FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

# Study on SLAM for Autonomous Racing

**Marcelo Henriques Couto**


Mestrado em Engenharia Informática e Computação

Supervisor: Prof. Vítor Hugo Machado Oliveira Pinto

Second Supervisor: Prof. Francisco Manuel Barbosa Ribeiro

October 5th, 2025

# Study on SLAM for Autonomous Racing

**Marcelo Henriques Couto**

Mestrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

President: Prof. Rosaldo José Fernandes Rossetti
Referee: Prof. José Lima
Referee: Prof. Vítor Hugo Machado Oliveira Pinto

October 5th, 2025

# Abstract

Autonomous driving has been a hot topic for the last decade. For an Autonomous vehicle to function properly, it requires a precise notion of its state and its surroundings at all times. SLAM (Simultaneous Localization and Mapping) is the problem of computing a representation of an agent's surroundings (map) while simultaneously estimating the agent's position in it. Although it is a well-studied problem, it is still a very challenging one.

Formula Student is a worldwide initiative that challenges students to design and build a single-seater racing car. Some competitions organized in this context have come to include autonomous disciplines, in which the vehicle must navigate a track without human intervention. FS FEUP is a team of students from the University of Porto that participates in these competitions.

This thesis aims to study different approaches to solve the SLAM problem effectively for autonomous racing scenarios, specifically targeting the ones characterized by Formula Student dynamic disciplines, providing a functional solution for the FS FEUP 02 prototype.

A Graph SLAM based approach was implemented, using loosely-coupled EKF Velocity Estimation as a velocity estimate source. Different optimization strategies were tested, as well as different motion models and the inclusion of LiDAR odometry as an additional source of motion information. The final system uses a Sliding-Window Levenberg-Marquardt solver for optimization. The implementation introduces a novel way of introducing pose nodes and factors to the graph, holding temporary pose estimates in a separate structure and only incorporating them into the graph once a certain pose difference threshold is reached.

This approach is capable of providing map and pose estimates in real-time for competition scenarios, having been validated in competition conditions at FSG 2025. The results obtained in simulation provided compare the efficiency and efficacy of different solvers in this context, including iSAM2, Batch and Sliding-Window Levenberg-Marquardt optimizers. The results obtained on ground also prove the usability of LiDAR odometry in this context and incentivize further research in this area.

# Acknowledgments

I would like to begin by thanking my supervisor, Prof. Vitor Hugo Pinto, for having supported me throughout this rather long journey for a master's thesis, supporting my decisions all the while letting me be the master of my own project.

Secondly, I would like to thank my family and my dear girlfriend, who have always been there, through thick and thin. Their encouragement and understanding have been invaluable to me, and I am truly grateful to have them in my life.

I would like to thank my colleagues at FS FEUP, who have inspired me to push the boundaries of my knowledge and skills. There was no better investment of my time than the hours spent working in the team, for they have truly made be a greater engineer, and a more complete person. On top of this, this dissertation would truly not have been possible without the thousands of hours of work poured into the FS FEUP 02 prototype, which served as testing bench and main objective of my work. I am eternally grateful for the opportunity to be part of such an incredible adventure, and for the possibility of making this project my dissertation.

I would like to express my heartfelt gratitude to all the members of the Autonomous Systems department for their constant help and support throughout this project. In particular, I am deeply thankful to the department leader, Davide Teixeira, for his patience and for generously taking the time to review my work.

Last but by no means least, I want to sincerely thank my colleague and friend, Lourenço Rodrigues, who was responsible for State Estimation at FS FEUP, for his unwavering support and collaboration throughout this journey. His dedication to the FS FEUP project, his invaluable insights, and the developments he carried out in parallel were essential, and without him, this dissertation would not be nearly as complete or polished. I am truly grateful to have shared this experience with him.

Marcelo Couto

*"Se fosse fácil não era para nós."*

Unknown at FS FEUP

# Contents

# List of Figures

# List of Tables

# Abbreviations and Symbols

| | |
|---|---|
| SLAM | Simultaneous Localization and Mapping |
| ABS | Antilock Brake System |
| ACC | Adaptive Cruise Control |
| AEB | Autonomous Emergency Braking |
| LKA | Lane Keeping Assistance |
| FSG | Formula Student Germany |
| FS | Formula Student |
| SAE | Society of Automotive Engineers |
| FSAE | Formula Society of Automotive Engineers |
| EKF | Extended Kalman Filter |
| UKF | Unscented Kalman Filter |
| IMU | Inertial Measurement Unit |
| GPS | Global Positioning System |
| LIDAR | Light Detection and Ranging |
| RADAR | Radio Detection and Ranging |
| CAN | Controller Area Network |
| ROS | Robot Operating System |
| GNSS | Global Navigation Satellite System |
| GSS | Ground Speed Sensor |
| RTK | Real Time Kinematic |
| LOAM | Lidar Odometry and Mapping |
| RMSE | Root Mean Square Error |
| JCBB | Joint Compatibility Branch and Bound |
| LeGO-LOAM | Lightweight and Ground-Optimized Lidar Odometry and Mapping |
| LIO | LiDAR Inertial Odometry |
| CoG | Center of Gravity |
| EV | Electric Vehicle |
| DV | Driverless Vehicle |
| CV | Combustion Vehicle |
| FSAE | Formula Society of Automotive Engineers |
| FSG | Formula Student Germany |

# Chapter 1

# Introduction

Autonomous Driving is currently a hot topic in the automotive and engineering world. In the last decade, an estimated $200 billion have been invested into Autonomous Driving technology development [1]. One of the tasks an Autonomous Driving system must be able to accomplish is to localize itself in the environment and map its surroundings. This thesis goes over the problem of Simultaneous Localization and Mapping in the context of Autonomous Driving, more specifically in the context of Formula Student competitions and Autonomous Racing.

## 1.1 Contextualization

This section will give context to the problem at hand, by going over what Localization and Mapping is and where Formula Student and autonomous racing come into play.

### 1.1.1 Localization & Mapping and SLAM

Localization and Mapping are two of the most important tasks in the field of Mobile Robotics. Localization is defined by the act of estimating the position or pose (position and orientation) of an agent in a given environment, while mapping is the task of creating a representation of the environment. The Simultaneous Localization and Mapping (SLAM) [2] problem is defined as the challenge of estimating the pose of an agent and the map of the environment simultaneously. It is often referred to as a *chicken-egg* problem due to the relationship between the estimation of a map and the pose of an agent in said map.

Most SLAM techniques utilize data from exteroceptive and proprioceptive sensors (among other sources), to generate an estimate of the variables of interest. The challenge comes from the inherent noise and error these sensors carry, which can lead to a divergence in the estimation of the variables. SLAM solutions often interpret the problem as a probabilistic inference problem, where the goal is to estimate the most likely state of the system given the data available.

There are numerous variations of the SLAM problem, as well as many approaches to the different challenges that encapsulate it. For instance, solutions to the SLAM problem can be divided into filtering-based and optimization-based methods. The processing of the input data can

also be performed in different manners, using different models for each of the tasks at hand. This has given rise to a wide array of solutions, each tailored to address specific challenges within the SLAM problem.

### 1.1.2 Formula Student

Formula Student competitions are engineering competitions that provide university students with a platform to develop and showcase their technical, organizational, and communication skills through the conception, design, manufacture, and validation of small, single-seater, formula-style racing cars. The term Formula Student is better connected to a worldwide initiative that loosely establishes the guidelines for the competitions. The competitions themselves are organized by different identities, such as FSAE, Imeche and FSG, leading to slight changes in the rules and format of the competition. Nevertheless, all of these share the common goal of improving the preparation of the next generation of engineers (among other professionals).

#### Competition Format

Within each competition, there are multiple divisions in which teams can compete. Some competitions divide the teams by the type of powertrain, meaning Electrical and Combustion prototypes compete separately, while others do not. On top of these, the Concept Class is a division sometimes present that allows teams to compete without a car, only competing in the Static Events. A recent addition to these divisions in some competitions is the Driverless Class, where teams compete with fully autonomous vehicles.

All competitions are divided into static and dynamic events. Each event is assigned a sum of points and a formula for its calculation. The winner of the competition is the team that manages to rack up the most points in all its events. Table 1.1 shows the point system for the FSG 2024 competition.

The static events are often in the format of presentations or discussions with judges, involving only the participants and aiming to evaluate the team's methods and knowledge. Even though these are of paramount importance, the dynamic events are the proper racing events, where the cars are put to the test in a series of challenges, which are the ones of most interest for this dissertation:

- the **Acceleration Event**, where the car must traverse a 75 m straight line as fast as possible;
- the **Skid Pad Event**, where prototypes are challenged in a fixed figure 8 track;
- the **Autocross Event**, where the car must perform one lap in a custom cone delimited track as fast as possible;
- the **Endurance Event**, where the car must complete a track similar or equal to the Autocross Event, but with multiple laps, where the challenge shifts focus from absolute speed to also take economy and reliability into account. The **Efficiency** points are awarded based on the car's fuel consumption during this event;

| **Static Events** | **Combustion & Electric Vehicles** | **Driverless Cup** |
|---|---|---|
| Business Plan Presentation | 75 | - |
| Cost Report | 100 | - |
| Engineering Design Event | 150 | 150 |
| **Dynamic Events** | | |
| Acceleration Event | 50 | - |
| Driverless Acceleration Event | 75 | 75 |
| Skid Pad Event | 50 | - |
| Driverless Skid Pad Event | 75 | 75 |
| Autocross Event | 150 | - |
| Driverless Autocross Event | - | 100 |
| Endurance Event | 250 | - |
| Efficiency | 50 | - |
| Trackdrive Event | - | 200 |
| **Total Points** | 1000 | 600 |

Table 1.1: Point System for FSG 2024 Competition

Figure 1.1 shows the representations of the tracks and the rules regarding each of them on the FSG 2025 handbook [3].

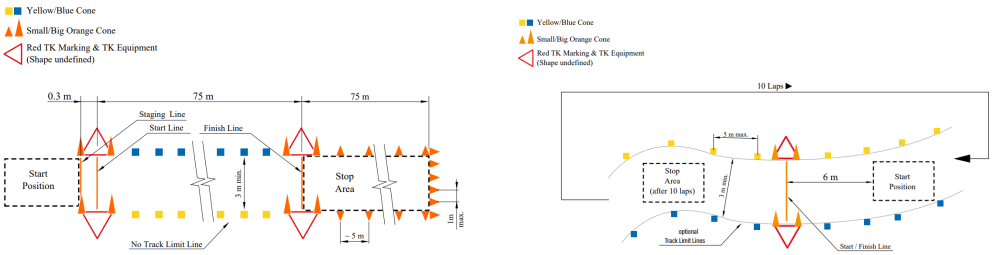
### Driverless Class

The Driverless Class, also known as Driverless Cup, aims to challenge students to build a vehicle that can traverse the tracks of multiple events without any human intervention. It was an initiative first introduced by FSG in 2017, one of the most renowned competitions in the world of Formula Student. Organizations keep pushing in the direction of the development of autonomous vehicles, with some driverless events now being part of the Class 1 (Electric) division in FSG.

Most competitions include the Engineering Design Event as the only Static Event of the Driverless Class, Formula Student UK being an exception. The dynamic events are similar to the ones available for the other divisions. While the Acceleration and Skidpad events feature the exact same track, which is fixed by rules, the Autocross event's track is often smaller for the Driverless Class. The Endurance event is usually renamed to Trackdrive and also suffers from a reduction in the number of laps, aiming to remove the economy challenge from the equation.

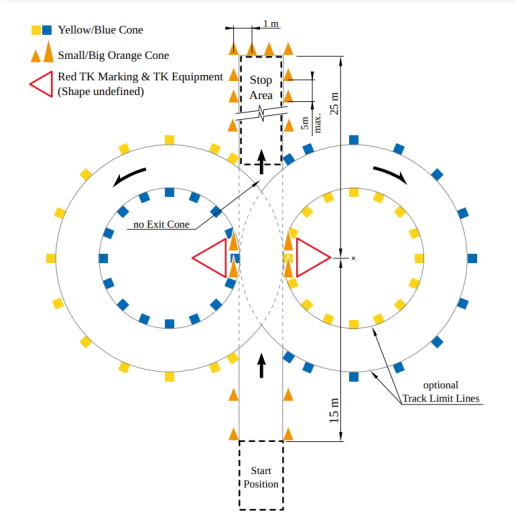
On top of this, and likely the most important difference, is the delimitation of the tracks: all the Driverless events' tracks are delimited by specific cones, defined, for instance, in the FSG 2024 handbook as well, portrayed in Figure 1.2. Blue and Yellow cones are used to delimit the left and right side of the track respectively, while large orange cones delimit the starting line and small orange cones are used for entrance and exit of the track.

The Driverless Class events require the vehicle to complete the track without any human interaction, apart from the initial start. The vehicle can be turned on manually, and the rest of the communication is dealt with using a Remote Emergency System (RES), which allows sending the "Go Signal" and an eventual "Emergency Signal", if necessary.




(a) Acceleration Event Track Regulation

(b) Trackdrive / Autocross Track Regulation




(c) Skidpat Event Track Regulation

Figure 1.1: Dynamic Events Track Schematics from FSG 2025 Handbook [3]

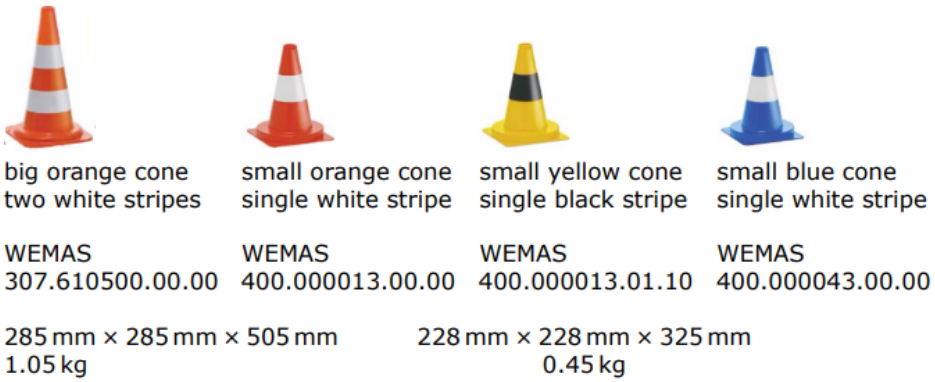



Figure 1.2: Cones used to delimit Driverless tracks in Formula Student Competitions, from FSG 2024 Handbook [3]

The fact that the physical properties of the cones are known beforehand is a key aspect of the problem, as it deeply affects the mapping challenge. Another characteristic of this challenge that can deeply affect the localization and mapping problem is that the maps of the tracks of two of the events are known beforehand, while the other two are not. This means that the system must be able to adapt to the environment and the track, which can be a challenge in itself.

The Formula Student Driverless environment is particularly challenging due to two main reasons:

- vehicles travel at a relatively high-velocity and exerting significant acceleration outputs, with 75 m acceleration times averaging the 4 seconds mark;
- the tracks the vehicle traverse are quite narrow, with tracks being as narrow as 3m (which is quite tight for a vehicle with a track measurement of more than one meter).

#### Formula Student FEUP

Formula Student FEUP has been representing the University of Porto in international Formula Student competitions for two years now. The team is now developing its first autonomous vehicle and second prototype ever, to compete in the Driverless and Electric classes of Formula Student competitions in the summer of 2025. A LiDAR-based perception was chosen for the vehicle, to render it capable of identifying the track limits, and the vehicle is to be fitted with wheel encoders, a steering angle sensor and an IMU, to allow for accurate state estimation of the vehicle's state. The current system is divided into 4 main modules as depicted in Figure 1.3. Currently, an **EKF SLAM** is responsible for both **pose, velocities and map estimation**. The team's present implementation is not capable of withstanding high accelerations, suffering from increased drift and error accumulation. It is also incapable of handling large tracks, with a latency of 100ms in not that complex scenarios, which is not suitable given the velocities these vehicles travel at.

## 1.2 Motivation

In this section, a reasoning will be made on why the problem at hand is relevant and the motivation behind this dissertation will be clarified.

### 1.2.1 Automation and Safety in the Automotive Industry

The automotive industry has been in constant and rapid evolution since its dawn. One of the biggest motivators for this evolution has been the desire for safer and more efficient methods of transportation. The last few decades have seen a rising contribution of electronics and computing technology in vehicle safety, as depicted in Figure 1.4, mainly with the implementation of Advanced Driver Assistance Systems (ADAS). Example features of such systems are Automatic Emergency Braking (AEB), Lane Keeping Assistance (LKA) and Adaptive Cruise Control (ACC).

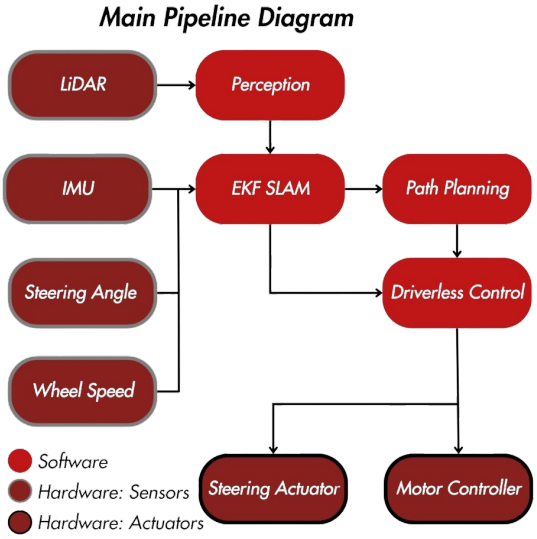



Figure 1.3: Formula Student FEUP Autonomous Pipeline Structure

These systems have been shown to have a significant impact on road safety [4] and are now standard on any personal or industrial road vehicle. That being said, there is still much improvement to be made, as the number of deaths caused by accidents each year is still in the six figures worldwide [5].

Even with the inclusion of the aforementioned features, a human being is still at the wheel and responsible for controlling the vehicle for a large portion of a supposed trip. As technology advances, vehicles become more secure, which also means a smaller percentage of road accidents occur due to machine failures and a larger percentage occur due to human error. The concept of an Autonomous Vehicle (AV) envisions the removal of the human element partially or totally from the equation, as a means to increase road safety. In the last few years, technological developments in the fields of Computer Vision, Artificial Intelligence and Automation [7], among others, have allowed substantial advancements in Autonomous Driving technologies. This concept, though controversial and likely to continue to undergo significant legal scrutiny in the upcoming years, has already been accepted to a great extent.

Propelled by the potential this technology has for impacting road safety, Autonomous Vehicles are soon to become a reality in our daily lives, with many companies already introducing autonomous driving capabilities into their vehicles in the present. Personal vehicles have only yet seen partial vehicle automation, as per the SAE Levels of Driving Automation [8]. Tesla is a prime example of this, whose vehicles have packed the Tesla Autopilot suite since 2016's model S first release [9]. However, there are already multiple examples of fully capable autonomous vehicles in operation, such as the Waymo One service [10] in Phoenix, Arizona, depicted in Figure 1.5a, Stagecoach's autonomous bus fleet in the UK [11], shown in Figure 1.5b, and an autonomous shuttle bus in Cascais, Portugal.

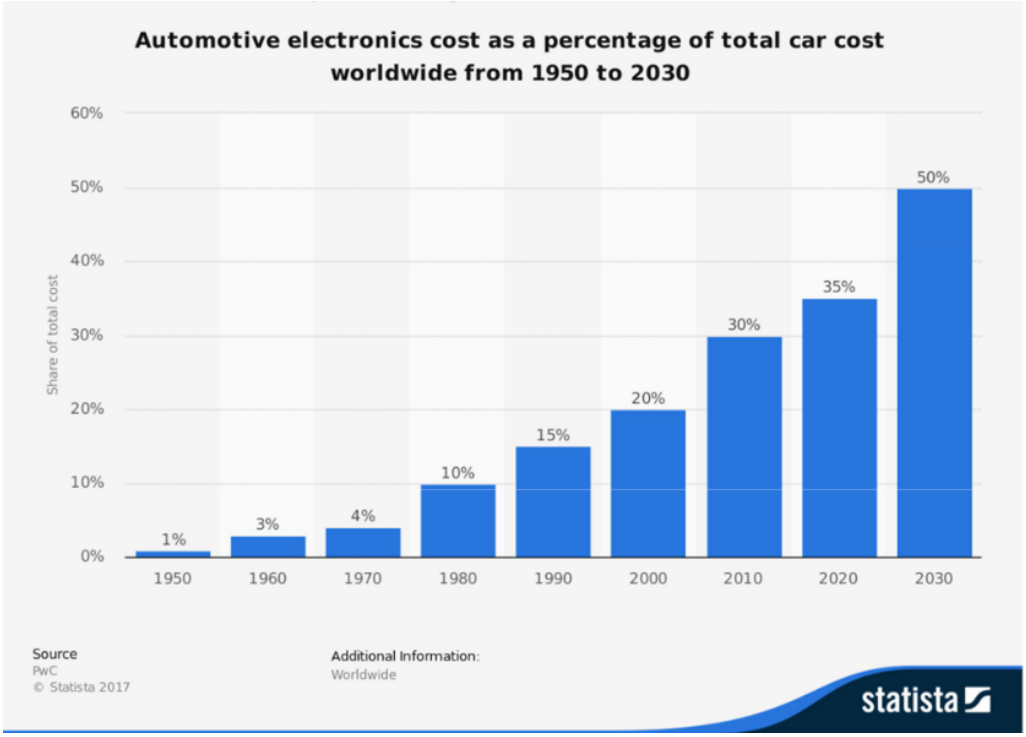



Figure 1.4: Automotive electronics cost as a percentage of total car cost [6]

(a) Waymo One Jaguar I-Pace Autonomous Vehicle [10]


(b) Stagecoach's Autonomous Bus [11]

Figure 1.5: Examples of Autonomous Vehicles in Operation

### 1.2.2 SLAM in Autonomous Vehicles

One of the key parts of an Autonomous Driving system is the State Estimation module. More specifically, a vehicle must be able to identify its state and the state of the environment surrounding it at all times. The state is often abstracted as a group of variables detailed enough to allow the system to make decisions regarding the course of action. In an Autonomous Driving scenario, both the pose and a representation of the vehicle's surroundings are almost always necessary, which comprises a Localization and Mapping problem.

In many scenarios, a representation of the map is already available, which makes the problem a localization problem. Even so, these maps are often not complete and generally do not include all the details necessary for the vehicle to navigate safely, such as dynamic objects. This leads to the need to solve a SLAM problem, in an environment often characterized by highly complex scenes and/or high speeds, where every millisecond counts. Moreover, SLAM implementations can often be used with a previously available map, improving its and the localization's accuracy [12].

### 1.2.3 Competition as a Driver for Innovation

It is no secret that competition is one of the biggest motivators for effort and innovation. For this reason, it is a common event to have important technological advancements come from sports and other highly competitive areas of human activity. The Automotive Industry is one of the best examples of this phenomenon [13], with racing having been a key ingredient for the developments thus far, being a phenomenon present since the beginning of the last century.

Formula Student's main purpose is to be a platform for engineering students to apply their knowledge in a real-world highly competitive scenario, creating a breeding ground for innovation and creativity. It was on par with this motto that many organizations within the Formula Student world included an Autonomous Driving category in their competitions.

## 1.3 Problem Statement

The problem this dissertation aims to address is to study and draw conclusions on **which techniques have the most success at estimating the vehicle and map state in the context of Autonomous Racing**. More specifically, it desires to address the problem as it is presented in Formula Student competitions. The knowledge generated from such exploratory work is relevant inside and outside the scope of autonomous racing since many characteristics of the SLAM problem in this scenario are also present in other contexts, such as generic Autonomous Driving.

In more detail, on the constraints and characteristics of the SLAM problem and the estimation of the vehicle's speed inherent to dynamic disciplines:

- The environment is to be represented as a **feature-based 2D map**, corresponding to the **cones position** in the track, as the goal of the mapping task is to determine the tracks' limits. The pose of the vehicle is also to be estimated only in 2 dimensions;

- On top of solving the SLAM problem, the system must be able to **estimate the vehicle's current velocity** with high precision, to allow for smooth control of the vehicle. This estimation is usually inherent to the SLAM problem, as the estimation vehicle's pose often necessitates such estimates;
- Due to the nature of the sport, the vehicles travel at **relatively high speeds** (often reaching upwards of 25 m/s) and outputting **accelerations of more than 1.5g**. A SLAM system for Formula Student (or any other type of Autonomous Motor racing) must be able to estimate the pose and map with high precision in these conditions, being **resilient to the increased noise and error** that comes with them;
- As previously mentioned, in some disciplines the track is known *a priori*, while in others it is not. Therefore, the algorithms must be suitable to be used both in a **SLAM mode and a Localization-Only mode**, taking advantage of the known map of the track when available;

## 1.4 Objectives

Following the problem statement, the objectives of this dissertation are listed and described more concretely below:

1. Implement a fully capable State Estimation system that fits the requirements of the SLAM problem for a Formula Student Autonomous Race car:
    - The system is adapted to the sensor suite and perception pipeline available in the vehicle developed by Formula Student FEUP, which uses a 3D LiDAR for external perception;
    - The system is design to withstand the accelerations and velocities characteristics of the environment;
    - The system is capable of maintaining sufficiently low drift for accurate mapping of track of up to 0.5 km;
    - The system performs under very low latencies, with vehicle state estimates under 1 ms and with perception processing times of at most 50 ms (the perception pipeline runs at most at 20 Hz);

    In this sense, the task of track limit detection is not to be considered, as it has already been solved with some success by the team;
2. Provide a modular and flexible implementation, that allows for further development and changes to be done in the future, allowing the system to evolve with the rest of the prototype:
    - The implementation shall be done using ROS2, as it is the middleware used by the team's system;

- The implementation shall provide extensive Doxygen documentation for comprehension of the team;
- The implementation shall follow SOLID principles and have extensibility in mind, providing interfaces for every function to which an alternative is possible;

3. Explore different solutions for different parts of the system with the goal of understanding which techniques have the biggest impact:
   - Evaluate the impact of different State-of-the-art techniques in terms of performance for the aforementioned problem;
   - Evaluate the performance of the different system setups implemented in both simulated and real-world high-acceleration scenarios, using the vehicle developed by Formula Student FEUP.

## 1.5 Document Structure

The remainder of this document will be organized as follows:

- **Chapter 2** goes over the theoretical concepts necessary to comprehend the implementation and the rest of the document in general;
- **Chapter 3** a literature review is made in SLAM methodologies, focusing on the ones that are most relevant to the problem at hand;
- **Chapter 4** details the implementation of the system and algorithms that encompass it;
- **Chapter 5** presents the methods for evaluation of the system and the experiments carried out and discusses the results obtained in the experiments;
- **Chapter 6** focuses on the conclusions and future work on the subject.

# Chapter 2

# Theoretical Concepts

Simultaneous Localization and Mapping, as previously stated, is a problem explored for some decades now, with ever-growing interest among the scientific community. This chapter goes over the basic principles necessary to comprehend the problem at hand. It is divided into two parts: the first part goes over the definition of the SLAM problem and its categorization, while the second part goes over the most common solutions to the problem. Additionally, a deeper explanation on the inner workings of the most relevant algorithms and techniques for this dissertation is given.

## 2.1 Nomenclature and Definitions

The terms used in the field of SLAM, mobile robotics and computer vision are often used interchangeably and can lead to some confusion. This section aims to clarify the most common terms used in the field.

State Estimation is the process of estimating the state of a system, which can be defined as the set of variables that describe the system at a given point in time. In the context of SLAM, the state of the system is composed of the pose of the agent and the map of the environment [14]. The pose of the agent is usually defined by its position and orientation in the environment, while the map of the environment can take up many forms, as discussed later in this section.

SLAM, however, is often a problem solved in the context of a larger goal, meaning that the state estimation task at hand might not be confined to estimating the pose and map. For instance, the work in this thesis also aims to estimate the velocity of the agent. This challenge is very tightly related to the SLAM problem and is fundamental for the stable operation of any vehicular control system. From here on, vehicle state estimation refers to the estimation of the variables that describe the state of the vehicle, which includes the pose and the velocity of the agent.

Odometry is another term used with multiple meanings. Originally, it meant the estimation of the motion of an agent through the use of wheel velocity. However, it is nowadays used in the literature to signify the estimation of movement of autonomous agents in general, including pose and velocity alike, as one is often derived from the other.

Besides an important problem in the world of computer vision and robotics, SLAM is essentially a field which encapsulates multiple techniques used to infer the state of a moving robot. For that matter, a great deal of work related to SLAM does not necessarily focus on the SLAM problem, but on localization using techniques that are common to SLAM.

## 2.2 SLAM Problem Definition and Categorization

Given a series of controls/odometry data $u_t$ and sensor observations $z_t$ over discrete time steps $t$, the SLAM problem is to compute an estimate of the agent's states $x_t$ at each point in time and a map of the environment $m$. Despite this rather straightforward definition, there is a multitude of characteristics to a SLAM problem that can change the way it is approached, leading to a wide array of different solutions to the problem (or rather, problems). In the next paragraphs, the multiple variables that can affect the SLAM problem are discussed, giving an insight into the different types of SLAM problems that can be solved.

### Full SLAM and Online SLAM

SLAM can first be divided into the **Full SLAM** and the **Online SLAM** problems [15]. While the former aims to estimate the full trajectory of the agent and the full map of the environment (as described in the previous paragraph), the latter aims to estimate only the current pose of the agent and the map of the environment up to that point. The Online SLAM problem is often more relevant in real-time applications, such as Autonomous Driving, where the agent must be able to make decisions based on the current state of the environment.

### Mapping

SLAM can also be importantly categorized by the type of map to be generated. Maps can vary in a multitude of characteristics, originating a wide array of different types. However, SLAM implementations tend to focus on generating a few types of scenes or maps:

- **Feature/Landmark-based (sparse) scenes:** composed of a set of landmarks or features that are extracted from the environment [16], representing the world in a sparse abstracted manner [17]. Note that these maps can still vary in density of feature widely, depending on the type of sensor, the algorithm and even the problem at hand and its objective;
- **Occupancy Grid maps:** denote the existence of an obstacle in a certain cell of the grid, representing the environment in a more abstract manner (more common for 2D mapping);
- **Dense scenes:** which represent the environment with the least amount of abstraction, usually in the form of a Point Cloud [18].

