuracy on the map.

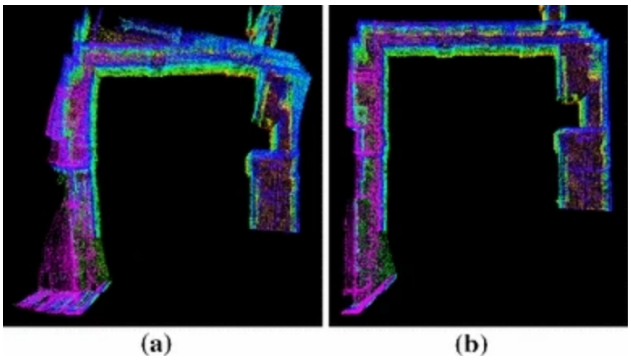

Figure 4.7: Comparison between (a) LiDAR odometry output and (b) final LiDAR mapping, on a corridor scene, with a trajectory of 32m in length, at a speed of 0.5m/s output.[10].

Understanding this, it can be easily assumed that the LiDAR odometry and mapping have different outputs. In the figure 4.7, the output of both algorithms can be compared, where (a) represents the odometry output achieved with direct registering of the laser points and on (b) the further output optimized by the LiDAR mapping. The higher distortion can be seen on the odometry output. Both algorithms were running using the same computational power, where the LiDAR odometry was called 10 times, while LiDAR mapping only once. On table 4.1 the comparison of both algorithms in terms of Computation break-down and accuracy can be observed.

Table 4.2: Motion estimation drift relative errors [80].

Environment	Test 1		Test 2	
	Distance (m)	Error (%)	Distance (m)	Error (%)
Corridor	58	0.9	46	1.1
Lobby	52	2.3	67	2.8

Table 4.3: Motion estimation errors with and without inertial measurements [80].

Environment	Distance (m)	Error		
		IMU (%)	Original (%)	Original+IMU (%)
Corridor	32	16.7	2.1	0.9
Lobby	27	11.7	1.7	1.3
Vegetated road	43	13.7	4.4	2.6
Orchard	51	11.4	3.7	2.1

This system was also tested in indoor and outdoor environments. The figure 4.8 represents the matching errors comparison between different environments. Overall, the indoor tests achieved a relative accuracy around 1% and 2.5% for the outdoor tests. This result indicates that for indoor environments there is a smaller number of matching errors compared with outdoor. The relative errors in motion estimation drift can be seen in the table 4.2, for one indoor and one outdoor test. This result is expectable, since the feature matching in natural environments is less exact. For instance, it is harder to extract feature points from surfaces as: leafs, grass and tree logs, rather than from walls, doors and corners on manufactured environments.

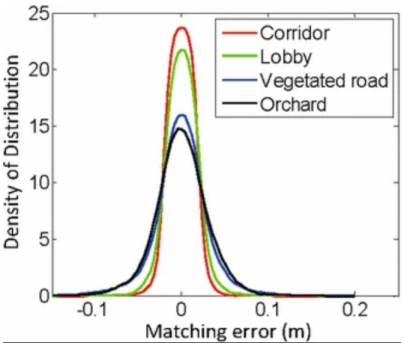

Figure 4.8: Matching errors in different test environments. Starting from: a tight and long corridor represented in red, a large room or lobby in green, a vegetated road in blue and an orchard among two rows of trees on black. The tests were performed at a speed of 0.5m/s. [40].

One of the purposes of the design of the LOAM system was to rely on laser range measurements with independence of inertial measurements. Even though not necessary, if an IMU is present, it can help to provide a better motion estimation for fast velocity changes. With this objective, a small test implementation with an IMU was made. This implementation consists in use of the orientation provided by the IMU, to pre-process the point cloud, before delivering it to the forward algorithms. The orientation is used to align the point cloud received from one sweep with the initial orientation of the LiDAR in that sweep. The acceleration measurements, are also used to remove part of the motion distortion. The IMU orientation is computed with the aid of a Kalman filter [53], by integrating angular rates provided by the gyroscopes and readings from the accelerometers. After this extra pre-processing step, the point cloud is then feed to the LiDAR odometry and mapping and processed as explained previously, on 4.2.1 and 4.2.2.

In the Table 4.3 the relative errors in motion estimation with the use and without use of the IMU can be compared. As previous tests the LiDAR is moving at a speed of 0.5 m/s. The overall

results show that, the use of the IMU assisted method gave the highest accuracy. While the use of IMU orientation only returned the lowest accuracy. This result shows that the IMU is effective in the handling of nonlinear motion. Whereas, the original method is a design to deal with the linear motion. This result is important to note, since the objective of this work is to explore a similar solution. It proves that the use of sensor fusion with inertial measurements can indeed help to improve the accuracy of the system. This implementation can be found in the next chapter 5.

Chapter 5

System implementation and results

As explained before, the LOAM system receives as input a 3D point cloud originating from a 3D laser scanner, then divides the work load into two different algorithms. The LiDAR odometry runs at fast speed, in order to be able to estimate velocity and simultaneously remove motion distortion in the point clouds in real time. But this comes with a cost- since the odometry works with fast speed and low amount of points, it has low-fidelity and cannot ensure accurate mapping. For this purpose, it is where the LiDAR mapping comes in. It works at slower speed, around a tenth of the speed of the odometry. It receives the undistorted point clouds and performs fine scan matching, with a big amount of points to ensure accuracy on the map. The combination of both algorithms ensures an efficient real time performance. The detailed theoretical explanation of the LOAM system can be found in the previous chapter 4.

The objective of this chapter is to explain in practical terms how the original loam system works and how it was implemented, jointly with the proposed improvements, as well as to present the obtained results. Firstly, the used hardware is revealed in the section 5.1.1, followed by the software 5.1.2. Secondly, the implementation and testing of the original loam algorithm is presented in the section 5.1.2.2. Finally, the implementation of the proposed improvements and results is shown in the section 5.3.

5.1 System overview

5.1.1 Hardware

A brief description for each hardware component is presented as follows:

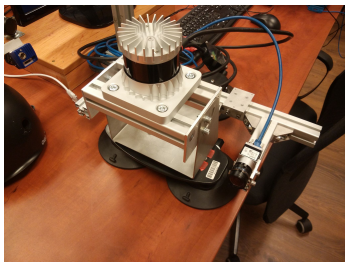

Figure 5.1: Ouster OS1, Mid-Range High-Resolution Imaging LiDAR.

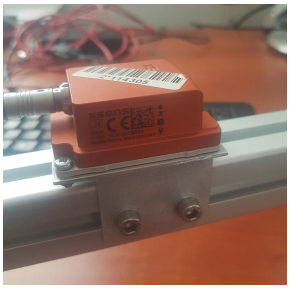

Figure 5.2: Xsens MTi-30 AHRS IMU.

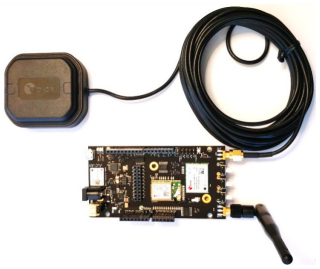

Figure 5.3: GPS C099-F9P board kit. [8].

- *Ouster OS1* Ultra-Wide View High-Resolution Imaging LiDAR [70]. This sensor is designed for indoor and outdoor all-weather environments. With a Range from 45m to 120m with accuracy of ±3 cm to ±10 cm depending on conditions. It is capable of a scanning Rate 10 or 20 Hz. The resolution is of 2,621,440 Points Per Second.

- *InvenSense ICM-20948* IMU [32]. This IMU is integrated in the OS1 LiDAR and is capable of 9 DOF motion tracking. It is MEMS based and constituted from: a 3-axis gyroscope, 3-axis accelerometer, 3-axis compass and a Digital Motion Processor. It is designed for Smartphones, Tablets, Wearable Sensors, and IoT applications.

- *Xsens MTi-30 AHRS* IMU [35]. This IMU is a Industry grade, MEMS based, with full-featured sensor fusion algorithm, capable 9-DOF motion tracking. constituted from: 3-axis gyroscope, 3-axis accelerometer and a Magnetometer.

- *C099-F9P* board with *zed-f9p ublox* module[8]. The *zed-f9p* positioning module is capable of multi-band GNSS, and rubust for high-volume industrial applications. It has an update rate up t0 20hz, position accuracy 0.01 m + 1 ppm CEP and Convergence time2RTK < 10 sec.

- *Melex 385H* [42]. This is a small electric powered vehicle with differential drive, used for ground tests at PUT [23]. The roof was adapted and a rigid frame was built for the assembly of a previously referred sensors.

Figure 5.4: Melex 385H electric vehicle.

Figure 5.5: Rigid frame assembly, monted on the Melex veicle with the: LiDAR, GPS and IMU sensors.

5.1.2 Software

5.1.2.1 The mobile robotic platform

For the implementation of this work the The Robot Operating System framework (ROS) [60] was used. It is an open-source set of software libraries and tools specifically designed for robotics applications and for an easy integration of several software parts needed for robot operation, from drivers to state-of-the-art algorithms. The main software libraries and tools used on this implementation, can be divided as follows:

- *Ubuntu 18.04.5 LTS (Bionic Beaver)* [76]: Operating system used on this work, chosen do to it's open source and stable build and good compatibility of the other chosen software libraries and tools.
- *ROS Melodic Morenia* [43]: The Ros distribution used on this work, chosen do to the stable build and good compatibility of the other chosen software libraries and tools.
- *Robot localization* [57]: This package is a collection of state estimation nodes, each of which is an implementation of a nonlinear state estimator for robots moving in 3D space. It is used to run the EKF node responsible of the sensor fusion.
- *Imu tools* [34]: Contains various tools for IMU devices, it is used to run the Complementary filter node responsible for the creation of orientation from the Ouster Imu data.
- *LOAM Velodyne* [24]: This is the main package containing the implementation of the algorithms necessary for the LiDAR Odometry And Mapping, namely LOAM.
- *Gps umd* [28]: This package is a space to stage messages and common GPS-processing routines. It is used to convert the raw GPS data, (longitude, latitude) in to odometry on the local frame.
- *Rviz* [62]: Main 2D/3D visualizer tool in ROS, used to visualize in 3D, the recorded data, live odometry and mapping trajectories.

- *Plotjuggler* [52]: It is a time series visualization tool. Used for 2D plotting and validation.
- *RGB-D benchmark* [14]: This is a collection of tools used to pre-process datasets and evaluate SLAM/tracking results. From here the Absolute trajectory error (ATE) scrip is used, in order to properly evaluate the results.

5.1.2.2 Architecture

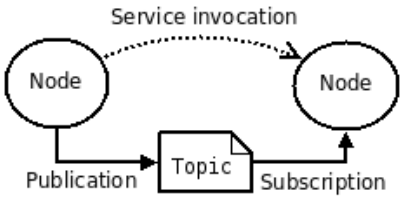

Figure 5.6: ROS basic computation concept.[59].

Since the implementation of this work is based on ROS, as first step, the basic principles of the ROS architecture are introduced. The figure 5.6 exposes the basic ROS computation concept [59]. This concept includes different elements, being the most notable: nodes, *Master*, messages, services, topics, and bags. Briefly, the nodes are the programs that are continually running and performing computations while simultaneously interchanging or not data between them, this process follows a publish-subscribing messaging system. The topics can be seen as communication channels, that contain the messages published (written) by nodes. The nodes can be publishers, in the case they intend to write messages, or subscribers if they want to have access (to read) the messages published on the topics of interest. On the diagram 5.6, this process is exemplified, where the node on the left wants to establish communication to the one on the right. It does it firstly by publishing a message on a topic, so afterwards the other node, that is subscribing to that topic, can read the message. The arrow pointing from a node to a topic means that the node publishes in the topic. The arrow pointing from a topic to a node means that the node is subscribed to the topic. The *Master*, which is the main node, stores topics and services registration information for nodes, ensuring their communication. The services are basic functions that can be called on ROS to ensure basic interactions on the system. Finally, the ROS bags are files of recorded information, that can be used to play ROS message data. They are very useful when it comes to the developing and testing of algorithms.

