



UNIVERSITÀ
DEGLI STUDI
DI PADOVA

UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

MASTER'S DEGREE IN COMPUTER ENGINEERING

A Practical Multi-sensor Localization and Mapping Approach for the Formula Student Driverless Competition

Supervisor:

PROF. ALBERTO PRETTO

Co-supervisor:

EMILIO OLIVASTRI, PHD CANDIDATE

Student:

NAVID KHALILI

ID number:

2044010

Academic year 2023/2024

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Professor Alberto Pretto, for the invaluable time and effort he dedicated to guiding me throughout this research. His profound knowledge and unwavering support have been instrumental in the completion of this thesis. I am deeply grateful for all the wisdom and expertise he has shared with me.

I also wish to extend my heartfelt thanks to my co-supervisor, Emilio Olivastri, PhD candidate, for his invaluable advice and continuous assistance. His dedication and contributions have significantly enriched this project, and I am sincerely thankful for all the help he has provided.

Additionally, I want to thank my mom, dad, little brother, and little sister. Their support and sacrifices made the opportunity of working on this thesis possible for me. Without their encouragement and understanding, this journey would not have been possible.

“The only way to achieve the impossible is to believe it is possible”

Abstract

Formula Student is an international engineering competition where student teams design, build, and race small formula-style race cars. The competition provides a platform for students to apply their engineering knowledge in a practical, hands-on project and involves various disciplines such as mechanical, electrical, and computer engineering. One category of this competition is Driverless. This competition challenges student teams to design, build, and program autonomous vehicles that can navigate and compete in various dynamic and static events without a human driver.

This thesis focuses on building a localization and mapping system for the Formula Student team from the University of Padova, RaceUp. The main goal is to transition the algorithms from a simulator environment to a real-world setup. The focus of this work is on vehicle sensorization, visual perception algorithms, and simultaneous localization and mapping (SLAM).

The SLAM process is typically divided into two main parts: the front-end, which involves sensor data collection and feature extraction, and the back-end, which focuses on optimizing the vehicle's estimated trajectory and the map of its surroundings.

The evaluation of the pipeline was conducted using real-world data obtained by equipping sensors onto the vehicle and traversing the track. By employing a blend of contemporary and traditional methodologies, we analyze data generated by the stereo camera to localize the colored cones outlining the track. The derived positions of the cones subsequently feed into critical modules of the vehicle, including the control system and the SLAM pipeline.

Contents

1	Introduction	1
1.1	Problem statement	1
1.2	Formula Student	3
1.3	Thesis objectives and outline	8
2	Related work	11
3	Theoretical background	15
3.1	Visual Perception	16
3.2	Simultaneous Localization And Mapping (SLAM)	18
4	Test environment	23
4.1	EUSSIM	25
4.2	Test mule	26
5	Real data acquisition	29
5.1	About the car	30
5.2	Car sensorization	30
5.3	Dataset	33
5.4	Updated setup	34
6	Front-end	39
6.1	Deep Vision-based cone detection	39
6.2	Stereo matching	41
7	Back-end building blocks	45
7.1	Graph-Based SLAM	45
7.2	Algorithm implementation	48
7.2.1	g ² o framework	48
7.2.2	Graph optimization	49

7.2.3	Anti-Ackerman steering and Motion model	51
7.2.4	Data association	53
7.2.5	Loop closure	55
7.2.6	ICR motion constraint	57
7.3	Evaluation on simulator and real world	59
7.3.1	Ground Truth	59
7.3.2	Experiments and results	60
8	Conclusion and future works	69
	References	71

Chapter 1

Introduction

Self-driving cars are emerging as a transformative technology with the potential to revolutionize transportation. As urban populations grow and traffic congestion increases, traditional methods of mobility are becoming less efficient and more unsustainable. Self-driving cars offer a promising solution to these challenges by enhancing road safety, reducing traffic congestion, and improving transportation efficiency.

Many fields can benefit from self-driving cars. Autonomous vehicles can streamline supply chains and delivery services, improving efficiency and reducing costs. In addition, Self-driving buses and shuttles can provide efficient and reliable public transportation solutions. In the field of agriculture, autonomous tractors and machinery can enhance precision farming, increasing productivity and reducing labor costs.

The development of self-driving cars has made significant strides in recent years. Significant progress has been made in sensor technology, artificial intelligence, and machine learning, enabling self-driving cars to perceive and interpret their environment accurately. LiDAR, radar, cameras, and GPS systems are being integrated to provide comprehensive situational awareness.

1.1 Problem statement

The design of a self-driving vehicle requires encountering several factors, from arranging hardware components to developing complex software systems. For an autonomous vehicle to navigate independently in unfamiliar situations, it requires various interconnected modules, and the ones we are going to focus on are perception, localization, and mapping. Naturally, the entire software stack would

be ineffective without a properly mounted set of sensors on the vehicle.

Perception in autonomous vehicles is analogous to human senses, as it involves collecting information from the surrounding environment. This ability to perceive the nearby world is fundamental to autonomous navigation. Perception data is used to determine whether there are obstacles in the vehicle's path, whether the road curves, or remains straight. Perception, as the initial stage of the self-driving pipeline, forms the backbone of the entire system. Reliable perception provides high-quality input to SLAM, while noisy perception hinders the overall system's performance.

SLAM [1] stands for simultaneous localization and mapping and is arguably the most extensive component of the self-driving pipeline. The SLAM problem revolves around estimating the map of the environment while simultaneously localizing the vehicle within that map. As a result, it is possible to determine the pose of the vehicle relative to the world and the associated uncertainty of that estimate.

As previously mentioned, there are numerous scenarios where autonomous driving technologies can be applied. One such scenario is the world of car racing. In this work, we will focus on a driverless racing event that is part of the Formula Student competition. This project is specifically developed for RaceUp [2], the Formula Student team of the University of Padova. Over the past 17 years, RaceUp has participated in both national and European-level races with electric and combustion vehicles. With competitions increasingly shifting towards driverless formats, there is now a pressing need for a reliable driverless system to compete in racing events.

In general, designing a vehicle capable of competing at a high-level event is far from simple. It requires seamless integration of numerous components to ensure the complex system functions effectively in dynamic situations. Moreover, redundancy in modules is essential to ensure fault recovery capabilities. Racing scenarios demand a fast processing computing system that is lightweight and has low power consumption, adding further complexity to the design process.