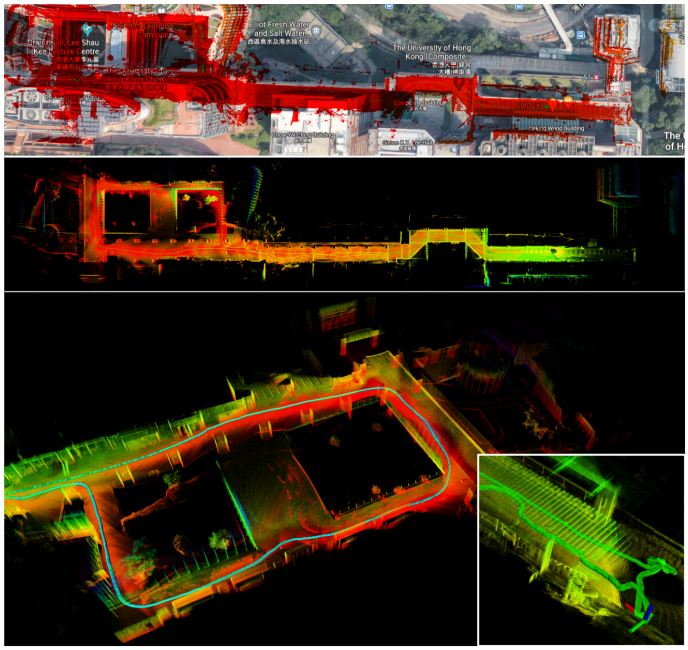


Figure 2.1: Dense Map Representation generated by FastLIO2 [18]

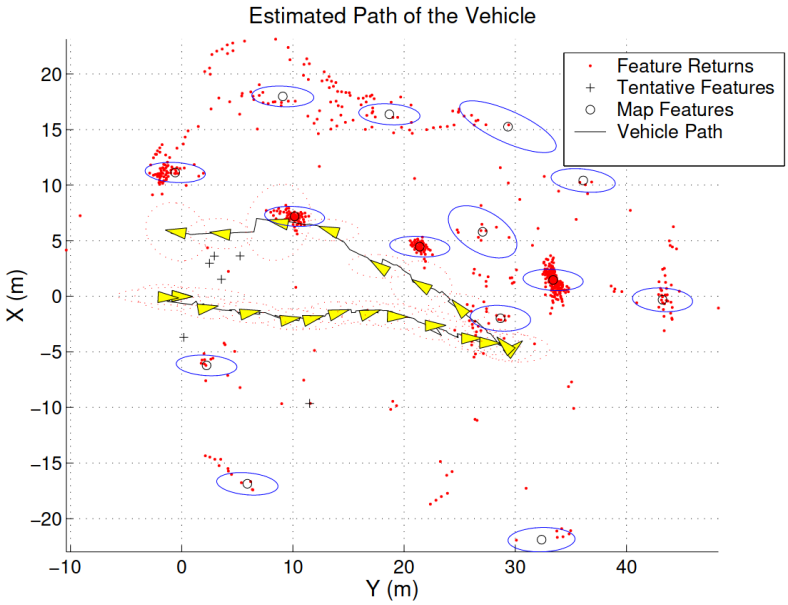



Figure 2.2: Feature-based Map

Figure 2.3: Occupancy Grid Map

### Environment

The environment in which the SLAM problem is to be solved also plays an important role in defining the potential solutions for the problem. On top of constraints to the type of sensors that might have success and the extent to which they can perform, the fact that moving objects are present in the environment can also have a significant impact on the SLAM problem. **Dynamic SLAM** [19] focuses on solving the problem in environments affected by change. This small change has a huge impact on the techniques used, as most systems developed thus far stand on the assumption that the environment is static to be able to converge into or maintain a stable representation.

### Sensors

Finally, one final crucial division of SLAM techniques is caused by the type of data/sensors used to solve the problem. The robot must be equipped with some sort of sensor to be able to perceive the environment surrounding it. For that reason, and due to the notable differences between them, methodologies are usually categorized by the type of sensor they require. The most prominent ones are **LiDAR and Camera-based SLAM techniques**, with subclassifications often emerging tying the algorithms and implementations to the need for a secondary source of information. LiDAR (Light Detection and Ranging) sensors are capable of providing a dense representation of the environment, with each point in the cloud representing a distance from the sensor to a point in the environment. There are many types of LiDAR sensors. It is important to note that many LiDAR sensors contain moving parts and create a scan of the environment by rotation of a column of lasers, or other similar structures. This is also the case for the LiDAR available for this dissertation.

Besides the main perception sensor, some SLAM solutions include/require proprioceptive sensors, such as IMUs. One typical combination is the use of Camera and IMU (Visual Inertial SLAM) [20] or LiDAR and IMU (LiDAR Inertial SLAM) [18] [21]. While a LiDAR outputs a cloud of points with their relative locations to the sensor, an IMU gives insight into the linear accelerations and angular velocities felt by the agent, which can be used to estimate the movement of the agent, as well as correct distortion of point clouds caused by high velocity (due to the LiDAR's rotation).

Other sensors used for velocity or position estimation include:

- **Ground Speed Sensors** (GSS), which estimate velocity by matching subsequent images or laser scans of the ground (among other techniques);
- **GNSS** (Global Navigation Satellite System), which is a system that allows for the localization of the agent by using signals from satellites;
- **Wheel Encoders**, which estimate the movement of the agent by measuring the rotation of the wheels.

## 2.3 SLAM Techniques

Localization and Mapping tasks used to be solved separately and were thought of as independent problems [22]. This was standard practice until the late 1980s and 1990s, in which two tasks were proven to be correlated. The early conceptual groundwork for SLAM was laid by Smith, Self, & Cheeseman in their seminal 1986 paper [23], which introduced the idea of probabilistic mapping using Kalman filters to address the uncertainties inherent in robot perception and motion. In the 1990s, the SLAM problem was first formally approached by [2], using much of the fundamental ideas presented by [23]. The methodologies presented and developed by these authors are still the basis for most SLAM solutions developed today, and define the problem as one of probabilistic inference:

$$p(x_{0:T}, m | z_{1:T}, u_{1:T}) \tag{2.1}$$

where $x_{0:T}$ is the trajectory of the agent, $m$ is the map of the environment, $z_{1:T}$ are the sensor observations and $u_{1:T}$ are the controls/odometry data. Note that this equation (often denoted as the SLAM posterior) defines the Full SLAM problem, while the Online SLAM problem is defined as:

$$p(x_t, m | z_{1:t}, u_{1:t}) \tag{2.2}$$

where $x_t$ is the current state of the agent.

To solve the SLAM problem as defined in equations 2.1 and 2.2, a probabilistic framework is typically employed, leveraging two key components: the **motion prior** and the **observation model**. The motion prior encodes how the state of the robot evolves over time given the control inputs (or other motion data), and is typically expressed as:

$$p(x_t|x_{t-1}, u_t) \tag{2.3}$$

This models the belief about the current state $x_t$ conditioned on the previous state $x_{t-1}$ and the control input $u_t$. It captures the uncertainty associated with the robot's movement, including wheel slippage, drift, and actuator noise.

The observation model, on the other hand, describes how measurements relate to the map and the robot's state. It is used to update beliefs about the world based on sensor data and is typically written as:

$$p(z_t|x_t, m) \tag{2.4}$$

Here, $z_t$ represents the observation at time $t$, and the model defines the likelihood of receiving that observation given the current pose $x_t$ and the map $m$.

Since these seminal works were presented, the development of strategies for SLAM and localization are often similar, as they typically stem from approaches which combine the agent's movement and data from the environment to improve both estimates. The Localization problem is often also solved with the aid of GNSS systems, which due to the latest technological advancements can be extremely precise. Even so, localization without the use of GNSS is still an extremely relevant problem [12] as:

- GNSS is not always available, as it requires a clear line of sight to the satellites;
- the precision of GNSS often comes at an increased monetary cost;
- a robust localization sensor should rely on multiple sources of information, as GNSS can be unreliable in some scenarios or suffer from failure. This is often a necessity, as many applications of localization systems are safety-critical.

### 2.3.1 Filtering-Based Techniques

Filtering-based approaches to SLAM formulate the problem as a recursive state estimation task, where the joint posterior over the robot pose and the map is updated as new control inputs and sensor measurements are received. At the core of these methods lies the Bayes Filter, a probabilistic framework that recursively estimates the belief over the system state. It operates in two stages: a prediction step, where the previous belief is propagated through the motion model to account for the robot's dynamics and control inputs, and a correction step, where this belief is refined using the most recent sensor observations. Since the Bayes Filter only defines the recursion and not the explicit form of the belief representation, different implementations emerge depending on how this posterior is approximated.

### Kalman Filters and EKF SLAM

Many algorithms have been used to solve the SLAM problem. However, a few specific algorithms are worth mentioning. The **Extended Kalman Filter (EKF) SLAM** presented in [2] is one of the most common solutions to the SLAM problem. The EKF SLAM is a recursive algorithm based on the Bayes Filter. The Kalman Filter assumes the state follows a Gaussian distribution, represented by its mean ($\mu$) and covariance ($\Sigma$), the state usually being both the Robot's pose and the variables that represent the map. To maintain this notion, all operations applied to the state must be linear. The EKF extends the latter to non-linear systems, by linearizing the models applied to the filter around the current state estimate using a first-order Taylor expansion.

The Kalman Filter (or any other Kalman Filter) incorporates the odometry and observation data in two steps: prediction step and correction step (or update step). The prediction step applies the motion model 2.3 to the odometry data, which will inform on a variation of the state (in SLAM's case, typically only on the Pose). Conversely, the correction step applies the observation model to the observation data, which informs directly on the state of the system. It is in the correction step that makes the Kalman Filter more than applying models. The full algorithm is represented by the set of equations 2.5:

$$\hat{\mu}_t = A_t * \mu_{t-1} + B_t * u_t \tag{2.5}$$

$$\hat{\Sigma}_t = A_t * \Sigma_{t-1} * A_t^T + R_t \tag{2.6}$$

$$K_t = \hat{\Sigma}_t * C_t^T (C_t * \hat{\Sigma}_t * C_t^T + Q_t)^{-1} \tag{2.7}$$

$$\mu_t = \hat{\mu}_t + K_t * (z_t - C_t * \hat{\mu}_t) \tag{2.8}$$

$$\Sigma_t = (I - K_t * C_t)\hat{\Sigma}_t \tag{2.9}$$

where

- matrices $A_t$ and $B_t$ are $n * n$ matrices that denote the process and motion models respectively[1];
- matrix $C_t$ corresponds to the observation model and describes how to map the state $x_t$ to an observation $z_t$;
- $R_t$ and $Q_t$ represent the process and observation noises;
- $K_t$ is the Kalman Gain.

The Kalman Gain ($K_t$) is essentially a term that can leverage the influence of the observations on the value of the next step. In sum, the Kalman Filter is a weighted mean between the observations and the state where the weight (the Kalman gain) evolves with the progression of the system

[1] In SLAM, only the 'motion model' nomenclature is typically used for both models, as they often act as a single function in implementation, with the 'process model' being more common in other applications of the Kalman Filter.

and is fundamentally influenced by the noise matrices $R_t$ and $Q_t$ provided for the models. According to Equation 2.10, if the observation noise is 0, the current state can be completely modelled through the observations. The converse is also true (if the noise tends to infinity, its influence will be negligible).

$$Qt = 0 \implies Kt = C^{-1} \implies expected\_now(x_t) = C^{-1} observations(z_t) \tag{2.10}$$

The difference from the base Kalman Filter presented and the EKF is that matrixes $A_t$ and $C_t$ are substituted by their jacobians $G_t$ and $H_T$, executing the linearization previously mentioned.

Despite this explanation focusing on EKF SLAM, these concepts stand for other applications of the algorithm, with the only changes being the usage of different data for prediction and correction steps and the abolishment of the 'motion model' term.

EKF SLAM originally refers to including the map representation in the state. This approach scales very poorly (quadratically) with the size of the map. This is likely the reason for the current FS FEUP's approach not handling large maps well.

Even though the EKF is the most popular derivate of the Kalman Filter being applied to SLAM, it is not the only one. The UKF (Unscented Kalman Filter) is a very similar approach, which instead of linearizing the models around the current state, samples points from the distribution and propagates them through the non-linear model (Unscented Transform). Other variants of the EKF are also used in more recent publications, such as the Iterated Kalman Filter in which, for each iteration, the correction step is repeated with a new linearization around the current estimate until the change in the state is below a pre-defined threshold. Although EKF SLAM is not a common approach due to the devastating fallback mentioned, the EKF and its relatives are still widely used for SLAM in slightly different manners, as will become apparent throughout the next chapters.

### Non-Parametric Filters

**Particle filters** differ from Kalman filters by the method the probability distribution of the state is represented: instead of using the mean and covariance, the state is represented by a group of particles, each with its weight. This allows for the relaxation of the Gaussian distribution assumption, consequently enabling non-linear models to be used. Despite its increased accuracy for applications to which these assumptions do not hold, it is a computationally heavier solution. Particle filters have been used for localization with relative success in the form of **Montecarlo Localization** [24].

Montecarlo localization works because the state space of the localization problem is often short, composed only of 3 variables in the case of two-dimensional space. Using a particle filter similarly for the SLAM problem would not be computationally efficient, largely due to the number of particles necessary to converge in such a larger state space (SLAM's state space includes the pose of the agent and all landmarks being mapped). **Rao-blackwellized Particle Filter** [25] exploits the idea of conditional independencies in the state space to separate the state space into smaller, more manageable problems, which can be solved with a Particle Filter. As depicted in

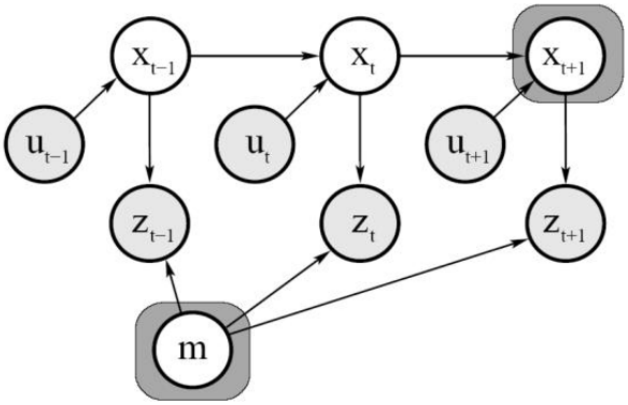



Figure 2.4: Bayes Network for the SLAM problem [26]

Figure 2.4, the multiple observations' likelihoods are conditionally independent in the case of the state of the agent being known. When using a particle filter, the state of the agent can be considered to be known for each particle.

The **FastSLAM** algorithm [27] applies the Rao-Blackwellized particle filter to SLAM by factoring the posterior 2.2 as:

$$p(x_{1:t}, m \mid z_{1:t}, u_{1:t}) = p(x_{1:t} \mid z_{1:t}, u_{1:t}) \prod_{i=1}^{K} p(m_i \mid x_{1:t}, z_{1:t}) \tag{2.11}$$

FastSLAM uses a particle filter to represent the posterior over the robot trajectory $p(x_{1:t} \mid z_{1:t}, u_{1:t})$, while within each particle, $K$ independent Extended Kalman Filters (EKFs) are used to estimate the positions of landmarks. The algorithm is divided into four steps:

1. **Sampling (Motion Update)**: For each particle $k$, sample a new pose $x_t^{[k]}$ based on the motion model and previous pose:
$$x_t^{[k]} \sim p(x_t \mid x_{t-1}^{[k]}, u_t)$$

2. **Measurement Update**: For each observed landmark $m_i$, update its EKF using the measurement $z_t$ and the predicted pose $x_t^{[k]}$:
$$p(m_i \mid z_{1:t}, x_{1:t}^{[k]}) \quad \text{via EKF}$$

3. **Importance Weighting**: Compute the likelihood of the observation given the particle's pose and landmark estimates, to assign a weight $w_t^{[k]}$ to the particle:
$$w_t^{[k]} = p(z_t \mid x_t^{[k]}, m^{[k]})$$

4. **Resampling**: Resample the set of particles based on their weights to focus computational effort on the more likely hypotheses.

The computational complexity of FastSLAM is $O(M \log K)$ per time step, where $M$ is the number of particles and $K$ is the number of landmarks, due to the use of data structures like KD-trees to manage per-particle landmark maps. Despite posing an improvement in comparison to EKF SLAM, FastSLAM can suffer from particle depletion and inaccuracies in data association if the number of particles is insufficient, particularly in ambiguous or feature-sparse environments.

**FastSLAM 2.0** [28] improves upon the original FastSLAM algorithm by addressing a key limitation related to the proposal distribution used during sampling. In FastSLAM 1.0, the motion model $p(x_t \mid x_{t-1}, u_t)$ is used as the proposal distribution, which does not incorporate the latest sensor observations. As a result, many sampled particles may be unlikely given the actual measurements, leading to particle depletion and poor localization. FastSLAM 2.0 solves this by using a more informed proposal distribution that incorporates the current observation $z_t$. This leads to a more robust algorithm, which needs fewer particles to achieve the same level of accuracy as FastSLAM 1.0, although of considerably higher implementation difficulty [29].

This leads to a more accurate approximation of the posterior and allows for better particle diversity with fewer particles. Additionally, FastSLAM 2.0 uses a more precise computation of the importance weights based on the full sensor model, improving the consistency and efficiency of the filter. These changes significantly enhance performance in environments with noisy sensors or ambiguous features, making FastSLAM 2.0 more robust for practical SLAM applications.

### 2.3.2 Optimization-Based Techniques

The techniques described (as well as all other Filter-based solutions [30]) were only designed to solve the Online SLAM problem. This was not necessarily an issue, as this version of the problem is the one with the most practical applications. However, these solutions are incapable of taking advantage of the knowledge of the past states of the agent to improve the current estimate, which could potentially seriously improve the accuracy of a SLAM system. Later, another family of techniques was born, **based on optimization** or smoothing, which estimated the full trajectory of the mobile agent by interpreting the SLAM problem as a least-squares minimization problem. This formulation of the problem was introduced in 1997 by Lu and Milios [31]. Even so, only a decade later, with advancements in the fields of sparse linear algebra, did this optimization-based approach become feasible and popular [32]. **Graph-based SLAM** (Pose Graph Optimization) is included in this type of solution, one of the most popular solutions to the SLAM problem today.

#### Graph SLAM as a Least Squares Problem

Graph SLAM formulates the problem as a factor graph, a representation of a complex function into multiple factors. This representation is a bipartite graph connecting variable nodes to function nodes. For example, the function $g$, defined by Equation 2.12, can also be represented by the graph in Figure 2.5.

$$g(X_1, X_2, X_3) = f_1(X_1) * f_2(X_1, X_2) * f_3(X_1, X_2) * f_4(X_2, X_3) \tag{2.12}$$

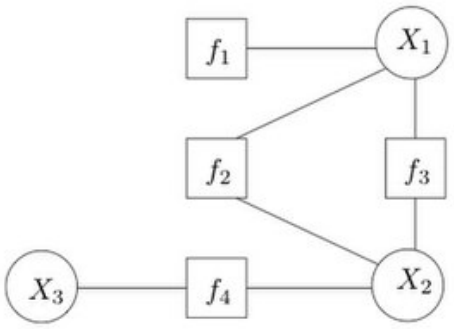



Figure 2.5: Factor Graph Example, with variable nodes $X_1$, $X_2$ and $X_3$ and factor node $f_1$, $f_2$, $f_3$ and $f_4$. Image taken from [33].

In this scenario, each variable node represents a pose or a landmark and each factor node represents the constraints between them arising from odometry and sensor measurements. In this context, factor-graphs are often visualized as pose graphs as these are easier to comprehend, the difference being factor nodes are edges themselves[2]. Each factor (or edge) encodes a probabilistic constraint, typically modeled as a Gaussian distribution, that captures the relative transformation between connected nodes along with its uncertainty. This uncertainty is often denoted by a covariance matrix $\Sigma$, but due to the these SLAM solvers nature, the information matrix $\Omega$, which is the inverse of the covariance matrix ($\Omega = \Sigma^{-1}$), is more often used in Graph SLAM. Contrary to the covariance, the greater the values in the information matrix, the less uncertainty on the constraint and the stronger the edge's influence on the final result. These observations often conflict with the current estimate of the state, leading to an error between the two. Figure 2.6 illustrates this idea well.

The SLAM problem can then be defined as a nonlinear least-squares optimization task: finding the configuration $x^*$ of all nodes (poses and landmarks) that best satisfies the set of constraints,

[2]This only applies for cases in which the factor graph only has factors that link two variables, which is often the case for Pose Graph SLAM

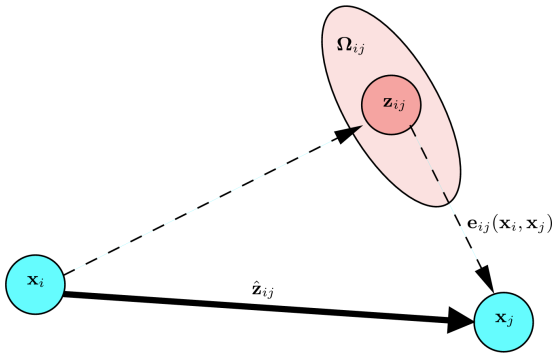



Figure 2.6: Pose Graph Representation Example: the dotted edge on the left represents the observation of j corresponding to i, while the solid arrow represents the expected observation from the state, leading to an error between the two. Figure obtained from [32].

with the representation as a least-squares problem shown by Equation 2.13.

$$x^* = argmin \sum_i ||f_i(x) - z_i||^2_{\Sigma_i} \tag{2.13}$$

where for each constraint $i$

- $f_i$ is the expected observation from the state estimate
- $z_i$ is the observation
- $x$ is the current state estimate
- $x^*$ is the best state estimate
- $\Sigma_i$ is the covariance associated to the measurement

The entire term inside the norm is often called the error function $e(x)_i = f(x) - z_i$, which is often also defined relative to the nodes of the graphs instead of the edges (constraints) $e_{ij}(x_i)(x_j)$ (often simplified to $e_{ij}(x)$). The Mahalanobis norm $||e_{ij}(x)||^2_{\Sigma_{ij}}$ has the form $e_{ij}(x)^T * \Sigma_{ij}^{-1} * e_{ij}(x)$, meaning we can write the function to minimize as

$$E(x) = \sum_{ij} e_{ij}(x)^T * \Omega_{ij} * e_{ij}(x) \tag{2.14}$$

**Classical Solution for the SLAM Least Squares**

This least squares problem is usually solved using Non-linear optimization solvers like Levenberg-Marquartdt or Gauss-Newton. The way these algorithms work is by iteratively calculating better estimates of the system based on the previous one, stopping when the change between iterations is below a configurable threshold. For these solvers, each iteration begins by approximating the error function around an initial estimate $x_0$ using the Taylor expansion:

$$e_{ij}(x) = e_{ij}(x_0 + \Delta x) \simeq e_{ij}(x_0) + J_{ij}(x_0) * \Delta x \tag{2.15}$$

with $J_{ij}(x_0) = \frac{\partial e_{ij}(x_0)}{\partial x_0}$ and $\Delta x$ being the increment you are trying to find for this iteration. By substituting this approximation in 2.14, we get:

$$E(x) = E(x_0 + \Delta x) \simeq \sum_{ij} (e_{ij}(x_0) + J_{ij}(x_0)\Delta x)^T \Omega_{ij} (e_{ij}(x_0) + J_{ij}(x_0)\Delta x) \tag{2.16}$$

$$= \sum_{ij} e_{ij}(x_0)^T \Omega_{ij} e_{ij}(x_0) + 2\underbrace{e_{ij}(x_0)^T \Omega_{ij} J_{ij}(x_0)}_{b_{ij}^T} \Delta x + \Delta x^T \underbrace{J_{ij}(x_0)^T \Omega_{ij} J_{ij}(x_0)}_{H_{ij}} \Delta x \tag{2.17}$$

$$= \sum_{ij} e_{ij}(x_0)^T \Omega_{ij} e_{ij}(x_0) + 2b_{ij}^T \Delta x + \Delta x^T H_{ij} \Delta x \tag{2.18}$$

$$\tag{2.19}$$

where $b_{ij}$ is the Gradient vector and $H_{ij}$ is the Hessian matrix. To find the minimum of this equation, the algorithm solves the system of equations by taking the derivative of the error function and setting it to 0 for $\Delta x$:

$$H\Delta x = -b \tag{2.20}$$

All there is left is to update the current estimate of the state and continue until the difference between iterations is small enough:

$$x^* = x_0 + \Delta x \tag{2.21}$$

What was just described was the classic Gauss-Newton algorithm for solving the SLAM Least Squares, one of the most used algorithms for this. The Levenber-Marquardt algorithm builds on top of this by simply adding a control variable that helps the algorithm change the convergence rate depending on how far from the solution it might be:

$$(H + \lambda I)\Delta x = -b \tag{2.22}$$

It is a controlled hybrid between the Gradient Descent and the Gauss-Newton methods: when the variation of the error decreases, $\lambda$ is increased, and vice-versa. A small value will lead to Levenber-Marquadt approximating to the Gauss-Newton method, while a large one will lead it to approximate to Gradient-Descent. This allows the Levenberg-Marquadt method to be resilient to non-linearities and poor initial estimates, while maintaining the convergence speed natural to the Gauss-Newton method.

One important characteristic of the Taylor expansion approximation is that the Jacobian matrix $J_{ij}(x)$ is often sparse, as the error between two nodes is often influenced by a small subset of variables. Consequently, the Hessian Matrix $H_{ij}$ is sparse as well, allowing for the application of sparse linear algebra when solving the system of equations 2.20. Some methods often used are Sparse Cholesky and QR Factorization.

### Incremental Smoothing and Mapping (iSAM and iSAM2)

Incremental Smoothing and Mapping (iSAM) [34] is an alternative solver to the classical batch optimization solutions just presented. The iSAM and iSAM2 algorithms are incremental methods for solving the SLAM problem efficiently by reusing previous computations rather than relinearizing and resolving the full nonlinear system from scratch at every iteration. These methods are especially valuable in real-time applications like autonomous driving, where measurements arrive continuously, and computational efficiency is critical.

Traditional batch solvers (e.g., Gauss-Newton or Levenberg-Marquardt) require solving a large linear system iteratively, meaning this relinearization at each time step. iSAM (Incremental Smoothing and Mapping) avoids this by factorizing the system once, using QR factorization, and incrementally updating that factorization as new measurements arrive. The system is also solved

only once per iteration, in contrary to batch optimizers, which do so until convergence criteria is met. This makes iSAM efficient, especially for systems with a sparse and gradually evolving structure. A major limitation of iSAM is that it does not relinearize previously linearized factors, which can lead to reduced accuracy over time.

iSAM2 [35] addresses this by introducing a more flexible and accurate framework based on the Bayes tree: a data structure that organizes the solution of the SLAM problem into a tree of cliques derived from variable elimination. This structure enables the system to efficiently identify which parts of the solution are affected by new measurements and selectively relinearize and update only those parts. This data structure also supports periodic relinearization and variable reordering, which helps maintain both accuracy and computational efficiency over long time horizons.

The construction of the factor graph occurs in the same fashion for iSAM2 as in other implementations. But instead of rebuilding the entire linearized system, iSAM2 identifies a minimal set of variables whose linearization is impacted by the new factors added. This is done using the Bayes tree, a hierarchical structure that encodes the result of previous variable elimination. Each node (or clique) in the Bayes tree corresponds to a conditional distribution over a subset of variables.

The partial update process proceeds as follows:

1. **Affected Subtree Identification**: iSAM2 identifies the cliques in the Bayes tree that involve any variables connected to the new factor. This defines a subtree of the Bayes tree that must be relinearized;

2. **Relinearization**: Only the factors involving the variables in the affected subtree are relinearized. This ensures that the system accurately reflects the updated nonlinear model;

3. **Re-elimination**: The affected subtree is removed from the Bayes tree, and a new factor graph is constructed using the relinearized factors. Variable elimination is performed again (typically using COLAMD or similar heuristics for ordering) to rebuild this part of the tree;

4. **Reinsertion**: The newly eliminated portion is reinserted into the Bayes tree, and the solution is updated incrementally.

The evolution of a Bayes-Tree with the addition of new factors is depicted in Figure 2.7. This selective reprocessing ensures that only a small, localized portion of the problem is updated, typically in logarithmic or sublinear time with respect to the total number of variables. The global structure of the SLAM problem remains intact, and the linearization of unaffected parts of the graph is reused. Additionally, iSAM2 maintains a delta representation of the solution, allowing fast access to variable estimates without recomputing the full solution at every step.

#### Performance Analysis

In terms of performance, graph-based SLAM methods such as batch Graph SLAM and iSAM2 generally outperform filtering-based approaches like EKF-SLAM in both accuracy and scalability.