5.2 Original Implementation

With the ROS introduction accomplished, the original LOAM implementation can now be addressed. The UML-based illustration presented in the figure 5.7 exposes the implemented architecture sustained by ROS based concepts. This diagram represents the different elements of the system: ROS-nodes, ROS-topics and the interactions between them. Primarily, the nodes are represented by the ellipse shape and topics shown as rectangular boxes. In this implementation the

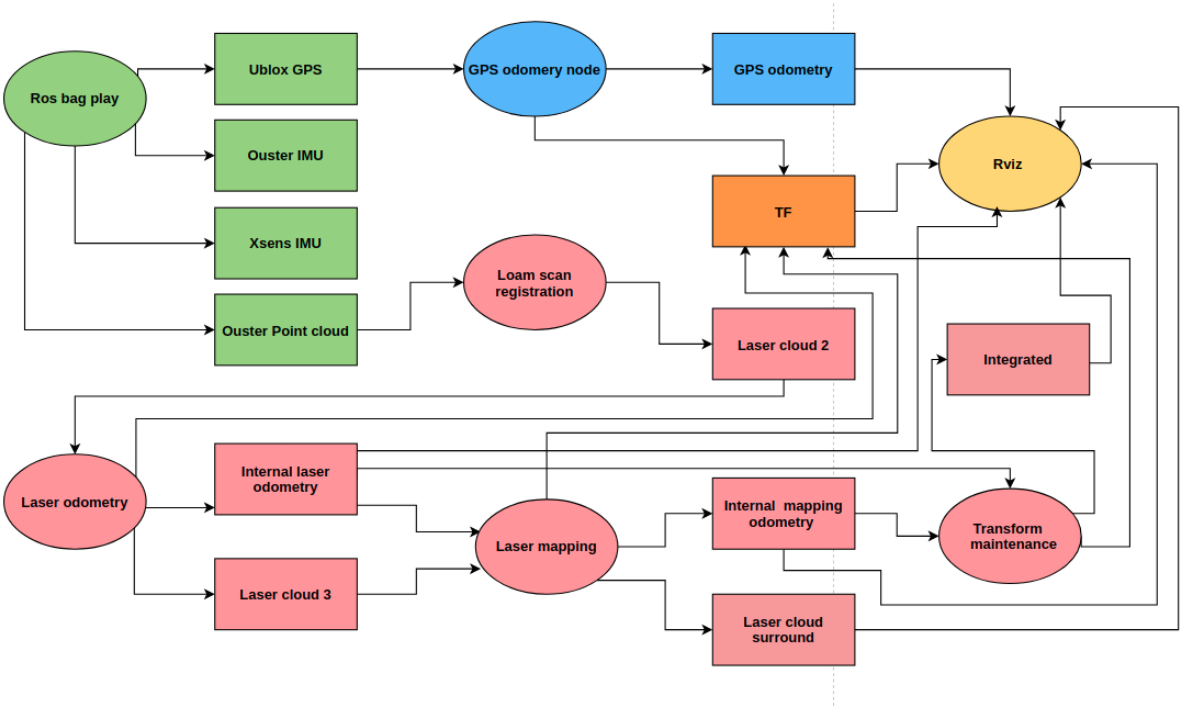

Figure 5.7: UML-based illustration of the implemented LOAM-ROS system. In the figure, the nodes are represented by the ellipse shapes and the topics are the rectangular boxes. The green color represents the *Ros-bag play* node and the related published data topics, the *GPS* node and GPS odometry are represented in blue. The LOAM nodes and respective published/subscribed topics are illustrated in pink, the orange color represent the transform topic and and *Rviz* node, respectively. Arrows represent the flow of information, in the form of subscribe-publish relations between nodes and topics.

access to live sensor data wasn't available, instead recorded data was used in the format of Ros-bags [58], this data contains all the data obtained simultaneously from all the sensors exposed previously. Starting by:

- The *Ouster LiDAR*: 3D point cloud.
- The *InvenSense IMU*: 3D linear accelerations, angular velocities and associated co-variances.
- The *ublox GPS*: geodetic location, latitude, longitude altitude and associated co-variances.
- External *Xsens IMU*: orientation and associated co-variances.

This data comes in the format of ROS-messages [46] that have their own standards, depending on the type of data. These bag files can be played and can publish messages exactly as the original sensors do. The *ROS-bag play* node and the previous referred published topics are the first element in the system, represented in green color. The following element in the system is the *GPS odometry node* and the published topic illustrated in blue. Here, the role of the GPS is not to improve the system by aiding in the pose estimation, but to provide a ground truth. Since this tests were performed in an open environment, the GPS provides an absolute trajectory with a good accuracy. Later on, this trajectory can be matched with the LOAM estimated trajectory and evaluate its performance. This node starts by subscribing to the *ublox GPS topic*, where it reads the published *NavSatFix* messages [65]. These messages encode the location in geodetic latitude ϕ, longitude λ and altitude h. This standard is converted to the local Ros-Map frame, in the ENU standard using the equations and matrix (2.11 to 5.1) presented in the chapter 2.1. Afterwards, the GPS node publishes the GPS estimated 6-DOF pose in the format of *nav_msgs/Odometry* [48] in the respective topic. Due to the fact that the GPS is not able to estimate orientation, this pose includes a "dummy" 3D orientation, represented with quarternion (x:0, y:0, z:0, w:1). Simultaneously, a TF message is published containing the associated geometry transformations between frames. The next element is the Loam system, denoted with pink. It is the most complex element, constituted by 4 distinct nodes and 6 topics that run the algorithms responsible for different computation steps in the Loam. The first step starts with the *Scan registration node*. This node is responsible for filtering the outlier points. It removes invalid points, which are considered to be too close or at infinity. Simultaneously, it assigns the time of registration of each point. These time stamps will be required later for the undistortion of the point cloud. This node subscribes to the ouster point cloud messages, that are encoded in the *velodyne_cloud_2* format [66]. Afterwards, it performs the previous computations and publishes the new filtered cloud in the topic called *Laser cloud 2*. The next steps involve the *Laser odometry* node. This node starts by subscribing the *laser cloud 2* topic. From this point, it reads the previous filtered laser clouds. As explained in the previous chapter 4.2.1, this node is responsible for estimating velocity and simultaneity remove motion distortion in the point cloud, after performing the need computations, it publishes the undistorted point cloud in the *laser cloud 3* topic. At the same time, it publishes its estimated 6-DOF pose on the *Internal laser odometry* topic and the associated geometric transform on the *TF* topic. At this point the *Laser mapping* node comes in, this node starts by subscribing the *Internal laser*

odomerty and *Laser cloud 3* topics. At this point, it is able to read the laser odometry published estimated pose and the undistorted point cloud. As explained in the previous chapter 4.2.2, this pose is used as initial guess, for the fine matching performed with the undistorted point clouds. As a final step, the mapping node publishes the matched point clouds, that represent the map itself in the topic *Laser cloud surround* and the estimated 6-DOF pose in the topic *Internal mapping odomertry*, along with the equivalent geometrical transform on the *TF* topic. To finalise, the last element is the *Transform maintenance* node, this node subscribes both pose estimates topics, from odometry and mapping, *Internal mapping odomertry* and *Internal laser odomerty*, respectively. It is responsible for integrating both poses estimates, as explained in the section 4.2.2.1. Both estimates are received at difference frequencies, the odomerty is received at 10hz and mapping at 1hz. To ensure smooth output and best accuracy, this node picks the mapping output at 1hz and fills the gaps with the fastest output from the odometry. This results in a final fast and smoother output at 10hz. This final pose is published in the *Integrated* topic and simultaneously the geometric transform is published in the *TF* topic. The *TF* topic is noted in a distinct color, orange. This topic is common to all the other elements in the system for publishing geometric pose transforms between frames. Finally, the last element in this system is the *Rviz* node, this node is responsible for creating a 2D/3D visualizing environment and displaying all chosen elements from system, as the estimated poses and point clouds. It accomplishes this by subscribing to all the output topics of the different elements in the system and by reading its equivalent geometric transform in the *TF* topic.

After this explanation, most of the work of the system should be well understood. To cover any other doubts, the figure 5.8 represents the ROS-transform tree of the system. This tree follows the standards introduced in the section 2.1.1 and exemplified on the figure 2.3. ROS defines a frame by default [74], as a coordinate system, that is always in 3D, right-handed, with X-forward, Y-left, and Z-up. The relation among two frames is represented by a 6- DOF relative pose, a translation proceeded by a rotation. Given an example if W and B are two distinct frames, the pose of B in W is represented by the translation from W origin to B origin, followed by the rotation of B coordinate axes in W. This rotations are calculated internally by ROS using rotation matrices, in similar ways and presented in the section 2.1.2.4. This tree represents the main frames used in this ROS implementation, where as explained in the previous paragraph, always when a node publishes pose data, it is associated to a frame and with its equivalent geometric transform to his parent frame. The LOAM system creates 2 different inertial coordinate systems, named as, *map* and *loam_map*, both right-handed. This can be better observed on the figure 5.10 where the *map* frame is the first as the ROS standards defines 2.1.1, and the *loam_map* is the second connected to it. The *loam_map*, as the name suggests, is the map frame that is used internally by loam topics, it serves to compensate mounting translations and rotations of the Laser sensor and allow better visualization on *Rviz*. Due to this fact, this frame uses a standard typically used on visual sensors, where the axis are organized as follows: the positive Y-axis pointing up, X-axis pointing left and Z-axis pointing forward to the direction the of motion. In practical terms this represents a positive rotation of 90 degrees around the x-axis compared to the parent frame *map*, the same happens

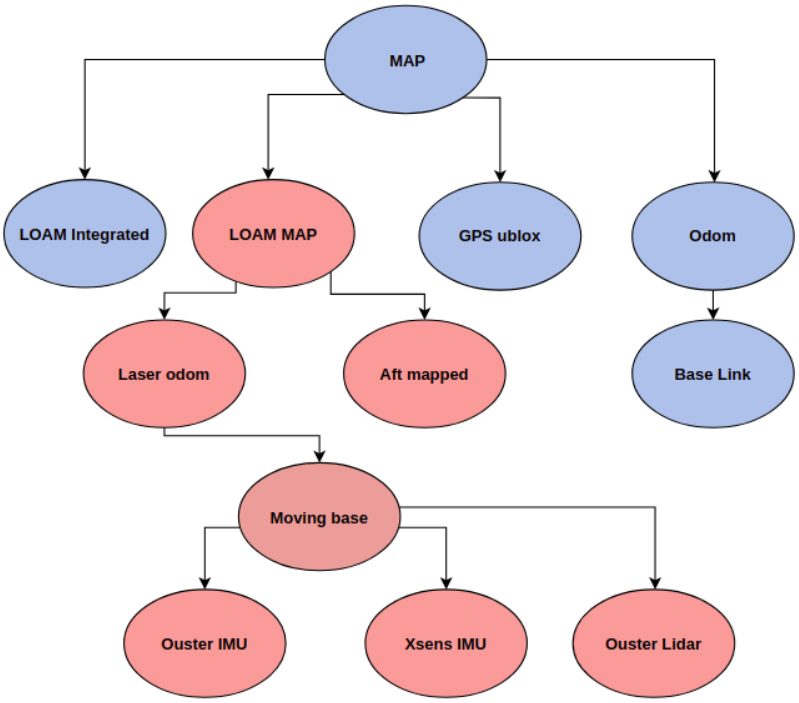

Figure 5.8: This figure represents the ROS main frames used actively for he system computations. Here the direction of the arrows are pointing from a parent to a child frame. The blue color represents the frames using the *map* standard, the positive Z-axis pointing up, X-axis pointing left and Y-axis pointing back. The pink frames are equivalent to the *Loam map*, the positive Y-axis pointing up, X-axis pointing left and Z-axis pointing forward.

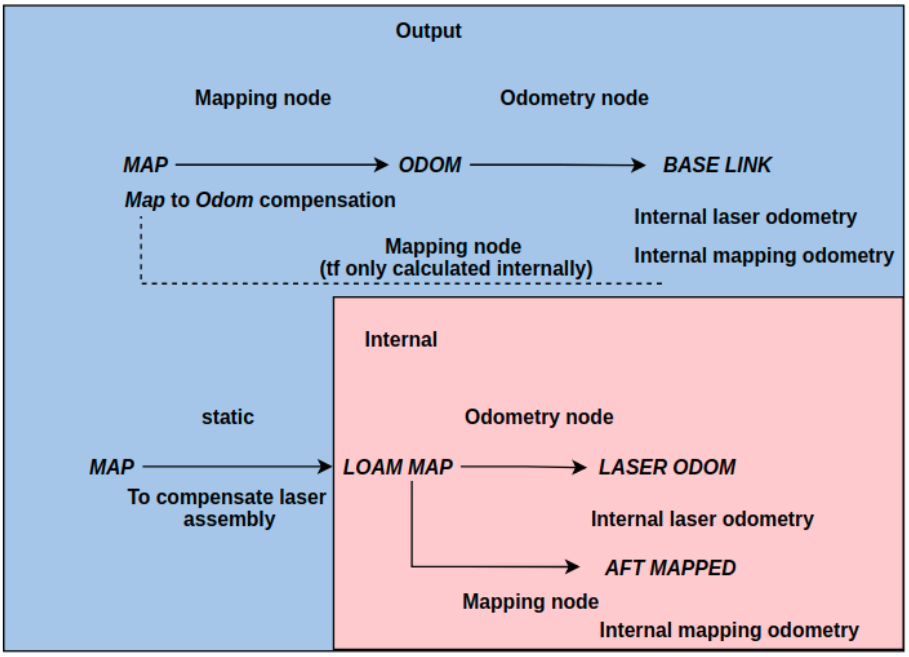

Figure 5.9: This diagram represents the different type of transforms between the frames and it's respective publishers nodes together with the published topics. The arrows represent the connections between frames, the pointing direction goes from the parent frame to the respective child frame. The blue represent the frames using the *Map* standard and the pink represents the *loam map*, respectively.

with all the frames connected after, this is represented by the pink color on the figure 5.8. The transforms that represent this relations can be static or dynamic, given as example, the transform between the *map* and *loam_map* is a static transform, defined bye the hardware assembly. In opposite the next transforms between the frames *laser_odom* and aft_mapped are calculated by the odometry and mapping nodes, respectively. The frames associated with sensors are also static transforms defined by the hardware assembly related to the center of the vehicle, this frame is represented as the *moving_base* frame. This frame is a auxiliary frame, added with the objective of attach the inertial sensors need for the next step of implementation, since the ROS Standard does not let sensors to be attached directly to a *odom* frame, it is created using a static transform, the physical frame can be seen in the picture 5.5. The *map* frame is not used by the LOAM nodes to perform computations, but as output for the "external" world. This is the function of the *Odom* and *Base link* frame, in case of a external entity needs to have access to the odometry pose or the center of the vehicle's pose. Here the odometry node publishes the *Base link* transform referent to *Odom* frame and the mapping node also publishes mapping estimated pose referent to the *Base link* frame. This creates a problem since the convention doesn't allows a ROS-transform tree to have two parents. Instead the transform between the *map* to *Base link* is calculated internally by the mapping node taking in account the previously computed transform from the *Odom* to *Base link* frame. The pose estimate in the *Odom* frame is an improved version of the internal odometry corrected by the mapping algorithm. This problem is better represented in the figure 5.9. The

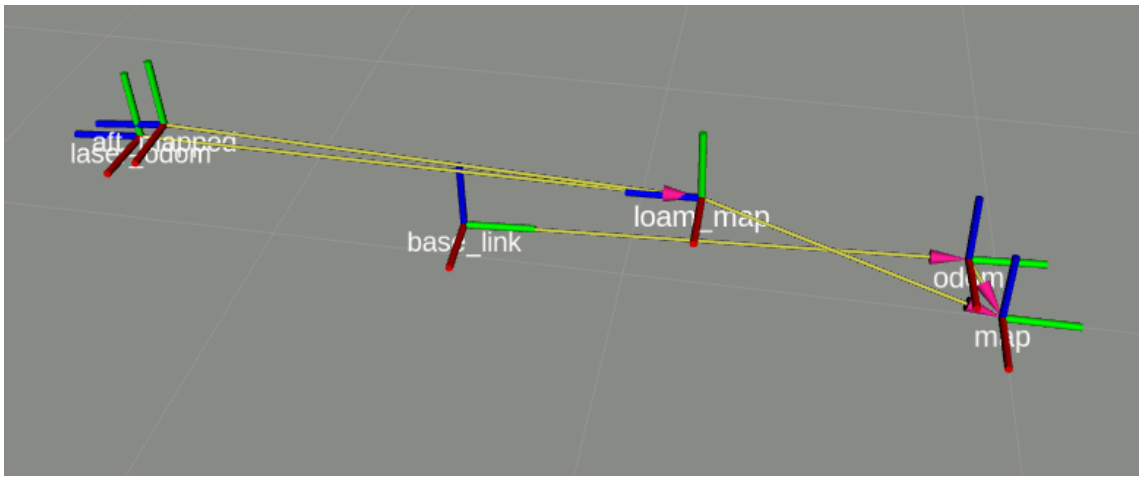

Figure 5.10: This figure is a *Rviz* screenshot taken on the *map* frame plane, exposing the main frames of the system. The x-axis is represented in red, followed by the y-axis in green and z-axis in blue. The arrows represent the connections between frames, the pointing direction indicates the parent frame.

remaining *GPS ublox* and *Loam integrated* frames represent the *GPS odometry* and *Transform Maintenance* node output frames, its transforms are produced by the respective nodes.