If we consider specifically the RaceUP Driverless case, the project is particularly challenging not only from a technical point of view but also because:

- the autonomous car prototype is modified every year with changes concerning the kinematics of the car, low-level software structure, and sensors related to each component of the car.
- the choice of sensors to be mounted on the vehicle is constrained to univer-

sity funds, weight of sensor, and power consumption;

- sensor shape and mounting position affects the design of aerodynamics and weight distribution, so all these aspects need to be discussed with the members belonging to many different departments;
- being this project the first concept of driverless vehicle at all, it is a learn-by-doing activity for everyone.

Given that Formula Student may not be widely recognized outside of those directly involved in the field, the following section will offer an overview of this event, with a specific focus on the Driverless category.

1.2 Formula Student

Formula Student (also known as Formula SAE, organized by the Society of Automotive Engineers) is a prestigious student competition founded in 1981. The competition challenges university students to design, build, and race a prototype single-seater, high-performance Formula-style race car. These cars are tailored to participate in nationally and internationally recognized racing events.

Formula-style vehicles are characterized by their open-wheeled design, single seat, open cockpit, and four wheels that are not aligned in a straight line [3]. Participating in Formula Student is a comprehensive challenge as it requires students to conceive, design, fabricate, develop, and compete with their cars.

Teams are composed exclusively of university students from diverse backgrounds, providing knowledge across various fields from all around the world (see figure 1.1). Essential skills for the competition include mechanical, electrical, and software engineering, as well as economics, management, and marketing.

Each team must build their own cars in compliance with a series of official rules that define specific characteristics of the chassis and safety conditions for the races. Adhering to these regulations also enhances the problem-solving capabilities of future engineers, who will seldom have total freedom in their professional projects.

Historically, Formula Student started with the development of a combustion vehicle with a traditional thermal motor. In 2010, an electrical division was introduced, requiring teams to also provide fully electrically-powered prototypes. In 2017, the German Formula Student committee initiated the Driverless division, aiming to design self-driving vehicles capable of completing all events without



Figure 1.1: All participants at Formula Student Germany 2023 [4]

human driver assistance or remote control. An example of these driverless cars is presented in figure 1.2.

From this point forward, all references to Formula Student will implicitly refer to the driverless race car, as this thesis focuses on designing and developing components of an autonomous racing system. The information provided pertains to the 2024 competition, as stated in the official documents [3] at the time of writing.

Specific regulations concerning both the car and the track have been established to ensure the safety of all participants. According to the Formula Student Germany Competition Handbook 2024 [3], tracks will be marked with colored cones: blue on the left side, yellow on the right, small orange on the entry and exit lanes, and large orange before and after the start, at the finish, and time-keeping points (see figure 1.3). Cones along the driving direction will be no more than 5 meters apart, and specific zones will be marked with colored paint.

Before the competition begins, each vehicle undergoes a Technical Inspection to ensure rule compliance. For the autonomous system, this includes providing data sheets for all perception sensors and documentation certifying their compliance with local legislation. The Remote Emergency System, a remote-controlled module on the vehicles that activates emergency behaviors, is also verified. This



Figure 1.2: Example of a fully sensorized driverless formula student car made by AMZ team [5]

system is activated if:

- the autonomous vehicle seems to be out of control;
- the vehicle gets visibly damaged (mechanically or electrically);
- the minimum average speed during the track drive event is not respected (2.5 m/s for the first three laps, and 3.5 m/s for the following ones);
- the presence on the track of something/someone not allowed to be there.

Other inspections include a tilt test, to check the wheels' contact with the ground and the presence of fluid leaks. A rain test for the safety of the electrical system in case of adverse weather conditions.

If the Technical Inspection has been successfully completed, the prototype is then judged according to two types of tests: static events, to rate cost analysis, business presentation, and engineering design capabilities of the participants, and dynamic events for the performance and technical reliability.

A brief overview of the two evaluation events is provided now, together with the official track configuration for the racing events, when available:

1. STATIC EVENTS





			
big orange cone two white stripes	small orange cone single white stripe	small yellow cone single black stripe	small blue cone single white stripe
WEMAS 307.610500.00.00	WEMAS 400.000013.00.00	WEMAS 400.000013.01.10	WEMAS 400.000043.00.00
285 mm × 285 mm × 505 mm 1.05 kg	228 mm × 228 mm × 325 mm 0.45 kg		

Figure 1.3: Official Driverless competition cones.

Business Plan Presentation: teams must deliver a comprehensive business model to convince potential investors or partners of the project’s profitability.

Cost and Manufacturing: this event evaluates the team’s ability to make cost-effective decisions during manufacturing, including make-or-buy choices. Teams must submit detailed Cost Report Documents for every material and component used in the vehicle.

Engineering Design: in this event teams explain their design choices and highlight features, concepts, or methods that add value to the vehicle

2. DYNAMIC EVENTS

Skidpad: an eight-shaped track, figure 1.4, consisting of two pairs of concentric circles, has to be cleared twice. This means that the vehicle must perform two laps around each circle, as well as autonomously enter and exit from the test area.

Acceleration: the track here is a simple straight line with specific length and width, figure 1.5, delimited by a starting and a finish line. The event consists of an acceleration test from a standing start. The primary metric that is used for evaluating this event is the taken time to complete the track.

Autocross: the autocross track does not have a fixed layout, but it is designed adhering to predefined constraints on the length, width, and number of curves and straight sections. The goal is to complete two runs consisting of one lap each, in the minimum possible time. The evaluation is obtained