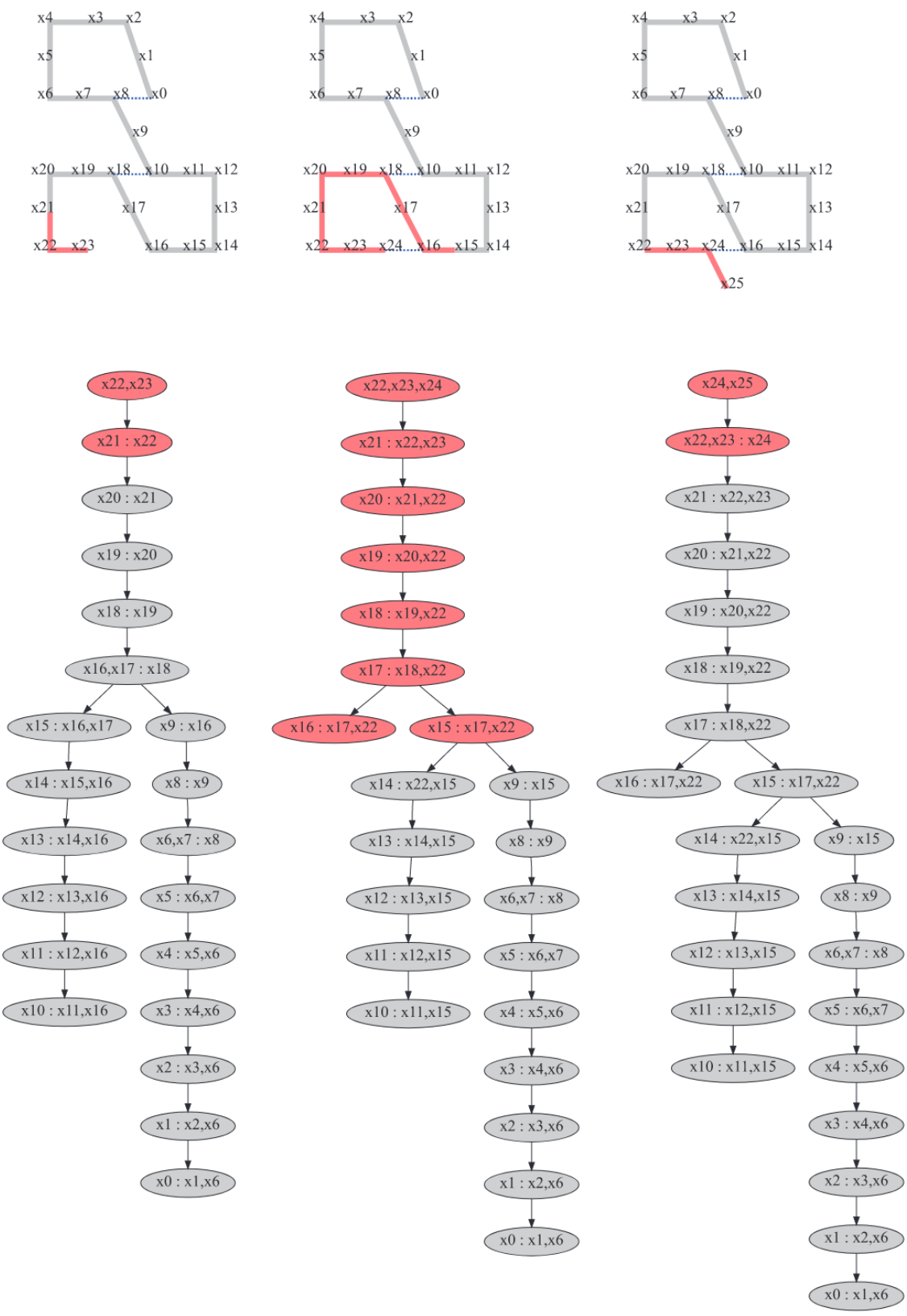



Figure 2.7: Evolution of the Bayes tree with the original factor graph in parallel [35].

Pose-graph optimization approaches solve the full SLAM problem, which on its own poses an accuracy benefit, as it does not discard the influence of previous states on the next estimate and is able to retroactively correct past poses and landmarks through loop closures. However, its batch nature makes it computationally intensive, with typically $O(n * d^2)$ complexity, where $n$ is the number of variables and $d$ the degree of the graph. iSAM2 addresses this by incrementally updating the solution using a Bayes tree representation and selectively relinearizing only affected variables, allowing it to achieve real-time performance with near-constant update times in practice. While slightly less accurate than full batch optimization due to its incremental nature, iSAM2 maintains high-quality estimates and is significantly more accurate than filtering methods. Overall, graph-based methods offer a better balance between accuracy and efficiency, especially in large-scale and highly dynamic environments.

#### Libraries and Implementations

Nowadays, most optimization-based solutions do not implement their own least-squares solver, often using libraries that contain implementations for these. Some of the most popular libraries are:

- g2o [36]: an open-source C++ library focused on ease of use, which provides implementations of multiple algorithms used both for SLAM and Bundle Adjustment, such as Gauss-Newton and Levenberg-Marquardt;
- Ceres Solver [37]: an open-source solver focused on the least-squares problem;
- GTSAM [38]: an open-source c++ library that provides implementations for multiple computer vision and robotics problems, using factor graphs and Bayes networks at its core and providing an implementation of the iSAM2 algorithm, among others.

While g2o is the most popular library, especially notorious among the Formula Student community, GTSAM has been shown to outperform g2o's implementation. In [39], a comparison between the three solutions listed above and a third less well-known (SE-Sync) was conducted, running the implementations on multiple benchmarking datasets. The results showed that, while all implementations were fairly capable, GTSAM and SE-Sync outperformed g2o considerably in the most complex scenarios.

### 2.3.3 SLAM Systems Structure

Graph SLAM, EKF SLAM and Fast SLAM only define the central component of the SLAM system, which is responsible for combining the processed information to create the estimates, sometimes referred to as **SLAM back-ends** (graph construction itself is typically not considered back-end). However, there is often a need to process the sensor data into a format capable of informing on the state of the system and the changes occurring. This leads to the existence of multiple components in a SLAM system, as depicted in Figure 2.8.

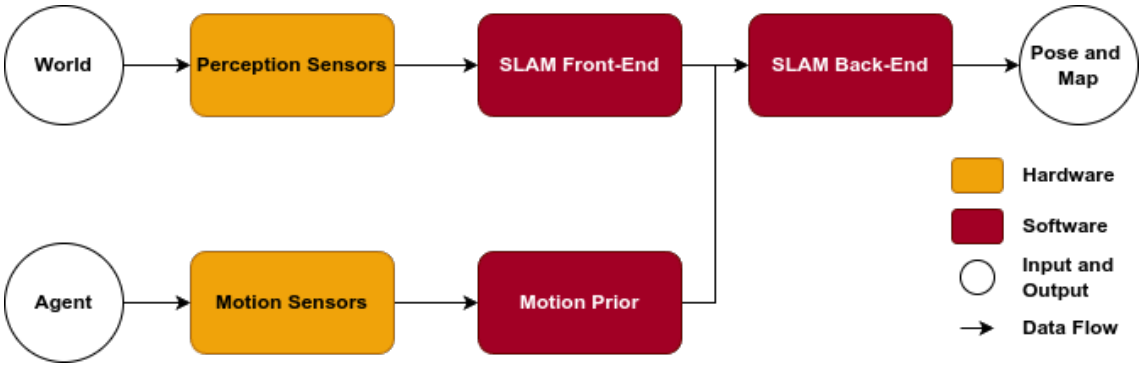



Figure 2.8: SLAM System General Structure depicted by a Data Flow Diagram

Depending on the sensor used, perception data processing is often required. Often referred to as **SLAM front-ends** [40], these modules usually include:

- **feature extraction** algorithms and **data association** algorithms, which are often required in feature-based mapping or for LiDAR odometry (estimation of movement based on LiDAR data)
- **scan matching** algorithms, which can be used for LiDAR odometry and dense map construction
- **object detection** and localization algorithms, which are often required for feature-based maps

### Data Association

When performing landmark-based SLAM, an additional module is necessary for a complete system: a data association strategy. Each time observations are received, the system must determine for each if they correspond to a previously observed landmark or if they are a new one. This is often done by comparing the current observation with the previous ones, using a distance metric to determine if they are similar enough to be considered the same landmark.

The choice of distance metric plays a crucial role in the performance of data association. The Euclidean distance is often used due to its simplicity, but it assumes isotropic and uncorrelated measurement noise. To account for varying uncertainties and correlations in sensor measurements, the Mahalanobis distance is frequently employed instead [15]. This metric incorporates the covariance of the observations and is defined as:

$$d_M(x) = \sqrt{(z_i - \hat{z}_j)^T P_{ij}^{-1} (z_i - \hat{z}_j)} \quad \text{where } P = C \cdot \Sigma \cdot C^T + R \tag{2.23}$$

$z_i$ is the $ith$ observation, $\hat{z}_j$ is the expected observation for the $jth$ landmark taking into account the observation model and $P$ is the covariance matrix associated with the difference between the observation and the state.

Apart from the metric, there are more than one general approaches to dealing with the multiple possible associations. One of the most common approaches to data association is the Nearest Neighbor (NN) approach, which simply associates each observation with the closest landmark in the map, performing Individual Compatibility (IC) tests. This method is also often called Maximum Likelihood (ML) data association, especially when using the Mahalanobis distance 2.23 as the distance metric (discussed further below).

More sophisticated techniques include Joint Compatibility Branch and Bound (JCBB), which evaluates entire sets of associations jointly rather than individually (Join Compatibility instead of Individual Compatibility). In JCBB, a tree structure is formed in breadth-first fashion, with depth equal to the number of observations. Each node in the tree represents a potential association. At the end, each branch is evaluated for Joint Compatibility of the solution, meaning the entire set of associations is evaluated together. While building the three, if a certain branch is found to be unable to beat the current best, it is pruned, leading to a better performance than brute force Joint Compatibility.

#### Motion Modeling and Motion Priors

Although front-ends are usually used to refer to data processing of perception sensors, SLAM and Localization systems that utilize other sources of information require processing of that data as well. The other most typical sources of information are control commands, IMUs and odometry sources (like Wheel Encoders). The **motion model**, which was previously referred while explaining the basis of SLAM systems 2.3, represents the set of functions that map this data into a variation of the pose of the agent (in many articles, this term is not used, simply including this model into the process model). In this dissertation, we will separate the motion model into two parts: first part of the modelling corresponds to the transformation from sensor or control data to the velocity of the agent, while the second part corresponds to the transformation from velocity to pose.

The complete motion models vary depending on the assumptions made about the agent, its movement the type and source of the data used, among other factors, leading to a wide array of options.

Motion models often incorporate other types of models that emulate certain characteristics of the agent or data source. For instance, when leveraging wheel rpm data, a particular **vehicle model** has to be taken into account, which is a set of functions that represent the kinematics and sometimes dynamics of a vehicle (difference illustrated in Figure 2.9), with the goal of accurately predicting its motion. These can be as simple as a point-mass model, in which the vehicle is reduced to its center of gravity. However, many of the works presented later leverage more complex models, which use vehicle models that factor in conditions such as tire slip to increase their accuracy.

Regarding the second part of the motion model, two-dimensional pose estimation originates just a small subset of typically used models, which mostly vary regarding the assumptions made about the type of movement the agent is performing between each time step. One example of such

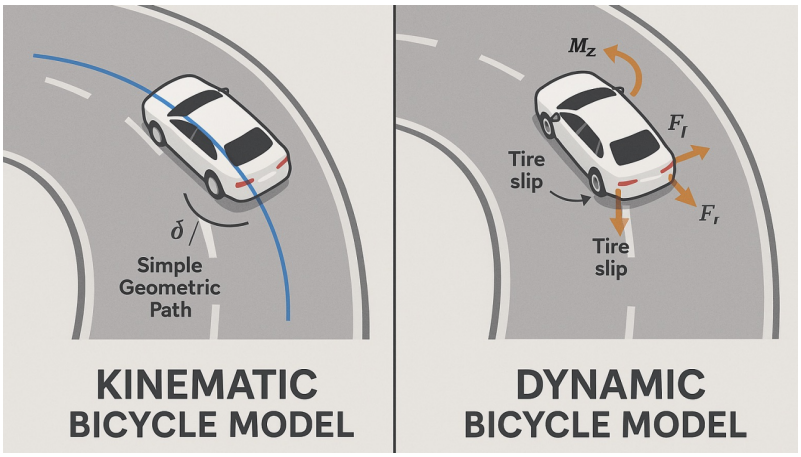



Figure 2.9: Kinematic Models vs Dynamic Models [41]

a model is the **constant velocity model**, which assumes that the agent is moving at a constant speed and direction between each time step. This model is often used in SLAM systems, as it is simple to implement and can provide good results in practice. Another example is the **constant acceleration model**, which assumes that the agent is accelerating at a constant rate between each time step. This model can be more accurate than the constant velocity model, but it is also more complex to implement and requires more computational resources.

### Loop Closure

With the first works on Graph SLAM, the notion of **loop closure** was also introduced [42]. Loop Closure algorithms have the objective of recognizing when the agent has returned to a previously visited location and are crucial for the convergence of many SLAM systems. Essentially, by identifying a loop closure, many *SLAM backends* can correct the drift that is inherent to the problem and drastically improve the accuracy of the system. This is particularly relevant for implementations using a factor graph representation because this approach uniquely allows for the creation of specific constraints for this type of event i.e. when the agent returns to a previously visited location, a **loop closure constraint** can be introduced connecting the original pose and the current one.

Additionally, worth noting is that there are two popular ways of performing loop closure: **appearance-based** and **geometric-based**. Appearance-based algorithms use visual information to identify loop closures, while geometric-based algorithms use geometric information, such as the relative position of landmarks or features in the environment.

### Tightly-Coupled vs Loosely-Coupled Approaches

Throughout this document, SLAM approaches are often categorized as either **tightly-coupled** or **loosely-coupled**:

- tightly-coupled systems fuse raw sensor data together;

- loosely-coupled systems often fuse the output from different modules that process sensor data instead of fusing sensor data directly.

While tightly-coupled solutions tend to be more accurate as complementary data from sensors can be taken into account to a further degree. However, loosely-coupled approaches are usually easier to implement, boast reduced computational load and provide a more modular architecture [43].

It is worth noting that this definition is not very stable and simply an interpretation of the author, which might ease in the comprehension of the works present.

# Chapter 3

# Literature Review

A good understanding of the SLAM problem at hand and knowledge of the most common solutions available are crucial for the development of a new system. In this chapter, a review of the relevant literature for the formulation of a solution to the presented problem is carried out. The most recent advancements in the field of SLAM are presented, with a focus on solutions catered towards LiDARs and environments where high accelerations are common. Given the wide applicability and success of Machine Learning and Deep Learning in the field of Computer Vision, a review of the most important works in the application of these techniques to SLAM is also presented. Then, a review of the existing work in SLAM for autonomous racing is made. To conclude, the last section wraps the analysis made, outlining the most important points and the position of this dissertation in the field.

## 3.1 LiDAR SLAM and Odometry

With the advancement of the field, SLAM has seen a division in focus into a multitude of different challenges. Vision SLAM has seen an increase in popularity with the advent of Deep Learning. New Neural models have renovated the hope that Cameras to be able to perform the SLAM task with accuracy comparable to the one boasted by LiDAR sensors. Alongside this, cameras are usually cheaper and lighter than LiDAR sensors, which makes them an attractive option for many applications. Even so, LiDAR SLAM is still at the forefront of the field, with many of the most accurate and reliable solutions being based on LiDAR data. Moreover, advancements in LiDAR technology and the emergence of Solid State LiDARs, manufactured and sold at a lower price, have made LiDAR SLAM more accessible to a wider audience [44]. The following sections will examine the most significant solutions of the last decade. Additionally, attention will be given to approaches relevant to high-dynamic scenarios, such as those encountered in drone navigation.

### 3.1.1 Lidar Odometry and Mapping

One of the most important works developed regarding LiDAR SLAM was Zhang and Singh's Lidar Odometry and Mapping (LOAM) [45], which presented a loosely coupled approach to SLAM

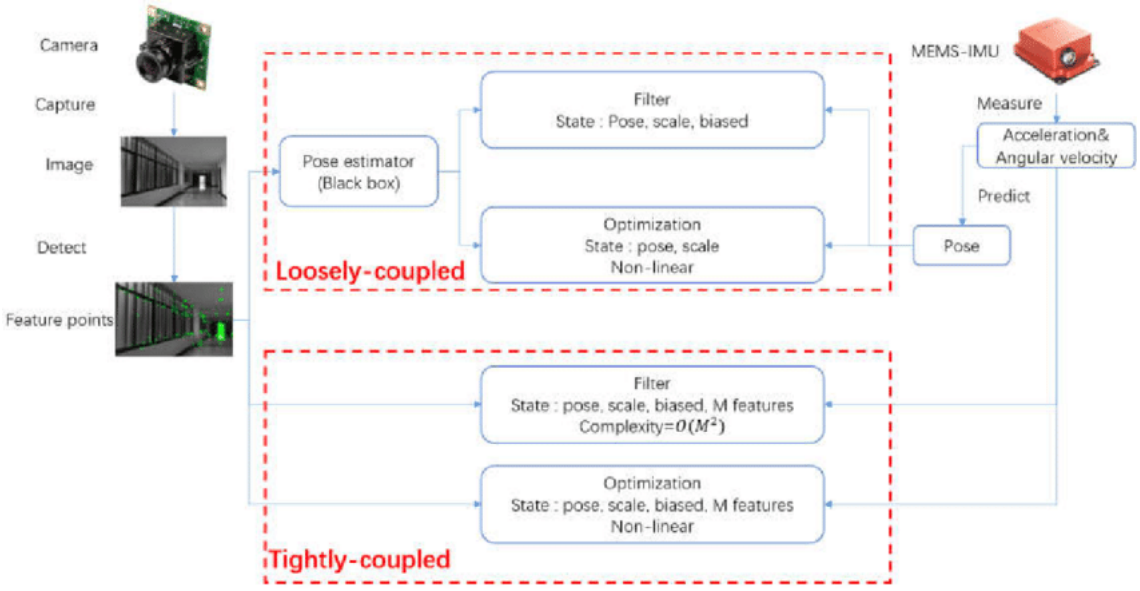



Figure 3.1: Loosely Coupled vs Tightly Coupled SLAM Schematic [46]

i.e. sensor data is processed independently instead of being directly fed into the back-end algorithm [46], as depicted in Figure 3.1. LOAM uses a feature matcher to extract edge and plane features and uses them for two parallel tasks: calculation of odometry is performed during the LiDAR sweep by minimizing the differences between the current and previous sweeps' extracted features; mapping is performed at the end of each LiDAR sweep by matching 10 times the features used in the odometry algorithm to the accumulated point cloud map. LOAM is capable of running under a 10Hz frequency and has been proven to handle high-velocity scenarios better than other popular algorithms, such as Google Cartographer [47].

Zhang and Singh's approach has since been widely used and adapted in many other works. One example is **LeGO LOAM** [48], which is a lightweight version of LOAM with ground-optimized feature extraction, focused on providing a LOAM technique suitable for UGV's that traverse rough terrain and carry low processing power. By including a segmentation phase before the feature extraction which separates ground from non-ground features, this solution is able to obtain better results in environments where the soil is the most reliable source for odometry estimation. The presented results in the original paper suggest these efforts were successful. LeGO LOAM is not only able to reduce the overall processing time of the algorithm, executing at nearly one-tenth the rate of its predecessor in the setup described, but also increases the robustness of the system, presenting significantly smaller pose estimation errors in most scenarios tested (granted it was tested in ). Other works arose from LOAM with success, such as F-LOAM [49] and V-LOAM [50], which cemented this approach's position in the field.

### 3.1.2 LIO-SAM

LOAM [45] set the ground for loosely-coupled implementations that allow for efficient and accurate odometry estimates using LiDAR. However, LOAM was still far from a perfect solution, showing drift problems in very long trajectories. **LIO-SAM** aims to provide a solution to this problem, additionally being a method easier to integrate with other sensors. Their proposal is to fuse LiDAR Odometry with IMU measurements following a Smoothing and Mapping approach (SAM), using a factor-graph. The authors also decided to perform scan matching against a local map instead of the global map matching proposed by LOAM, which contributes positively to the efficiency of the algorithm. This approach also has the additional benefit of being easy to integrate with other sensors and measurements, as well as loop closure modules, due to the underlying factor graph.

### 3.1.3 Fast LIO

**Fast LIO** [51] is another important work in the field of LiDAR SLAM. Fast LIO is a LiDAR-Inertial Odometry system tailored for Solid State LiDARs. Xu and Zhang make the point that loosely coupled feature-based approaches like [48] tend to fail in unstructured environments or with LiDARs with a small field of view, which is usually the case with Solid State LiDARs. The work presents a tightly-coupled approach, which fuses the LiDAR data with the IMU data using an Iterated Error State EKF, a Kalman Filter which operates in the error state space, which usually is smaller and involves simpler, more efficient operations. Normally, fusing all the data in a tightly coupled manner would be unfeasible in real-time for such an application. However, the authors present a novel method for computing the Kalman Gain which does not scale with the amount of measurements (which in this case are many, as they correspond to LiDAR captured features), but with the size of the state (which in this case is fixed size and contains only information related to the 3D Pose of the agent).

The same authors later presented **FastLIO2** [18], which doubled down on the objective of robustness against unstructured featureless environments and resulted in one of the best all round solutions out there. It did so by directly registering the Point Cloud from the sensor instead of extracting features, which allegedly allows for the exploitation of subtle features in the environment. It also transitioned to store the map in an incremental k-d tree, a new data structure which improved execution time and compensated for the increase in storage data. The result was one of the best LiDAR Odometry solutions out there, with better accuracy in most scenarios in comparison to its predecessor while maintaining efficiency. More notably, despite being a thightly-coupled solution, execution times remained lower than the most popular loosely-coupled solutions out there, like LIO-SAM. This implementation can run at 100Hz and work in a wide array of scenarios, including drone trajectory estimation, shown both in the original article and in other works [52].

Despite somewhat recent, these works have served as ground for many other solutions, such as Faster-LIO [53], and Fast-LIVO [54], including one other work that will be reviewed later in this chapter.

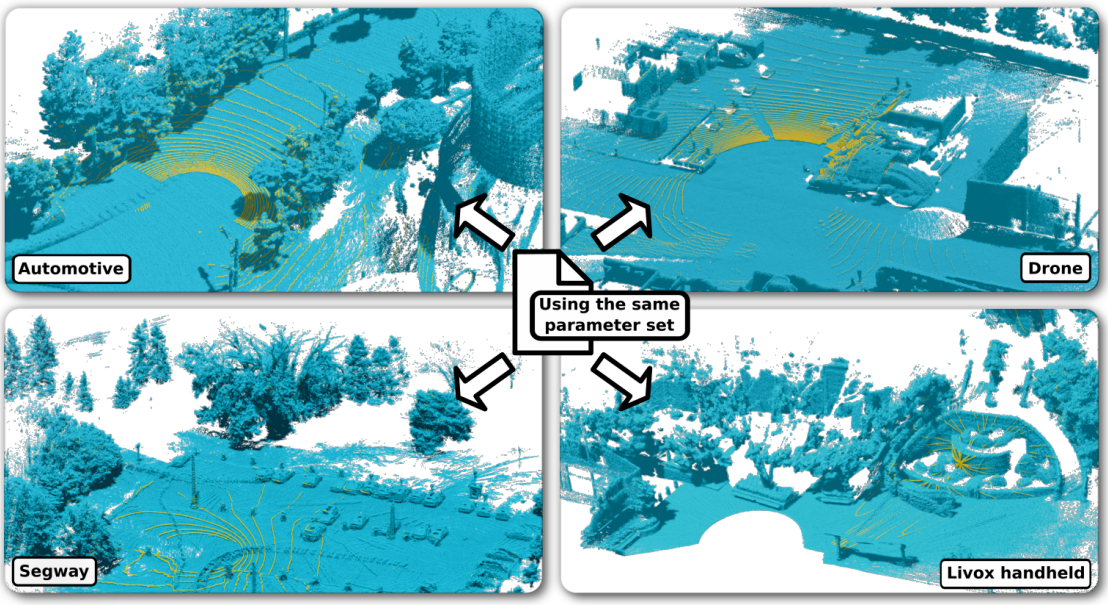



Figure 3.2: KISS ICP Results in multiple scenarios [55]

### 3.1.4 ICP Based Methods

Other widely adopted approaches for performing SLAM using a LiDAR sensor are based on the raw application of the **Iterative Closest Point** (ICP) algorithm. ICP is a point-to-point matching algorithm that iteratively minimizes the distance between two point clouds. KISS ICP attempts to make a point for the algorithm, arguing there is strength in simplicity and creating a LiDAR Odometry module that can run at 100Hz [55] and performs reasonably in almost any scenario, having been evaluated both for handheld, drone and car motion profiles, as shown in Figure 3.2.

Another equally successful approach is CT-ICP [56]. In contrast with the former, CT-ICP is fitted with a SLAM back-end built with g2o [36] and presents both an innovative manner to perform LiDAR odometry and a fast method for loop closure detection. Both of these have an open-source implementation available and score amazing results in the Kitti Dataset [57], one of the most popular benchmark datasets for SLAM and odometry.

In their recent work, Lee et al. (2025) presented GenZ-ICP [58], a LiDAR odometry method aimed at improving reliability across both typical and degenerate environments. Instead of depending on a single error metric, the algorithm intelligently blends point-to-plane and point-to-point formulations, with the weighting adjusted according to the geometry of the scene. This strategy helps avoid ill-conditioned optimizations, especially in long corridor settings. It was shown to achieve competitive accuracy on several benchmark datasets, boasting a similar performance to SOTA implementation in the KITTI Dataset [57] such as KISS-ICP, and outperforming such solutions in corridor scenarios such as the Ground-Challenge Dataset [59].

### 3.1.5 LiDAR SLAM Analysis

The works mentioned in this section represent the forefront of LiDAR SLAM research. All the implementations have one thing in common: they are focused on creating reliable and robust systems that are capable of running in a variety of scenarios. There are no consistently available and well-identifiable features in all environments the algorithms are designed to function in. Consequently, feature extraction, direct point cloud registering and scan matching are core components of most of these approaches, with the works giving significant importance the front-end modules of the SLAM solution.

This problem, however, applies to a lesser extent in the Formula Student Driverless scenario, as the track is composed of cones. The presence of such a distinguishable, unique and consistent feature/object renders most of these approaches unnecessary and unfeasible from a mapping point of view. Moreover, there is little direct motivation for generating dense and detailed scenes of the environment in a Formula Student scenario, as the objective of the mapping task is to identify the track boundaries.

Even so, some of the ideas introduced might apply to the issue analyzed in this dissertation. For instance, the concept of LiDAR odometry can improve the velocity and pose estimates at elevated rates. Many of the articles mentioned also make a point for the robustness of loosely coupled systems, obtaining remarkable results without loop closure or tightly fusing data from multiple sensors. On the other hand, the concept of continuous-time odometry adopted by most of them [45, 56] shows some potential. Generating odometry measurements multiple times per scan can have a positive impact on the fulfillment of the system's requirements, in the face of the low latency needed for racing scenarios.

## 3.2 Deep Learning in Localization and Mapping

Deep Learning has been a trending theme in a multitude of different areas of science, due to the phenomenal potential of Deep Neural Networks to learn complex patterns and adapt to different scenarios. Computer Vision has been one of the most affected fields, having seen a drastic advancement in most disciplines with the application of Convolutional Neural Networks [60], among other Deep Learning architectures. Thus, despite the somewhat ephemeral nature of the DL application to SLAM, a review of the most important works is still in order.

Camera-based solutions for SLAM suffer from the inherent challenge of depth estimation, as the sensors they use at its base output two-dimensional representations of the environment. Stereo and RGB-D setups have been widely used to solve this issue. However, the advent of Deep Learning has brought forth a new solution to this problem, with works having been carried out to make Monocular Visual SLAM viable. One notable example is CNN-SLAM [61], which uses a Convolutional Neural Network to estimate the depth of the environment. Previous work in Monocular SLAM's depth estimation revolved around the assumption that a moving camera generates consecutive frames that can be treated as a stereo system. The authors of [61] attempt

to solve the problem of a lack of a good estimation of the absolute scale of the environment from which these other approaches suffer. They do so by fusing the depth estimation from this method with the one provided by pre-trained CNNs. The system proposed, although far from generating clear and dense maps, yields considerably better results than previous methods.

Other works have been carried out attempting to use Deep Learning in Visual SLAM. Namely, [62] presents a fully learning-based Visual Odometry system. It leverages the TartanAir [63] dataset to train a model which is capable of performing on a wide variety of scenes, which is a common pitfall of learning-based approaches.

Despite the clear predominance of Machine Learning approaches in Visual SLAM, LiDAR data is also capable of being leveraged by said techniques. Nurbert et al. [64] present a self-supervised learning approach to LiDAR odometry which, in contrast to most systems, does not require any ground truth data for training. The motivation behind the work is the common problem reviewed in the previous section: processing of the entire Point Cloud is too costly and geometry-based feature extraction or down-sampling approaches can often be unreliable. With this in mind, using learning-based approaches can solve the high computational cost required to process all sensor data. The system developed does not encompass a pre-processing step, making it capable of being used in real-time applications with only a light processing unit. The approach also shows to be capable of yielding results on par with other state-of-the-art methods.

Not many other articles have been published on the topic, with [65] following a similar unsupervised approach, yet focusing on building a geometry-aware neural network, fed with vertexes as inputs rather than the raw point cloud. This approach originated similar, if not slightly worse results, according to [64].

## 3.3 State Estimation in Autonomous Racing

The SLAM problem is commonly required to be solved for most mobile robotics applications. When it comes to competition and racing, the prospect is no different. There are two major types of competitions where SLAM is a key component: 4-wheel vehicle racing competitions, to which the problem addressed in this work pertains, and drone racing competitions.

In the context of drone racing, the SLAM problem differs in a multitude of ways, namely the fact that drones require localization and mapping in three-dimensional space, as opposed to the two-dimensional space that is required for most ground vehicles. For this reason, drone racing will not be further discussed in this work.

When it comes to 4-Wheel Autonomous Vehicle Racing, some main competitions and prior applications come up. There are a few that generated research work in the area of state estimation, such as the **DARPA Grand Challenge**, the **F1 Tenth** and the **Indy Autonomous Challenge**. More recently and famously, the **Abu Dhabi Autonomous Racing League** took place, but not enough time has passed for publications to arise. Despite the variety of competitions, the majority of work found was in the context of two competitions: **Roborace**, a championship for autonomous

electric vehicle racing, and the **Formula Student Driverless** competitions [66], which naturally applies to the problem at hand.