5.2.1 Experimental Results

With the Hardware 5.1.1 and software 5.1.2 presentation finished, the experimental results from the original system implementation can be exposed. Coming along with difficulties connected to implementations which were needing the correction in order to run correctly the system. The figure 5.11, represents an untreated point cloud from the LiDAR scanner.

During the first attempts to run the system, a problem was found that didn't allow to visualize the data from different sensors simultaneous, on *Rviz*. After close inspection of the data inside of the bag, it was found that messages coming from different sensors had different starting timestamps. This as due to the fact, that each sensor has it's own internal clock and they weren't previously synchronized. This would represent a problem, since the hardware step wasn't available, this synchronization couldn't be performed. Having only this data available, this would create problems specially in the case of the implementation of the improved version of the system, since the inertial measurements need to be synchronized with the point clouds, to successfully perform its integration on to the system. This problem was address by rewriting the timestamps of the messages inside of the bag. Using a simple algorithm represented in pseudo code 3. That reads each message and attributes a new timestamp using the global bag time.

After solving the previous problem, the first successful testing runs were performed. Here other problem was detected, with the alignment of the trajectories. Namely the GPS and the LOAM trajectory, these errors were mainly visible in the yaw angle or in the rotation of the ground floor around the Z-axis. This error can be seen on figure 5.12. In this place, the LOAM estimated trajectory follows the mapped road correctly and the GPS estimation goes diagonal to their side.

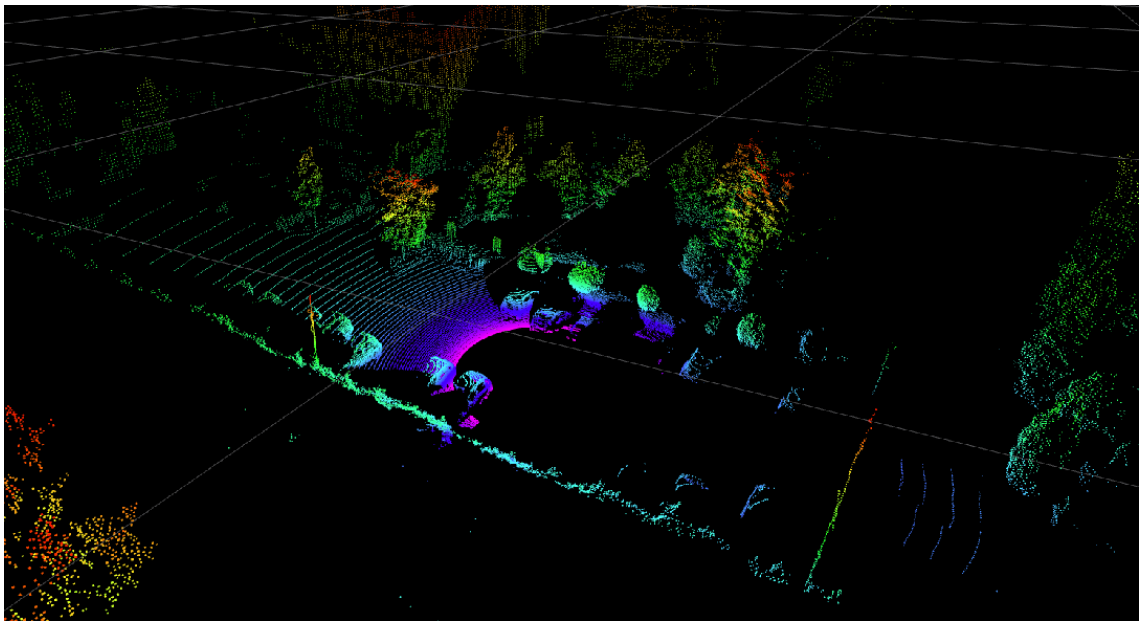

Figure 5.11: This figure is a *Rviz* screenshot taken on the *map* frame plane, exposing the output of the laser scanner. While performing the scanning of an outdoor environment, namely a street. Where the raw point cloud can observed, without any filtering and the capability of this scanner can noted, where there is enough resolution to identify several objects like cars, trees, walls etc. The colors represent the altitude of the points, start from cold colors in the ground, to warmer as the altitude increases.

Algorithm 2: LiDAR Odometry

Data: Read message timestamps from bag:

- The Ouster LiDAR 3D point cloud.
- The Ouster internal IMU.
- The ublox GPS.
- External Xsens IMU.

Result: Rewrite: all the previous messages timestamps.

```
initialization;
Open the bag file.
Start to read messages from bag.
while Didn't reach Bag end do
    for each message topic on the bag do
        if New message found then
            Read current global time of the bag T.
            Rewrite message timestamp with T.
        end
        Move to next topic.
    end
end
```

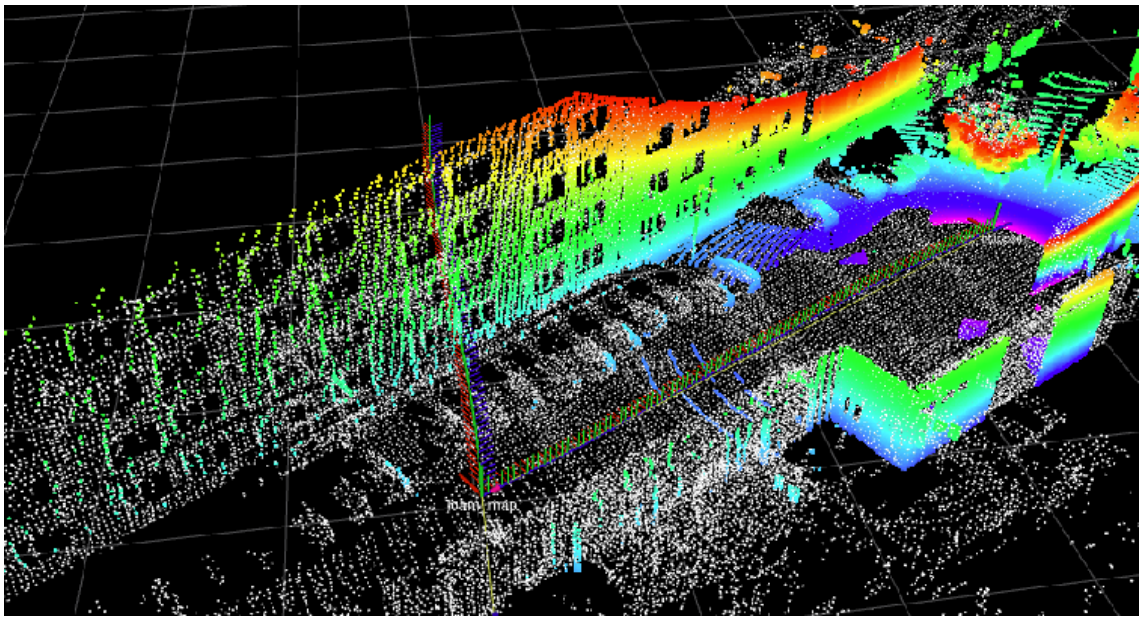

Figure 5.12: This figure is a *Rviz* screenshot taken on the map frame plane. While performing the scanning of an outdoor environment, namely a street. Here, the live laser output can be observed, noted by the colored points, simultaneity the mapped points are represented in white. The LOAM estimated trajectory can be seen, as the one aligned with the street. The GPS estimated trajectory is the one going diagonally with the street.

With a closer inspection, it was found that both algorithms were working as expected and above mentioned problem was originated from their own different mechanism of pose estimation. When it comes to the case of the GPS, it estimates an absolute pose on the planet, that is aligned with the Cardinal points. The LOAM does not have any feedback from where it is related to the earth, it's local map will have it's origin and alignment with the starting point of the run. After apprehension of the problem, it was decided that the best way to solve it, in order to be able to correctly mach the the points of both trajectories and calculate the accuracy of LOAM trajectory, was to edit the GPS node algorithm. It was done after the conversions from geodetic coordinates to Earth Centred Earth Fixed (ECEF) and ECEF coordinates to ENU. Since the rotation to be performed was only in the 2D plane, the problem was fixed by applying a 2D rotation with the desired offset (θ) on the the X-coordinate and Y-coordinate, using the following matrix.

$$\begin{bmatrix} Xaligned \\ Yaligned \end{bmatrix} = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix} \tag{5.1}$$

Finally, after the correction of the alignment of the previous obstacle, the system should be ready for accuracy testing. However, after many repetitive runs, another problem was discovered. In every run the output of the odometry and consequently the mapping were showing slightly different trajectory drifts, in the exactly same conditions. The first hypothesis, was that the computer used for running the algorithms did not have enough processing power to ensure the computation

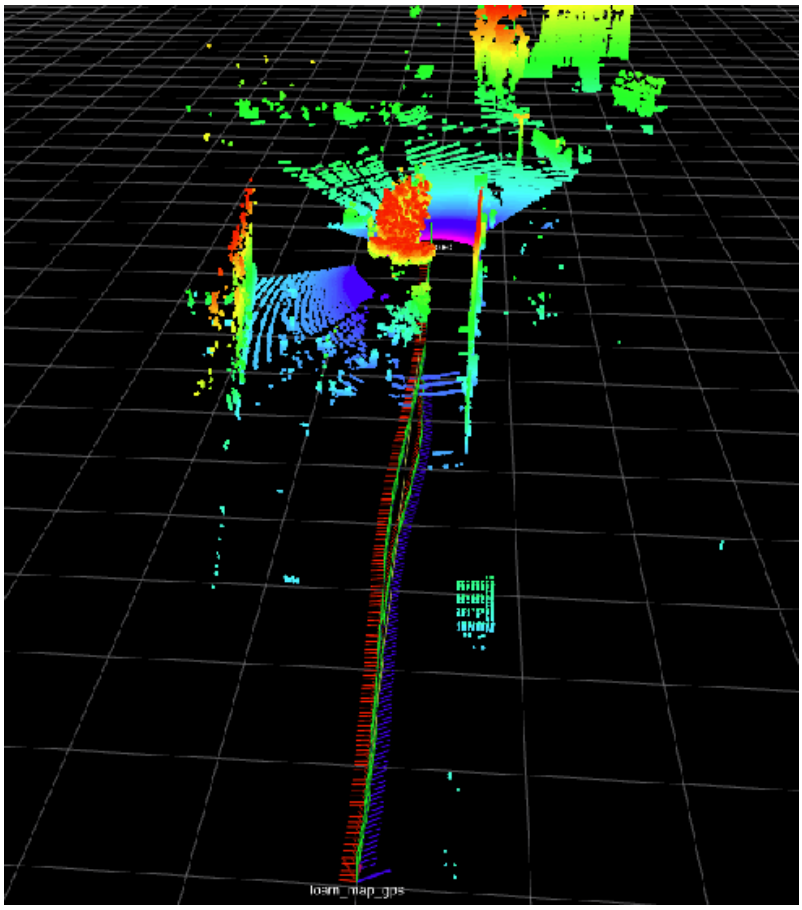

Figure 5.13: This figure is a *Rviz* screenshot taken on the *map* frame plane. Taken in the same conditions as the figure 5.12. With the particularity that the mapped points were removed for better visualization and the the figure was aligned with the trajectories. Here, the GPS estimated trajectory is the straightest or the one identified with red in the left side, the LOAM is the one making the small curve or with blue on the Right side. This behaviour is expected since the GPS trajectory should be the one containing less distortion.

needed to perform the runs with the data being played at full speed. Even tough the computer used in the current work is equipped with a similar hardware (*Intel Core i7-4700HQ 2.4GHz* processor and *8GB RAM*), than the one originally used when LOAM was firstly developed (*2.5GHz quad core processor* and *6GB RAM*), the LiDAR used here has much more dense point clouds. This could result in a slower processing speed. After this conclusion, the first solution was to slow down the playing of the bag. It was performed by using the rate parameter on the Ros-bag play node. However, after several tests with a speed slowed 10 times and after with 100 times, there was still occurring significant difference. Knowing those results, the conclusion was that it did not constitute the problem. The next candidate was the Ros-bag player node itself, since every time the system is being run, there is a sequence of events that start by: the system nodes initializing, them the bag starts playing, after this the nodes finish to subscribing to respective topics and finally the first messages are received by the nodes callbacks. With this, the timing of the messages being played, combined with time the nodes actually completes initialization, means that there will be always some variation from run to run. Affecting mainly the odometry algorithm were exact points being perceived will arrive with slightly different timestamps resulting in different point matching and and motion distortion. After apprehension of the problem, it was decided that the best way to solve it, would be to implement a custom Ros-bag player, that cloud wait for the mapping or odometry nodes to finish the processing of a scan, and them release the next set of messages. For this a feedback loop was introduced, were the Bag playing node subscribes to the outputs topics of the odometry. This new algorithm is presented in a format of a pseudo code, Algorithm 3, to allow the easier understanding. With this new bag player, the previous problem was solved. It allowed the playing of the bag at full speed, while maintaining consistent results, for runs performed in the same conditions.