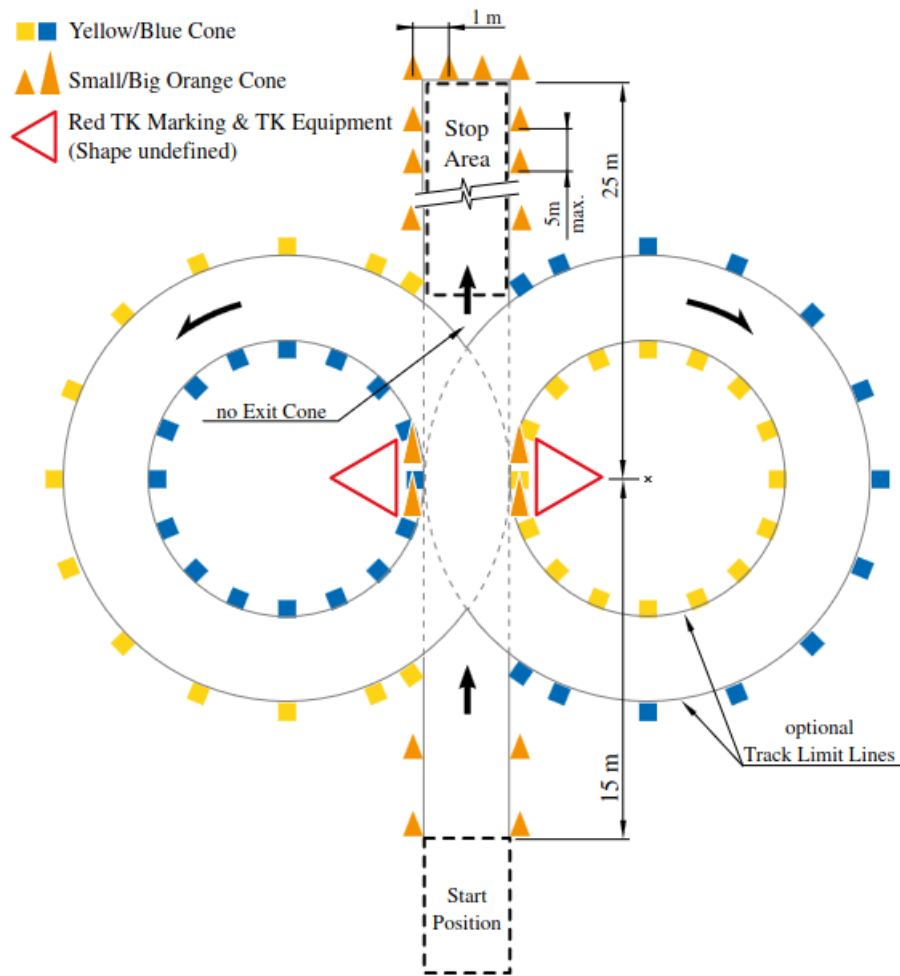


Figure 1.4: Skidpad track configuration.

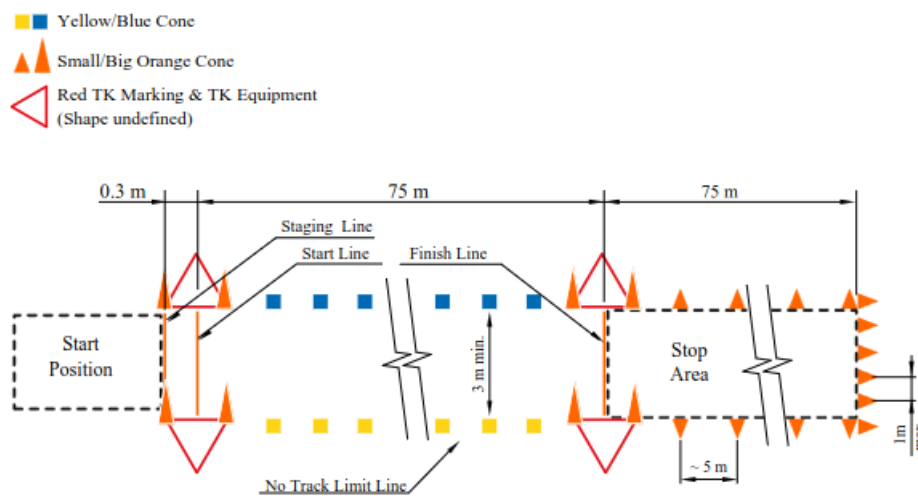


Figure 1.5: Acceleration track configuration.

according to the completion of the aforementioned laps and to the comparison between the elapsed time and the maximum allowed time.

Trackdrive: this event is only for the driverless cup and consists of a closed loop circuit, shorter than the ones described above, designed according to some constraints. Teams have to complete ten laps, counted by the vehicle itself: this means that there will be no explicit signals or indications on the track, implying that the autonomous system must be able to keep track of the completed laps. Points received in this test depend on the number of completed laps, and on the individual elapsed time.

During dynamic events, penalties can be assessed in many different cases. The most relevant for the self-driving category are knocking over cones, going outside the track with all four wheels and not re-entering within a certain time, or stopping in an unsafe manner, meaning outside of the specified area, for example.

1.3 Thesis objectives and outline

The aim of this thesis is to transition the work already completed on the simulator [6] to the real world. The simulator platform used for this project is EUFS, where the complete SLAM algorithm has been implemented and tested. Transitioning from a simulation environment to the real world can be challenging due to:

- many noise factors not being considered in the simulator. These noises affect both perception (changes in light exposure, unattended objects in the scene, changes in calibration due to car movements, etc.) and localization (different wheel slip on various surfaces, error margins on each sensor, asynchronous sensors);
- In the real world, each sensor (such as wheel encoders, steering sensors, cameras, IMUs, etc.) requires a software driver that ensures real-time performance;

Beginning with a literature review (in chapter 2) of scientific papers and articles related to autonomous racing scenarios, the state-of-the-art solutions employed by other Formula Student teams were identified and analyzed. Special attention was given to the various sensor configurations and the different approaches proposed for addressing the mapping problem.

Chapter 5 details the initialization phase of the project, which involves acquiring a comprehensive real-world dataset following the proper sensorization of an official Formula Student car. Real sensors were mounted on the prototype in a manner closely replicating the setup used in simulation. Additionally, a set of sensor drivers were developed to synchronize incoming data and ensure real-time storage of the received information.

As the front-end of Simultaneous Localization and Mapping, a perception module has been implemented to utilize images from a pair of stereo cameras. The visual cone detection pipeline employs a deep learning approach, using a fine-tuned version of YOLOv7 [7] to obtain bounding boxes around the landmarks in the stereo camera images. Preliminary qualitative results will be presented in Chapter 6.

In Chapter 7 we will focus on the final choice for the SLAM back-end system: a graph-based approach. This approach involves the mathematical optimization of spatial constraints linking vertices of a graph. A detailed explanation of this implementation is provided by introducing *g2o* as the framework for graph optimization, the anti-Ackerman steering system and its integration into the motion model, a nearest-neighbor data association algorithm, and the loop closure techniques used to tackle the difficulties of loop closure in a real-world racing environment. And finally, the results will be presented by comparing the real-world performance with the simulator environment performance. The simulator is considered to be a benchmark for the evaluation of real-world performance.