In the following section, some of the most important work carried out regarding state estimation and SLAM in the context of Autonomous Racing will be reviewed. A special focus will be given to the Formula Student Driverless competition, as it is the one that most closely resembles the problem at hand, as well as the one that provides the largest amount of research material.

### 3.3.1 AMZ Racing

AMZ Racing is a Formula Student team from ETH Zurich that has marked the field of Autonomous Racing with their work. They won the first 3 FSD competitions from FSG and published multiple articles describing their approaches, which are widely renowned in the Formula Student community.

#### First System

In 2018, the team described the perception and state estimation system they developed for their first autonomous vehicle [67], which amounted to great success, winning the 2017 Formula Student Driverless competition from FSG. The system was composed of a complex and vast array of sensors, both for localization and for perception, including a stereo camera, a 3D LiDAR sensor, a Ground Speed Sensor, an IMU and wheel encoders, among others. The State Estimation module is divided into two separate programs:

- **velocity and pose estimation** is performed with the aid of an Extended Kalman Filter (EKF), fusing information from the sensors available;
- **SLAM and Localization** problems are solved with an implementation of FastSLAM [27], changed to perform localization when after a first lap is completed and fitted with a Loop Closure module to detect said event.

Such a complex array of sensors and data sources could easily lead to delays in data processing, *i.e.* data from a sensor is waiting to be processed by the EKF while it is performing an update step. To avoid this, while the EKF implemented is in the midst of an update step, it degenerates to a Steady State Kalman Filter, which processes the incoming data assuming no alteration of the covariance matrix, leading to a constant time Kalman gain calculation (rather than the usual linear complexity). The vehicle state estimation module also includes an outlier rejection module based on a chi-squared test.

The FastSLAM implementation is fed by a rather simple perception pipeline composed of motion undistortion, ground removal using RANSAC, clustering based on cluster size and density relative to the car's distance and a Nearest Neighbor filter, which removes clusters of cones near other clusters. The SLAM system is fitted with a loop closure detection method: each particle has an inherent state machine which derives its state from the relationship between its current location

and its starting one; once all particles are close enough to their initial position, loop closure is considered.

The system proposed is evaluated in multiple racing scenarios in which the test vehicle reached velocities of 90 km/h. The article mentions an RMSE of 0.18 m for localization when compared to the ground truth, which is remarkable given the speeds and accelerations at which the autonomous vehicle was travelling. According to the authors, outliers from the Ground Speed Sensor were also systematically rejected by the chi-squared test. The Localization and Mapping module took between 7 and 28ms to perform an iteration, which is under the latency of a typical LiDAR sensor. Results on the accuracy of the Fast SLAM implementation are not provided in this article.

### Updates to the Pipeline

A few months later, in September of the same year, another work was published, focusing once again on the updates for the perception and state estimation pipelines [68]. The team portrays their pursuit of a highly reliable system, inferred by the title itself.

In the new version of the State Estimation and Mapping system, the velocity and pose estimation are changed to include acceleration in the state space and remove the pose, now being fully independent of the SLAM module, as depicted in Figure 3.3. Aiming for a more robust velocity estimation, tire slip is also added to the state vector:

$$\left[v_x, v_y, \omega_z, a_x, a_y, sr\right] \tag{3.1}$$

With the new state estimator, acceleration, angular velocity and consequently linear velocity estimates take into account a dynamic vehicle model, which includes tire modelling following the Pacejka Magic Formula [69]. This new EKF is therefore more robust, as taking into account dynamics will improve the accuracy of the overall velocity estimates and presents a good complement for the IMU based estimation, as they tend to falter in different scenarios.

The system now boasts both a mono and a stereo setup for camera-based cone detection, on top of the 3D LiDAR. The LiDAR pipeline is updated to have a more robust pre-processing step, as well as a cone coloring module, based on a CNN trained to distinguish yellow cones from blue cones through the intensity gradients of the cone's clusters. Both camera pipelines are shown to use YOLOv2 for cone detection. The mono setup exploits the prior knowledge of the cone's dimensions using Keypoint Regression to match the detected cones and infer the distance at which they are at. In this system, fusion of the different perception pipelines is performed by the SLAM module itself. The previous FastSLAM implementation was upgraded to a **FastSLAM 2.0** algorithm [28].

Later in the same competition year, the team published another paper [70], one of the most renowned works in the competition. The complete autonomous driving system is described, including the perception and state estimation systems. Figure 3.4 provides an overview of the full state estimation system at this point.

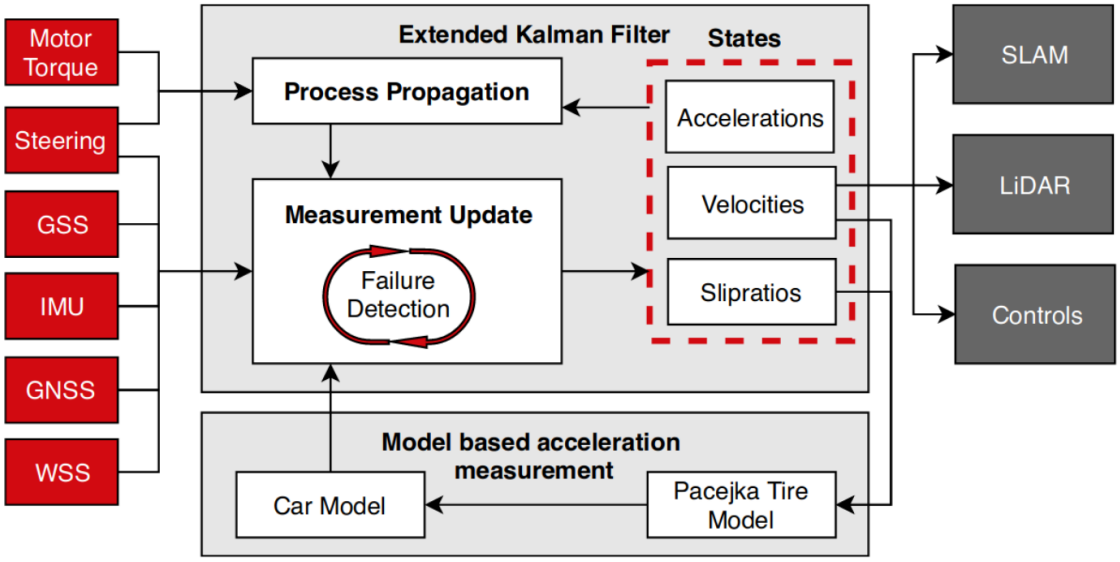



Figure 3.3: AMZ Racing Velocity Estimation Pipeline [68]

The results presented in these articles show that the transition to Fast SLAM 2.0 improved the accuracy of the SLAM module, requiring less particles to reach the same results as the previous iteration. Moreover, the position estimate from the SLAM modules presented only a RMSE of 0.2 m over a 230 m distance, using the same method for ground-truth formulation. It is worth noting that no explicit reference is made to the velocities at which the vehicle was travelling during mapping tests.

### Mapping Task Velocity Increase

In 2020, the same team published updates to the mapping and planning-related modules of their system during the 2019 season [71]. Focusing on the mapping task, the first significant change involves the integration of the perception pipelines. In contrast with the previous system, 2 separate LiDARs are used with the same pipeline, while the stereo setup seems to have been removed. The new system also performs an additional fusion between the two pipelines before feeding the information from them into the SLAM back-end, as depicted in Figure 3.5.

This time around, the SLAM algorithm was completely revamped. As the SLAM algorithm requires cones with sufficient certainty to be able to perform the mapping task with quality, but path planning requires cones with sufficient distance to be able to plan the path, the team decided to split the SLAM module into two separate modules. A local map is generated utilizing a Kalman Filter, which takes only the cones' positions and colors into account. The global map is now generated by a Graph SLAM implementation, with the Ceres solver [37] at its core, which is fed by the local map. Not using direct measurements in this algorithm might improve its accuracy, as admitted by the team, with the justification that its main use is to combat pose error accumulation with loop closure.

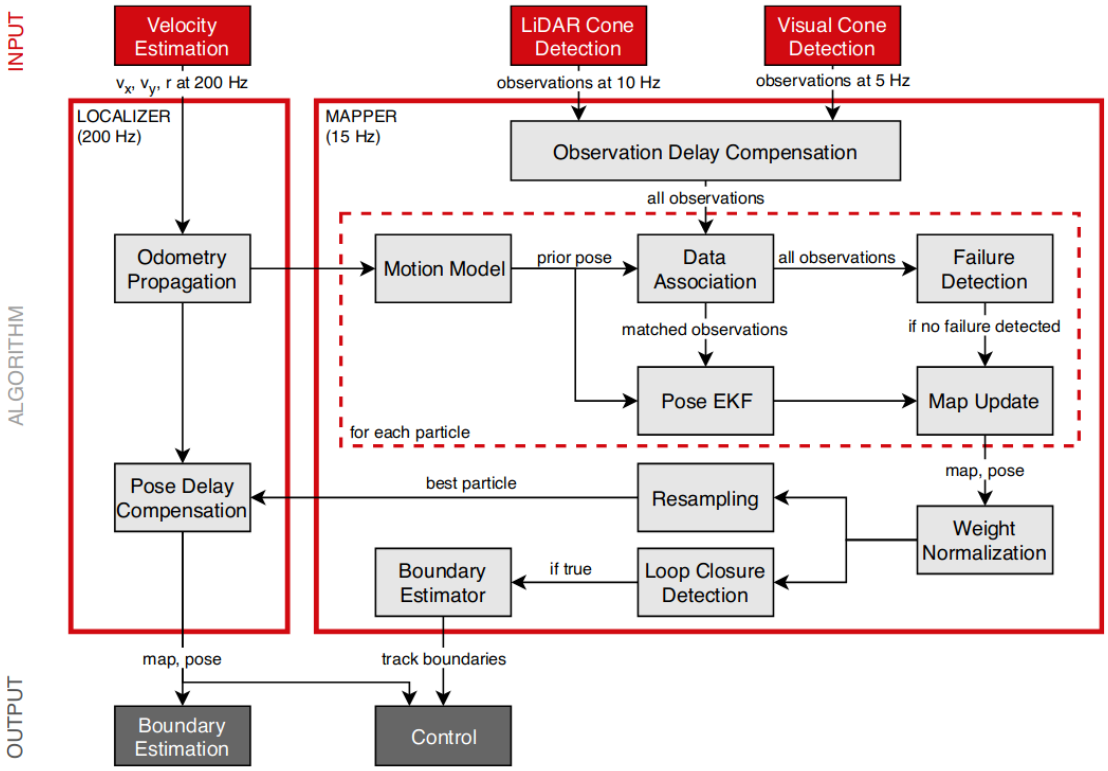



Figure 3.4: AMZ Racing SLAM System Overview [70]

The new approach is evaluated with the aid of a GNSS RTK system for ground truth generation. It is shown that the new approach achieves a slightly smaller RMSE of 0.16 m, revealing that the previous mapping tests were likely carried out at 2.8 m/s, as the result of the previous pipeline used for comparison with this one is of 0.2 m RMSE at that speed. The authors also indicate that this new system is capable of performing the mapping task with similarly low RMSEs (0.29 m) at 12 m/s, a considerable increase, proving the separation into two modules was fruitful.

### Final Analysis

AMZ Racing's system over the years has been well described until 2020. Despite being 5 years-old, the last systems developed are on par or even superior to what is seen in competition today. Some conclusions from the analysis carried out are:

- both Fast SLAM and Graph SLAM present a viable solution for the Formula Student scenario;
- the separation of the velocity estimation from the SLAM module did not improve accuracy, which means that, despite not being shown, was likely carried out to improve estimation speed;
- Graph SLAM was shown to be capable of improving the mapping task accuracy, but a method to fend off the increased computational weight is probably desirable.

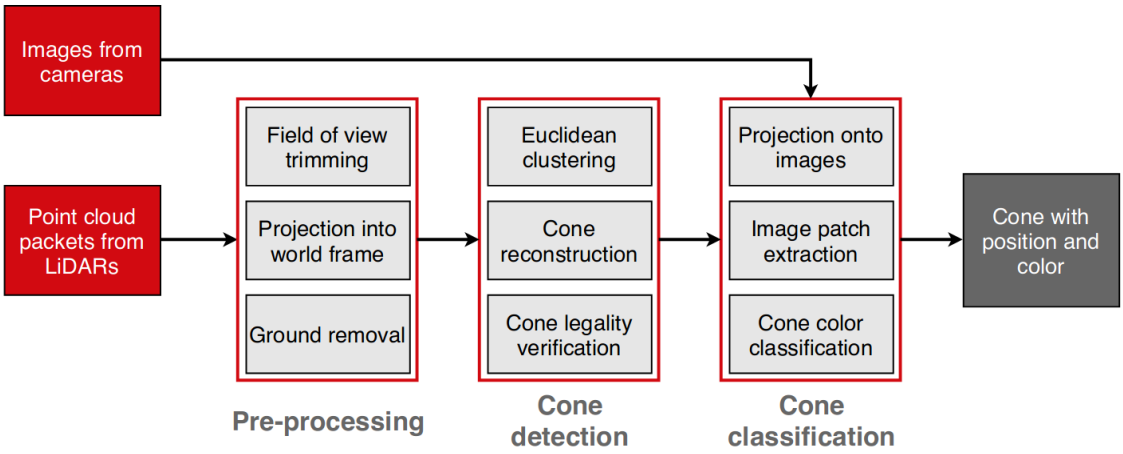



Figure 3.5: AMZ Racing Perception Sensor Fusion Pipeline [71]

The students behind the system were extremely focused on reliability, which led to an extremely robust system, albeit very complex. In essence, the system is quite the feat of engineering, but it is plausible that it is not the most time-efficient solution for the problem at hand. It is also worth noting that the complexity achieved in these works is the result of an entire team of students throughout multiple years.

### 3.3.2 KA Raceing

KA Raceing is a Formula Student team from the Karlsruhe Institute of Technology, which has a long history in the competition. It has been designing autonomous cars since the introduction of the category in 2017. The work published by the team regarding state estimation is analyzed in this section.

The team first described the components of the driverless pipeline in 2020 [72], detailing the system which had competed one year earlier. Despite the completeness of the system and robustness in areas such as vehicle control, in which an implementation of a Model Predictive Controller is used, the state estimation system is rather simple. It is composed of a single EKF SLAM algorithm. The EKF is fed cone positions from a perception pipeline which derives the location of the landmarks solely from a 3D LiDAR sensor, using a separate camera system only for validation and color inference. For motion prediction, data from two-wheel speed sensors installed in the rear wheels is used, along with the yaw rate provided by an IMU. Despite the simple setup, the team obtained fantastic results, finishing behind only AMZ Racing in the 2019 season. Unfortunately, no further data regarding the system's performance was made available.

#### SLAM Solutions Comparison

In the same year, Le Large presented its work to the team regarding the development of the SLAM module [73]. Its master's thesis explored two different solutions for SLAM in the context of formula student:

- a filtering solution - the improved version of the EKF already developed previously;
- an optimization-base solution - a Graph SLAM implementation using g2o [36].

Both solutions use the same LiDAR processing modules (the perception module, similar to this thesis' case, is not part of LeLarge's work), motion priors/motion model and data association model. When it comes to data association, both algorithms use an implementation of the Joint Compatibility Branch and Bound (JCBB) algorithm from the MRPT library. Aiming to improve efficiency, it is mentioned that only landmarks within the plausible perception's range are considered. A constant velocity model as in previous work is used, which approximates the longitudinal speed of the vehicle as the average of the rear wheel speeds and receives the yaw rate from an IMU:

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ \theta_{k+1} \end{bmatrix} = \begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix} + \begin{bmatrix} cos(\dot{\theta}_k)v_{x,k}\Delta t - sin(\dot{\theta}_k)v_{y,k}\Delta t \\ sin(\dot{\theta}_k)v_{x,k}\Delta t + cos(\dot{\theta}_k)v_{y,k}\Delta t \\ \dot{\theta}_k\Delta t \end{bmatrix} \tag{3.2}$$

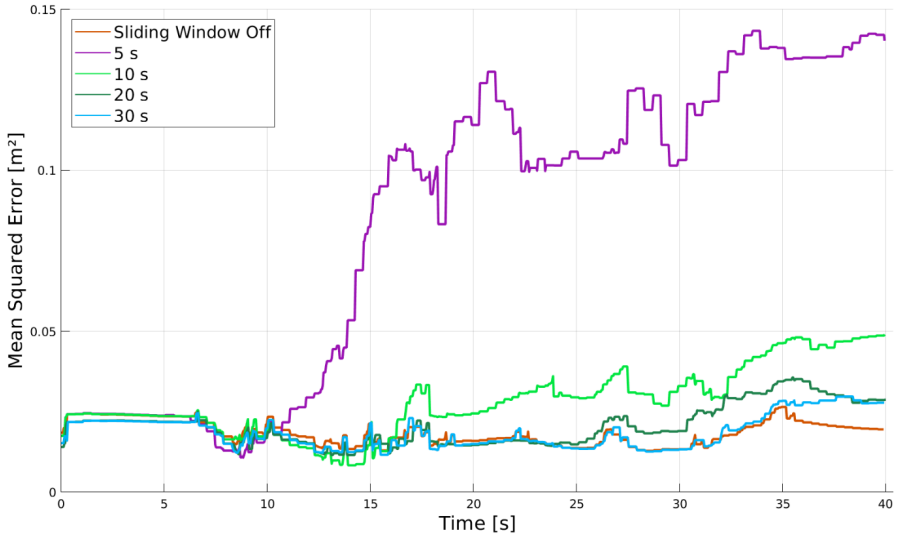
where $v_{x,k}$ and $v_{y,k}$ are the longitudinal and lateral speeds of the vehicle at time step $k$, respectively, and $\dot{\theta}_k$.

Both implementations rely on fairly standard algorithms, with a small twist. To fulfil the real-time requirements of the scenarios the system will face, parallelization was leveraged in both implementations to enable more frequent estimates:

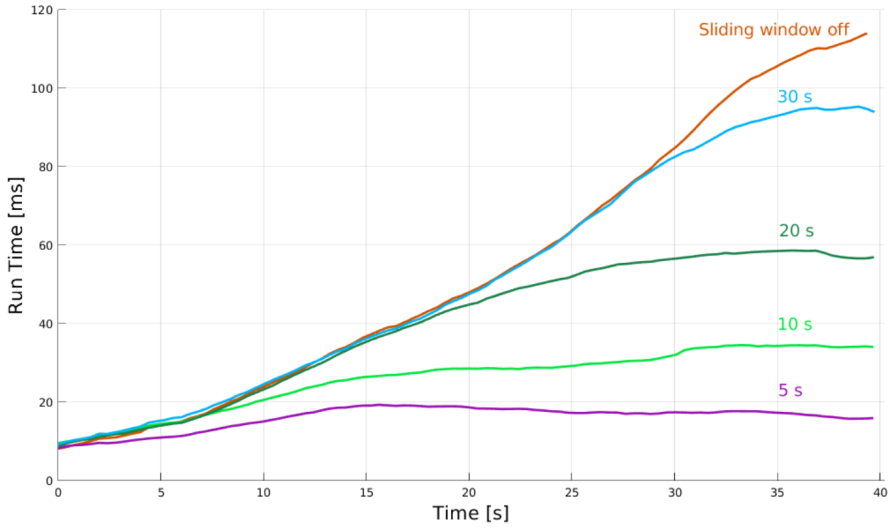
- In the case of the EKF, a second thread is used to perform the update step, while multiple prediction steps are made. This avoids not only possible odometry measure discards, but also increases the frequency at which estimates are made. When the update step finishes, the difference between the vehicle state estimate from the last prediction and the update/-correction is calculated and generates a final new predicted estimate;
- In the case of the Graph SLAM implementation, the first thread is responsible for taking in all the sensor and odometry data and providing a short-term prediction, while the second thread is tasked with cyclically building the graph and solving the optimization problem. On top of that, doubling down on the pursuit of efficiency, the optimization can be performed using a sliding window, which limits the number of edges taken into account.

Both systems are compared with each other and with the previous EKF SLAM implementation. The results show that the new EKF implementation performs slightly better than the old version, while the Graph SLAM outperforms both in terms of accuracy: both EKFs reached a rate of 50% of cones mapped with a displacement higher than 30 cm from the ground truth, with the optimization-based implementation performing always under 30%. RMSE values show the same trend, with Graph SLAM achieving values under 0.05. The data also show that the impact of the sliding window on the efficiency of the algorithm most likely compensates for the negative influence on the accuracy, as shown in Figure 3.6. All implementations are capable of providing

position estimates at a rate of 100Hz, with the Graph SLAM taking the win, consistently performing under 1 ms in both scenarios. The pose estimation accuracy was not evaluated. There is also no mention of the velocity at which the experiments were carried out.




(a) RMSE through lap time in Autocross




(b) Computation Time throughout lap time in Autocross

Figure 3.6: Impact of Sliding window in both accuracy and efficiency of the Graph SLAM algorithm [73]

The system described went on to win the 2021 Formula Student Driverless competition, at FSG showcasing the robustness and reliability of the system developed. One year later, similarly to AMZ, the full autonomous racing system was described in an article [74]. The algorithms remain largely the same, with the exception of a switch to NN-based data association module, which is claimed to be more efficient while maintaining largely similar accuracy.

The work developed by Le Large and the KA Raceing team once again vouch for both the

usage of Graph SLAM in a Formula Student scenario and for the inclusion of a twist in the implementation to allow for quicker pose estimates. The sliding window approach has also been shown to be a viable solution for the minimization of optimization step time reduction. The system is however mostly evaluated at unknown vehicle speeds.

### 3.3.3 FST Lisboa

FST Lisboa is another Formula Student team that has published work regarding their Autonomous System solution for a Formula Student Driverless vehicle. The team has proven its worth, scoring 3rd place in the last two FSG driverless competitions. At the origin of the base system were multiple master theses, which will be reviewed in this section, focusing on the work related to SLAM and state estimation.

The development of a robust dual sensor perception pipeline is detailed in [75]. Morgado develops a system similar to one developed by AMZ and KA Raceing, in which the 3D LiDAR performs localization of the cones, while the camera is mostly used to validate this positioning and identification and infer cone color. A diagram of this system is depicted in Figure A.1. No further details on this pipeline will be explored, as it is not the focus of this work.

At the same time this perception pipeline was developed, another master thesis took place, focusing on the development of a SLAM system for the vehicle [29]. Both systems seem to, in a sense, have been developed for one another. Similarly to [73], this work envelops the conceptualization and implementation of two approaches to SLAM. This time, a Fast SLAM 2.0 is implemented and compared with a Graph SLAM implementation.

Similarly to KA-Raceing's (and this work's) scenario, the SLAM module previously developed in the team was not meeting the requirements, which was a Fast SLAM 1.0 implementation. The author took it upon himself to come up with an improved Fast SLAM 2.0 implementation, developed from scratch in C++ and using ROS [76], based on the original description in [28].

Loop closure is detected in a similar manner to AMZ Racing's FastSLAM implementation, based on a finite state machine. However, this time around, the state machine is not applied to the particles, but rather to the landmarks themselves. Each cone is considered either in view, to have exited view or to have returned to view. A loop closure is detected when the number of landmarks in the last state is greater than a certain dynamic threshold, which depends on the number of seen landmarks and the standard deviation of particles.

The Graph SLAM implementation is implemented on top of the g2o library [36]. The author developed an implementation that once again strays away from performing full optimization steps at each iteration. Two key novelties are presented: the usage of an EKF for each landmark, in FastSLAM fashion, and two different modes of optimization. These differences can be easily understood when framed into the division between the SLAM and localization-only modes:

1. while in Autocross (SLAM mode), the system performs optimization only taking into account the current observations and the pose. The EKF for each landmark enable the system

to correct the initial estimate of the landmark's position, as the first ones are typically the worst given the landmark is being observed at the furthest distance possible;

2. when loop closure is detected, using the same method as described previously, the system performs optimization taking the full map into account. The EKF update is then deactivated and the system enters a localization-only mode.

One other focus of the work conducted by Lopes was to improve the efficiency of the data association module, which was based on the Maximum Likelihood algorithm. To avoid iterating through the entire set of landmarks, observations are matched to the previous observations inside the perception module using the Bhattacharyya Distance. The matched landmarks carry an ID when sent to the SLAM module, which is used to perform the matching. Maximum likelihood is used on the SLAM's side when this matching is not possible.

In this work, two motion models are also analyzed, based on the Unicycle Model and the Kinematic Bicycle Model respectively. Because the first one assumes the center of rotation is the center of the vehicle in all scenarios, it naturally falls short in terms of accuracy. For that reason, the latter is used, which follows Equation 3.3:

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ \theta_{k+1} \end{bmatrix} = \begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix} + \begin{bmatrix} v \cdot cos(\beta + \theta_{t-1}) \\ v \cdot sin(\beta + \theta_{t-1}) \\ \frac{tan\delta \cdot cos\beta}{L} \cdot v \end{bmatrix} \tag{3.3}$$

As stated, this motion model takes as input data from a separate velocity estimation pipeline that uses the encoders of 4 in-wheel motors to derive the rotation of the wheels. This pipeline accomplishes the estimation with an EKF fusion of the sensor data and a 4-wheel vehicle model derived from the LuGre tire model. This algorithm was developed as an observer for an MPC controller, whose objective was to perform torque vectoring in a four-wheel drive electric vehicle [77].

The systems are evaluated and compared both in a simulated environment and in a real scenario. The execution times recorded shown are far below $100\mu s$ for all methods, which is quite the feat for a SLAM algorithm. Even so, the context in which they were retrieved and the method used to do so are not disclosed. For the on-ground scenario, the tasks were evaluated at a speed of 3.5 m/s and ground truth was generated using a Total Station connected to a GNSS RTK system. Both simulation and on-ground results show once again that Graph SLAM is the clear winner, with a lower value for the finishing RMSE value over both scenarios.

Curiously, the results presented in terms of RMSE are significantly higher than the ones presented by [73] throughout the run, as depicted in Figure 3.7. This may be explained by the fact that optimization is only performed at the end of the run, in comparison to the sliding window approach used by KA Raceing.

The results vouch, once again, for the usage of Graph SLAM in this context. The work also presents a unique approach to pose graph SLAM, fusing the typical approach with characteristics from the FastSLAM algorithm.

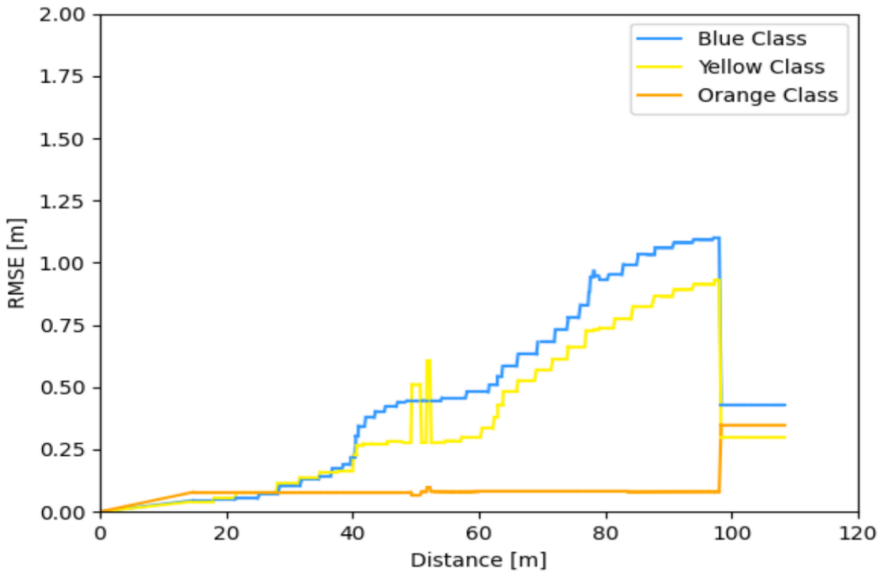



Figure 3.7: Graph SLAM RMSE in real-world mapping task [29]

### 3.3.4 BCN eMotorsport and LIMO-VELO

A student member of the BCN eMotorsport from ETSEIB-UPC has developed quite a different approach to the SLAM problem in Formula Student. Instead of following the route of creating a SLAM module that loosely fuses data from a Perception pipeline and other sensors, it invested on using a LIO method. LIMO-VELO [78] is inspired by FAST-LIO2, having created a solution with similar basis but better suited for the Formula Student environment and the team's prototype, most notably:

- Like FAST-LIO2, interprets the point-clouds as continuous streams of data;
- Performs mapping only with points from past scans, in order to avoid matching map points to features from the same scan;
- To fend off degeneration caused by small fields of view, substitutes the LiDAR estimates by IMU predictions in the degenerate directions.

This approach leads to the generation of a dense representation of the map, which is harder to use and integrate with other modules, as coordinates of track limiting cones are not identified by the SLAM module.

Even so, Segarra's work as proven the applicability of LiDAR odometry to Formula Student scenarios. LIMO-VELO was able to accurately localize BCN eMotorsport's vehicle quite accurately and in real-time, with performance comparable to most implementations described above, leveraging less complex sensor suites and velocity estimation pipelines.