The figure 5.14 represents the resulting Loam, odometry and mapping trajectory estimates, together with the GPS ground truth in the form of 2D plots using the *Plotjuggler* tool. Here on the top, the 2D ground plane (x/y) is represented and in the bottom the altitude (z). Can be easily observed that at the ground level, the mapping (blue) is almost overlapped witht the GPS (green), this means that the mapping achieved a good estimate, for the other side the odometry (pink) shows a big drift especial in the Y-axis. At the altitude estimate, clearly the odometry achieved a poor performance. To better compare the mapping estimate, on the figure 5.15 the odometry was removed. It can be seen that, although the mapping achieved a better performance it is far from perfect since the GPS indicates that the ground has a small slope, less than 2m, the mapping estimate a slop of more than 6m.

5.3 Improved implementation

Upon the original LOAM implementation explained and tested, the next step is to start implementing the proposed improvements. As it was observed in the previous runs, the Z-axis is the one that suffers from more drift. In this approach, the inertial measurements will be used to give a more accurate orientation heading to try to minimize this effect. The initial idea is to experimentally

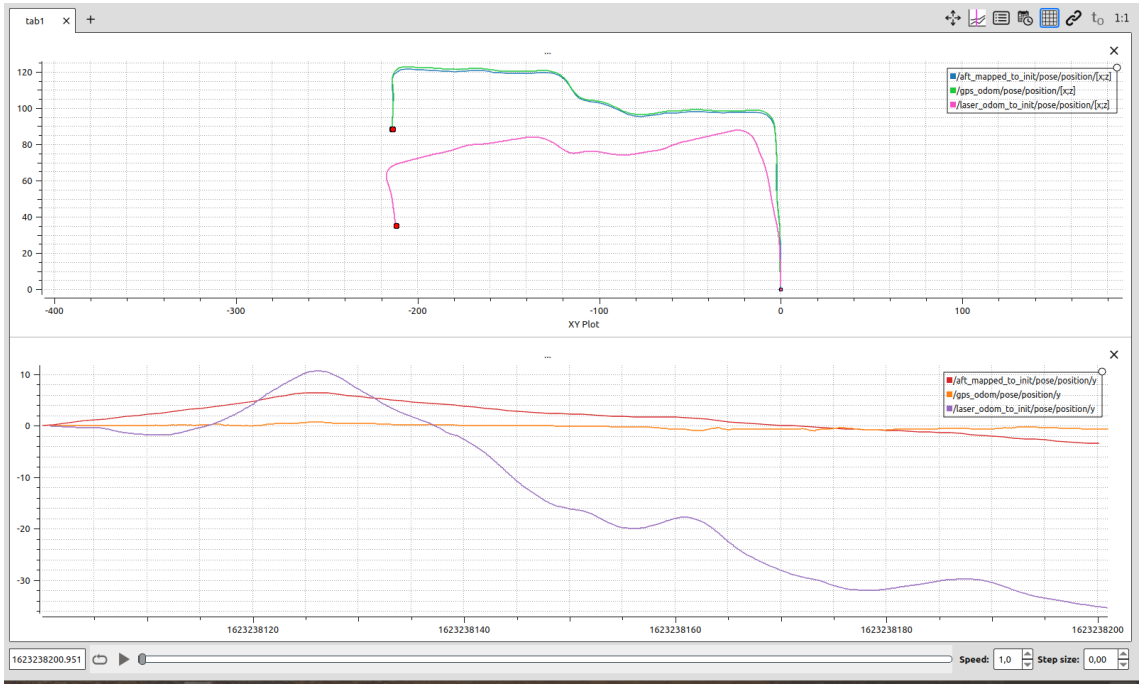

Figure 5.14: This figure represents the resulting LOAM liDAR odometry (pink) and mapping (blue) trajectory estimates, together with the GPS (green) ground truth in the form of 2D plots using the *Plotjuggler* tool. Here on the top, the 2D ground plane (x/y) is represented and in the bottom the altitude (z).

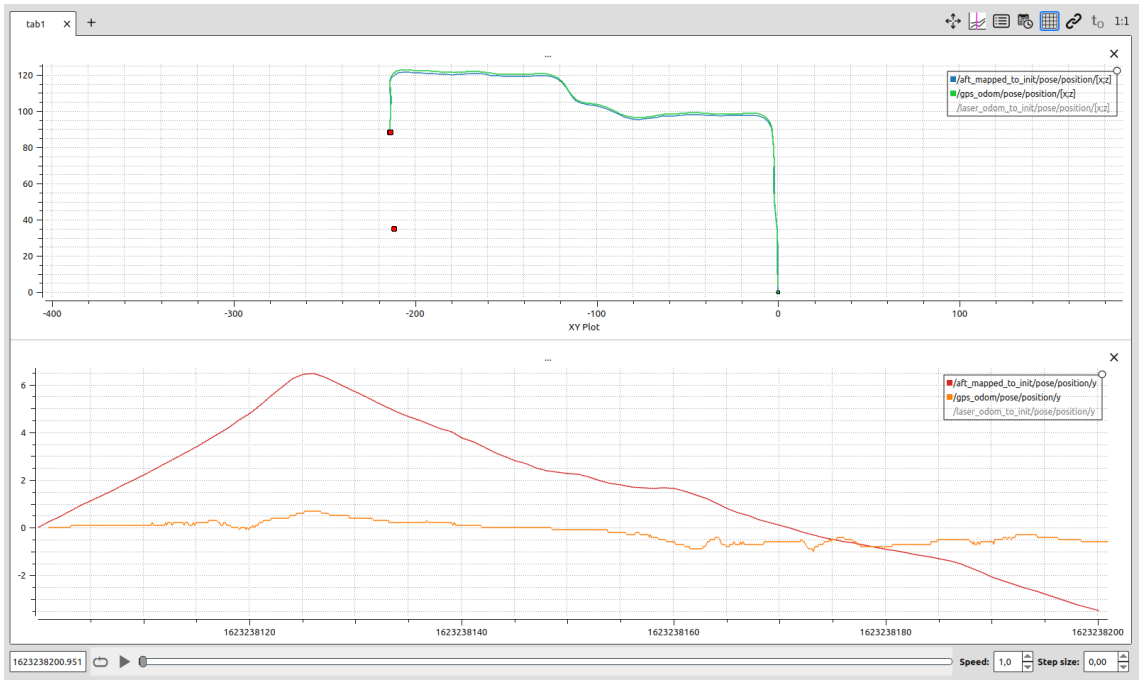

Figure 5.15: This figure represents the resulting LOAM liDAR mapping (blue) trajectory estimate and the GPS (green) ground truth in the form of 2D plots using the *Plotjuggler* tool. Here on the top, the 2D ground plane (x/y) is represented and in the bottom the altitude (z).

Algorithm 3: LiDAR Odometry

Data: Read messages from bag:

- The Ouster LiDAR 3D point cloud.
- The Ouster internal IMU, 3D linear accelerations and angular velocity's.
- The ublox GPS, geodetic location, latitude, longitude altitude and associated co-variances.
- External Xsens IMU, 3D linear accelerations, angular velocity's and orientation.

Result: Publish: all the previous messages.

```
initialization;
while odometry node not initialized do
    if odometry node initialized then
        Subscribe to Lasercloud3 topic.
        Start to read and store messages from bag.
        if Point cloud message found then
            publish the previously stored set of messages,
            including the current Point cloud message.
        end
        break
    end
end
while Didn't reach Bag end do
    if New Lasercloud3 message received then
        Start to read and store messages from bag.
        if Odometry message found then
            publish the previously stored set of messages,
            including the current Point cloud message.
        end
    end
end
```

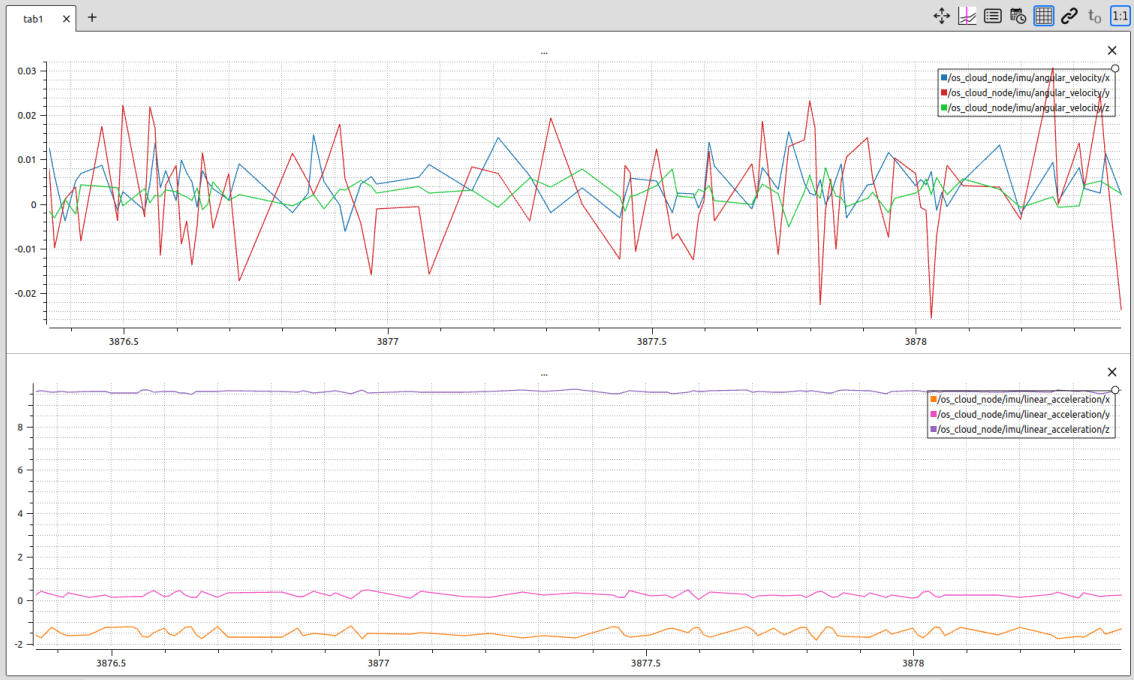

Figure 5.16: This figure exposes a plot using the *Plotjuggler* tool. Here, the linear accelarations and angular velocities can be observed. These measurements were taken at the beginning of the trajectory with the vehicle in an immobile state, using the *InvenSense IMU* 5.1.1.

evaluate if the IMU integrated on the LiDAR has enough precision to be used for this purpose. Bearing that in mind, the IMU output can be observed in the figure 5.16. The figure 5.16 presents a plot of the linear accelarations and angular velocities. These measurements were taken in the beginning of the trajectory with the vehicle in a immobile state. At this point, the plotted corresponding accellarations and angular velocities should be 0, excluding the gravity vector. However, as it can be observed, there is considerable amount of high frequency noise. This result is expected, since it was known from the previous chapter 2.2.2- this technology is very vulnerable to noise. In order to minimize this, the measurements will not be fused directly, they will be pre-processed first.

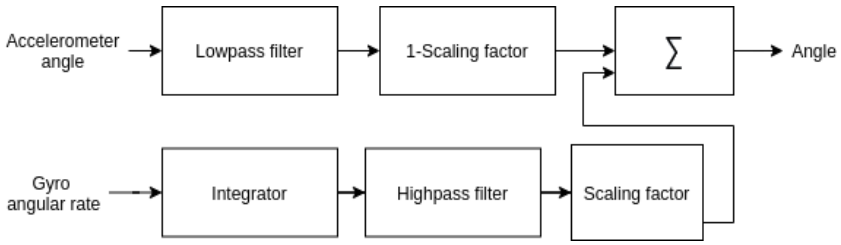

Figure 5.17: This figure exposes a simplified block diagram, representing the complementary filter.

5.3.1 Complementary filter

For this purpose a Complementary filter was chosen, due to its simple implementation, light computation demand compared with other approaches using Kalman filters [77], while simultaneously achieving a good performance. The working principle [67] of this filter is summarized in the figure 5.17, where it starts by extracting the angles from the gravity vector from the accelerometer measurements. Then the angles are filtered with a low-pass filter. In parallel, the angular rate measurements from the gyroscope are integrated and to finalize, filtered through a high-pass filter. Since the frequency response of the low-pass and high-pass filters need to add up to 1 at the frequency domain, a scaling factor is multiplied to ensure this requirement. The final step is to sum both outputs. The following equation 5.2 summarizes this relations:

$$\theta_{new} = \alpha \cdot (\theta_{old} + \theta \cdot \Delta t) + (1 - \alpha) \cdot \beta \tag{5.2}$$

where,

θ_{new} = new output angle

α = scaling factor.

θ_{old} = previous output angle.

Δt = sample time.

θ = gyroscope rate.

β = accelarometer angle.

For the implementation of this filter, the previously exposed *IMU tools* [34] Ros-package was used. It offers a configurable implementation of a complementary filter, with ability to perform auto bias estimation and remove noise within chosen thresholds. The scaling factor used was a α of 0.99, that corresponds to a gain of 0.99 to the gyroscope rate and 0.1 to the accelarometer angle. The chosen thresholds were equal to, 0.2rad/s for angular Velocity and 0.1m/s^2 for acceleration. The considered gravity is 9.81m/s^2. After the configuration, the filter was successfully tested and the plot of a small trajectory (90s) can be seen on the figure 5.18 and 5.19. This output should result in a more reliable and precise orientation compared with the one provided by the laser odometry itself. To verify this, the output of the complementary filter was compared with the output of the *Xsens IMU*. Since this IMU is considered industrial grade, it should have better precision than the *InvenSense IMU* present in the *Ouster LiDAR*, that is a simpler cheaper consumer grade IMU. In the first figure, the existence of a constant offset in the yaw and pitch angles can be easily observed. This can be due to differences in the mouthing position of both IMUs and also due to the fact that *Xsens IMU* is equipped with a Magnetometer that can set the reference of the yaw angle to be relative to the orientation of the earth magnetic field. Since these offsets are static they can be removed for better comparison of the orientations, after the compensation of these offsets (+138 degrees to yaw and +4 degrees to the pitch in the *Xsens IMU*). It can be observed that besides a small expected noise on the complementary filter, both outputs match, confirming that the filter is working properly.