Chapter 2

Related work

The competitive nature of Formula Student makes it challenging to find scientific literature specific to this application, as most teams are reluctant to publish their work or share their research with competitors. Additionally, publicly available papers often describe earlier versions of the race cars used in previous seasons. Another effective way to become familiar with the work done in this field is to attend competitions and observe the cars in action on the tracks. Direct observation can provide valuable insights into the practical implementation of various technologies and strategies used by different teams. In addition, more precise and detailed information about the systems used in race cars can be found directly on the websites of the involved universities or teams. This is because most of the work in this field is not published as formal research.

Despite these challenges, analyzing available works can provide valuable insights into the fundamental algorithms and setups currently used by other participants. For new teams entering the competition, starting with simpler yet effective systems is likely the best approach.

In [6], Tonin from the University of Padova introduces a pipeline for SLAM in the EUFS simulator [8], utilizing two approaches in graph SLAM optimization: global optimization of the entire graph and incremental local optimization. In the first approach, the entire graph is optimized at once, while in the second, optimization occurs after a certain number of edges are added. The results presented in this work demonstrate a significant improvement in the accuracy of localization and mapping within the simulator environment while the optimization is done incrementally.

In [9] the team from Beijing university introduces a LiDAR-vision method for detecting the traffic cones. Their sensor positioning is interesting due to the

positioning of the LiDAR under the nose of the car, making the focus of the emitted rays to be on the cones. Also for navigation they use a combination of GPS and INS (Inertial Navigation System) which provides high-accuracy positioning critical for vehicle localization, especially during the second lap when the vehicle relies on pre-mapped trajectories. In addition, they use a complementary filter for coupling GPS-INS data with LIDAR odometry to enhance overall positional accuracy.

In [10], the KIT team utilizes a redundant sensor suite with three cameras and four LiDARs for robust environment perception. This multi-sensor approach helps mitigate the limitations of any single sensor and improves the overall reliability of the system. The paper highlights the presence of two independent pipelines for processing camera and LiDAR data. This redundancy ensures that the system can function even if one sensor encounters issues. In addition, a CNN-based approach has been used to detect bounding boxes of the cones. For the SLAM pipeline, they also use an Extended Kalman Filter, with (x, y, θ) as parameters. Data association is handled using the renowned Joint Compatibility Branch and Bound (JCBB) [11] algorithm.

In [12], the Austrian team TUW from Wien developed a prototype that does not use a LIDAR but instead relies on a planar laserscanner along with the other standard sensors like a camera, IMU, and GPS. Similar to other teams, their perception module employs a mixed approach to leverage both track images and laserscan points. The perceived data is then used as input for an Extended Kalman Filter implementation to construct the map of the track.

In [13], [14], and [15], the AMZ team from ETH Zurich presents *Gotthard* and *Pilatus*, their cars used in the 2018-2021 seasons. These papers show that the sensor setup remained largely consistent over the years, with the exception of an additional LIDAR added in 2019, placed on the main hoop alongside the cameras. Regarding the perception module, both systems integrate a camera-based cone detection pipeline with a LIDAR-based one. For the visual pipeline, both cars use a version of YOLO as the detector, while the LIDAR is also utilized to recognize color through intensity data analysis. In terms of the SLAM module, a significant change occurred from the 2018 car to the 2019 car: they switched from a particle filter algorithm, FastSLAM 2.0 [16], to graphSLAM [17].

Finally, the MIT Driverless team, as detailed in [10], uses a YOLO-based pipeline to detect cones but does not provide extensive information about the rest of their system.

Recent advancements in visual SLAM (V-SLAM) have shown promise in improving the performance and reliability of autonomous race cars. For instance, ORB-SLAM, a well-known V-SLAM algorithm, has been adapted for use in high-speed scenarios. ORB-SLAM utilizes oriented FAST and rotated BRIEF (ORB) features to achieve real-time performance, robustness, and accuracy in large-scale environments [18]. This algorithm’s ability to handle dynamic and changing environments makes it highly suitable for Formula Student applications.

Graph-based SLAM methods, such as g2o, have introduced efficient optimization techniques for large-scale SLAM problems. g2o (General Graph Optimization) provides a flexible framework for optimizing nonlinear error functions, which is crucial for accurate and scalable SLAM [19]. This method is particularly beneficial for maintaining high accuracy in extensive and complex racing tracks.

To gain a broader understanding of potential approaches worth further study, it can also be beneficial to consult the numerous theses written by university students about their prototypes. However, it’s important to remember that theses are not published or officially reviewed works, so the information they contain needs to be carefully verified.

The main contribution of this thesis is the development of a real-world simultaneous localization and mapping (SLAM) module based on graph optimization, while focusing on both the front-end (visual perception) and back-end (localization and mapping optimization), specifically designed for the *SG-05* RaceUp electric car.

Chapter 3

Theoretical background

This chapter explores the foundational concepts and methodologies underpinning the development of a robust autonomous localization and mapping system. The theoretical background is divided into two primary sections: Visual perception and SLAM (Simultaneous Localization and Mapping).

The visual perception section addresses the initial stage of the autonomous driving pipeline, which is crucial for understanding the vehicle's surroundings. We will first discuss the fundamental principles of how cameras work, introducing the pinhole camera model as the basis for various camera systems in computer vision. Then the key tasks in achieving a 3D representation of the environment, such as calibration and triangulation, are explained in detail. Calibration involves extracting intrinsic and extrinsic parameters of the cameras to ensure accurate image representation and depth estimation. Triangulation is then used to derive the 3D positions of objects from the stereo images, which is crucial for tasks like 3D reconstruction and obstacle detection.

The SLAM section explores the widely studied problem in robotics of enabling a system to navigate autonomously in an unknown environment. It begins by defining the core components of SLAM: the state vector, control vector, landmark location, and observation vector. These components are used to formulate the SLAM problem, which involves estimating the vehicle's position and the map of its surroundings simultaneously. We then explain two problems: the observation model and the motion model. The observation model translates sensor measurements into environmental data, while the motion model describes how the vehicle's state evolves over time. Together, these models update the vehicle's state and the world map as the vehicle moves and detects new landmarks. The discussion then focuses on graph-based SLAM, one of the three main approaches

to addressing the SLAM problem, alongside Kalman filters and particle filters. The mathematical formulation of graph-based SLAM is presented, illustrating how the problem can be represented by a graph where nodes correspond to the robot's poses and edges define constraints between these poses.