On top of this, a comparison between is method and other popular implementations such as FAST-LIO2 and LIO-SAM [21] is provided for a Formula Student scenario. The analysis confirms

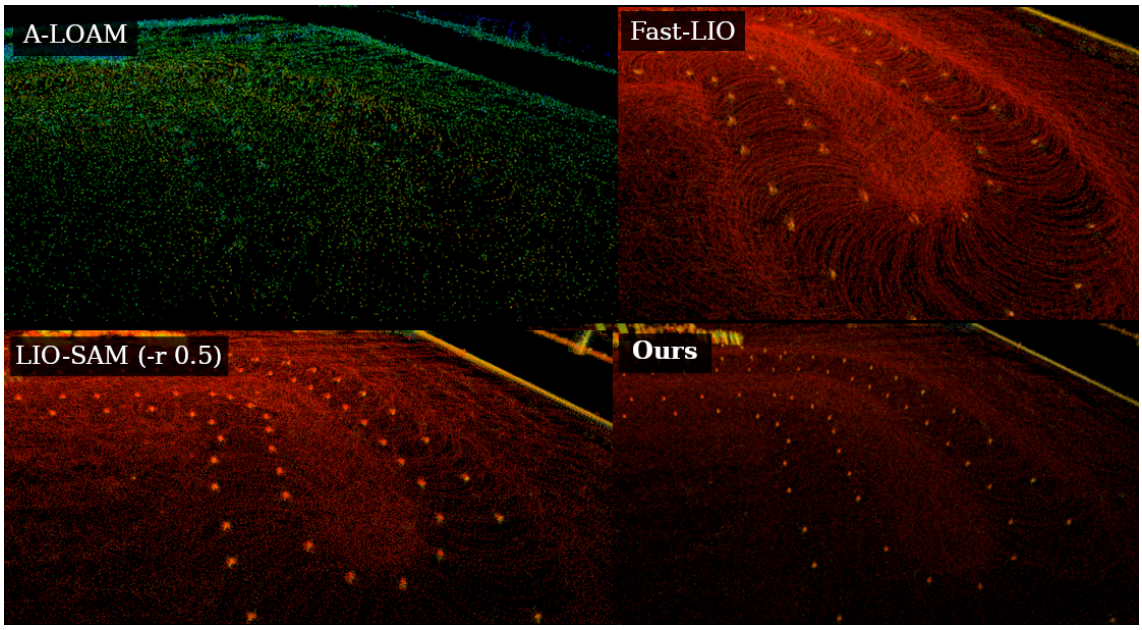



Figure 3.8: LIMO-VELO vs FAST-LIO2 performance in a Formula Student scenario [78]

the trend of other studies, validating FAST-LIO2's usability for such scenarios, depicted also in Figure 3.8. It also indicated that LIO-SAM was not capable of handling the real-time requirements of the presented scenario, having been run in half the real speed. LOAM based approaches were also evaluated, presenting significant drift in comparison to other approaches.

### 3.3.5 Other Teams

Many other teams compete in Formula Student Driverless competitions throughout the world. Surprisingly though, the articles and thesis analyzed above not only represent the most renowned and well fundamented work but also most of the work available, with other papers published often having only brief mentions of an EKF implementation for SLAM [79, 80]. Even so, information regarding the work of other teams has been obtained through other methods, such as information sharing during competitions and events. Moreover, some other shorter articles deserve a mention in this section, as they provide further insight into the trends in the field.

Through attendance at the Autonomous Racing Workshop (ARWo), organized by former team members of the e-gnition Hamburg e.V. team, the team was able to gather statistics regarding the systems used.

The great majority of teams use a Graph SLAM implementation based on the g2o optimization library [36], many of which implement a sliding window approach, fed by an EKF fusion of data from wheel encoders and IMUs mostly, sometimes including GSSs. AMZ, Lund and Revolve NTNU stood out of the pack by using iSAM2 as a backend for SLAM. The second most common choice for SLAM was the Fast SLAM algorithm, with the EKF still being presented by some teams. Some teams used different algorithms for localization and for SLAM modes, like Oxford

Brookes Racing and Rennteam, which used EKF for localization but FastSLAM 2.0 [28] and Graph SLAM for the first lap of autocross respectively.

Machine Learning was used by FSUPV to estimate velocity from wheel encoders and steering angle, rather than the classical kinematics-based approach. Ecurie Aix and GET Racing used the *robot_localization* ROS package for velocity estimation, the former fusing information from 2 IMUs and the SLAM module itself, while the latter used IMU and wheel encoder data. It is worth mentioning, however, that GET Racing intends to switch to a self-developed EKF implementation.

Most teams estimate cones' relative positions using a perception pipeline along the lines of the ones described before, based on either LiDAR or LiDAR and camera data. BCN eMotorsport and FaSTTUBe were the exceptions, as they reversed the order of operations: while BCN eMotorsport developed an EKF implementation to fuse IMU and LiDAR data and generate a dense point cloud map, FaSTTUBe used FAST-LIO. The dense point cloud maps were then passed through a typical point cloud perception pipeline, removing ground points and identifying the cones' positions.

Combining this information with data available in the FSG website Table 3.1 was constructed. Both from the table and the analysis carried out, some conclusions can be drawn:

- Graph SLAM is the most popular choice for SLAM in the competition;
- EKF Sensor Fusion is very well established for velocity estimation;
- Alternative approaches are rare and have not scored as well. The teams presented are already among the best in the world, as FSG is the most competitive in terms of driverless racing, which means other approaches may exist but are not as successful. For instance, it is a well-known fact that many other teams use EKF SLAM, but do not score as well as the ones presented here;

It is worth noting that this table does not constitute solid statistical basis alone, but helps to confirm tendencies already inferred from other sources, such as the other SLAM related articles analyzed before.

### 3.3.6 Deep Learning in Formula Student Driverless

Formula Student Driverless projects, has evident from the analysis carried out, are a great source of research in the field of SLAM and state estimation. Deep learning techniques are often applied in the processing of camera data. Formula Student was no exception to this trend, with CNNs being used for object classification in multiple occasions [68] [74]. Neural Networks have also been tampered with for the estimation of velocity from motor, wheel and IMU data. For instance, [81] have developed and compared an approach based on Recurrent Neural Networks (RNNs) to a more traditional EKF approach for velocity estimation, concluding that the RNN approach is capable of providing better results in terms of accuracy and robustness. KA-Raceing students have also leveraged Neural Networks to serve as process and sensor models for the EKF used in their velocity estimation pipeline [74].

Table 3.1: Matching between core state estimation system choices and competition results for Formula Student teams in FSG in the last two years.

| **Team** | **SLAM Back-end** | **Velocity Estimation** | **Map Type** | **Placement 2022** | **Placement 2023** |
|---|---|---|---|---|---|
| Chalmers | Graph SLAM (g2o) | EKF Sensor Fusion | Landmark | - | 1/30 |
| KA Raceing | Graph SLAM (g2o) | EKF Sensor Fusion | Landmark | - | 2/30 |
| FST Lisboa | Graph SLAM (g2o) | EKF Sensor Fusion | Landmark | 3/19 | 3/30 |
| TU Fast | FastSLAM 1.0 | EKF Sensor Fusion | Landmark | 4/19 | 8/30 |
| AMZ Racing | Graph SLAM (iSAM2) | EKF Sensor Fusion | Landmark | 6/19 | 7/30 |
| e-gnition Hamburg e.V. | Graph SLAM (g2o) | EKF Sensor Fusion | Landmark | 7/19 | 15/30 |
| Rennteam | Graph SLAM (g2o) and EKF | Not Found | Landmark | - | 10/30 |
| Ecurie Aix | Graph SLAM (g2o) | EKF Sensor Fusion | Landmark | 9/19 | 14/30 |
| FSUPV | Graph SLAM (g2o) | Machine Learning Approach | Landmark | - | 12/30 |
| FS Team Talinn | FastSLAM 2.0 | Not Found | Landmark | - | 13/30 |
| BCN eMotorsport | Iterated Kalman Filter | IEKF Sensor Fusion with LiDAR Odometry | Dense Pointcloud | 8/19 | 17/30 |
| FaST TUBe | FAST-LIO | Not Found | Dense Pointcloud | 13/19 | 16/30 |
| Revolve NTNU | Graph SLAM (iSAM2) | EKF Sensor Fusion | Landmark | 10/19 | 17/30 |
| DART Racing | Graph SLAM (g2o) | EKF Sensor Fusion | Landmark | 5/19 | 23/30 |
| Elbflorace | Graph SLAM (g2o) | EKF Sensor Fusion | Landmark | 14/19 | 20/30 |

However, the usage of Machine Learning techniques is still quite limited, with most teams opting for more traditional approaches. On top of that, the usage of Deep Learning for SLAM is still in its infancy, and without precedents in this context apart from the application to camera data processing.

### 3.3.7 Autonomous Racing outside Formula Student

As mentioned at the beginning of this section, other autonomous racing competitions exist and have generated some work in the field. Most of these scenarios rely heavily on the existence of a GNSS signal to perform localization. For example, Stanley, the winner of the Darpa Grand Challenge in 2005, used an EKF to combine data from a GPS, an IMU and wheel encoders to estimate the vehicle's pose [82]. This is also the case of the Indy Autonomous Challenge, in which a solid GNSS signal is a safe assumption. Still, EKF cements itself as a sensor fusion methodology for localization or velocity, with team TUM Autonomous Motorsport using it to fuse GNSS data with IMU information for this challenge [66].

As mentioned before, Roborace includes a greater amount of research carried out in the field of localization without GNSS. Among these, two articles stand out. Schratter et al [83] uses the Normal Distributions Transform (NDT) algorithm to generate a 3D map of the racetrack offline and match the point clouds being received during the challenge to said map, providing a pose estimate. This setup can achieve errors of less than 20 cm with the vehicle travelling at more than 45 m/s. The other approach is presented in [84], which uses the same concept of matching the point cloud received from LiDAR with a pre-built map. This time, a variant of the Adaptive Montecarlo Localization (AMCL) (Montecarlo localization with dynamic adaptation of the number of particles based on the uncertainties) algorithm (IAMCL) is used to that extent. The system builds upon this base with two EKFs: one aiming to provide odometry from IMU and wheel speeds to the AMCL algorithm, another fed by both the LiDAR odometry algorithm, the wheel speeds and the IMU data, providing a high-frequency estimate. Once again, this system is robust enough to provide pose estimates within the 5-meter range at speeds of 200 km/h.

## 3.4 Final Analysis

The literature review performed in this chapter provides a comprehensive overview of the SLAM problem, granting insight into the details of this complex problem and overviewing the most recent and relevant work, with a special focus on LiDAR SLAM and state estimation for autonomous racing. This analysis allows for a better understanding of to what extent this work could innovate.

Most LiDAR odometry implementations are designed for slightly different challenges, providing a different type of map than the one necessary for this problem. Even so, it outlines the potential of using LiDAR information for motion estimation, as entire SLAM systems are built without the usage of any other sensor, often based on loosely coupled approaches, with no loop closure. This is a clear indication that the usage of LiDAR for motion estimation is a viable solution, as it is capable of providing accurate and reliable information for the task.

Machine Learning approaches, although quite interesting and popular for many related problems, are deemed too underdeveloped to be used in this context with confidence. The success of these approaches for loop closure is notorious. However, the challenge of loop closure in the Formula Student scenario is quite different, as the vehicle is very likely to visit almost the exact same position when completing a lap. This results in the fact that simpler and more efficient techniques can be used with the same or greater success.

Turning away from Formula Student, most other work in the context of autonomous racing that did not leverage GNSS intensely employed rather simple algorithms. This can be justified by the fact that the increased speeds of these scenarios vouch for the simplicity of the approaches, as it often results in a more efficient system. This simplicity is not seen in the Formula Student scenario mostly because, in contrast with these scenarios, tricks like offline map registration are not allowed, increasing the complexity of the problem and the accuracy requirements for the system. This challenge is precisely what motivates the development of systems that can be used for localization-only and SLAM modes. Teams often focus on the development of a single system aiming to not divide the development efforts into two separate algorithms. On top of this, despite greater velocities posing a challenge, track margins are often much more relaxed in comparison to this case.

When it comes to particular techniques, Pose-Graph Smoothing and Mapping (Graph SLAM) was shown to be a clear winner in the context of Formula Student, as it was featured in multiple works providing better results than filtering approaches. Moreover, the trend among the best-scoring teams is to switch to this type of solution.

One interesting fact is that almost all state estimation solutions analyzed performed velocity estimation separately from the SLAM module, most of them not using information from the perception pipeline or the LiDAR sensors whatsoever. This can mostly be explained by the typical robustness of these pipelines, given the increased amount of sensors that they fuse, and the need for short latencies.

While some work already presented a comparison between two approaches to the SLAM problem in Formula Student, some gaps are noticeable in the existing literature:

- Apart from [78], no serious study on the impact of the usage of LiDAR and perception data for velocity estimation has been carried out in the context of Formula Student;
- Although adoption is rising, there are no studies comparing different optimization backends for Graph SLAM in the context of Formula Student;
- Most works published are quite thorough on the evaluation of the State Estimation systems, but fail to fully categorize the environment in which they were tested by rarely mentioning the velocity at which the tests were carried out. This is a key piece of information, as it is a crucial factor in the performance of the system;

With this, a motivation to invest in certain specific points arises, on top of achieving the core objectives for the state estimation system. This thesis intends to fill these gaps by providing a

study on the impact of the mentioned techniques and carrying out a thorough and characterizing the results for different scenarios, focusing on the impact of the techniques for highly-dynamic agents.

# Chapter 4

# Practical Implementation

In this chapter, the proposed methodology to achieve the outlined objectives is defined. This methodology is naturally inspired by the Literature Review and outlines made in the previous chapter. Additionally, the vehicle setup is revised and the implementation carried out is presented in detail.

## 4.1 Proposed Approach

As mentioned in the introduction of this document, the objective of this dissertation is to study the best methodologies to perform vehicle state and map estimation in a Formula Student Driverless context. To that extent, a solution to the problem is to be developed, which solves the inaccuracy and inefficiency problems of the current state estimation system used in FS FEUP's autonomous pipeline. This implementation is to be modular and flexible, allowing for easy integration with the rest of the pipeline and with the ongoing changes in the system.

The main implementation proposed generally consists of:

- A loosely-coupled SLAM solution, with separate modules for velocity estimation and SLAM:
    1. Performing velocity estimation tightly-coupled with the SLAM system using a pose graph would potentially hinder the modularity of the implementation and result in less real-time-prone system;
    2. The usage of a separate module for velocity estimation has been proven to work in similar scenarios;
- Velocity Estimation performed through fusion of wheel encoder and IMU data in an EKF:
    1. The EKF has been widely proven to suffice in the context of mobile robotics state estimation in general and been shown to work for velocity estimation in the context of Formula Student Driverless, as shown through the works analyzed in Chapter 3;

2. The EKF, in comparison to other sensor fusion methods (Neural Networks, Non-Parametric Filters), is easier to implement and debug and requires less computational efforts;

- SLAM implemented in a smoothing approach using the GTSAM library [38]:

1. a smoothing approach has been proven more reliable and has had more success in Formula Student contexts;
2. GTSAM library usage entails a different approach from the most popular ones reviewed, which are developed using g2o. This additionally allows for the comparison of traditional smoothing optimizers with more modern and complex approaches, such as iSAM2.

**Note:** the claims made for justifying the proposed approach are based on the literature review performed in this dissertation.

This implementation should outperform the currently employed and fulfill the first objective of this project.

Regarding the second objective, the directives described in it will be followed, with an implementation with as few dependencies as possible and using ROS2 as its base. In the implementation section, specific methods used to achieve a modular and scalable implementation are described in more detail.

In light of what was mentioned in the last section of the previous chapter, a study of the impact of the inclusion of LiDAR odometry techniques in the state estimation system is of utmost interest, as this is an emerging technology mostly overlooked in the context of Formula Student until recently.

Moreover, multiple motion priors and vehicle models that fit the sensors and data available are to be implemented and their impact on the system's performance is to be analyzed.

## 4.2 System Integration

This section goes over the overall system integration, outlining the structure of the driverless system the implementation will be integrated into. The simulation environment and test vehicle are also described, as they are fundamental to the development and testing of the system.

### 4.2.1 Software Stack

As previously mentioned, the resulting work of this dissertation is meant to be included in the team's autonomous driving pipeline. The latter is composed of multiple modules, most of which interact directly with the State Estimation nodes, as shown in Figure 4.1.

The Path planning and Control modules simply define restrictions for the output of the systems developed in this project, these being the need for landmark-based map generation and estimation of two-dimensional pose and longitudinal velocity. The Path Planning module generates a path

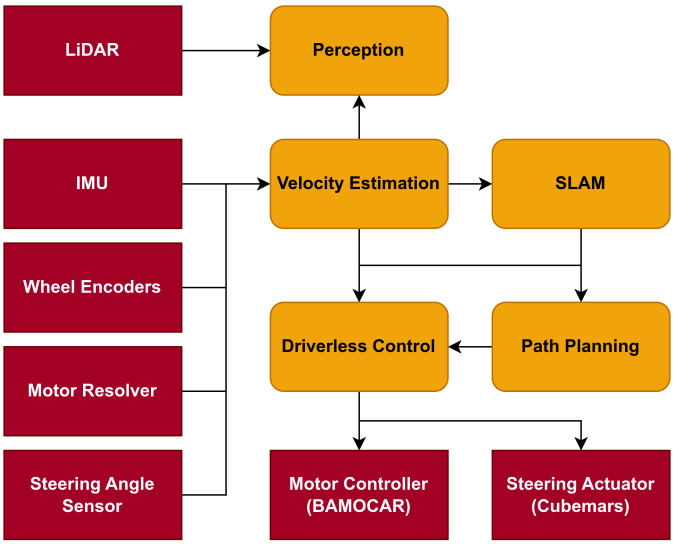



Figure 4.1: Autonomous Driving Pipeline Overview

that aims to follow the midline of the track leveraging Delaunay Triangulations and B-Splines for smoothing, focusing on reducing the odds of track boundary violation. Each point in the path is associated with a velocity based on path curvature, to avoid dealing with scenarios where the limits the car's grip are tested. The Control module performs outputs steering and throttle commands using two separate controllers: a Pure Pursuit controller and a PI-D Controller for lateral and longitudinal control, respectively. This is referred to as a high-level controller, as these inputs are further processed by hardware components with integrated control loops to translate them into actuation.

The Perception module however can affect directly the results obtained by the SLAM system, as the former would traditionally be considered part of the SLAM system's front-end and is fundamental to the generation of the map. This will be taken into account in the results analysis and is somewhat mitigated by the inclusion of the chosen simulator as a validation mechanism. Additionally, due to the heightened importance of this subsystem for the SLAM module, a basic description of the processing of the LiDAR point-clouds is performed:

1. **Pre-Processing:** The point-cloud is filtered using a low-pass filter, removing points that are too far away from the vehicle or out of a 120º FOV. This is done both with the decrease of LiDAR quality data with distance and to cater for the weaknesses of the perception pipeline itself. The point-cloud is further reduced by identifying and removing the points that correspond to the ground. This is done by dividing the point-cloud into a cylindrical grid and applying a RANSAC plane fitting algorithm to identify the ground points in parallel to each of the segments;

2. **Object Identification:** Due to the nature of the environment, the cones that limit the track are more or less evenly spaced. For this reason, the DBSCAN algorithm is quite successful

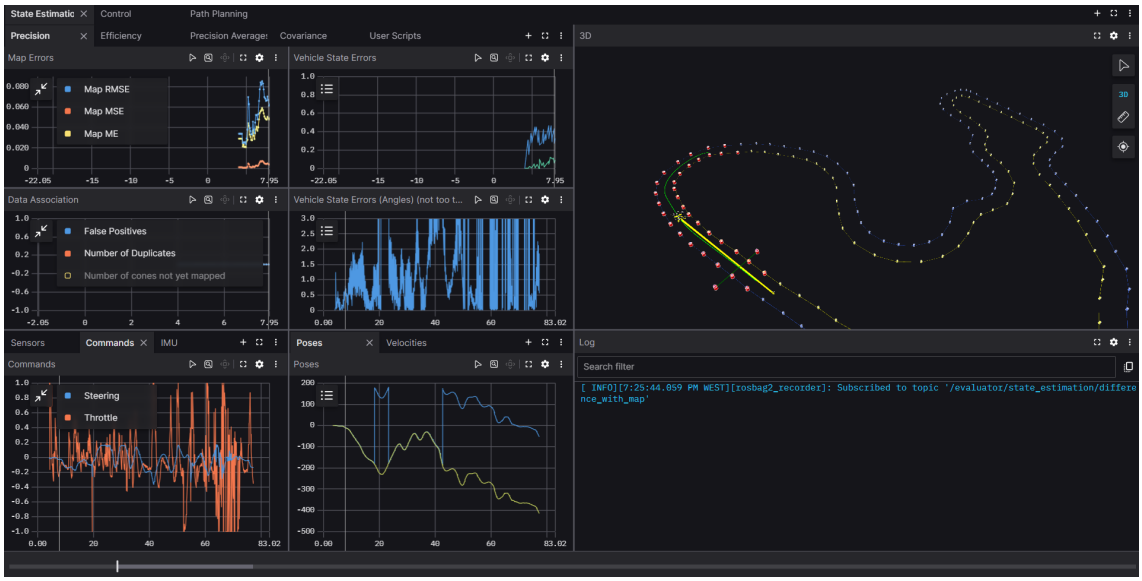



Figure 4.2: Foxglove Dashboard showcasing PacSim Simulation

at identifying the objects in the track, by simply generating clusters based on density. This leaves us with a set of potential cones;

3. **Object Classification / Validation:** The identified cones are then classified using a set of statistical and geometric heuristics, based on the characteristics of the cone clusters and their distribution in the track.

#### External Libraries and Frameworks

On top of depending on the implemented modules by the team, the state estimation system heavily uses the structures and algorithms in the GTSAM library [38] and data structures from the Eigen library [85]. GTSAM (Georgia Tech Smoothing and Mapping) is a C++ library for factor graphs and Bayes networks, which is used to implement the SLAM module. It provides a set of tools for factor graph construction, optimization and evaluation, including an iSAM2 [35] implementation. Eigen is a C++ template library for linear algebra, which is used to perform matrix operations and linear algebra computations in the state estimation system.

The system is fully implemented in C++ and uses ROS2 Humble [86] at its base. The functionalities presented henceforth are implemented as ROS2 packages that integrate two ROS2 nodes.

### 4.2.2 Simulation Environment

The simulation environments have already been explored and set up, being based on the PacSim simulator. PacSim is a autonomous driving simulator specifically developed to emulate Formula Student environments. It was initially developed by the Elbflorace team in C++, using ROS2, but has been adapted to the vehicle used by the FS FEUP team and its sensors. It models the vehicle's movement using a dynamic bicycle model that includes both lateral and longitudinal

weight transfer. Additionally, the simulator incorporates tire models based on the Pacejka Magic Formula. The choice of this simulator is based on the team's experience that such a tailored simulator results in a more trustworthy representation of the real scenarios, mainly due to the lack of tools that both model a Formula Student vehicle's dynamics accurately and provide modelling for the sensors used by the team. One important note is that the simulator does not model LiDAR point-clouds, but instead provides a cone observation noise model, acting as a mock of a perception module. This cone noise model mimics the behavior of the detection of cones' relative coordinates, with controllable noise, outliers and other parameters, allowing for the simulation of this part of the system with decent control. This further allows the development of the SLAM module without dependency to the team's Perception module throughout the year.

PacSim does not provide a GUI, with user interaction carried out through external tools with ROS2 interface. For our case, we will use Foxglove Studio, a visualization and data management tool that allows for the creation of dashboards that are designed for ROS2 compatibility. An example of the simulation system with the interaction of the two tools is showcased in Figure 4.2.

For the context of this dissertation, sensor modules will function at the same rate as in the vehicle, and the noise modeled after the values obtained in the real-world sensors.

### 4.2.3 FS FEUP 02 Prototype

The FS FEUP 02 prototype is the team's second prototype, the first ever fully autonomous. It features a RWD electric powertrain based around a single permanent magnet synchronous motor and a chain-drive transmission with a 4:1 gear ratio. The vehicle features multiple electronic control units, with the two main ones located in the front and rear of the vehicle, connected through a CAN bus. Most electronic components present in the vehicle are developed in-house, including the ECUs responsible for low-level sensor processing.

#### Computational Unit

Additional to the ECUs, the vehicle includes an onboard computer, which is responsible for DV functionality and logging, the Autonomous System Computational Unit (AS CU). The ASRock 4x4 Box-7840U features an AMD Ryzen 7 7840U processor, with 8 cores and 16 threads, and 16 GB of RAM. It runs Ubuntu 22.04 and ROS2 Humble, and hosts the all the autonomous driving modules of the pipeline, including the state estimation system.

#### Actuators

The vehicle's motor is controlled by a commercial controller and inverter, the BAMOCAR-PG-D3-700/400, which is connected to the AS CU through a CAN bus. It is capable of performing multiple forms of control, including torque control, which is the one used in the vehicle. The controller-inverter set-up is also capable of performing regenerative braking, which is the active form of braking used in the autonomous driving mode.


Figure 4.3: FS FEUP 02 Prototype

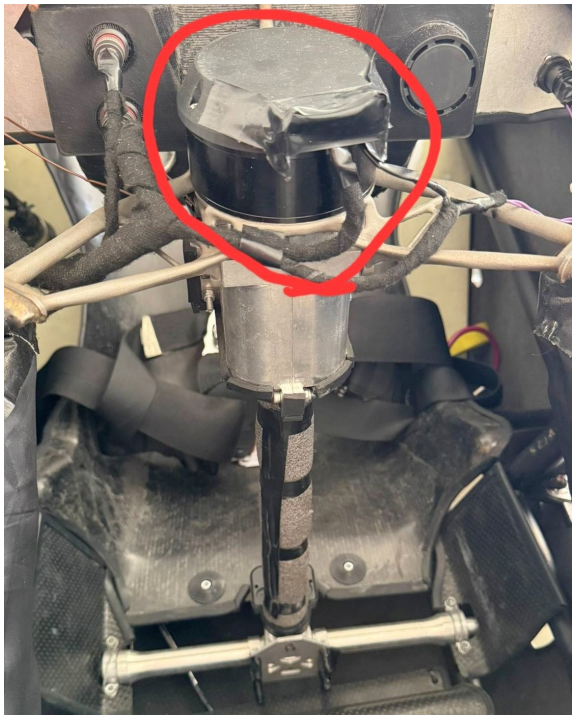


Figure 4.4: Steering Actuator Integration in FS FEUP 02

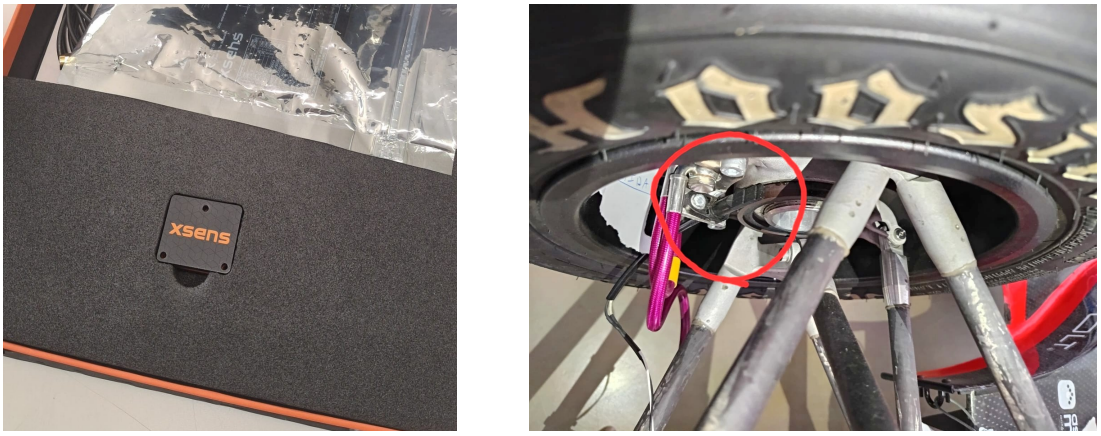
For steering actuation, the Cubemars AK70-10 is used, introduced directly into a gear-box system that connects to the steering column, as shown in Figure 4.4. The steering motor once again includes a control module that allows for direct steering angle control.

#### Sensor Suite

The vehicle is equipped with a Hesai Pandar40P, a three-dimensional rotating mechanical LiDAR. The LiDAR is positioned below the main hoop of the vehicle, oriented parallel to the ground. Its FOV is mainly limited by the two bars composing the main hoop, as the vehicle does not feature a rear wing. The LiDAR is directly connected the AS CU via Ethernet (100Mbps).

A Xsens AHRS unit is installed at the CoG of the car, the Mti-670 (shown in Figure 4.5a), providing high-frequency filtered IMU data. The unit is a 9-axis IMU, providing acceleration and angular velocity data at 100Hz, as well as orientation data at 50Hz. The unit is connected to the vehicle's main ECU through a USB connection.

This sensor-suite was to be complemented by a set of wheel encoders, one in each wheel, implemented using commercial Bosch ABS Sensors. Unfortunately, due to design and integration mistakes, the rear wheel encoders were not able to be integrated, leaving only the front wheels. Their integration can be seen in Figure 4.5b. The motor resolver is also used for estimating the velocity of the vehicle, as well as a steering angle sensor, included in the steering column. Both of these sensor modalities communicate with the AS CU through the CAN bus.




(a) Xsens MTi-670 AHRS Unit (b) Wheel Speed Sensor Integration

Figure 4.5: Sensors used in FS FEUP 02

## 4.3 Solution Overall Design

This section goes over the overall structure and behavior of the state estimation system, both describing its architecture and packaging and the execution of the programs themselves.