Figure 5.18: This figure exposes the output orientation of the complimentary filter, labeled as (imu/data) and the Xsens IMU labeled as (xsens/data). The orientation is exposed in Euler angles (pitch , roll and yaw) measured in degrees.

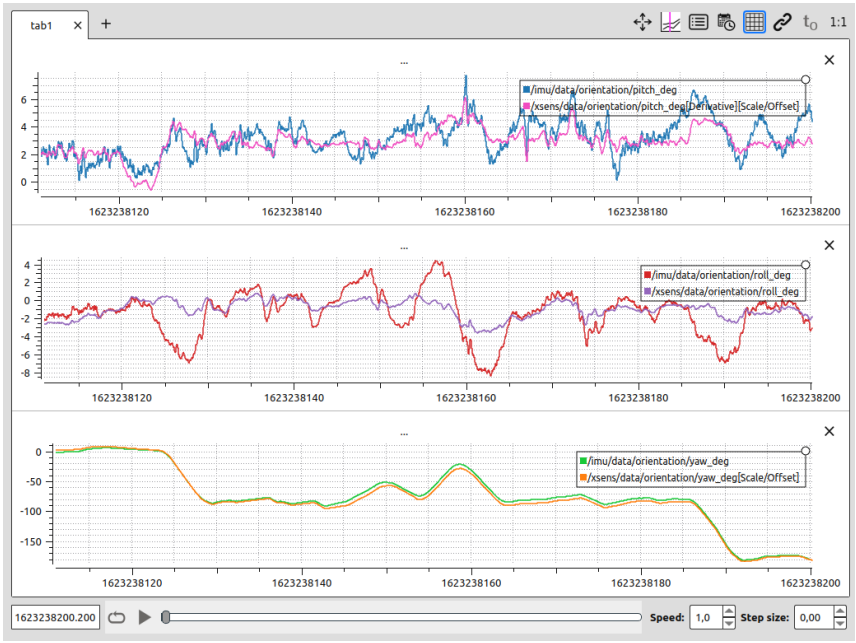

Figure 5.19: This figure exposes the output orientation of the complimentary filter, labeled aas (imu/data) and the Xsens IMU labeled as (xsens/data). The orientation is exposed in Euler angles (pitch , roll and yaw) measured in degrees. In this case the offsets in the yaw and pitch angles were compensated for a better comparison.

```
# Frames
map_frame: loam_map
odom_frame: laser_odom
base_link_frame: moving_base
world_frame: laser_odom
# other params
frequency: 10
publish_acceleration: true
predict_to_current_time: false
two_d_mode: false
transform_time_offset: 0
print_diagnostics: true
debug: false

# sensor configuration
# ordered as x, y, z, roll, pitch, yaw, vx, vy, vz, vroll, vpitch, vyaw, ax, ay, az

imu0: /xsens/data
imu0_config: [false, false, false,
              false, false, false,
              false, false, false,
              false, false, false,
              false, false, false]
imu0_differential: false
imu0_queue_size: 10
imu0_remove_gravitational_acceleration: false

imu1: /imu_ordered/data
imu1_config: [false, false, false,
              true, true, true,
              false, false, false,
              false, false, false,
              false, false, false]
imu1_differential: false
imu1_queue_size: 10
imu1_remove_gravitational_acceleration: false

odom0: /laser_odom_to_init
odom0_config: [true, true, true,
                false, false, false,
                false, false, false,
                false, false, false,
                false, false, false]
odom0_relative: true
odom0_differential: false
odom0_queue_size: 10
```

Figure 5.20: This figure exposes the parameters of the *ekf localization node*, used in this work.

5.3.2 Extended Kalman Filter

In the moment that the orientation from the IMU is ready to use, there is still the lack of the mechanism to fuse the new orientation estimate. It will be done with the objective of improving the pose estimated through the odometry algorithm. At this point, the extended kalman filter is introduced. The theoretical explanation of this filter can be found in the 3.3 section. Here, only the practical implementation will be addressed. For the implementation of this filter, the previously exposed *Robot localization* [57] Ros-package was used. From this package the *ekf localization node* was used, this node offers configurable fusion of an arbitrary number of sensors with different modes of operation. This node is based in the implementation of an extended kalman filter using the ROS-framework, that uses an omnidirectional motion model to project the state forward in time, and corrects that projected estimate using perceived sensor data. This internal model is a standard 3D kinematic model derived from Newtonian mechanics. The characteristics of robot state are encoded in a 15-dimensional state vector, X. That comprises the vehicle's 3D pose, 3D orientation, and their respective velocities, and linear acceleration. This details can be better viewed in the 3.1.1 section.

The configuration of the sensors is made using a configuration matrix that has the same variables as the state vector, where simply a false or true is filed to tell to the filter that the specific measurement should be fused. For easier explanation, the configuration parameters used in this work can be seen on the following figures 5.20 and 5.21. First, for the correct working of the filter the desired frames need to be specified, that follow the previous explained Ros-standards 2.1.1. The *loam_map* was chosen to represent the map frame, this was specified because the measurements that are going to be computed in the filter are going to be given back to the mapping node This node is using the frame as reference, creating the need to keep the vectors of the measured components in agreement to the respective original axis. The same is true for the odom frame defined as *laser_odom* and base link frame *moving_base*. The world-frame is a special parameter that defines the internal mode of working. In the case of fusing only continuous position data, such as odometry or IMU data, that is the present case, the world frame needs to be defined as the same value of the odom frame. If the sensor fusion is to be performed with global absolute position data, that is subject to discrete jumps, like in the use of GPS. The world frame should be defined with the same value as the map frame. This two modes will define in which frame the estimated output pose should be published and which respective transform to compute and publish. As next, there are the optional parameters, where the $frequency$ parameter can be noted, that is the operation rate, in frequency (Hz), at which the filter produces a state estimate. The *two_d_mode*, that defines if the internal model will work only in two dimensions. The explanation of the remaining parameters can be found at [73]. The following parameters, are the sensor configuration, where the first step is to specify the topic to be subscribed in order to successfully read the desired sensor messages. In this case, the first sensor is the *Xsens IMU*, being the topic to be subscribed the $xsens/data$, then on the matrix the desired measurements to be fused should be set to true having in attention the order of the variables. In the case of the *Xsens IMU*, it was not currently been used, with this all the components are false. As next, the $differential$ parameter defines the fusion mode, if it is set to true, the measurement is fused deferentially. In practical terms, this means for a measurement at time t from the sensor in question. Firstly, it is subtracted from the previous measurement at time $t-1$, then the result is converted to a velocity and finally fused. This parameter is not currently being used. The *queue_size* is the size of messages to store in the buffer. Last but not least, the *remove_gravitational_acceleration* is a parameter that allows the automatic removal of the acceleration due to gravity, from linear acceleration data. Since only orientation is fused, this parameter is not used. The next sensor is the orientation from the complementary filter, that has the same configurations as the previous IMU. With the exclusion of the subscribed topic, that in this case is $/imu\_ordered/data$ and the components related to the orientation, pitch, roll and yaw, respectively are set to true. As final input of the filter, there is the laser odometry at the topic $/laser\_odom\_to\_init$. On the matrix only the pose, x, y and z components are set to true. The remaining parameters are the same of the previous sensors with the exclusion of the $relative$ parameter, that can be used, if defined to true, to set the initial conditions to 0. As last, on the figure 5.21 the noise covariance matrices can be observed namely, the process noise known as Q and the initial estimate referred as P0.

```
# ordered as x, y, z, roll, pitch, yaw, vx, vy, vz, vroll, vpitch, vyaw, ax, ay, az. Defaults to the matrix below if
# unspecified.QQ
process_noise_covariance: [1, 0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
                           0,    1, 0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
                           0,    0,    1, 0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
                           0,    0,    0,    1, 0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
                           0,    0,    0,    0,    1, 0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
                           0,    0,    0,    0,    0,    1, 0,    0,    0,    0,    0,    0,    0,    0,    0,
                           0,    0,    0,    0,    0,    0,    1, 0,    0,    0,    0,    0,    0,    0,    0,
                           0,    0,    0,    0,    0,    0,    0,    1, 0,    0,    0,    0,    0,    0,    0,
                           0,    0,    0,    0,    0,    0,    0,    0,    1, 0,    0,    0,    0,    0,    0,
                           0,    0,    0,    0,    0,    0,    0,    0,    0,    1, 0,    0,    0,    0,    0,
                           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    1, 0,    0,    0,    0,
                           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    1, 0,    0,    0,
                           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    1, 0,    0,
                           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    1, 0,
                           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    1]

# ordered as x, y, z, roll, pitch, yaw, vx, vy, vz, vroll, vpitch, vyaw, ax, ay, az. Defaults to the matrix below
#if unspecified.PO
initial_estimate_covariance: [0.1, 0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
                              0,    0.1, 0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
                              0,    0,    0.1, 0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
                              0,    0,    0,    0.1, 0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
                              0,    0,    0,    0,    0.1, 0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
                              0,    0,    0,    0,    0,    0.1, 0,    0,    0,    0,    0,    0,    0,    0,    0,
                              0,    0,    0,    0,    0,    0,    1, 0,    0,    0,    0,    0,    0,    0,    0,
                              0,    0,    0,    0,    0,    0,    0,    1, 0,    0,    0,    0,    0,    0,    0,
                              0,    0,    0,    0,    0,    0,    0,    0,    1, 0,    0,    0,    0,    0,    0,
                              0,    0,    0,    0,    0,    0,    0,    0,    0,    1, 0,    0,    0,    0,    0,
                              0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    1, 0,    0,    0,    0,
                              0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    1, 0,    0,    0,
                              0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    1, 0,    0,
                              0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    1, 0,
                              0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    1]
```

Figure 5.21: This figure exposes the noise covariance matrices parameters of *ekf localization node, used in this work.*

These covariances were defined experimentally with high values, in order to help the filter converge fast when new measurements were added. With the parameterization explained, the figure 5.22 exposes the computation steps, namely rotations or necessary conversions, to perform by the EKF node to fuse a new measurement. This is required due to the fact, that the state vector components are referenced to Wold frame, by opposite the sensors measurements are referent to the body frame of the vehicle, with the exception of the LiDAR linear position. Now, that the necessary components to fuse with the inertial measurements were added to the system, namely the complementary filter and extended kalman filter nodes. The final system architecture can be observed in the new UML-based illustration presented in the figure 5.23. It exposes the improved implemented architecture sustained by ROS based concepts. In this diagram the recently added *complementary filter* node and topic can be observed with violet color, subscribing to the *Ouster IMU* topic and publishing on the *IMU data* topic and the corresponding geometric transform. The *Extended kalman filter* node and the related topic are illustrated with gray color. This node starts by subscribing the output of the complementary filter (3D orientation) and the estimated output of the odometry node (6-DOF pose) and then reads the associated geometric transforms from the *TF* topic. After performing the necessary computations explained in the previous section 5.3.2, the filter publishes its new 6-DOF pose estimate at the *Improved odometry* simultaneity with the respective transform at the *TF* topic. Now, the *Laser mapping* and the *Transform maintenance* nodes instead of subscribing to the odometry estimated pose, they receive the new improved pose from the *Improved odometry* topic. Here is also visible the new *Ros bag play* node, with the addition of feedback from the *Internal laser odometry* topic, as explained on the section 5.2.1.

Providing all the explanation, the improved system was finally tested. The figure 5.24 is a *Rviz* screenshot, on the map level, that displays the estimated trajectories- from the GPS, LiDAR mapping, LiDAR odometry and improved odometry nodes, in a outdoor scenario. It can be noticed, that in this case both odometry estimates are overlapped, meaning that the new estimate did not

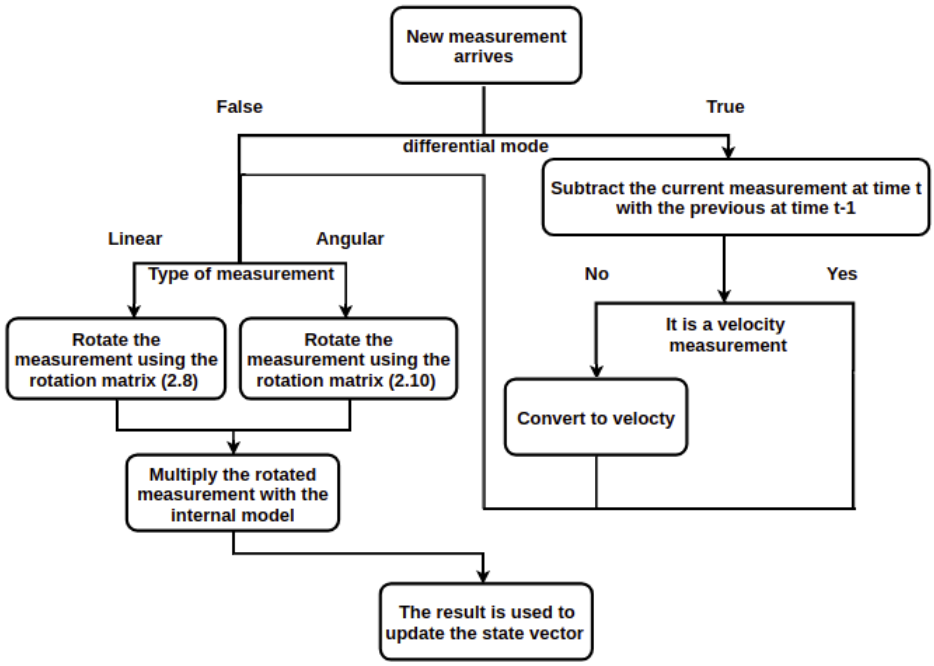

Figure 5.22: This figure exposes the computation steps performed by the *ekf localization node*, in a flowchart style, necessary to apply to the measurements in order to fuse them on the system.

show improvements. After a closer look to the problem, it was discovered that the odometry estimates were overlapped in terms of position, however the pose estimated with the Kalman had the improved orientation from the complementary filter. This means, that the EKF node is performing the fusion. However, the original purpose of improving the LiDAR odometry location estimate was not working. The reason behind this was the internal model of the Kalman itself. After a more detailed evaluation of the equations present at 4.10, that represent the kinematic model of the system and also after analysing the rotations matrices (2.8 and 2.10), that perform the necessary geometric transformations between frames, it was found that on the linear position equations, that depend from: the position, velocity and acceleration. Only the position was being given, refereed to the map frame, this value does not require any rotation and is independent of the orientation. This means, that in order to correct the problem, the internal model needs to be changed or the measurements need to be fused in a different mode. At this point, the $differential$ mode can help, by fusing the position as velocity. It will be depending also on the angular components. Knowing this, the $differential$ parameters were turned on and the system run again, achieving the desired results.