3.1 Visual Perception

In the domain of simultaneous localization and mapping (SLAM), two primary components are commonly distinguished: the front-end and the back-end. The front-end handles sensor data processing to determine the robot's present position and map environmental landmarks. More specifically, in our scenario, we focus on identifying spatial relations and associations among landmarks, particularly the cones on the track.

While humans use their senses while driving a car, autonomous robots rely on sensors for gathering information about the surrounding environment. In this project, the sensor used for sensing the environment to make a map of the track, are stereo cameras.

Before investigating the principle of stereo vision, it worth mentioning the theory behind how cameras work. A camera is a device containing an image sensor that converts light reflected by objects into current signals, ultimately producing an image. The basic camera model, known as the pinhole camera model, depicted in figure 3.1, represents the projection of light rays onto the sensor through a small aperture, mimicking the behavior of the human eye. This model forms the foundation for various camera systems, serving as a fundamental concept in the field of computer vision and imaging technology.

A stereo system, a common setup in computer vision and robotics, consists of two or more usually identical cameras rigidly mounted and capturing the same scene from different perspectives. This configuration enables the system to perceive depth information by exploiting the geometric principles of triangulation. By correlating corresponding points in the images captured by each camera, the system can calculate the disparity between them, which directly relates to the depth of the scene. This depth estimation capability is essential for tasks such as depth mapping, 3D reconstruction, obstacle detection, and environment perception in various applications, including autonomous driving, augmented reality, and robotics.

To achieve a 3D representation of landmarks in the surrounding environment,

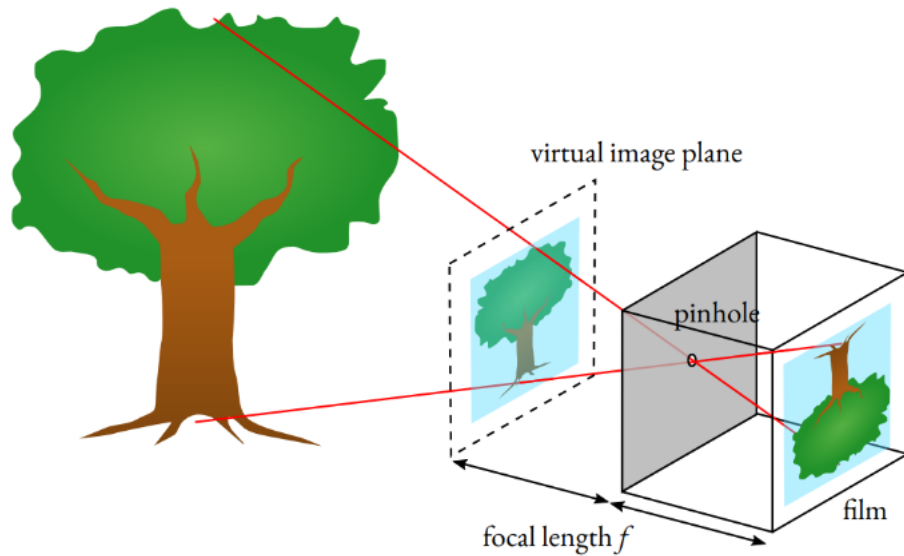


Figure 3.1: The pinhole camera model, with added virtual image plane.

two main tasks must be completed: calibration and triangulation.

The primary objective of calibration is to extract the intrinsic and extrinsic parameters of the cameras. However, it's crucial to conduct denoising and undistortion before proceeding with calibration. The intrinsic parameters include the focal length, optical center, and distortion coefficients, which are unique to each camera and necessary for accurate image rectification and 3D reconstruction. The extrinsic parameters define the spatial relationship between the cameras, including their relative position and orientation.

As the first step, we should conduct mono camera calibration for each camera to extract the intrinsic parameters. This involves capturing images of a known calibration pattern, such as a checkerboard, from various angles and distances. Specialized software then analyzes these images to compute the camera's intrinsic parameters, correcting for lens distortion and ensuring accurate image representation.

After mono camera calibration, we proceed with stereo camera calibration to determine the extrinsic parameters. This process involves capturing synchronized images from both cameras of the same calibration pattern and calculating the relative transformation between the cameras. This transformation is essential for accurate triangulation, as it allows the system to map points in one camera's image to the corresponding points in the other, enabling precise depth estimation and 3D representation of the scene.

Triangulation is a crucial technique used to derive the 3D position of an object

point from two or more images taken from different viewpoints. Once the cameras are calibrated, triangulation can be performed using the intrinsic and extrinsic parameters obtained during calibration. The steps for triangulation include:

- **Feature Matching:** Identify corresponding points in the images taken by the stereo cameras. This involves finding points in one image that match points in the other image, typically using feature detection and matching algorithms;
- **Compute the disparity,** which is the difference in the positions of the corresponding points in the two images. Disparity is inversely proportional to the depth of the point in the scene—the greater the disparity, the closer the object is to the cameras;
- **Use the disparity information and the known baseline distance** (the distance between the two cameras) to calculate the depth (z-coordinate) of the points. The depth can be calculated using the formula:

$$z = \frac{f \cdot B}{d} \quad (3.1)$$

where z is the depth, f is the focal length of the cameras, B is the baseline distance between the cameras, and d is the disparity.

- **3D Point Reconstruction:** Combine the depth information with the intrinsic parameters to reconstruct the 3D coordinates (x, y, z) of the points in the scene. This involves back-projecting the 2D points in the image plane to 3D space using the calculated depth.

3.2 Simultaneous Localization And Mapping (SLAM)

SLAM, which stands for simultaneous localization and mapping, is a widely studied problem in robotics. It is applicable whenever a system needs to navigate autonomously in an unknown environment. To achieve its goal, the robot must construct an accurate model of its surroundings and the trajectory it is following. Additionally, it must be capable of localizing itself within the created map.

To achieve these goals, the robot uses its sensors to observe the world and gather information about the scene, such as identifying landmarks. These landmarks are then used to create a map and serve as reference points for localization.

To formally define the problem [1], at each time instant t , we can define:

- \mathbf{x}_t : the state vector describing the current position and orientation of the vehicle;
- \mathbf{u}_t : the control vector to move from state \mathbf{x}_{t-1} to state \mathbf{x}_t ;
- \mathbf{m}_i : the vector describing the i -th landmark location;
- \mathbf{z}_t : the observation of the landmarks at time t .