This project aimed to design and implement a state estimation system that not only was able to fulfill the performance requirements of the team, but was also capable of being easily expanded

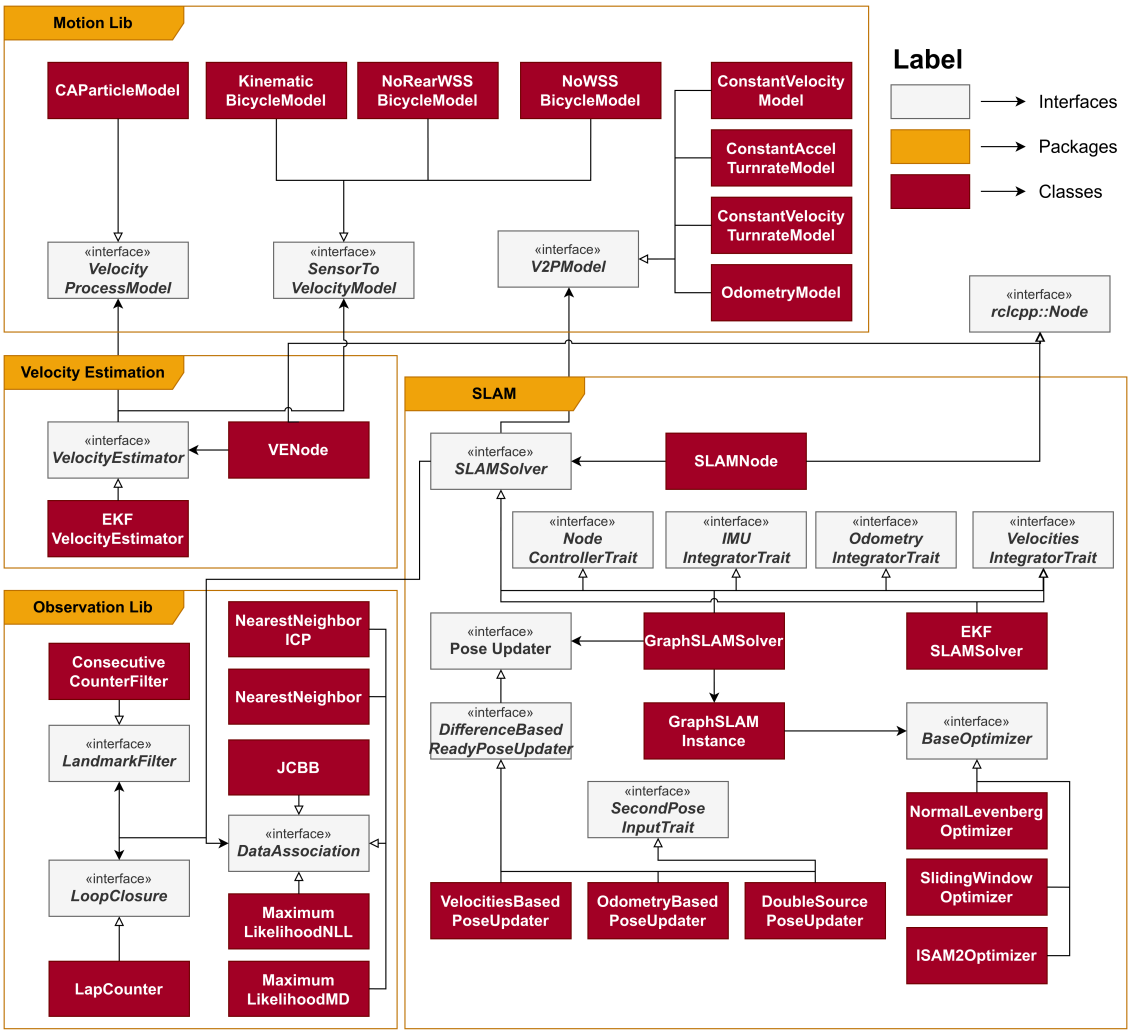



Figure 4.6: Overall Architecture of the State Estimation System

and tampered with. To that extent, modularity was taken very seriously when designing the code architecture. The overall architecture of the software developed is showcased in Figure 4.6.

The system can be divided initially into two main modules, as previously mentioned: the Velocity Estimation module and the SLAM module. Each of these modules is implemented as a separate ROS2 package, with its own nodes, executing completely in parallel.

For experimental purposes, 3 different implementation of LiDAR (or LiDAR-Inertial) odometry were also set-up, to evaluate their impact in the SLAM module's performance:

- Fast-LIMO (LIMO-VELO) [78]: a multi-threaded version of the original LIMO-VELO with ROS2 integration - chosen for its proven track in Formula Student;
- KISS-ICP [55] - chosen for its simplicity, ease of setup and overall position in the field;
- GenZ-ICP [58] - chosen for its generalizability and robustness while using solely LiDAR.

The SLAM Package includes two solvers (for now): EKF and Graph SLAM. The SLAM-Solver class provides a standard interface for any SLAM implementation, extendable by trait classes, which add extra data processing functionalities to the base class.

Motion and Observation Models, as well as Data Association methods, are extracted to independent classes, all inheriting from common interfaces, increasing the ease of extending functionality with new models and methods and respecting the Dependency 'Inversion Principle'. 'Trait' classes are also created to extend the functionality of multiple objects without forcing other child classes to implement unused methods, following the 'Interface Segregation' principle.

This specific SLAM implementation also makes use of a simple Loop Closure module and a Perception Filter, both implementations carried out by the team and designed to integrate the project included in this dissertation seamlessly. Additionally, the implementation of the Graph SLAM solver included the addition of a Pose Updater class, which aims to provide a pose estimate in very reduced times. The same design paradigm described in the previous paragraph is applied to optimization methods, allowing for the easy addition of new optimization techniques without modifying the existing codebase.

The overall behavior of the state estimation pipeline is depicted in Figure 4.7. Both nodes execute asynchronously, through callbacks defined for their respective input topics.

- **Velocity Estimation** receives Accelerations and Angular Velocities from the IMU, RPMs from the Wheel Encoders and Motor Resolver at 200Hz and Steering Angle at 100Hz and publishes linear and angular velocities in a 2D plane at nearly 200Hz;
- **SLAM** receives cones' relatives coordinates from the LiDAR based Perception module at 10Hz. Depending on the configuration, can use odometry information from a LiDAR odometry node, which publish at different rates, from 10Hz to 100Hz, or information from the Velocity Estimation node (or both). The SLAM node publishes pose and map every time there is an update to it, which equates to every time one of the three callbacks is executed. The SLAM node also implements a parallel execution optimization feature, controlled by launch parameters, which switches the optimization procedure from a synchronous optimization after the inclusion of new factors, to an asynchronous one on a fixed time interval.

## 4.4 Velocity Estimation

As noted in the beginning of this chapter, an EKF was implemented to perform velocity estimation. The Extended Kalman Filter has been already reviewed. Therefore, this section will focus on the implementation details and the specificities of the EKF used in this project.

### 4.4.1 State Vector

In contrast to the implementations presented by other authors, the state of this EKF includes only the objective estimation variables and no others, resulting in a state vector of the form:

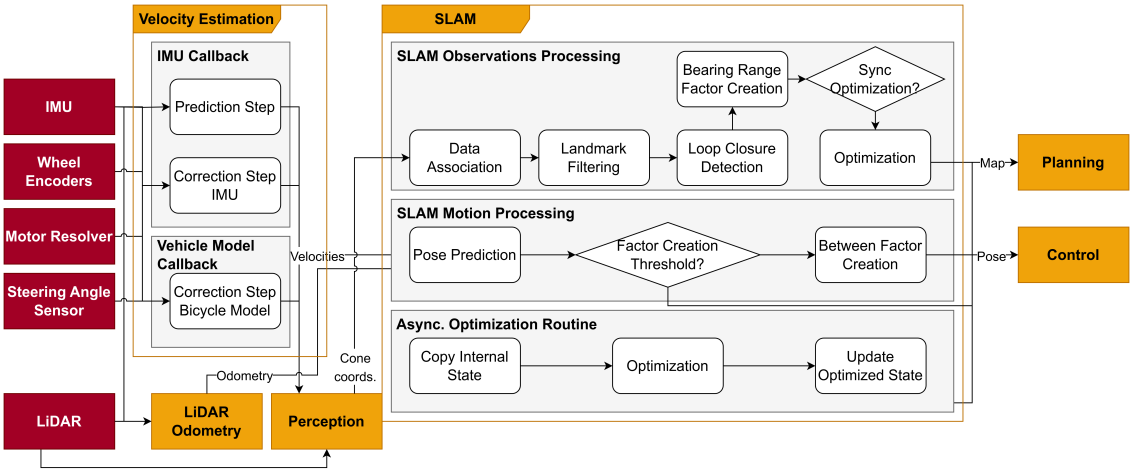



Figure 4.7: Overall Process Flow of the State Estimation System

$$\begin{bmatrix} v_x \\ v_y \\ \omega_z \end{bmatrix} \tag{4.1}$$

where $v_x$ and $v_y$ are the longitudinal and lateral velocities of the vehicle, respectively, and $\omega_z$ is the yaw rate of the vehicle, in the vehicle coordinate system. Acceleration data is not included in the state vector because it is only observable by the AHRS unit, which already includes filtering of these variables.

As depicted in Figure 4.7, the EKF node includes two callbacks, one for the IMU data and another for the rest of the sensors. The IMU callback performs the prediction step of the EKF and a correction step exclusively for the angular velocity $\omega_z$, while the other performs a correction based on a Kinematic Bicycle Model. The Kalman Gain calculation and the rest of the EKF algorithms are implemented as described in Chapter 2.

### 4.4.2 State Transition Model

The state transition model serves as a prior distribution over the state space. In this case, it uses the accelerations obtained from the IMU to predict the next state from the previous one, rather than using solely the previous state, assuming constant acceleration and turn-rate.

$$\begin{bmatrix} v_{x,t} \\ v_{y,t} \\ \omega_{z,t} \end{bmatrix} = \begin{bmatrix} v_{x,t-1} \\ v_{y,t-1} \\ \omega_{z,t-1} \end{bmatrix} + \begin{bmatrix} (a_x + v_y \cdot \omega_z) \cdot \Delta t \\ (a_y - v_x \cdot \omega_z) \cdot \Delta t \\ 0 \end{bmatrix} \tag{4.2}$$

The formula includes both linear acceleration and the coriolis term, excluding centripetal acceleration as the IMU is located at the CoG of the vehicle.

As the state transition model (or process model) is non-linear, the Jacobian matrix of the state transition model with respect to the state vector $F$ is used to linearize the model around the

current state estimate [15]. The covariance matrix of the state vector $\Sigma$ is updated according to the following equation:

$$\hat{\Sigma_t} = F \cdot \Sigma_{t-1} \cdot F^T + Q, \quad \text{with } Q = G \cdot R \cdot G^T \tag{4.3}$$

where $G$ is the Jacobian matrix of the process noise with respect to the accelerations, and $R$ is the noise covariance matrix of the sensors.

### 4.4.3 Measurement Model

The measurement model relates the state vector to the measurements obtained from the sensors, being used to perform a correction on the state estimate through the filter. In this case, the measurements are obtained from the wheel encoders, motor resolver, steering angle sensors and IMU, resulting in two different correction steps.

Angular velocity is directly measured by the IMU, resulting in a direct observation model. The second callback utilizes the data from the rest of the sensors. The entire state is observable through the combination of the encoders, resolver and steering sensor measurements utilizing a **Kinematic Bicycle Model** [87], illustrated in Figure 4.8. This model was chosen instead of a more complex one due to its proven effectiveness in comparison to four wheel models, and rather simple formulas, resulting in a more efficient computation, which is crucial for this application and was a proven problem in the previous solution the team used. The model itself is defined as follows:

$$v_{\text{rear}} = \sqrt{v_x^2 + (v_y - \omega \ell_r)^2}, \tag{4.4}$$

$$v_{\text{front}} = \sqrt{v_x^2 + (v_y + \omega \ell_f)^2}, \tag{4.5}$$

$$\text{rpm}_{\text{front}} = \frac{60}{\pi D} v_{\text{front}}, \tag{4.6}$$

$$\delta = \arctan\left(\frac{\omega L}{v}\right), \quad \text{where } v = \sqrt{v_x^2 + v_y^2}, \tag{4.7}$$

$$\text{rpm}_{\text{motor}} = \frac{60}{\pi D} gr\, v_{\text{rear}}. \tag{4.8}$$

where $rpm_{\text{front}}$ and $rpm_{\text{motor}}$ are the front wheel encoders and motor resolver measurements, respectively, $\delta$ is the steering angle measurement, $L$ is the wheelbase of the vehicle, $\ell_f$ and $\ell_r$ are the distances from the CoG to the front and rear axles, respectively, $D$ is the diameter of the wheels and $gr$ is the gear ratio of the transmission.

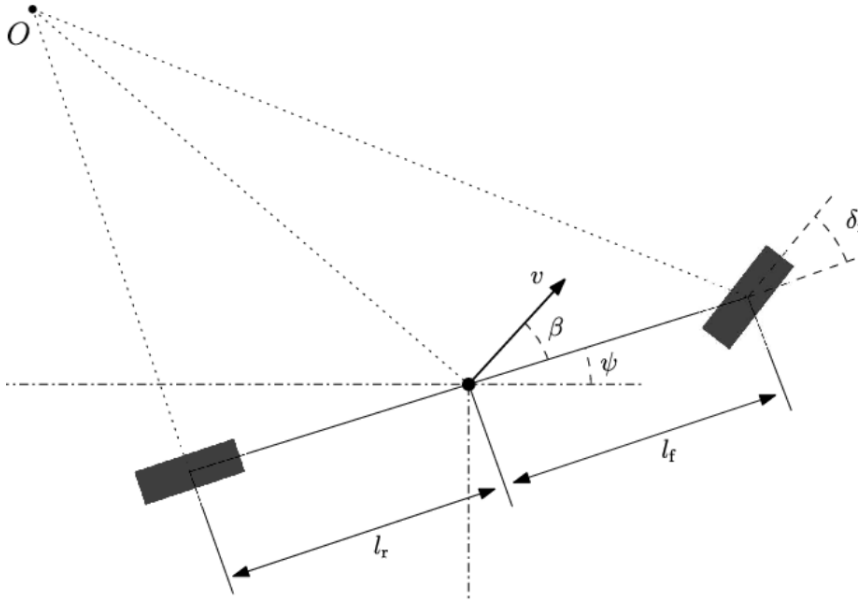



Figure 4.8: Kinematic Bicycle Model

## 4.5 SLAM Module Implementation

This section goes over the implementation of the Graph SLAM Solver and the surrounding modules that support the solver itself in the localization and mapping task.

### 4.5.1 Data Association, Loop Closure and Perception Filtering

**Data Association** is a crucial part of any SLAM system. Due to the parallel work inside the team, a multitude of data association algorithms had been implemented by the team member responsible for State Estimation in FS FEUP, including the Joint Compatibility Branch and Bound (JCBB) algorithm, and variants of the Maximum Likelihood using different distance formulas. These implementations were already validated internally and as such were used for this system as well. In this case, the Nearest Neighbor (Maximum Likelihood with euclidean distance) was chosen, given its superior performance in comparison to the others, which required more complex matching (JCBB) or calculation of covariance matrices and multiplications involving them.

The same applies for the Loop Closure and Perception Filtering techniques used in the SLAM system: they were implemented internally and were integrated into the system developed in the context of this dissertation during its development.

The **Loop Closure** consists of a simple matching algorithm that takes into account the distance of the current pose to the origin and uses the output from data association to validate that the cones in sight are the same as the ones observed when the vehicle was in the starting line. As we use landmark-based SLAM, the loop closure module does not have the same fundamental effect in the Graph's correction, as the association to the same landmarks will already perform this correction by itself. This module was implemented by the team mainly with the goal of counting

the laps performed by the vehicle, something necessary to correctly perform the dynamic events in competition.

The **Perception Filter** was implemented to aid filtering outliers from the Perception module. It is a rather simplistic implementation relying on repetitive consecutive observation of the same cones for validation. A statistical approach could have been followed, but this simple heuristic combats the sporadic erroneous observations quite well. This approach was designed following the observation that, during testing, there were very few outliers that remained as such for more than one consecutive observation from Perception.

### 4.5.2 Motion Model

One of the objectives of this project is to evaluate different solutions in a dynamically challenging Autonomous Driving scenario. With this objective in mind, three different Motion Models (second part, as per the nomenclature used in Chapter 2) were implemented, as documented in [88] and [89].

#### Constant Velocity Model

The Constant Velocity (CV) model is a simple kinematic motion model that assumes a vehicle moves with a constant linear velocity and heading between two consecutive time steps. The state transition can be described by the following equations:

$$\begin{bmatrix} x_{t+\Delta t} \\ y_{t+\Delta t} \\ \theta_{t+\Delta t} \end{bmatrix} = \begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} + \begin{bmatrix} v_x \cos(\theta_t) - v_y \sin(\theta_t) \\ v_x \sin(\theta_t) + v_y \cos(\theta_t) \\ \omega \end{bmatrix} \cdot \Delta t \tag{4.9}$$

where:

- $x, y$ are the Cartesian coordinates of the vehicle,
- $\theta$ is the vehicle's orientation (heading angle),
- $v$ is the linear velocity, assumed constant during the motion interval,
- $\omega$ is the angular velocity (yaw rate).

This model is simple and computationally efficient, making it suitable for real-time applications.

#### Constant Velocity and Turn-rate Model

The Constant Velocity and Turn-Rate (CVTR) model describes the motion of a ground vehicle assuming a constant linear velocity and a constant angular velocity (turn rate) over a short time interval. The vehicle's state evolution is governed by the following equations:

$$\begin{bmatrix} x_{t+\Delta t} \\ y_{t+\Delta t} \\ \theta_{t+\Delta t} \end{bmatrix} = \begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} + \begin{bmatrix} \frac{v}{\omega}\left[\sin(\theta_t + \omega\Delta t) - \sin(\theta_t)\right] \\ \frac{v}{\omega}\left[-\cos(\theta_t + \omega\Delta t) + \cos(\theta_t)\right] \\ \omega\Delta t \end{bmatrix} \tag{4.10}$$

where:

- $x, y$ are the Cartesian coordinates of the vehicle,
- $\theta$ is the vehicle's orientation (heading),
- $v$ is the linear velocity,
- $\omega$ is the angular velocity (yaw rate).

In the special case where $\omega \approx 0$ (i.e., straight-line motion), the model simplifies to:

$$\begin{bmatrix} x_{t+\Delta t} \\ y_{t+\Delta t} \\ \theta_{t+\Delta t} \end{bmatrix} = \begin{bmatrix} x_t + v\Delta t\cos(\theta_t) \\ y_t + v\Delta t\sin(\theta_t) \\ \theta_t \end{bmatrix} \tag{4.11}$$

**Constant Acceleration and Turn-rate Model**

The Constant Acceleration and Turn-Rate (CATR) model extends the Constant Velocity and Turn-Rate (CVTR) model by incorporating linear acceleration into the vehicle's motion over a short time interval $\Delta t$. This allows for more accurate modeling of real-world vehicle dynamics, especially in scenarios involving speed changes during turning maneuvers. The vehicle's position and orientation are updated using:

$$\begin{bmatrix} x_{t+\Delta t} \\ y_{t+\Delta t} \\ \theta_{t+\Delta t} \end{bmatrix} = \begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} + \begin{bmatrix} \frac{1}{\omega^2}\left[(v_t\omega + a\omega\Delta t)\sin(\theta_t + \omega\Delta t) + a\cos(\theta_t + \omega\Delta t) - v_t\omega\sin(\theta_t) - a\cos(\theta_t)\right] \\ \frac{1}{\omega^2}\left[-(v_t\omega + a\omega\Delta t)\cos(\theta_t + \omega\Delta t) + a\sin(\theta_t + \omega\Delta t) + v_t\omega\cos(\theta_t) - a\sin(\theta_t)\right] \\ \omega\Delta t \end{bmatrix} \tag{4.12}$$

where:

- $x, y$: global position coordinates,
- $\theta$: vehicle heading (yaw angle),
- $v$: linear velocity,
- $a$: linear acceleration,
- $\omega$: angular velocity (yaw rate).

In the special case where $\omega \to 0$, i.e., straight-line motion with constant acceleration, the equations simplify to:

$$\begin{bmatrix} x_{t+\Delta t} \\ y_{t+\Delta t} \\ \theta_{t+\Delta t} \end{bmatrix} = \begin{bmatrix} x_t + \left(v_t \Delta t + \frac{1}{2} a \Delta t^2\right) \cos(\theta_t) \\ y_t + \left(v_t \Delta t + \frac{1}{2} a \Delta t^2\right) \sin(\theta_t) \\ \theta_t \end{bmatrix} \tag{4.13}$$

This model assumes that the vehicle moves along a circular arc with increasing or decreasing speed, making it suitable for systems requiring high-fidelity motion prediction such as autonomous navigation or high-speed tracking.

### 4.5.3 Graph SLAM

The Graph SLAM solution implemented utilizes the GTSAM library for the factor-graph construction and optimization. The structure of the graph follows a traditional approach:

- Variable nodes correspond to either vehicle poses or landmarks;
- Factor nodes correspond to either measures from the perception pipeline or pose differences inferred from the velocity estimation module (or odometry modules).

The differences begin in the graph construction method: while perception observations are always directly processed and included in the graph, inputs that are used for the creation of new pose nodes are initially processed by an auxiliary structure, the 'PoseUpdater'. This auxiliary structure receives the velocity estimates and applies the Motion Model (second part of the motion model) to the latest pose stored.

On top of that, there are two other crucial features to outline:

- As foreshadowed in Section 4.3, the system implements three different modes of optimization: batch optimization using Levenberg-Marquardt, incremental optimization using iSAM2 and Sliding-Window optimization using Levenberg-Marquardt. The approach of optimizing only upon loop closure detection was deemed unworthy of maintaining due to the need for a very solid velocity estimation to be remotely usable;
- The system was implemented to operate with synchronous or asynchronous optimization, with the former being triggered only upon reception of perception observations.

Both of these features cater for this thesis' goal of exploring different solutions for parts of a SLAM system and evaluate their performance in real-time scenarios. The overall architecture of this part of the implementation is depicted in Figure 4.9.

#### Pose Updater - Motion Integration

The Pose Updater's task in the system is simply to apply the Motion Model and keep an up-to-date pose estimate. This separation allows for the control of which inputs result in the addition of new factors to the graph.

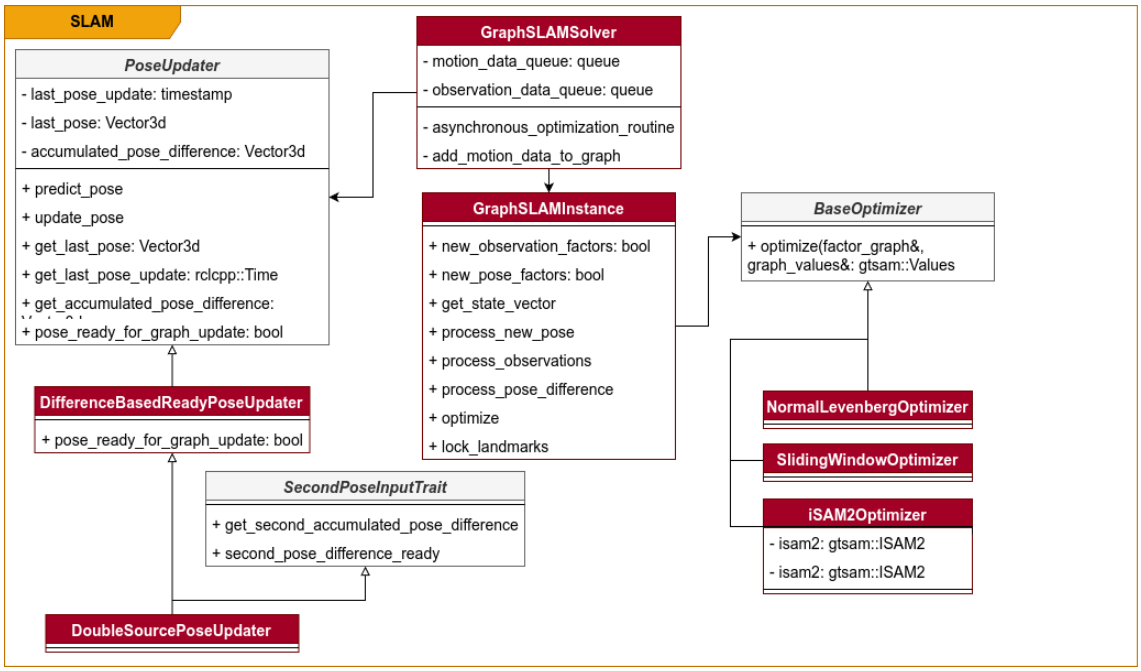



Figure 4.9: Graph SLAM Architecture Overview

As the optimization process grows in complexity with the number of factors, this last feature caters towards the real-time capability of the system due to resulting in less complex factor graphs. This auxiliary structure also allows for the implementation of different methods to decide how to merge different sources of motion data and decide how they are included in the Factor Graph. It also allows for the usage of different Motion Models, as previously described, allowing it to be used with both Odometry and Velocities data.

The one criteria implemented for the addition of new poses to the graph is a minimum pose difference norm, calculated as:

$$\|\Delta T\| = \sqrt{\Delta x^2 + \Delta y^2 + (\Delta \theta)^2} \tag{4.14}$$

where the pose difference $\Delta T$ is calculated between the last pose added to the graph and the current pose estimate in the frame of the latter.

One challenge that arises from this approach is that the noise modeling of the factors to be included differs significantly: in a normal implementation, each odometry or velocity input results in a new pose and factor in the graph, meaning the noise vector for that factor can be directly derived from the motion data source's noise, provided it is mapped through the Jacobian of the Motion Model in relation to that motion data, as such:

$$\Sigma_{\text{factor}} = J_{\text{motion}} \cdot \Sigma_{\text{motion}} \cdot J_{\text{motion}}^T \tag{4.15}$$

where $J_{\text{motion}}$ is the Jacobian of the Motion Model with respect to the motion data (odometry or velocity), $\Sigma_{\text{motion}}$ is the covariance matrix of the motion data (diagonal matrix with the variance of

each of the input variables) and $\Sigma_{\text{factor}}$ is the covariance matrix of the factor to be included in the graph.

However, since we are accumulating multiple inputs before adding a new factor to the graph, this direct mapping is not possible: the noise of the factor becomes the covariance of the accumulated pose difference, which belongs to the SE(2) space, meaning it cannot be calculated by simple addition of the noises each time the Motion Model is applied. The covariance calculated at t-1 is the covariance of the pose difference between t-2 and t-1. When we receive the data at time t, the noise matrix $Q$ is now referring to the pose difference between t-1 and t. The new covariance matrix $\Sigma_t$ must take into account the uncertainty of the previous pose difference as well as the uncertainty of the new pose difference. For this, the previous covariance matrix must be transformed to the new frame of reference, which is done through the adjoint representation of SE(2), $Ad_{\delta T_t}$ [90].

The calculation of the Covariance representing the entire pose transformation each time new velocity or odometry data is received is carried out the following manner:

$$\Sigma_{factor} = \Sigma_t = Ad_{\delta T_t} \cdot \Sigma_{t-1} \cdot Ad_{\delta T_t}^T + J_{\text{motion}} \cdot \Sigma_{\text{motion}} \cdot J_{\text{motion}}^T \tag{4.16}$$

$$\text{with} \quad Ad_{\delta T_t} = \begin{bmatrix} R(\Delta\theta)_t & J\rho_t \\ 0 & 1 \end{bmatrix}, \quad J = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}, \tag{4.17}$$

$$\delta T_t = T_{t-1}^{-1} \cdot T_t = \begin{bmatrix} R(\Delta\theta)_t & \rho_t \\ 0 & 1 \end{bmatrix}, \quad \rho = \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}, \quad R(\Delta\theta) = \begin{bmatrix} \cos(\Delta\theta) & -\sin(\Delta\theta) \\ \sin(\Delta\theta) & \cos(\Delta\theta) \end{bmatrix} \tag{4.18}$$

where $\delta T_t$ is the pose difference between the last two poses, $T_{t-1}$ and $T_t$, in SE(2) representation; $\rho_T$ is the translation vector and $R_T$ is the rotation matrix for the current transformation.

### Pose Graph Optimization

As previously mentioned, one of the goals of this dissertation is to compare the performance of different optimization approaches under the classical SLAM representation for Formula Student. With this in mind, the implementation abstracts the optimization process into an interface, which is then implemented in three different ways: an iSAM2 optimizer, a Levenberg-Marquardt batch optimizer and a Sliding-Window Levenberg-Marquardt optimizer:

- The **Levenberg-Marquardt optimizer** is one of the most used approaches for pose-graph smoothing. This implementation uses the GTSAM library's implementation directly and with no tweaks. Batch optimization means that the entire factor graph is optimized at once, which leads to very accurate results, albeit with a high computational cost. This is the most traditional approach to SLAM optimization and is widely used in the literature;

- The **iSAM2 optimizer** differs from the previous one in the sense that it is not a batch optimizer, as previously explained. iSAM2's incremental approach leads to a more real-time

friendly approach. The nature of the algorithm also allows control of the number of iterations (typically one iteration suffices, but more can be carried out). Once again, the GTSAM implementation of this algorithm is used;

- The **Sliding-Window Levenberg-Marquardt optimizer** is a technique widely applied in the context of Formula Student. The application of a Sliding-Window to a batch optimizer allows the execution time to stunt at a certain value, providing great control over the algorithm performance and allowing for real-time constraints to be met.

Once again, the system design allows for the easy addition of new optimization methods, as long as they follow the interface defined. This is particularly useful for future work, as it allows for the integration of more complex optimization methods without the need to change the rest of the system.

As previously mentioned, an asynchronous optimization routine can be used instead of performing optimization on a given input callback. This allows the continuation of processing of data by the SLAM program and subsequent update of the internal state while the optimization is running, which is crucial given the fact that the optimization is by far the slowest process of the entire pipeline and, depending on the configuration, can easily cause the pipeline to miss several velocity inputs. The method implemented to perform this asynchronous process is depicted in Figure 4.7 and is summed into a three step process:

1. Perform a copy of the 'PoseUpdater' and 'GraphSLAMInstance' objects (locked by a mutex);
2. Run the optimization routine in parallel;
3. Reprocess the missed inputs, if any (locked by a mutex):
    - Velocity inputs are stored in a queue upon arrival;
    - Observation inputs are stored in a queue after processing data association, filtering and loop closure.

Even though this process can still lead to problems for very slow optimization processes (ill-performed data association for instance), it mostly allows for the usage of much heavier optimization methods.

# Chapter 5

# Results Analysis

In this chapter, the results obtained from each of the different implemented setups will be analyzed and compared. The results are divided into two main sections: simulation analyses and on-ground results. Each section will present the most relevant findings from the experiments conducted and will discuss the possible implications of these results.

## 5.1 Simulation analyses and Results

As previously stated, simulation was largely used to test and validate the different implementations carried out. Unfortunately, due to delays and other issues that compromised the on-ground evaluation, simulation was also used to analyze performance and obtain results.