5.4 Evaluation

Having all the previous problems solved, the system can be finally successfully evaluated. At this point, to be able to compare the different trajectories, firstly they need to be run and saved as a file.

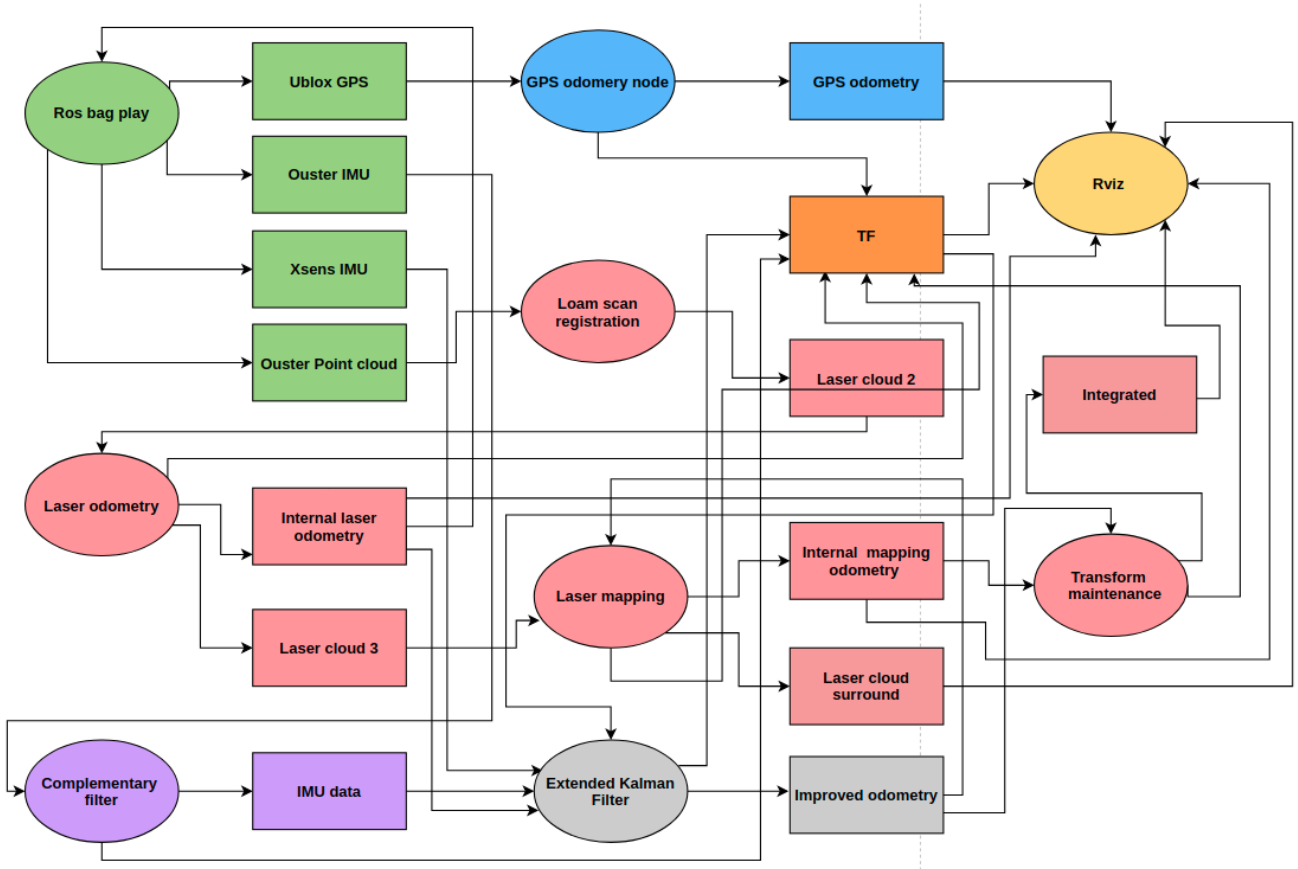

Figure 5.23: UML-based illustration of the improved implemented LOAM-ROS system. In the figure, the nodes are represented by the ellipse shapes and the topics are the rectangular boxes. The green color represent the *Ros-bag play* node and the related published data topics, the *GPS* node and GPS odometry are presented in blue. The LOAM nodes and respective published/subscribed topics are illustrated in pink, the orange color represent the transform topic and *Rviz* node, respectively. The violet color represents the *complementary filter* node and topic. To finalize, the gray color shows the *extended kalman filter* node and topic. Arrows represent the flow of information, in the form of subscribe-publish relations between nodes and topics.

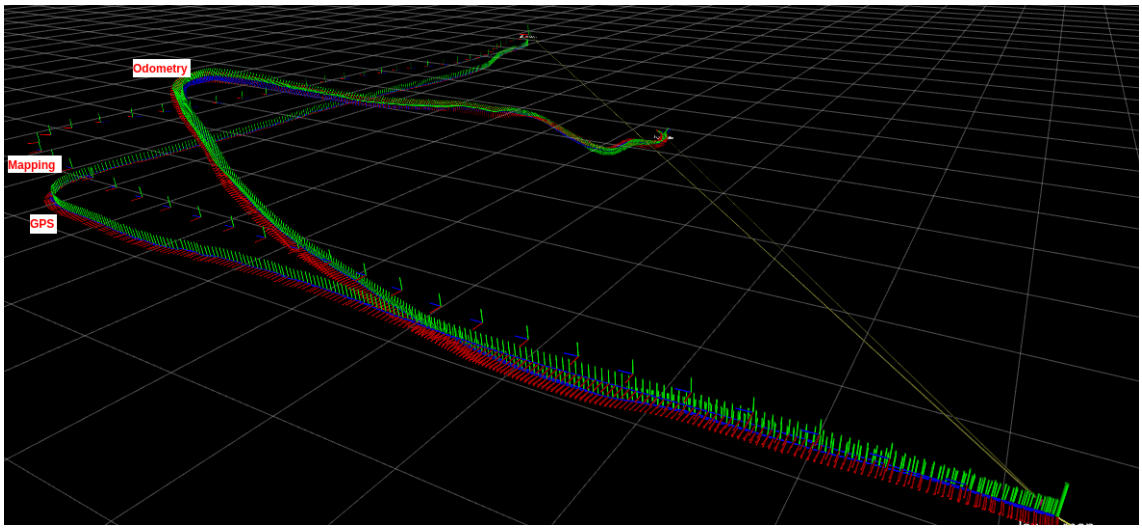

Figure 5.24: This figure is a *Rviz* screenshot taken on the map frame plane. Exposing the estimated trajectory, from the GPS, LiDAR mapping, LiDAR odometry and improved odometry nodes in a outdoor scenario. Both odometry estimates are overlapped.

For this purpose, a small modification to the odometry, mapping and GPS nodes need to be performed. That consists in saving all the estimated trajectories in a text file during each run. With the trajectories acquired, the next step is to chose the evaluation metric. In order to do that, the absolute trajectory error (ATE) is used. This metric consists of the direct measure of the difference between points of the ground truth and the estimated trajectory. Before this difference can be measured, it is necessary to perform a pre-processing step. This process starts with the use of each timestamp, to associate the estimated individual poses with the individual poses from the ground truth. Afterwards, both of the trajectories are aligned, using singular value decomposition [69]. With this algorithm, the difference between each pair of poses can be computed, such as the mean, median and standard deviation of these differences. For this purpose, scripts presented at the website of the *Computer Vision Group Department of Informatics*, from the Technical University of Munich (TUM) [14], were used. This website provides openly available tools, for performing the previously explained evaluations. These tools can be used to pre-process the data and to evaluate SLAM/tracking results and offer the option of plotting both trajectories. In the following tables (5.1, 5.2 and 5.3) the errors of the original system, the improved one and the comparison between both, respectively, can be observed. They were calculated from a trajectory of 250m in an open environment, using a threshold of 500ms to match the trajectories' points. The figures 5.25 to 5.36 represent the difference (in red) between these trajectories in the form of 2D plots, the distance. The final figure 5.37 represents a 2D plot, where the original Lidar odometry and the improved one can be better compared. An interactive 3D plot comparing the mapping output and GPS trajectories is available at [2]. With this data, it can be observed that on both, original and improved system, the LiDAR odometry estimate, as it was expected, achieves the worst performance. Followed by the integrated mapping estimate, that shows slightly worse results than the lone mapping estimate, due to

the integration of the odometry to fill its gaps. As last, the mapping achieves the best estimate. Finally, comparing both results shows a significant improvement on the performance of the improved LiDAR odometry estimate, specially in the altitude estimation, meeting the initial expectations. On the other side, the improved mapping and final integrated output are just marginally improved. It shows that a better initial estimate is not enough for improving, in an acceptable amount, the mapping performance.

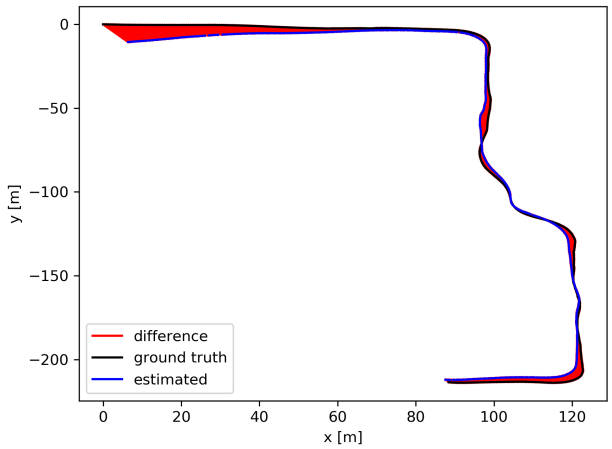

Figure 5.25: 2D plot representing the ATE error, of the odometry estimate of the x/y plane compared with GPS x/y plane ground truth.

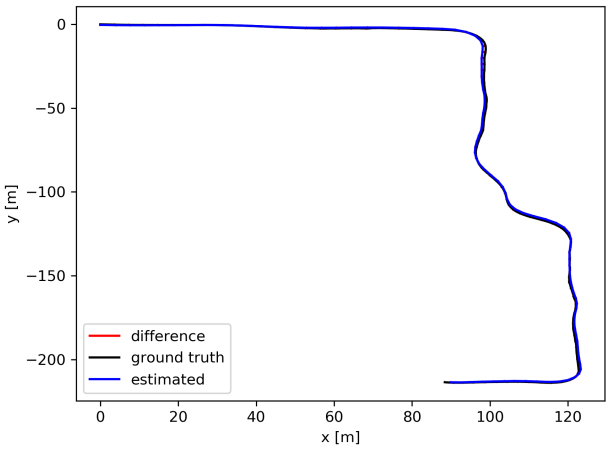

Figure 5.26: 2D plot representing the ATE error, of the mapping estimate of the x/y plane compared with GPS x/y plane ground truth.

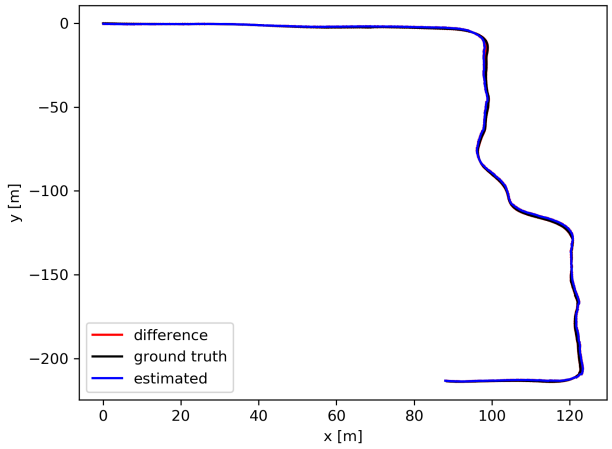

Figure 5.27: 2D plot representing the ATE error, of the integrated mapping estimate of the x/y plane compared with GPS x/y plane ground truth.

Table 5.1: ATE estimation errors, of the original system compared with GPS ground truth.

Tragectory (m)	Lidar Odometry	Lidar Mapping	Integrated
RMS (Root-mean-square deviation)	4.95	0.72	0.78
Mean	4.57	0.65	0.73
Median	4.39	0.59	0.69
STD (standard deviation)	1.9	0.29	0.3
Min	0.72	0.17	0.038
Max	12.55	1.29	1.44

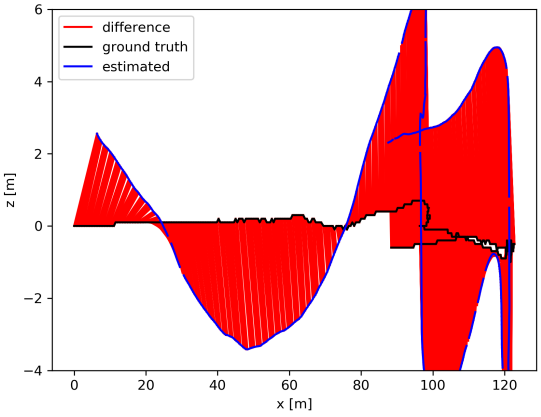

Figure 5.28: 2D plot representing the ATE error, of the odometry estimate of the x/z plane compared with GPS x/z plane ground truth.