According to this notation, the SLAM problem can be stated as

$$P(\mathbf{x}_t, \mathbf{m} | \mathbf{z}_{0:t}, \mathbf{u}_{0:t}) = P(\mathbf{x}_t, | \mathbf{z}_{0:t}, \mathbf{u}_{0:t}) P(\mathbf{m} | \mathbf{x}_{0:t}, \mathbf{z}_{0:t}), \quad (3.2)$$

where writing $0 : t$ denotes the series of the specified values until time t .

The probability distribution derived from the observations and control inputs up to time t , along with the vehicle's initial state, represents the joint posterior density of both the landmark locations and the vehicle's state at time t .

From Equation 3.2, it is evident that simultaneous localization and mapping (SLAM) comprises two sub-problems: the observation model and the motion model.

The former outlines how sensor measurements translate into the environment, defining the likelihood of a specific observation given the known locations of both the vehicle and landmarks:

$$P(\mathbf{z}_t, | \mathbf{x}_t, \mathbf{m}), \quad (3.3)$$

The latter represents the evolution of the system over time, describing how the previous state and control inputs influence the new state:

$$P(\mathbf{x}_t, | \mathbf{x}_{t-1}, \mathbf{u}_t), \quad (3.4)$$

In summary, the vehicle state and the world map undergo continuous updates as the vehicle moves: new landmarks, detected through sensors, are checked for association with existing ones on the map. If a positive match is found, the two landmarks are associated. Otherwise, the new landmark is added to the map.

The current state-of-the-art relies on three main approaches to address the SLAM problem: Kalman filters, particle filters, and graph-based methods. This thesis focuses solely on the third approach. Below, the mathematical formulation of graph-based SLAM will be formally presented, as detailed in [17].

The simultaneous localization and mapping problem can be represented by a graph, where nodes correspond to the robot's poses at various time points, and edges define constraints between these poses. By utilizing different sensors, environmental observations are collected and used to create these constraints.

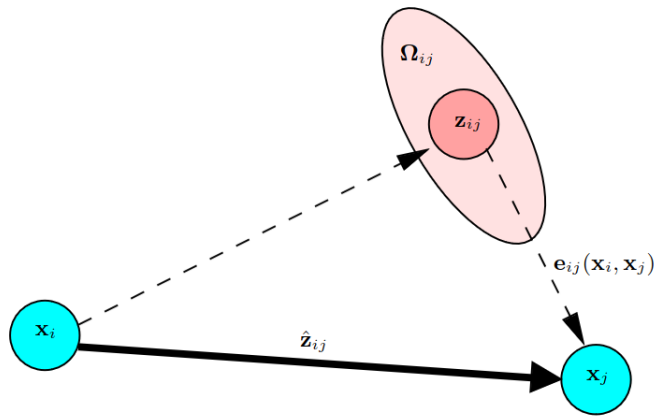


Figure 3.2: Mathematical configuration of the graph used to represent the SLAM problem.

A common graph configuration for the SLAM problem is depicted in figure 3.2. Each pair of vertices \mathbf{x}_i and \mathbf{x}_j is connected by an edge according to the measurement \mathbf{z}_{ij} . The expected measurement $\hat{\mathbf{z}}_{ij}$ is the prediction of the measurement \mathbf{z}_{ij} given a configuration for the nodes \mathbf{x}_i and \mathbf{x}_j . The ellipsoid around \mathbf{z}_{ij} in figure 3.2 is the information matrix Ω_{ij} , representing the uncertainty that we have on the measurement.

Finally, the error encoded in the edges will be defined as follows :

$$\mathbf{e}_{ij}(\mathbf{x}_i, \mathbf{x}_j) = \hat{\mathbf{z}}_{ij} - \mathbf{z}_{ij}, \quad (3.5)$$

In general, it represents the difference between the expected measurement and the actual measurement.

Defining \mathcal{C} as the set of pairs of indices for which an observation exists, the goal of the SLAM back-end optimizer is to minimize

$$F(\mathbf{x}) = \sum_{(ij) \in \mathcal{C}} \mathbf{e}_{ij}^T \Omega_{ij} \mathbf{e}_{ij} \quad (3.6)$$

in order to find the optimal configuration of the nodes \mathbf{x}^* such as

$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmin}} F(\mathbf{x}) \quad (3.7)$$

Chapter 4

Test environment

In the process of developing any complex system, in our case a self-driving car, there should be platforms to allow safe and repeatable testing. In this chapter, we will discuss two main tools used for testing in this project: **Virtual simulator** and **Test mule**.

When developing a Formula Student Driverless car, it is imperative to first rigorously test all systems and components within a dedicated simulator designed specifically for this purpose. This approach offers numerous benefits that are crucial to the success and safety of the project.

Using a simulator allows for comprehensive testing of the car's algorithms and systems in a controlled and repeatable environment. This ensures that all scenarios, including edge cases and potential failure points, can be explored without the risk of damaging the vehicle or endangering the team. The simulator replicates real-world conditions, enabling developers to fine-tune the car's responses to various dynamic situations, such as different track layouts and obstacles.

A crucial advantage of simulation is the availability of *ground truth data*, which is essential for proper benchmarking and algorithm evaluation. Ground truth data provides an accurate reference that can be used to measure the performance of various algorithms, such as perception, localization, and mapping. This allows us to precisely evaluate the accuracy and reliability of the systems, ensuring that they meet the necessary performance standards.

Another vital feature of simulation is sensor simulation. This allows for realistic modeling and calibration of the car's sensors, such as LiDAR and cameras. By simulating these sensors, developers can test how the car perceives its environment and ensure that the sensor data is accurately interpreted by the car's systems. Sensor simulation also helps in calibrating the sensors, which is essential

for maintaining the precision and reliability of the car's perception system.

Simulation also offers significant benefits in terms of scalability and rapid testing. Changes to the car's design or software can be quickly implemented and tested in the simulator, allowing for rapid prototyping and iteration. This accelerates the development process, enabling the team to test multiple configurations and algorithms in a short period. The scalability of simulation means that extensive testing can be conducted without the limitations and costs associated with real-world testing.

In addition to simulation, the use of a test mule is a vital component in the development of a Formula Student Driverless car. A test mule is a simplified version of the vehicle that is not intended for competition and does not need to adhere to the stringent rules and regulations governing the final competition car. This approach offers several critical advantages that enhance the development process.