The simulation environment modeled several tracks used in the Formula Student Driverless competitions, including the 2023 and 2024 FSG Autocross tracks, which are the ones the following analyses mostly stand on. Figure 5.1 showcases the FSG 2024 Autocross track mapped in the simulation environment. Each of the results obtained are an average of 5 runs to account for the inherent variability of the simulation and the reliability of the solutions. The results were obtained running the pipeline in the same computer used for on-ground testing on the FS FEUP 02, mentioned in Section 4.2.

### 5.1.1 Optimization Methods Comparison

The three different optimization strategies implemented were evaluated for their accuracy in the pose and map estimation, as well as their computational performance. Both the accuracy in pose and map estimation were inferior for the sliding-window approach in comparison to the other two methods, as can be seen in Table 5.1 and Figure 5.2. Even so, all three methods were able to perform with high enough accuracy for the other modules to comfortably operate, with no failure in simulations from any part of the pipeline due to poor pose or map estimation. These results are also natural: the Sliding-Window approach will only optimize over a certain horizon, which can lead to drift accumulation over time. The iSAM2 and Normal Levenberg-Marquardt approaches,

Figure 5.1: FSG 2024 Autocross track mapped in the simulation environment.

on the other hand, optimize over the entire trajectory, which helps in reducing drift and improving overall accuracy.

Conversely, when it comes to computational performance, the Sliding-Window Levenberg-Marquardt optimization was the most efficient, as shown in Figure 5.3. Once again, this is expected: the consequences of the windowed smoothing clearly lead to a stagnation of the growth of the execution time, as intended by the strategy.

Both iSAM2 and Normal Levenberg-Marquardt showed approximately linear growth in execution time, with the former having a more unpredictable growth pattern. The iSAM2 performed quite similarly to the Normal Levenberg-Marquardt, which is not usually to be expected. The CPU load follows a similar pattern, as shown in Figure 5.4, with the Sliding-Window approach being the least demanding and the batch optimization being the most demanding.

Parallel optimization was also tested for multiple solvers. Contrary to what might be expected, the results were worse for every scenario. The additional operations necessary overloaded the CPU further and led to overall worse performance.

| **Optimization Solver** | **RMSE $x$ (m)** | **RMSE $y$ (m)** | **RMSE $\theta$ (rad)** | Track |
|---|---|---|---|---|
| iSAM2 | 0.055187 | 0.051504 | 0.003819 | FSG 2024 |
| Normal Levenberg-Marquardt | 0.044054 | 0.035629 | 0.002321 | FSG 2024 |
| Sliding-Window LM | 0.089448 | 0.136029 | 0.003227 | FSG 2024 |
| iSAM2 | 0.051436 | 0.049285 | 0.003621 | FSG 2023 |
| Normal Levenberg-Marquardt | 0.042152 | 0.029710 | 0.002206 | FSG 2023 |
| Sliding-Window LM | 0.073291 | 0.131081 | 0.002913 | FSG 2023 |

Table 5.1: Pose accuracy results for pose estimation in FSG 2023 and 2024 Autocross in simulation environment.

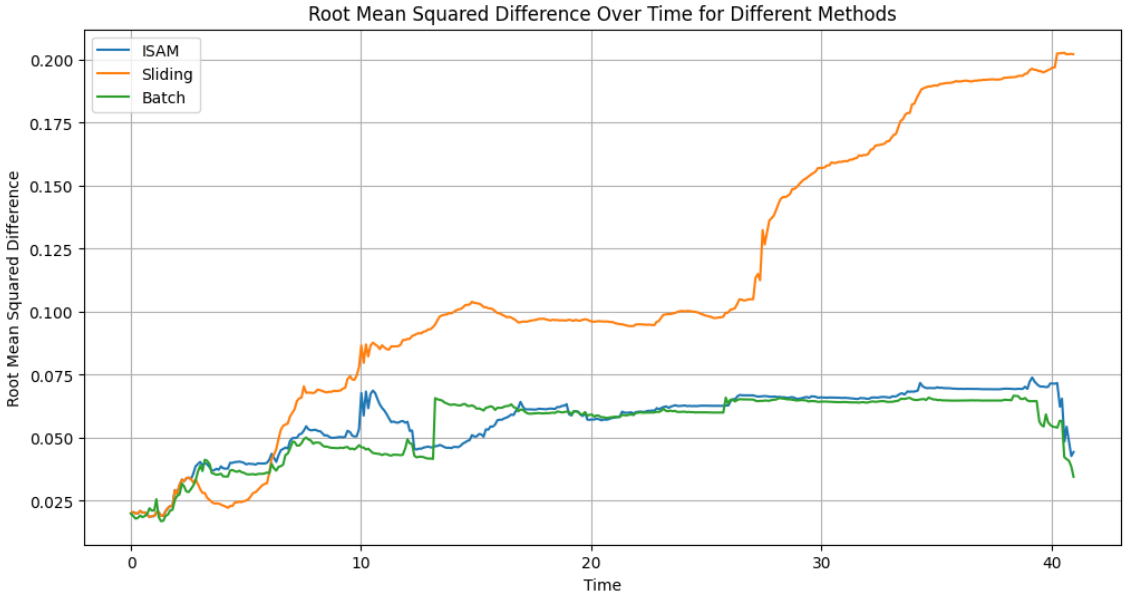



Figure 5.2: Mapping accuracy results for pose estimation in FSG 2024 Autocross in simulation environment for each optimization solver.

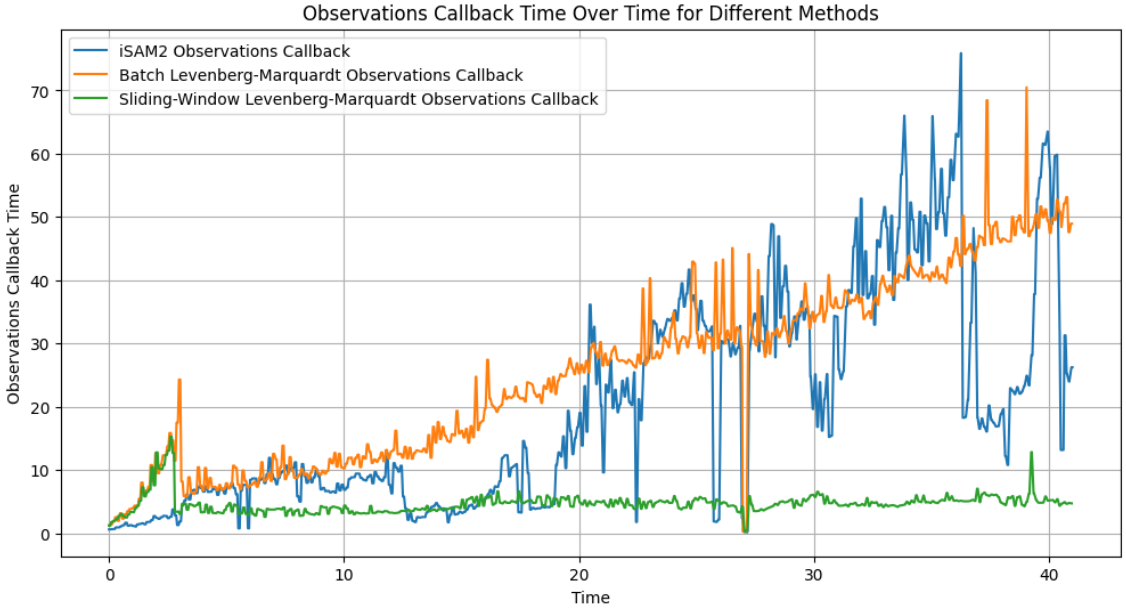



Figure 5.3: Execution time results for observation processing (mapping) in FSG 2024 Autocross in simulation environment for each optimization solver.

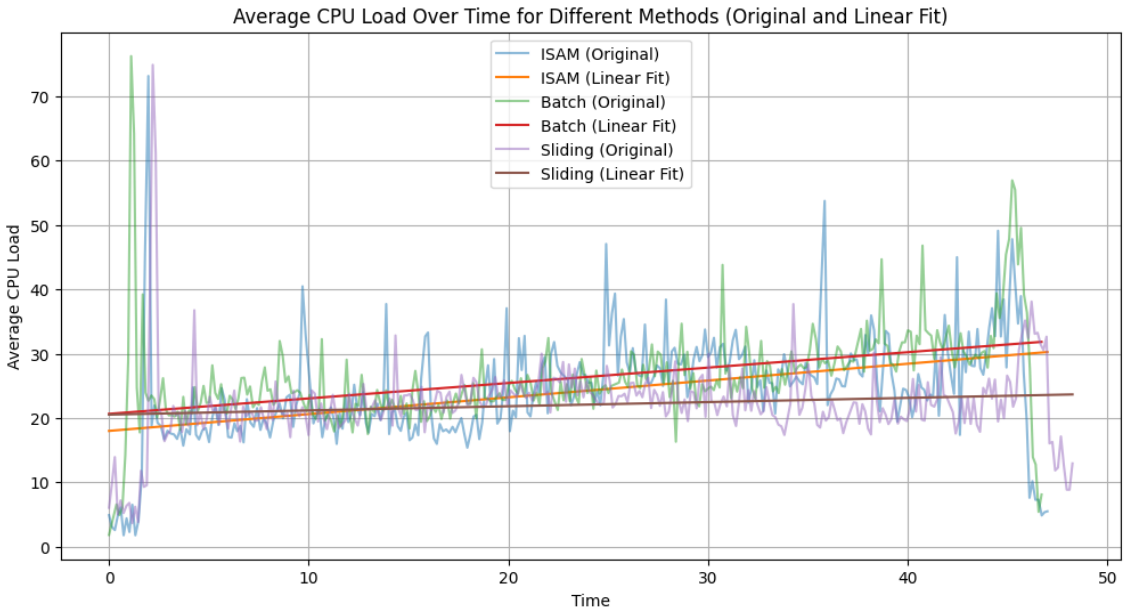



Figure 5.4: CPU load results for observation processing (mapping) in FSG 2024 Autocross in simulation environment for each optimization solver.

### 5.1.2 Motion Models Comparison

The three motion models presented were evaluated by turning off optimization and evaluating the accuracy of the pose estimation when only relying on the motion model and the velocity estimates from the EKF. As shown in Figure 5.5, all three models performed quite similarly for position accuracy (orientation accuracy can be seen in Figure C.1).

## 5.2 On-Ground Results

On-ground testing and evaluation was quite shorter and with lesser quality than initially intended. This was due to two main issues:

- The delays in the implementation and validation of the autonomous systems' hardware in the vehicle was only concluded much closer to the competition date, leaving only a few weeks to perform testing of the driverless systems already in between competition events;
- The hardware necessary to be able to infer accuracy of the pose or map estimation was not available in time for the tests.

This section will nonetheless showcase the results obtained by the system both during testing and competition runs.

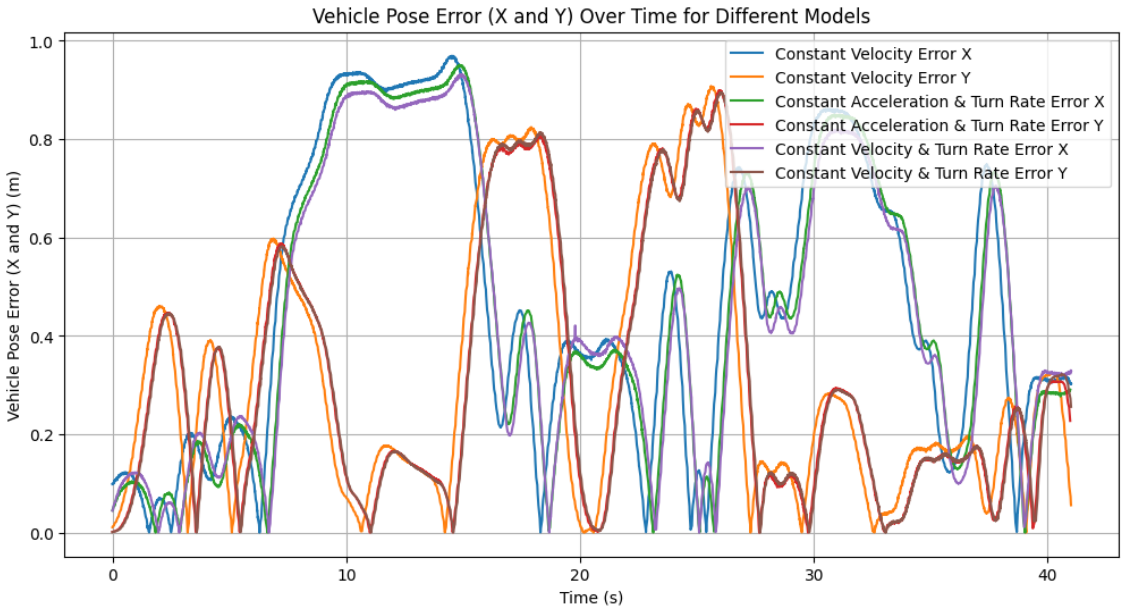



Figure 5.5: Position accuracy results for pose estimation in FSG 2024 Autocross in simulation environment for each motion model.

### 5.2.1 Comparison of Different Setups

The overall system was initially tested in-house, on a testing site with custom-built tracks. The combinations mentioned earlier were also tested on-ground in order to compare results and arrive at the best system. Contrary to what was observed in simulation, the Sliding-Window Levenberg-Marquardt optimization was the best performing optimization strategy. This comes down to computational performance: the iSAM2 and Normal Levenberg-Marquardt optimizations incur in a higher latency and consequently missing of valuable measurements. These two facts produced a degradation of the performance of the overall system, which was not observed in simulation due to the more idealized conditions.

From the motion models tested, the Constant Velocity and Turn-rate performed the best consistently. This can be explained by the fact that the IMU installed in the vehicle was providing less accurate acceleration data than expected, eventually with non fixed biases. This led to the acceleration-based motion model performing poorly, as it was relying on less accurate data. The Constant Velocity model also performed less accurately as it relied more on lateral velocity estimate than angular velocity, which was less accurate for the same reasons.

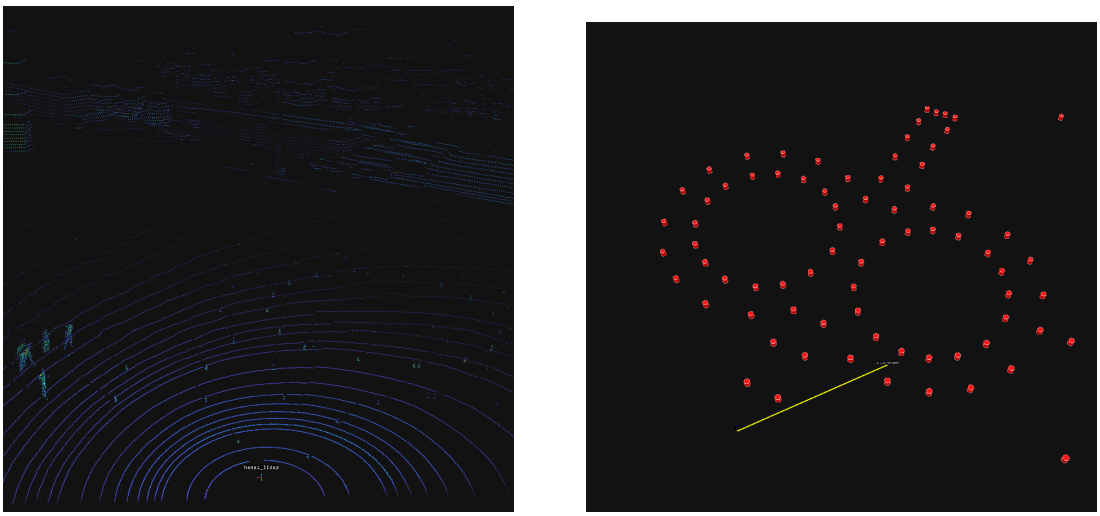
### 5.2.2 LiDAR Odometry Results

The LiDAR odometry methods were tested on-ground in two scenarios:

- Acceleration track: a straight line of 75m, with cones delimiting the track;
- Skidpad track: a figure eight track with a diameter of 18.25m, delimited by cones.

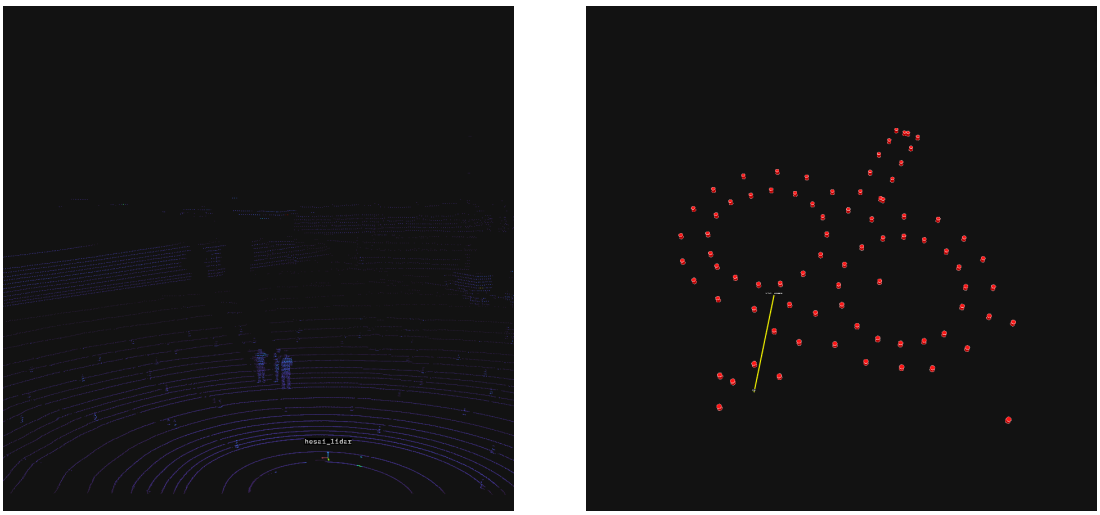
These scenarios complement each other well, providing both a high-speed straight-line scenario and a low-speed, high-curvature one. The vehicle was configured to reach 10 m/s in the Acceleration track and 3.5 m/s in the Skidpad track.

From the three LiDAR odometry methods tested, two were able to perform adequately in the Skidpad tests: LIMO-VELO (FAST-LIMO) and GenZ-ICP. These were consistently able to provide odometry estimates that allowed the vehicle to complete the course and map the track successfully. Figures 5.7 and 5.6 illustrate the results obtained with these methods. On the other hand, KISS-ICP was less reliable and failed a good percentage of the runs.



(a) GenZ-ICP LiDAR Pointcloud (b) GenZ-ICP Generated Skidpad Map

Figure 5.6: Skidpad scenario results using GenZ-ICP.



(a) LIMO-VELO LiDAR Pointcloud (b) LIMO-VELO Generated Skidpad Map

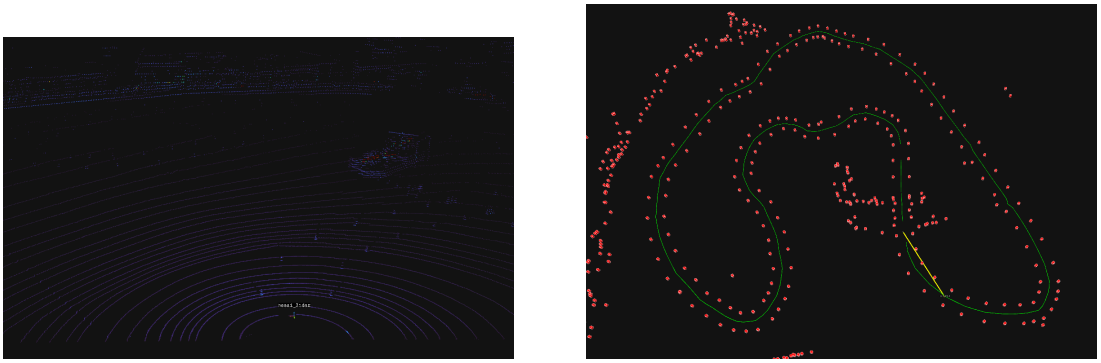
Figure 5.7: Skidpad scenario results using LIMO-VELO.

For the Acceleration tracks, not a single method was able to consistently provide odometry

estimates that allowed the vehicle to complete the course without mapping failures. This, however, can be explained by the fact that LiDAR used is not the most suitable for LiDAR odometry due to the reduced rates. The high speeds reached in this track, combined with the low LiDAR rate, lead to poor matching between frames, resulting in poor odometry estimates. Given more testing time, an alternative LiDAR driver, in which point clouds are published in segments at an increased rate, would have been tested.

### 5.2.3 Overall System Results

Despite the challenges, the final setup was able to map tracks for all the Driverless events and provide accurate pose estimates in real-time. Using Sliding-Window optimization and the Constant Velocity and Turn-rate model, the system provides map and pose estimates with less than 10 ms and 1 ms latency consistently, respectively. The system was able to map the FSG 2025 Autocross track in competition, as shown in Figure 5.8b, and provide accurate pose estimates for the vehicle to traverse the course (note that the outliers are originated by the Perception module and do not cause problems for the SLAM system).



(a) FSG 2025 Autocross LiDAR Pointcloud

(b) FSG 2025 Autocross Generated Map

Figure 5.8: FSG 2025 Autocross scenario.

# Chapter 6

# Conclusions & Future Work

This chapter summaries the work developed in this dissertation, presenting the main conclusions drawn from it and the future work that can be done to improve it.

## 6.1 Summary of Contributions

The first contribution of this work is a review of the State of the Art on SLAM in Autonomous racing was carried out: the latest advancements in SLAM algorithms were discussed and the challenges and known solutions for the SLAM problem in Autonomous Racing scenarios were presented.

In this thesis, a state estimation system composed of an EKF Velocity Estimation module and a Graph SLAM module was developed and testing in simulation and on-ground. This system fulfilled the objectives defined, being capable of performing at competition level and solving the problems the previous implementation was facing. The system was also designed with modularity and scalability in mind, allowing for easy replacement of components and future improvements.

The system also implemented different solutions to certain parts of the problem, allowing for a comparison of different approaches and the selection of the best one for the final implementation. This included the usage of iSAM2 as a solver, which had not been previously evaluated in this context. The inclusion and adoption of LiDAR odometry techniques was also briefly explored, by introducing three different implementations as odometry sources to the SLAM system implemented.

## 6.2 Conclusions

This dissertation suffered from some challenges and limitations regarding the testing and validation of the proposed solution. The main limitation was the lack of time available for on-ground testing. Despite this, experiments were still carried out and results were analyzed the best way possible.

From the results, we can conclude that the Sliding-Window method for optimization stands as the best solution for the Formula Student case, having enough accuracy for the tracks that a prototype will face in competition while maintaining low latencies all around. LiDAR odometry is confirmed to be utilizable and shown to work in a different paradigm that previously presented.

The objectives defined for the implementation were fulfilled, leaving the team with a state estimation system capable of being used in competition and serve as the basis for many others in the future.

## 6.3 Future Work

The work developed in this dissertation can be improved and extended in many ways. The most immediate one is the conduction of more extensive on-ground testing, with the acquisition of better quality data and the usage of higher accuracy ground truth systems. This would allow for a more thorough evaluation of the system and its performance in real-world scenarios. Secondly, a deeper exploration of the LiDAR odometry techniques could be done, with a more thorough evaluation of the different methods and further tuning and adaptation of the implementations. Lastly, the velocity estimation is to be improved, with the inclusion of more sensors and modeling of vehicle dynamics, which was not fundamental in this case, but will be crucial once the rest of the system allows for higher velocities and accelerations.

# References


[1] McKinsey & Company. *Mobility's future: An investment reality check | McKinsey*. URL: https://www.mckinsey.com/industries/automotive-and-assembly/our-insights/mobilitys-future-an-investment-reality-check (visited on 06/10/2024).

[2] J.J. Leonard and H.F. Durrant-Whyte. "Simultaneous map building and localization for an autonomous mobile robot". In: *Proceedings IROS '91:IEEE/RSJ International Workshop on Intelligent Robots and Systems '91*. Nov. 1991, 1442–1447 vol.3. DOI: 10.1109/IROS.1991.174711. URL: https://ieeexplore.ieee.org/document/174711 (visited on 06/09/2024).

[3] FSG. *FSG Competition Handbook 2025*. Ed. by FSG. FSG, 2024. URL: https://www.formulastudent.de/fileadmin/user_upload/all/2025/important_docs/FSG25_Competition_Handbook_v1.0.pdf.

[4] Leandro Masello et al. "On the road safety benefits of advanced driver assistance systems in different driving contexts". In: *Transportation Research Interdisciplinary Perspectives* 15 (Sept. 2022), p. 100670. ISSN: 2590-1982. DOI: 10.1016/j.trip.2022.100670. URL: https://www.sciencedirect.com/science/article/pii/S2590198222001300 (visited on 06/09/2024).

[5] World Health Organization. *Despite notable progress, road safety remains urgent global issue*. en. URL: https://www.who.int/news/item/13-12-2023-despite-notable-progress-road-safety-remains-urgent-global-issue (visited on 06/09/2024).

[6] Adam Torok et al. "Automatization in road transport: a review". en. In: *Production Engineering Archives* 20.20 (Sept. 2018), pp. 3–7. ISSN: 2353-7779. DOI: 10.30657/pea.2018.20.01. URL: https://www.sciendo.com/article/10.30657/pea.2018.20.01 (visited on 06/09/2024).

[7] Ekim Yurtsever et al. "A Survey of Autonomous Driving: Common Practices and Emerging Technologies". In: *IEEE Access* 8 (2020), pp. 58443–58469. DOI: 10.1109/ACCESS.2020.2983149.

[8] SAE International. *SAE Levels of Driving Automation™ Refined for Clarity and International Audience*. en. URL: https://www.sae.org/site/blog/sae-j3016-update (visited on 06/10/2024).

[9] Shantanu Ingle and Madhuri Phute. "Tesla Autopilot : Semi Autonomous Driving, an Uptick for Future Autonomy". en. In: 03.09 ().

[10] *Waymo Self-Driving Car Spotted in Violation of Traffic Rules on San Francisco's Van Ness Avenue*. en. May 2024. URL: https://hoodline.com/2024/05/waymo-self-driving-car-spotted-in-violation-of-traffic-rules-on-san-francisco-s-van-ness-avenue/ (visited on 09/04/2025).

[11] *Stagecoach's ground-breaking autonomous bus project: Pioneering a new era in public transport*. en. Section: Uncategorized. URL: https://www.intelligenttransport.com/transport-articles/170829/stagecoachs-ground-breaking-autonomous-bus-project-pioneering-a-new-era-in-public-transport/ (visited on 06/10/2024).

[12] Shuran Zheng et al. "Simultaneous Localization and Mapping (SLAM) for Autonomous Driving: Concept and Analysis". en. In: *Remote Sensing* 15.4 (Jan. 2023). Number: 4 Publisher: Multidisciplinary Digital Publishing Institute, p. 1156. ISSN: 2072-4292. DOI: 10.3390/rs15041156. URL: https://www.mdpi.com/2072-4292/15/4/1156 (visited on 06/26/2024).

[13] *Innovations From Racing That Found Their Way On Our Roads!* en-US. Section: Motorsports. May 2017. URL: https://www.endurancewarranty.com/learning-center/motorsports/racing-innovations-found-on-roads/ (visited on 06/10/2025).

[14] Tim Bailey and Hugh Durrant-Whyte. "Simultaneous Localisation and Mapping (SLAM): Part II State of the Art". en. In: ().

[15] Sebastian Thrun. "Probabilistic robotics". In: *Communications of the ACM* 45.3 (2002), pp. 52–57.

[16] Hidenobu Matsuki et al. *CodeMapping: Real-Time Dense Mapping for Sparse SLAM using Compact Scene Representations*. en. arXiv:2107.08994 [cs]. July 2021. URL: http://arxiv.org/abs/2107.08994 (visited on 06/25/2024).

[17] Jakob Engel, Vladlen Koltun, and Daniel Cremers. "Direct Sparse Odometry". en. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40.3 (Mar. 2018), pp. 611–625. ISSN: 0162-8828, 2160-9292. DOI: 10.1109/TPAMI.2017.2658577. URL: http://ieeexplore.ieee.org/document/7898369/ (visited on 06/25/2024).

[18] Wei Xu et al. "FAST-LIO2: Fast Direct LiDAR-Inertial Odometry". In: *IEEE Transactions on Robotics* 38.4 (Aug. 2022). Conference Name: IEEE Transactions on Robotics, pp. 2053–2073. ISSN: 1941-0468. DOI: 10.1109/TRO.2022.3141876. URL: https://ieeexplore.ieee.org/abstract/document/9697912 (visited on 06/27/2024).

[19] Mina Henein et al. "Dynamic SLAM: The Need For Speed". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. ISSN: 2577-087X. May 2020, pp. 2123–2129. DOI: 10.1109/ICRA40945.2020.9196895. URL: https://ieeexplore.ieee.org/abstract/document/9196895 (visited on 06/25/2024).

[20] Carlos Campos et al. "ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual–Inertial, and Multimap SLAM". In: *IEEE Transactions on Robotics* 37.6 (Dec. 2021), pp. 1874–1890. ISSN: 1941-0468. DOI: 10.1109/TRO.2021.3075644. URL: https://ieeexplore.ieee.org/abstract/document/9440682 (visited on 06/10/2025).

[21] Tixiao Shan et al. “LIO-SAM: Tightly-coupled Lidar Inertial Odometry via Smoothing and Mapping”. en. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Las Vegas, NV, USA: IEEE, Oct. 2020, pp. 5135–5142. ISBN: 978-1-72816-212-6. DOI: 10.1109/IROS45743.2020.9341176. URL: https://ieeexplore.ieee.org/document/9341176/ (visited on 06/26/2024).

[22] Julio A. Placed et al. “A Survey on Active Simultaneous Localization and Mapping: State of the Art and New Frontiers”. In: *IEEE Transactions on Robotics* 39.3 (June 2023). Conference Name: IEEE Transactions on Robotics, pp. 1686–1705. ISSN: 1941-0468. DOI: 10.1109/TRO.2023.3248510. URL: https://ieeexplore.ieee.org/abstract/document/10075065 (visited on 06/25/2024).