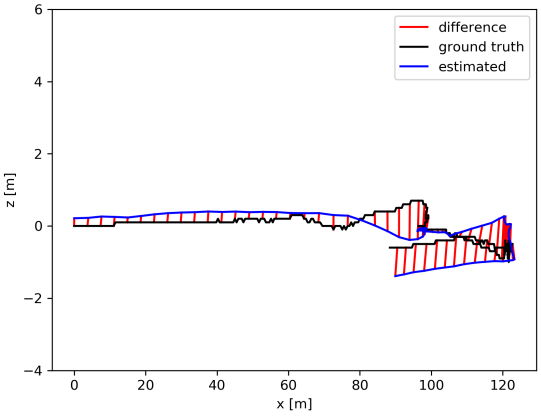

Figure 5.29: 2D plot representing the ATE error, of the mapping estimate of the x/z plane compared with GPS x/z plane ground truth.

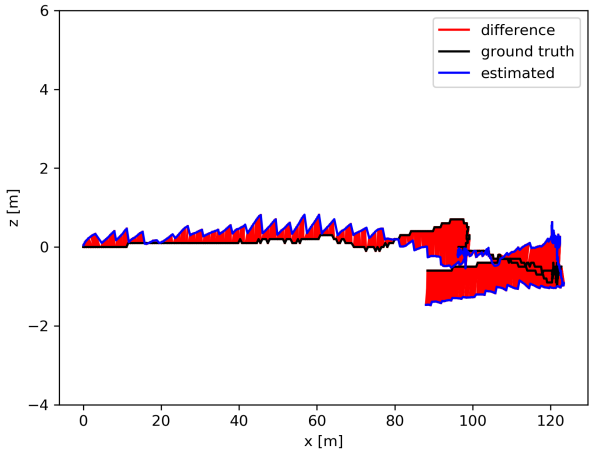

Figure 5.30: 2D plot representing the ATE error, of the integrated mapping estimate of the x/z plane compared with GPS x/z plane ground truth.

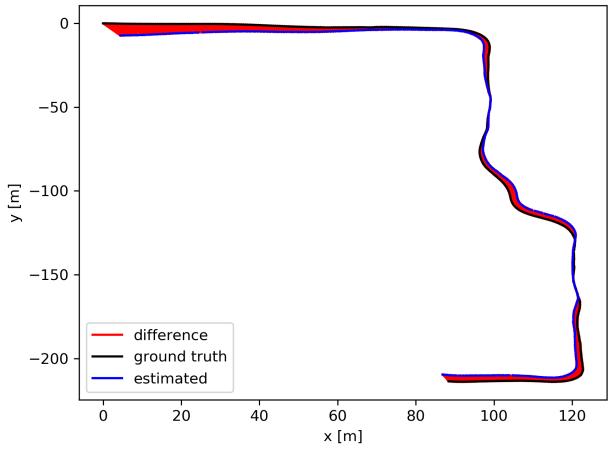

Figure 5.31: 2D plot representing the ATE error, of the odometry estimate of the x/y plane compared with GPS x/y plane ground truth.

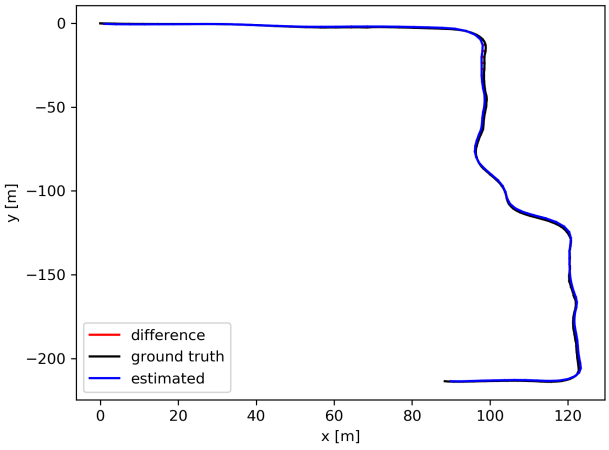

Figure 5.32: 2D plot representing the ATE error, of the mapping estimate of the x/y plane compared with GPS x/y plane ground truth.

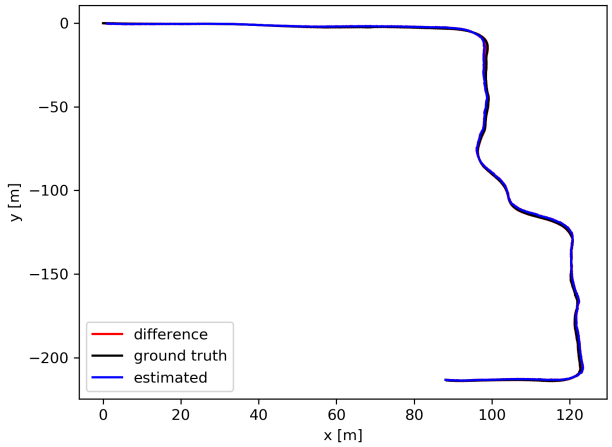

Figure 5.33: 2D plot representing the ATE error, of the integrated mapping estimate of the x/y plane compared with GPS x/y plane ground truth.

Table 5.2: ATE estimation errors, of the improved system compared with GPS ground truth.

Tragectory (m)	Lidar Odometry	Lidar Mapping	Integrated
RMS (Root-mean-square deviation)	3.11	0.7	0.77
Mean	2.75	0.64	0.72
Median	2.37	0.59	0.69
STD (Standard deviation)	1.45	0.28	0.28
Min	0.26	0.16	0.038
Max	8.45	1.25	1.4

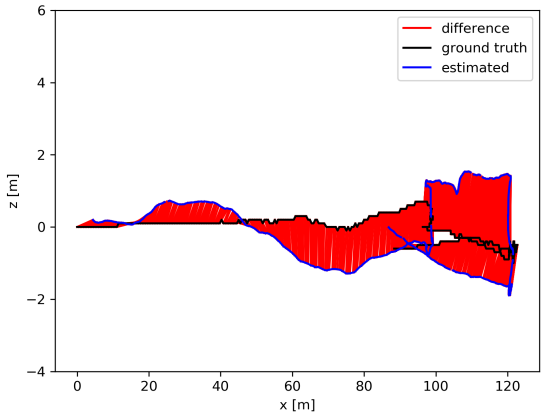

Figure 5.34: 2D plot representing the ATE error, of the odometry estimate of the x/z plane compared with GPS x/z plane ground truth.

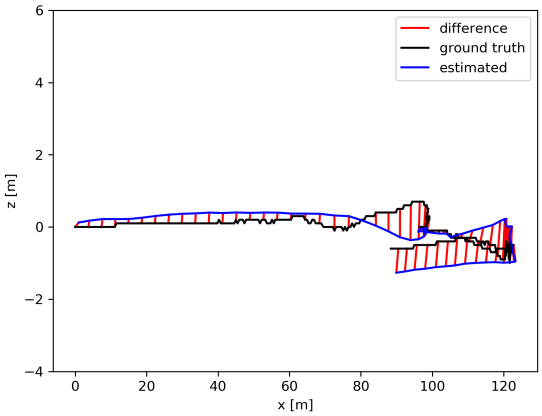

Figure 5.35: 2D plot representing the ATE error, of the mapping estimate of the x/z plane compared with GPS x/z plane ground truth.

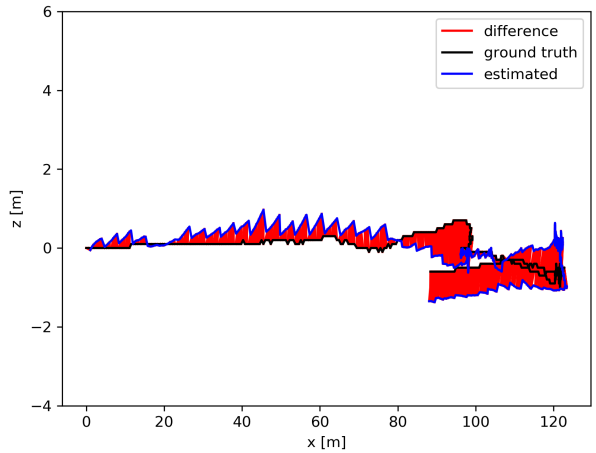

Figure 5.36: 2D plot representing the ATE error, of the integrated mapping estimate of the x/z plane compared with GPS x/z plane ground truth.

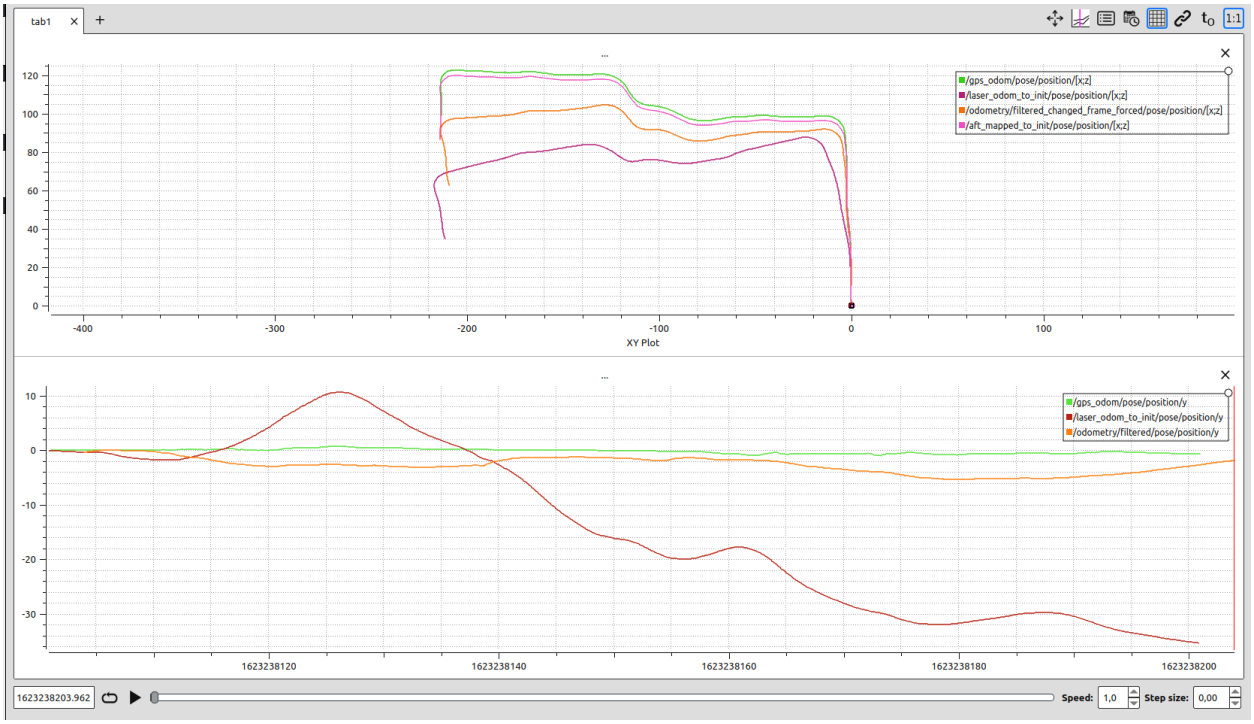

Figure 5.37: The figure on top exposes a 2D plot of the ground plane (x/y) representing the estimated trajectory, from the GPS, LiDAR mapping and LiDAR odometry, in a outdoor scenario, in meters. The *odometry_filtered* is the improved version of the *laser_odom*, the *aft_mapped* the mapping and the *GPS_odom* the GPS estimate, respectively. On the bottom, one dimensional plot of the altitude compares the previous odometry estimates with the GPS.

Table 5.3: Error comparison, of the original system and the improved one.

Trajectory (%)	Lidar Odometry	Lidar Mapping	Integrated
RMS (Root-mean-square deviation)	37	2.7	1.3
Mean	39	1.5	1.4
Median	46	0	0
STD (Standard deviation)	23	3.4	6.7
Min	63	5.9	0
Max	32	3.1	2.8

Chapter 6

Conclusions

After the realization of this work, a deep knowledge about motion estimation and mapping, using point clouds from a 3D LiDAR system, was acquired. Together with the advantages and limitations of the use of inertial measurements sensors, combined with sensor fusion through the use of Extended Kalman filters. Here, the matching of point clouds is a challenging problem, for the reason that it involves recovery of motion and correction of motion distortion. In order to use inertial measurements sensors (IMUs) correctly, this sensors should be carefully calibrated, since they are very vulnerable to noise. Also, the extended kalman filters are easy to use but if the data is biased or not properly calibrated, its estimation can easily start to drift. Nevertheless, the main objective of this work was accomplished. First goal was to understand and implement a state estimation algorithm that relied only on 3D LiDAR data and a second one to verify if the use of inertial measurements could indeed improve its performance. Several limitations affected the final results in this work, especially the access to the Laboratories at Poznan University of Technology. Due to the COVID-19 pandemic, its access was limited. In consequence, the direct use and experimentation of the hardware was limited, which led to the poor calibration of the IMU's. Even though, the obtained results were positive, the odometry pose estimation showed improvements especially in the drift of the altitude estimation and the Ouster IMU revealed to be accurate enough for the task. Several improvements can be pointed as future work, starting by the calibration of the IMU's, to properly find any offsets that can create drift, especially in the yaw angle. In case of the use of the same data, it is possible to extend further the state of the extended kalman filter, to take into the account the offset in the yaw angle and converge to the real value. Despite the fact that the odometry estimate was improved and could achieve even further improvements, the noticeable improvement on mapping results showed that this is not optimal approach. This comes from the fact that the odometry pose estimate is only used as initial guess for the point could matching. Even with the best initial point, the mapping algorithm converged typically to a similar result. Along with that, other approach can be used to take the maximum advantage of the inertial measurements. That is instead of the use of a motion estimation model, at the odometry algorithm that considers constant angular and linear velocities during a scan. This could be replaced with a model closer to reality, that could use the real measurements from the IMU sensor. Due to that, the motion distortion of

the point cloud could be better corrected. This improved point cloud could be given to the further algorithms and from there, achieve a best performance event at the mapping level.