Firstly, having a test mule allows for practical, real-world testing of the car's systems and components. While simulations provide a controlled environment for initial testing, real-world conditions can introduce variables and challenges that are difficult to replicate virtually. A test mule bridges this gap by allowing developers to observe how the car's systems perform under actual operating conditions, thereby providing invaluable insights into the car's behavior and performance.

The simplified nature of the test mule makes it an ideal platform for iterative testing and development. Without the constraints of competition rules, the test mule can be modified and adjusted more freely, enabling rapid prototyping and experimentation.

Moreover, the use of a test mule helps to identify and resolve potential issues early in the development process. By testing components such as sensors, actuators, and control systems on the test mule, developers can detect and address problems before they are integrated into the more complex and expensive competition vehicle. This proactive approach reduces the risk of encountering critical failures during the later stages of development or during competition.

Additionally, a test mule provides an invaluable opportunity for the team to practice and refine their testing and debugging procedures. This includes honing their skills in data collection, analysis, and troubleshooting, which are essential for diagnosing and solving issues efficiently. The experience gained through working with the test mule can significantly improve the team's preparedness and confidence when dealing with the final competition car.

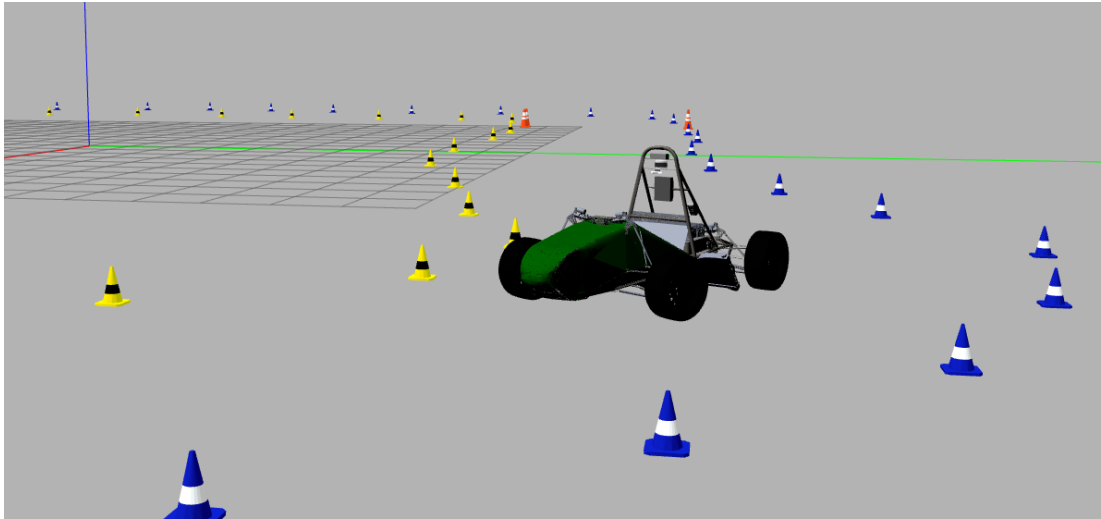


Figure 4.1: An example of simulated car and track in EUFS simulator.

4.1 EUFSSIM

As discussed earlier in section 1.2, Formula Student competitions adhere to a specific structure where the track layout consists of colored cones arranged according to the rules. For this project, we chose the EUFS simulator [8], developed by the Edinburgh University Formula Student team, among several available simulators tailored for Formula Student purposes.

The EUFS simulator, or `eufs_sim`, is an advanced simulation tool designed to aid Formula Student driverless teams in developing and testing their autonomous vehicle software. Utilizing the Gazebo simulator and ROS, `eufs_sim` allows for comprehensive testing on preset, rule compliant tracks as well as randomly generated tracks to simulate the dynamic events encountered in real competitions. The simulator is highly configurable, enabling users to select various vehicle models, weather conditions, sensor setups, and command modes, which is essential for realistic and robust testing of autonomous systems. In figure 4.1, it can be seen an example of a simulated track, with a custom car model inserted.

Another outstanding characteristic of this simulator is its compliance with 2020 rules of competition. This simulator also considers the car states as shown in 4.2. This scheme defines the conditions for switching between different car states. For instance, if the Emergency Brake System is activated, the Autonomous System enters the emergency state, deactivating both the Ready To Drive and the Tractive System.

Two key customization features of the simulator that were particularly useful for our project are track generation and the ability to customize the car model. As

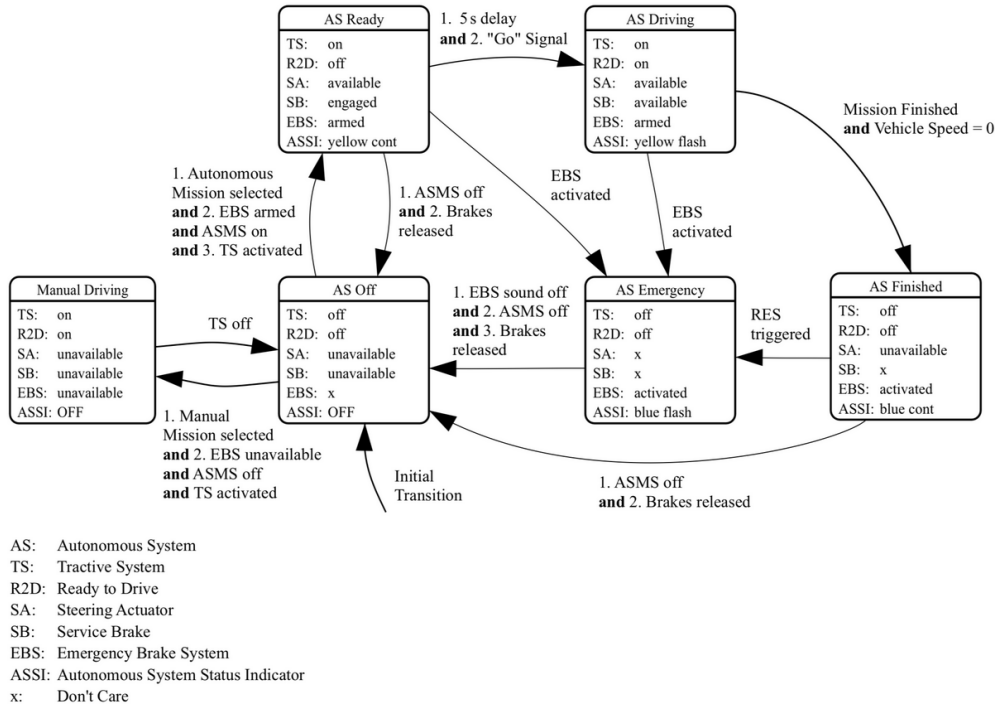


Figure 4.2: Scheme of the states used in the EUFS simulator.

we will discuss in chapter 5, we use a specific layout for our real data acquisition. This simulator allows us to replicate the same layout in the simulation environment. Additionally, it is possible to customize your vehicle by incorporating both the CAD model of the car and its kinematic model.