[23] Randall Smith, Matthew Self, and Peter Cheeseman. “Estimating Uncertain Spatial Relationships in Robotics”. en. In: *Autonomous Robot Vehicles*. Ed. by Ingemar J. Cox and Gordon T. Wilfong. New York, NY: Springer, 1990, pp. 167–193. ISBN: 978-1-4613-8997-2. DOI: 10.1007/978-1-4613-8997-2_14. URL: https://doi.org/10.1007/978-1-4613-8997-2_14 (visited on 06/25/2024).

[24] Frank Dellaert et al. “Monte carlo localization for mobile robots”. In: *Proceedings 1999 IEEE international conference on robotics and automation (Cat. No. 99CH36288C)*. Vol. 2. IEEE. 1999, pp. 1322–1328.

[25] Kevin Murphy and Stuart Russell. “Rao-Blackwellised Particle Filtering for Dynamic Bayesian Networks”. en. In: *Sequential Monte Carlo Methods in Practice*. Ed. by Arnaud Doucet, Nando de Freitas, and Neil Gordon. New York, NY: Springer, 2001, pp. 499–515. ISBN: 978-1-4757-3437-9. DOI: 10.1007/978-1-4757-3437-9_24. URL: https://doi.org/10.1007/978-1-4757-3437-9_24 (visited on 07/01/2024).

[26] Alaa Housein and Gao Xingyu. “Simultaneous Localization and Mapping using differential drive mobile robot under ROS”. In: *Journal of Physics: Conference Series* 1820 (Mar. 2021), p. 012015. DOI: 10.1088/1742-6596/1820/1/012015.

[27] Michael Montemerlo et al. “FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem”. en. In: (2002).

[28] Michael Montemerlo et al. “FastSLAM 2.0: An Improved Particle Filtering Algorithm for Simultaneous Localization and Mapping that Provably Converges”. en. In: *IJCAI*. Vol. 3. 2003, pp. 1151–1156.

[29] Luis Lopes. “A SLAM Method for the Formula Student Driverless Competition.pdf”. MA thesis. Instituto Superior Técnico, 2021.

[30] Sebastian Thrun et al. “Simultaneous Mapping and Localization with Sparse Extended Information Filters: Theory and Initial Results”. en. In: *Algorithmic Foundations of Robotics V*. Ed. by Jean-Daniel Boissonnat et al. Vol. 7. Series Title: Springer Tracts in Advanced Robotics. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 363–380. ISBN: 978-3-642-07341-0 978-3-540-45058-0. DOI: 10.1007/978-3-540-45058-0_22. URL: http://link.springer.com/10.1007/978-3-540-45058-0_22 (visited on 06/25/2024).

[31] F. Lu and E. Milios. “Globally Consistent Range Scan Alignment for Environment Mapping”. en. In: *Autonomous Robots* 4.4 (Oct. 1997), pp. 333–349. ISSN: 1573-7527. DOI: 10.1023/A:1008854305733. URL: https://doi.org/10.1023/A:1008854305733 (visited on 06/25/2024).

[32] Giorgio Grisetti et al. “A Tutorial on Graph-Based SLAM”. In: *IEEE Intelligent Transportation Systems Magazine* 2.4 (2010). Conference Name: IEEE Intelligent Transportation Systems Magazine, pp. 31–43. ISSN: 1941-1197. DOI: 10.1109/MITS.2010.939925. URL: https://ieeexplore.ieee.org/abstract/document/5681215 (visited on 06/25/2024).

[33] *Factor graph*. en. Page Version ID: 1259575473. Nov. 2024. URL: https://en.wikipedia.org/w/index.php?title=Factor_graph&oldid=1259575473 (visited on 06/10/2025).

[34] Michael Kaess, Ananth Ranganathan, and Frank Dellaert. “iSAM: Incremental Smoothing and Mapping”. In: *IEEE Transactions on Robotics* 24.6 (Dec. 2008). Conference Name: IEEE Transactions on Robotics, pp. 1365–1378. ISSN: 1941-0468. DOI: 10.1109/TRO.2008.2006706. URL: https://ieeexplore.ieee.org/abstract/document/4682731 (visited on 07/01/2024).

[35] Michael Kaess et al. “iSAM2: Incremental smoothing and mapping using the Bayes tree”. en. In: *The International Journal of Robotics Research* 31.2 (Feb. 2012). Publisher: SAGE Publications Ltd STM, pp. 216–235. ISSN: 0278-3649. DOI: 10.1177/0278364911430419. URL: https://doi.org/10.1177/0278364911430419 (visited on 07/01/2024).

[36] Rainer Kummerle et al. “$G^2o$: A general framework for graph optimization”. en. In: *2011 IEEE International Conference on Robotics and Automation*. Shanghai, China: IEEE, May 2011, pp. 3607–3613. ISBN: 978-1-61284-386-5. DOI: 10.1109/ICRA.2011.5979949. URL: http://ieeexplore.ieee.org/document/5979949/ (visited on 06/26/2024).

[37] Sameer Agarwal, Keir Mierle, and The Ceres Solver Team. *Ceres Solver*. Version 2.2. Oct. 2023. URL: https://github.com/ceres-solver/ceres-solver.

[38] Frank Dellaert and GTSAM Contributors. *borglab/gtsam*. Version 4.2a8. May 2022. DOI: 10.5281/zenodo.5794541. URL: https://github.com/borglab/gtsam).

[39] Andela Juric et al. “A Comparison of Graph Optimization Approaches for Pose Estimation in SLAM”. en. In: *2021 44th International Convention on Information, Communication and Electronic Technology (MIPRO)*. Opatija, Croatia: IEEE, Sept. 2021, pp. 1113–1118. ISBN: 978-953-233-101-1. DOI: 10.23919/MIPRO52101.2021.9596721. URL: https://ieeexplore.ieee.org/document/9596721/ (visited on 06/26/2024).

[40] Cesar Cadena et al. “Simultaneous Localization And Mapping: Present, Future, and the Robust-Perception Age”. In: *IEEE Transactions on Robotics* 32 (June 2016). DOI: 10.1109/TRO.2016.2624754.

[41] *(23) Kinematic vs. Dynamic: Choosing the Right Model for Vehicle Simulations | LinkedIn*. Apr. 15, 2025. URL: https://www.linkedin.com/pulse/kinematic-vs-dynamic-choosing-right-model-vehicle-simulations-jain-g83gf/ (visited on 09/05/2025).

[42] J.-S. Gutmann and K. Konolige. “Incremental mapping of large cyclic environments”. In: *Proceedings 1999 IEEE International Symposium on Computational Intelligence in Robotics and Automation. CIRA’99 (Cat. No.99EX375)*. Nov. 1999, pp. 318–325. DOI: 10.1109/CIRA.1999.810068. URL: https://ieeexplore.ieee.org/abstract/document/810068 (visited on 06/26/2024).

[43] Xiaobin Xu et al. "A Review of Multi-Sensor Fusion SLAM Systems Based on 3D LIDAR". en. In: *Remote Sensing* 14.12 (Jan. 2022). Publisher: Multidisciplinary Digital Publishing Institute, p. 2835. ISSN: 2072-4292. DOI: 10.3390/rs14122835. URL: https://www.mdpi.com/2072-4292/14/12/2835 (visited on 09/02/2025).

[44] Nanxi Li et al. "A Progress Review on Solid-State LiDAR and Nanophotonics-Based LiDAR Sensors". en. In: *Laser & Photonics Reviews* 16.11 (2022). _eprint: https://onlinelibrary.wiley.com/doi/ p. 2100511. ISSN: 1863-8899. DOI: 10.1002/lpor.202100511. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/lpor.202100511 (visited on 06/26/2024).

[45] Ji Zhang and Sanjiv Singh. "LOAM: Lidar Odometry and Mapping in Real-time". en. In: *Robotics: Science and Systems X*. Robotics: Science and Systems Foundation, July 2014. ISBN: 978-0-9923747-0-9. DOI: 10.15607/RSS.2014.X.007. URL: http://www.roboticsproceedings.org/rss10/p07.pdf (visited on 06/01/2024).

[46] Cheng Yuan et al. "A Novel Fault-Tolerant Navigation and Positioning Method with Stereo-Camera/Micro Electro Mechanical Systems Inertial Measurement Unit (MEMS-IMU) in Hostile Environment". In: *Micromachines* 9 (Nov. 2018), p. 626. DOI: 10.3390/mi9120626.

[47] Qin Zou et al. "A Comparative Analysis of LiDAR SLAM-Based Indoor Navigation for Autonomous Vehicles". In: *IEEE Transactions on Intelligent Transportation Systems* 23.7 (July 2022). Conference Name: IEEE Transactions on Intelligent Transportation Systems, pp. 6907–6921. ISSN: 1558-0016. DOI: 10.1109/TITS.2021.3063477. URL: https://ieeexplore.ieee.org/abstract/document/9381521 (visited on 06/03/2024).

[48] Tixiao Shan and Brendan Englot. "LeGO-LOAM: Lightweight and Ground-Optimized Lidar Odometry and Mapping on Variable Terrain". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. ISSN: 2153-0866. Oct. 2018, pp. 4758–4765. DOI: 10.1109/IROS.2018.8594299. URL: https://ieeexplore.ieee.org/abstract/document/8594299?casa_token=EeVwFq5HcLIAAAAA:zVsHz_QdQvibxThD0qbcJUbpPr1qjh-3R1QcqcgGupwMM9jLH2lSPuComnsBEzOiRFBD_PY (visited on 06/01/2024).

[49] Han Wang et al. "F-LOAM : Fast LiDAR Odometry and Mapping". In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. ISSN: 2153-0866. Sept. 2021, pp. 4390–4396. DOI: 10.1109/IROS51168.2021.9636655. URL: https://ieeexplore.ieee.org/abstract/document/9636655?casa_token=3Nqp-20-cuwAAAAA:GGBbZYFZVWdegigHwdg-NliDPfxrLjOJ5zoPw-fJPxdcy44Sq6vBYcuSpsGvO0N (visited on 06/01/2024).

[50] Ji Zhang and Sanjiv Singh. "Visual-lidar odometry and mapping: low-drift, robust, and fast". en. In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. Seattle, WA, USA: IEEE, May 2015, pp. 2174–2181. ISBN: 978-1-4799-6923-4. DOI: 10.1109/ICRA.2015.7139486. URL: http://ieeexplore.ieee.org/document/7139486/ (visited on 06/26/2024).

[51] Wei Xu and Fu Zhang. *FAST-LIO: A Fast, Robust LiDAR-inertial Odometry Package by Tightly-Coupled Iterated Kalman Filter*. en. arXiv:2010.08196 [cs]. Apr. 2021. URL: http://arxiv.org/abs/2010.08196 (visited on 05/31/2024).

[52] João Graça Martins. "Multi-Sensorial Simultaneous Localization and Mapping in Unmanned Aerial Vehicles". eng. Accepted: 2023-12-18T01:16:33Z. MA thesis. Faculdade de Engenharia da Universidade do Porto, July 2023. URL: https://repositorio-aberto.up.pt/handle/10216/151903 (visited on 06/26/2024).

[53] Chunge Bai et al. “Faster-LIO: Lightweight tightly coupled LiDAR-inertial odometry using parallel sparse incremental voxels”. In: *IEEE Robotics and Automation Letters* 7.2 (2022), pp. 4861–4868.

[54] Chunran Zheng et al. “Fast-livo: Fast and tightly-coupled sparse-direct lidar-inertial-visual odometry”. In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2022, pp. 4003–4009.

[55] Ignacio Vizzo et al. “KISS-ICP: In Defense of Point-to-Point ICP – Simple, Accurate, and Robust Registration If Done the Right Way”. In: *IEEE Robotics and Automation Letters* 8.2 (Feb. 2023). Conference Name: IEEE Robotics and Automation Letters, pp. 1029–1036. ISSN: 2377-3766. DOI: 10.1109/LRA.2023.3236571. URL: https://ieeexplore.ieee.org/abstract/document/10015694 (visited on 04/17/2024).

[56] Pierre Dellenbach et al. “CT-ICP: Real-time Elastic LiDAR Odometry with Loop Closure”. en. In: *2022 International Conference on Robotics and Automation (ICRA)*. Philadelphia, PA, USA: IEEE, May 2022, pp. 5580–5586. ISBN: 978-1-72819-681-7. DOI: 10.1109/ICRA46639.2022.9811849. URL: https://ieeexplore.ieee.org/document/9811849/ (visited on 06/26/2024).

[57] Andreas Geiger, Philip Lenz, and Raquel Urtasun. “Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite”. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2012.

[58] Daehan Lee, Hyungtae Lim, and Soohee Han. “GenZ-ICP: Generalizable and Degeneracy-Robust LiDAR Odometry Using an Adaptive Weighting”. In: *IEEE Robotics and Automation Letters (RA-L)* 10.1 (2025), pp. 152–159. DOI: 10.1109/LRA.2024.3498779.

[59] Jie Yin et al. “Ground-challenge: A multi-sensor slam dataset focusing on corner cases for ground robots”. In: *2023 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE. 2023, pp. 1–5.

[60] Peiyuan Jiang et al. “A Review of Yolo Algorithm Developments”. In: *Procedia Computer Science*. The 8th International Conference on Information Technology and Quantitative Management (ITQM 2020 & 2021): Developing Global Digital Economy after COVID-19 199 (Jan. 2022), pp. 1066–1073. ISSN: 1877-0509. DOI: 10.1016/j.procs.2022.01.135. URL: https://www.sciencedirect.com/science/article/pii/S1877050922001363 (visited on 06/27/2024).

[61] Keisuke Tateno et al. “CNN-SLAM: Real-Time Dense Monocular SLAM With Learned Depth Prediction”. In: 2017, pp. 6243–6252. URL: https://openaccess.thecvf.com/content_cvpr_2017/html/Tateno_CNN-SLAM_Real-Time_Dense_CVPR_2017_paper.html (visited on 06/27/2024).

[62] Wenshan Wang, Yaoyu Hu, and Sebastian Scherer. “TartanVO: A Generalizable Learning-based VO”. en. In: *Proceedings of the 2020 Conference on Robot Learning*. ISSN: 2640-3498. PMLR, Oct. 2021, pp. 1761–1772. URL: https://proceedings.mlr.press/v155/wang21h.html (visited on 06/27/2024).

[63] Wenshan Wang et al. “TartanAir: A Dataset to Push the Limits of Visual SLAM”. In: (2020).

[64] Julian Nubert, Shehryar Khattak, and Marco Hutter. “Self-supervised Learning of LiDAR Odometry for Robotic Applications”. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. ISSN: 2577-087X. May 2021, pp. 9601–9607. DOI: 10.1109/ICRA48506.2021.9561063. URL: https://ieeexplore.ieee.org/abstract/document/9561063 (visited on 06/27/2024).

[65] Younggun Cho, Giseop Kim, and Ayoung Kim. "Unsupervised Geometry-Aware Deep LiDAR Odometry". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. 2020, pp. 2145–2152. DOI: 10.1109/ICRA40945.2020.9197366.

[66] Johannes Betz et al. "Autonomous Vehicles on the Edge: A Survey on Autonomous Vehicle Racing". In: *IEEE Open Journal of Intelligent Transportation Systems* 3 (2022). Conference Name: IEEE Open Journal of Intelligent Transportation Systems, pp. 458–488. ISSN: 2687-7813. DOI: 10.1109/OJITS.2022.3181510. URL: https://ieeexplore.ieee.org/abstract/document/9790832 (visited on 06/29/2024).

[67] Miguel I. Valls et al. "Design of an Autonomous Racecar: Perception, State Estimation and System Integration". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. ISSN: 2577-087X. May 2018, pp. 2048–2055. DOI: 10.1109/ICRA.2018.8462829. URL: https://ieeexplore.ieee.org/abstract/document/8462829?casa_token=v7RlVWIBEdsAAAAA:Mz665jjijSW3Z0X97fkhmuhF-GB-jOT9U2qiriHFB3aJ-EcIAv0qdwqfzhq8ENI3m8dCCQ8 (visited on 06/16/2024).

[68] Nikhil Bharadwaj Gosala et al. "Redundant Perception and State Estimation for Reliable Autonomous Racing". In: *2019 International Conference on Robotics and Automation (ICRA)*. arXiv:1809.10099 [cs]. May 2019, pp. 6561–6567. DOI: 10.1109/ICRA.2019.8794155. URL: http://arxiv.org/abs/1809.10099 (visited on 06/28/2024).

[69] Hans Pacejka. *Tire and vehicle dynamics*. Elsevier, 2005.

[70] Juraj Kabzan et al. "AMZ Driverless: The full autonomous racing system". en. In: *Journal of Field Robotics* 37.7 (2019). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.21977, pp. 1267–1294. ISSN: 1556-4967. DOI: 10.1002/rob.21977. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.21977 (visited on 06/10/2024).

[71] Leiv Andresen et al. "Accurate Mapping and Planning for Autonomous Racing". In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. ISSN: 2153-0866. Oct. 2020, pp. 4743–4749. DOI: 10.1109/IROS45743.2020.9341702. URL: https://ieeexplore.ieee.org/abstract/document/9341702?casa_token=pkMcS9Y0jUAAAAAA:P-wBeCW_vjseEZVToHmrKkhWhrRw7DyEP1nNVGQ3HN5IrAdloddZ0OKkdZmJ9FIUVo (visited on 06/16/2024).

[72] Sherif Nekkah et al. *The Autonomous Racing Software Stack of the KIT19d*. arXiv:2010.02828 [cs]. Oct. 2020. DOI: 10.48550/arXiv.2010.02828. URL: http://arxiv.org/abs/2010.02828 (visited on 06/10/2024).

[73] Nick Le Large. "A comparison of different approaches to solve the SLAM problem on a Formula Student Driverless race car". en. MA thesis. Karlsruhe Institute of Technology, 2020.

[74] Andres Alvarez et al. *The Software Stack That Won the Formula Student Driverless Competition*. en. arXiv:2210.10933 [cs]. Oct. 2022. URL: http://arxiv.org/abs/2210.10933 (visited on 06/10/2024).

[75] Diogo Morgado. "A Perception Pipeline for an Autonomous Formula Student Vehicle". en. MA thesis. Instituto Superior Técnico, 2021.

[76] Stanford Artificial Intelligence Laboratory et al. *Robotic Operating System*. Version ROS Melodic Morenia. May 23, 2018. URL: https://www.ros.org.

[77] Nuno Alexandre de Almeida Salgueiro. "Torque vectoring control of an electric vehicle with in-wheel motors". In: *examensarb., Universidade de Lisboa* (2021), pp. 4–18.

[78] Andreu Huguet Segarra et al. “LIMO-Velo: A real-time, robust, centimeter-accurate estimator for vehicle localization and mapping under racing velocities”. B.S. thesis. Universitat Politècnica de Catalunya, 2022.

[79] Alastair Bradford, Grant van Breda, and Tobias Fischer. *Racing With ROS 2 A Navigation System for an Autonomous Formula Student Race Car*. arXiv:2311.14276 [cs]. Nov. 2023. DOI: 10.48550/arXiv.2311.14276. URL: http://arxiv.org/abs/2311.14276 (visited on 06/10/2024).

[80] Zeilinger, Marcel et al. *Design of an Autonomous Race Car for the Formula Student Driverless (FSD)*. en. ISBN: 9783851255249. DOI: 10.3217/978-3-85125-524-9-10. URL: https://openlib.tugraz.at/download.php?id=5aaa45931188a&location=medra (visited on 06/10/2024).

[81] Sirish Srinivasan et al. *End-to-End Velocity Estimation For Autonomous Racing*. en. arXiv:2003.06917 [cs]. Aug. 2020. URL: http://arxiv.org/abs/2003.06917 (visited on 06/24/2024).

[82] Sebastian Thrun et al. “Stanley: The robot that won the DARPA Grand Challenge”. en. In: *Journal of Field Robotics* 23.9 (2006). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.20147, pp. 661–692. ISSN: 1556-4967. DOI: 10.1002/rob.20147. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.20147 (visited on 06/16/2024).

[83] Markus Schratter et al. “LiDAR-based mapping and Localization for autonomous racing”. In: *Proc. Int. Conf. Robot. Autom.(ICRA) Workshop Opportunities Challenges Auton. Racing*. 2021, pp. 1–6.

[84] Federico Massa et al. “LiDAR-Based GNSS Denied Localization for Autonomous Racing Cars”. en. In: *Sensors* 20.14 (Jan. 2020). Number: 14 Publisher: Multidisciplinary Digital Publishing Institute, p. 3992. ISSN: 1424-8220. DOI: 10.3390/s20143992. URL: https://www.mdpi.com/1424-8220/20/14/3992 (visited on 06/16/2024).

[85] Gaël Guennebaud, Benoît Jacob, et al. *Eigen v3*. http://eigen.tuxfamily.org. 2010.

[86] Steven Macenski et al. “Robot Operating System 2: Design, architecture, and uses in the wild”. In: *Science Robotics* 7.66 (2022), eabm6074. DOI: 10.1126/scirobotics.abm6074. URL: https://www.science.org/doi/abs/10.1126/scirobotics.abm6074.

[87] Alonzo Kelly. “A vector algebra formulation of kinematics of wheeled mobile robots”. In: *International conference on Field and Service Robotics*. 2010, pp. 1–14.

[88] Robin Schubert, Eric Richter, and Gerd Wanielik. “Comparison and evaluation of advanced motion models for vehicle tracking”. In: *2008 11th International Conference on Information Fusion*. June 2008, pp. 1–6. URL: https://ieeexplore.ieee.org/document/4632283/?arnumber=4632283 (visited on 01/19/2025).

[89] Robin Schubert et al. “Empirical evaluation of vehicular models for ego motion estimation”. In: *2011 IEEE Intelligent Vehicles Symposium (IV)*. 2011 IEEE Intelligent Vehicles Symposium (IV). ISSN: 1931-0587. June 2011, pp. 534–539. DOI: 10.1109/IVS.2011.5940526. URL: https://ieeexplore.ieee.org/abstract/document/5940526 (visited on 06/12/2024).

[90] Tim D. Barfoot. “State Estimation for Robotics”. In: 2017. URL: https://api.semanticscholar.org/CorpusID:65172180.

# Appendix A

# Literature Review Appendices

## A.1 SLAM in Autonomous Racing

### A.1.1 FST Lisboa

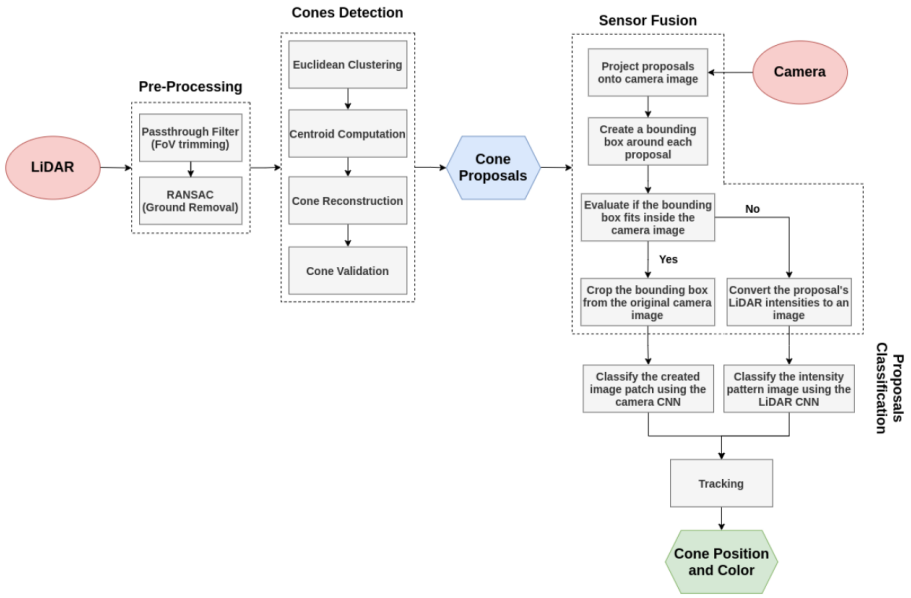



Figure A.1: FST Perception Pipeline from [75]

# Appendix B

# Practical Implementation Appendices

## B.1 Solution Overall Design

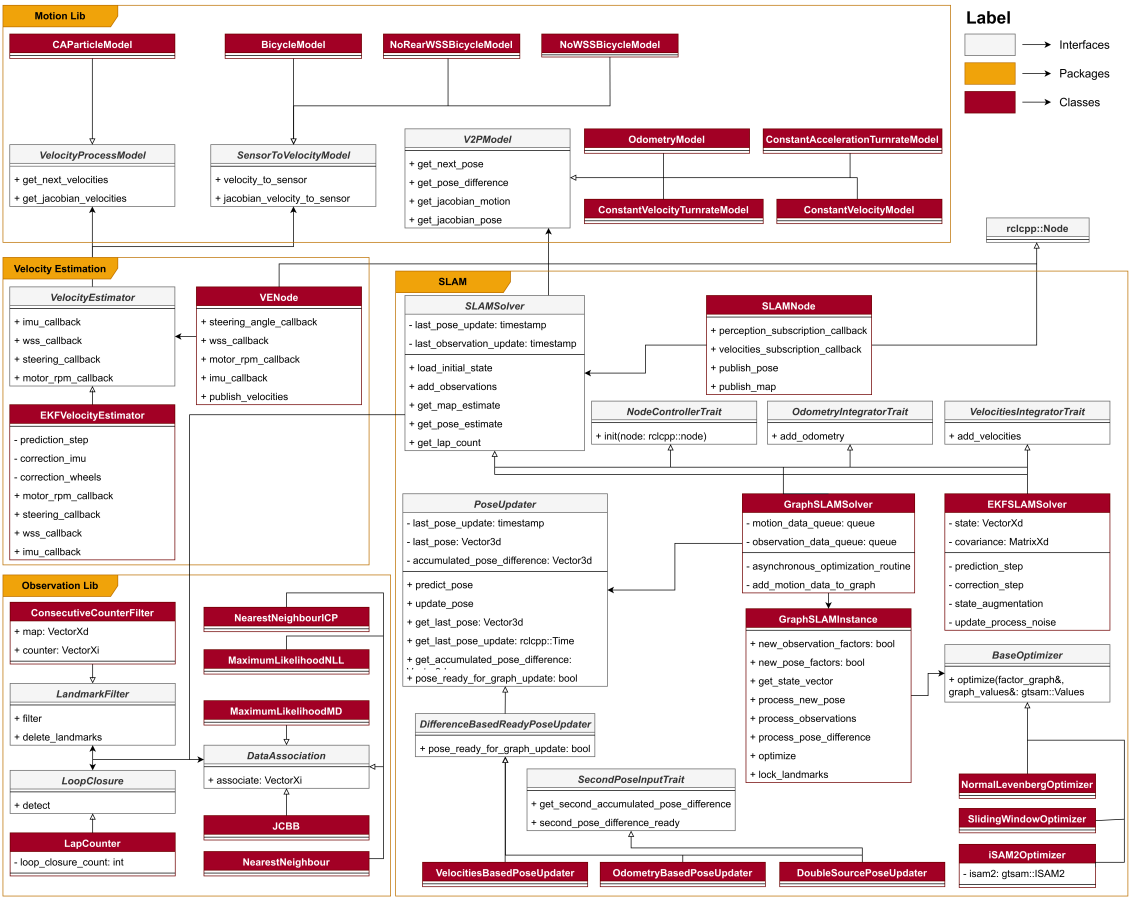



Figure B.1: Complete Class Diagram

# Appendix C

# Results Analysis Appendices

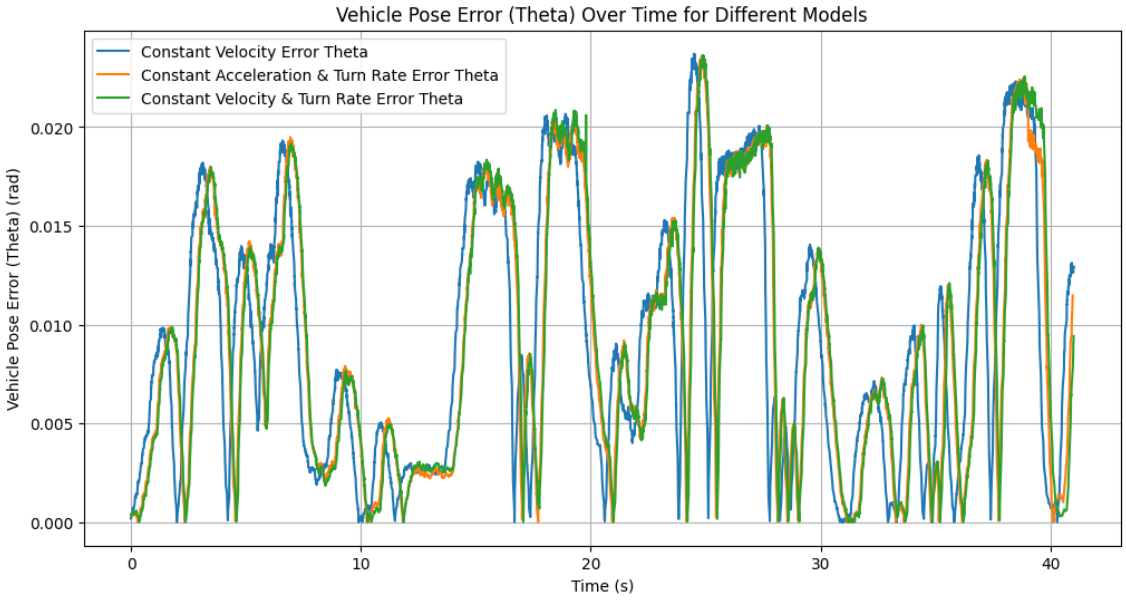



Figure C.1: Orientation accuracy results for pose estimation in FSG 2024 Autocross in simulation environment for each motion model.