References

[1] 3d is here: Point cloud library (pcl) | ieee conference publication | ieee xplore. https://ieeexplore.ieee.org/document/5980567. (Accessed on 10/01/2021).

[2] 3d plot of gps and loam path | elias ferreira. https://web.fe.up.pt/~up201502818/wiki/doku.php?id=3d_plot. (Accessed on 10/18/2021).

[3] Applied sciences | free full-text | an overview of lidar imaging systems for autonomous vehicles. https://www.mdpi.com/2076-3417/9/19/4093. (Accessed on 09/15/2021).

[4] Aspirador mi robot vacuum cleaner. https://mistoreportugal.pt/pt/produtos/aspirador-mi-robot-vacuum-cleaner-pro. (Accessed on 09/16/2021).

[5] Attitude estimation of multi-axis steering ugv using mems imu | ieee conference publication | ieee xplore. https://ieeexplore.ieee.org/document/9095122. (Accessed on 09/14/2021).

[6] Azure kinect dk | microsoft azure. https://azure.microsoft.com/pt-pt/services/kinect-dk/#overview. (Accessed on 10/28/2021).

[7] Block diagram of the lidar odometry and mapping software system | low-drift and real-time lidar odometry and mapping | springerlink. https://link.springer.com/article/10.1007/s10514-016-9548-2/figures/3. (Accessed on 09/22/2021).

[8] C099-f9p application board | u-blox. https://www.u-blox.com/en/product/c099-f9p-application-board. (Accessed on 10/06/2021).

[9] Comparing icp variants on real-world data sets | springerlink. https://link.springer.com/article/10.1007%2Fs10514-013-9327-2. (Accessed on 09/30/2021).

[10] Comparison between lidar odometry and lidar mapping output | low-drift and real-time lidar odometry and mapping | springerlink. https://link.springer.com/article/10.1007/s10514-016-9548-2/figures/12. (Accessed on 09/30/2021).

[11] Complementary filtering approach to orientation estimation using inertial sensors only | ieee conference publication | ieee xplore. https://ieeexplore.ieee.org/document/6224564. (Accessed on 09/14/2021).

[12] Complete basic operation of the extended kalman filter | diagram. https://www.researchgate.net/figure/showes-a-complete-basic-operation-of-\the-extended-Kalman-filter-combining-the-high-level_fig4_274373497. (Accessed on 10/03/2021).

[13] Computational geometry - algorithms and applications | mark de berg | springer. https://www.springer.com/gp/book/9783540779735. (Accessed on 09/27/2021).

[14] Computer vision group - datasets - rgb-d slam dataset and benchmark. https://vision.in.tum.de/data/datasets/rgbd-dataset. (Accessed on 10/06/2021).

[15] Continuous 3d scan-matching with a spinning 2d laser | ieee conference publication | ieee xplore. https://ieeexplore.ieee.org/document/5152851. (Accessed on 09/30/2021).

[16] Coordinate frames | tim whiteman - academia.edu. https://www.academia.edu/38172552/Frames. (Accessed on 09/14/2021).

[17] Coriolis force - wikipedia. https://en.wikipedia.org/wiki/Coriolis_force. (Accessed on 09/15/2021).

[18] Digital imaging and remote sensing image generation. http://www.dirsig.org/docs/new/images/coordinates/geo_coords.png. (Accessed on 09/12/2021).

[19] An enhanced visual-inertial navigation system based on multi-state constraint kalman filter | ieee conference publication | ieee xplore. https://ieeexplore.ieee.org/document/9184501. (Accessed on 09/14/2021).

[20] European geostationary navigation overlay service - wikipedia. https://en.wikipedia.org/wiki/European_Geostationary_Navigation_Overlay_Service. (Accessed on 09/15/2021).

[21] An evaluation of 2d slam techniques available in robot operating system | ieee conference publication | ieee xplore. https://ieeexplore.ieee.org/document/6719348. (Accessed on 08/06/2021).

[22] An example of extracted points | low-drift and real-time lidar odometry and mapping | springerlink. https://link.springer.com/article/10.1007/s10514-016-9548-2/figures/5. (Accessed on 09/26/2021).

[23] Frontpage | politechnika poznańska. https://www.put.poznan.pl/en?q=. (Accessed on 10/06/2021).

[24] Github - laboshinl/loam_velodyne: Laser odometry and mapping (loam) is a realtime method for state estimation and mapping using a 3d lidar. https://github.com/laboshinl/loam_velodyne. (Accessed on 10/06/2021).

[25] Global positioning system - theory and practice | b. hofmann-wellenhof | springer. https://www.springer.com/gp/book/9783211835340. (Accessed on 10/08/2021).

[26] The global positioning system on jstor. https://www.jstor.org/stable/24989397?seq=1#metadata_info_tab_contents. (Accessed on 09/15/2021).

[27] Gps.gov: Space segment. https://www.gps.gov/systems/gps/space/. (Accessed on 09/15/2021).

[28] gps_umd - ros wiki. http://wiki.ros.org/gps_umd. (Accessed on 10/06/2021).

[29] How gps receivers work - trilateration vs triangulation - gis geography. https://gisgeography.com/trilateration-triangulation-gps/. (Accessed on 09/15/2021).

[30] How nasa lost a spacecraft from a metric math mistake | simscale. https://www.simscale.com/blog/2017/12/nasa-mars-climate-orbiter-metric/. (Accessed on 09/15/2021).

[31] How things work: How does a mems gyroscope work — omnia mfg. https://www.omniamfg.com/mechanical/2019/4/14/how-things-work-how-does-a-mems-gyroscope-work?rq=mems. (Accessed on 09/15/2021).

[32] Icm-20948 | tdk. https://invensense.tdk.com/products/motion-tracking/9-axis/icm-20948/. (Accessed on 10/06/2021).

[33] Illustration of mapping process | low-drift and real-time lidar odometry and mapping | springerlink. https://link.springer.com/article/10.1007/s10514-016-9548-2/figures/8. (Accessed on 09/28/2021).

[34] imu_complementary_filter - ros wiki. http://wiki.ros.org/imu_complementary_filter. (Accessed on 10/06/2021).

[35] Inertial measurement unit. https://www.xsens.com/imu. (Accessed on 10/06/2021).

[36] Integration of pose transforms | low-drift and real-time lidar odometry and mapping | springerlink. https://link.springer.com/article/10.1007/s10514-016-9548-2/figures/9. (Accessed on 09/28/2021).

[37] Kalman filter equations and instruction | download scientific diagram. https://www.researchgate.net/figure/Kalman-filter-equations-and-instruction_fig3_283645465. (Accessed on 09/16/2021).

[38] Lidar remote sensing and applications - pinliang dong, qi chen - google livros. https://play.google.com/store/books/details?id=jXFQDwAAQBAJ&rdid=book-jXFQDwAAQBAJ&rdot=1&source=gbs_vpt_read&pcampaignid=books_booksearch_viewport. (Accessed on 09/14/2021).

[39] Master thesis website. https://web.fe.up.pt/~up201502818/. (Accessed on 10/15/2021).

[40] Matching errors | low-drift and real-time lidar odometry and mapping | springerlink. https://link.springer.com/article/10.1007/s10514-016-9548-2/figures/11. (Accessed on 10/02/2021).

[41] A mathematical introduction to robotic manipulation | richard m. murra. https://www.taylorfrancis.com/books/mono/10.1201/9781315136370/mathematical-introduction-robotic-manipulation-richard-murray\-zexiang-li-shankar-sastry. (Accessed on 09/29/2021).

[42] Melex. https://melex.com.pl/en/. (Accessed on 10/06/2021).

[43] melodic - ros wiki. http://wiki.ros.org/melodic. (Accessed on 10/10/2021).

[44] Microelectromechanical systems - wikipedia. https://en.wikipedia.org/wiki/Microelectromechanical_systems. (Accessed on 09/16/2021).

[45] Modern methods for robust regression - robert andersen - google livros. https://books.google.pt/books/about/Modern_Methods_for_Robust_Regression.html?id=ce5yKCu8HRoC&redir_esc=y. (Accessed on 09/29/2021).

[46] msg - ros wiki. http://wiki.ros.org/msg. (Accessed on 10/07/2021).

[47] Multiple view geometry in computer vision. https://www.cambridge.org/core/books/multiple-view-geometry-in-computer-vision/0B6F289C78B2B23F596CAA76D3D43F7A. (Accessed on 09/29/2021).

[48] nav_msgs/odometry documentation. http://docs.ros.org/en/noetic/api/nav_msgs/html/msg/Odometry.html. (Accessed on 10/08/2021).

[49] Nmea 0183 - wikipedia. https://en.wikipedia.org/wiki/NMEA_0183. (Accessed on 09/15/2021).

[50] Parallelization of a vine trunk detection algorithm for a real time robot localization system. https://www.researchgate.net/publication/335773466_Parallelization_of_a_Vine_Trunk_Detection_Algorithm_For_a_Real_Time_Robot_Localization_System. (Accessed on 09/16/2021).

[51] Performance assessment of the android smartphone's imu in a gnss/ins coupled navigation model | ieee journals & magazine | ieee xplore. https://ieeexplore.ieee.org/document/8915828. (Accessed on 09/15/2021).

[52] Plotjuggler. https://plotjuggler.io/. (Accessed on 10/06/2021).

[53] Probabilistic robotics | the mit press. https://mitpress.mit.edu/books/probabilistic-robotics. (Accessed on 09/16/2021).

[54] Project point cloud to the end of a sweep | low-drift and real-time lidar odometry and mapping | springerlink. https://link.springer.com/article/10.1007/s10514-016-9548-2/figures/6. (Accessed on 09/27/2021).

[55] Rep 103 – standard units of measure and coordinate conventions (ros.org). https://www.ros.org/reps/rep-0103.html. (Accessed on 08/20/2021).

[56] Rep 105 – coordinate frames for mobile platforms (ros.org). https://www.ros.org/reps/rep-0105.html. (Accessed on 08/20/2021).

[57] robot_localization wiki — robot_localization 2.7.3 documentation. http://docs.ros.org/en/noetic/api/robot_localization/html/index.html. (Accessed on 10/06/2021).

[58] rosbag - ros wiki. http://wiki.ros.org/rosbag. (Accessed on 10/11/2021).

[59] Ros/concepts - ros wiki. http://wiki.ros.org/ROS/Concepts. (Accessed on 10/07/2021).

[60] Ros.org | powering the world's robots. https://www.ros.org/. (Accessed on 10/06/2021).

[61] Rudolf e. kálmán - wikipedia. https://en.wikipedia.org/wiki/Rudolf_E._K%C3%A1lm%C3%A1n. (Accessed on 09/16/2021).

[62] rviz - ros wiki. http://wiki.ros.org/rviz. (Accessed on 10/06/2021).

[63] Sagnac effect - wikipedia. https://en.wikipedia.org/wiki/Sagnac_effect. (Accessed on 09/15/2021).

[64] Scanning rangefinder distance data output/utm-30lx product details | hokuyo automatic co. ltd. https://www.hokuyo-aut.jp/search/single.php?serial=169. (Accessed on 10/11/2021).

[65] sensor_msgs/navsatfix documentation. https://docs.ros.org/en/api/sensor_msgs/html/msg/NavSatFix.html. (Accessed on 10/08/2021).

[66] sensor_msgs/pointcloud2 documentation. http://docs.ros.org/en/noetic/api/sensor_msgs/html/msg/PointCloud2.html. (Accessed on 10/08/2021).

[67] Sensors | free full-text | keeping a good attitude: A quaternion-based orientation filter for imus and margs. https://www.mdpi.com/1424-8220/15/8/19302. (Accessed on 10/12/2021).

[68] Simultaneous localization and mapping: part i | ieee journals & magazine | ieee xplore. https://ieeexplore.ieee.org/document/1638022. (Accessed on 09/21/2021).

[69] Singular value decomposition tutorial. https://www.researchgate.net/publication/246546380_Singular_Value_Decomposition_Tutorial. (Accessed on 10/11/2021).

[70] Small unmanned aircraft. https://www.degruyter.com/document/doi/10.1515/9781400840601/html. (Accessed on 10/03/2021).

[71] Solving 2pi ambiguity problem of a laser scanner based on phase-shift measurement method or long distances measurement | ieee conference publication | ieee xplore. https://ieeexplore.ieee.org/abstract/document/6393323. (Accessed on 10/28/2021).

[72] Space and time: Inertial frames (stanford encyclopedia of philosophy). https://plato.stanford.edu/entries/spacetime-iframes/. (Accessed on 10/28/2021).

[73] State estimation nodes — robot_localization 2.7.3 documentation. http://docs.ros.org/en/noetic/api/robot_localization/html/state_estimation_nodes.html#parameters-common-to-ekf-localization-node-and-ukf-localization-node. (Accessed on 10/13/2021).

[74] tf/overview/transformations - ros wiki. http://wiki.ros.org/tf/Overview/Transformations. (Accessed on 10/09/2021).

[75] Towards ubiquitous identification of iot devices through visual and inertial orientation matching during human activity | ieee conference publication | ieee xplore. https://ieeexplore.ieee.org/document/9097600. (Accessed on 09/14/2021).

[76] Ubuntu 18.04.6 lts (bionic beaver). https://releases.ubuntu.com/18.04/. (Accessed on 10/10/2021).

[77] Using inertial sensors for position and orientation estimation [1704.06053]. https://arxiv.org/abs/1704.06053. (Accessed on 10/12/2021).

[78] Visual slam algorithms: a survey from 2010 to 2016 | ipsj transactions on computer vision and applications | full text. https://ipsjcva.springeropen.com/articles/10.1186/s41074-017-0027-2. (Accessed on 08/06/2021).

[79] Bruno Siciliano and Oussama Khatib. *SLAM: Problem Definition*, page 1154. Springer-Verlag, Berlin, Heidelberg, 2016.

[80] Ji Zhang and Sanjiv Singh. Low-drift and real-time lidar odometry and mapping. *Autonomous Robots*, 41:401–416, 02 2017.