4.2 Test mule

After the first developing phase has been successfully tested in the simulator, the next step in the development pipeline is to proceed with the test mule mentioned earlier. Having a test mule is crucial before proceeding with the actual racing car. It allows for practical, real-world testing and experimentation, enabling quick modifications and early issue resolution without risking damage to the primary car.

The selected platform for the test mule is one of the first RaceUp racing cars, MG-03 depicted in figure 4.3, which participated in various competitions in Germany and Italy in the year 2007.

The powertrain for this car was initially a combustion engine, but to align the car's control system more closely with the actual racing car, it was converted to an electric drivetrain.



Figure 4.3: RaceUp MG03, modified to be the test mule

Additionally, the entire braking system was redesigned to incorporate an actuated braking and emergency brake system. The new braking system includes a linear actuator connected to an air valve that releases air pressure from an air tank when a braking command is received.

Moreover, the steering system was upgraded by adding an actuator, a potentiometer to measure the steering angle, and an Arduino to control the actuator for achieving the desired angle. And finally, the wheels are equipped with encoders for odometry estimation.

Regarding the sensorization of the test mule, a pair of cameras have been mounted on a rigid support, as shown in Figure 4.4, and are used as a stereo pair.

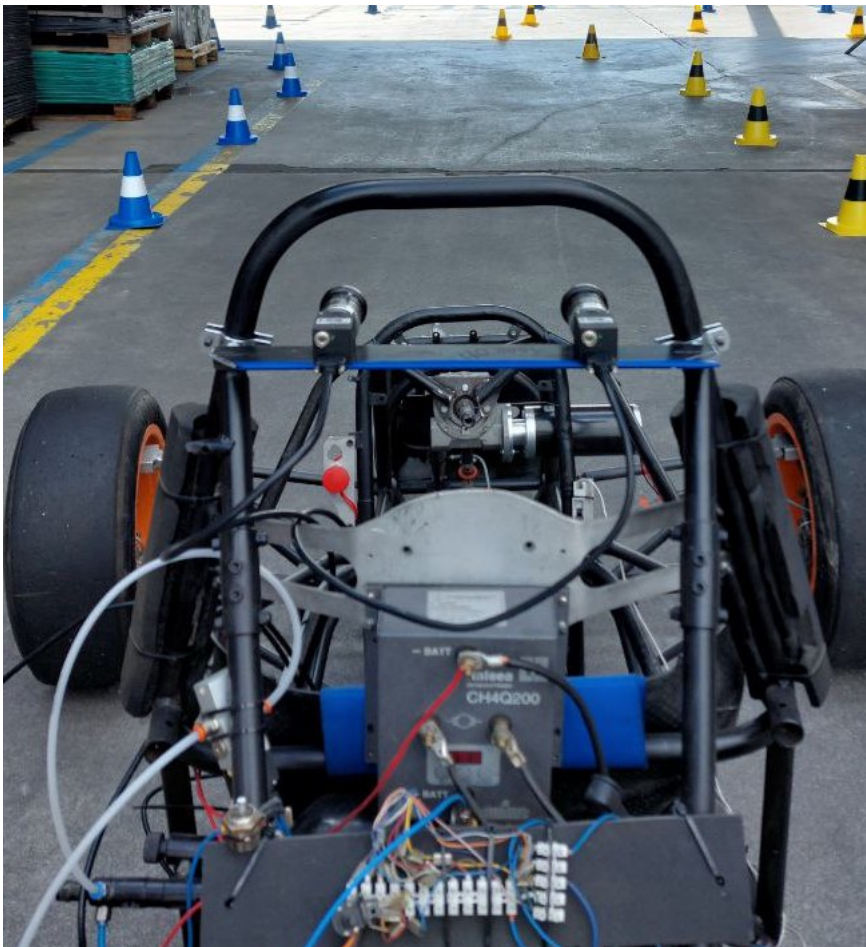


Figure 4.4: Flir cameras mounted on test mule

Chapter 5

Real data acquisition

As the main goal of this thesis is to transfer all the work done in the virtual simulator to a real scenario, we need to acquire real data. Testing in the real world and running the car around a physical circuit requires teamwork and collaboration from various engineers, including mechanics, electricians, powertrain technicians, and safety personnel. All of this has been made possible thanks to the efforts of my teammates at the RaceUp Electric team [2].

The car used for this data acquisition was SG-e 05 (fig. 5.1) electric single-seater.



Figure 5.1: Vehicle used for real data acquisition: SG-e 05 RaceUp electric.

In this case, we did not use a test mule, ensuring that the dataset is as realistic as possible for the actual competition car. For this reason, during the sensor installation on the car, it was not possible to make holes in the monocoque or change the car’s setup. The sensor supports had to be easily assembled and disassembled without causing any damage to the car.

In the following sections, we will first discuss the structure of the car and the sensorization procedure. Next, we will provide information about the dataset itself. Finally, after noticing the need for an updated sensor suite through testing, we will present the new sensor setup used to replace the previous one.

5.1 About the car

SG-e 05 car features four electric motors and each wheel is equipped with an encoder. The data from these encoders is used to calculate the speed of the car, by taking into account the radius of the wheels and the revolutions per second. The steering system of the car is also equipped with a steering angle sensor. Based on the readings from this sensor and understanding of it’s kinematics (an anti-Ackermann model in this case), we can calculate the angle of the front wheels at each moment.

All components of the car communicate with the Central Processing Unit (CPU) through CAN (Controller Area Network) communication. The CPU is responsible for processing the readings from the sensors, that will be then used for dataset acquisition. To access this data from the computing setup, we need to use a CAN Interface, which in our case we used Kvaser BlackBird v2 (fig 5.2). In addition to this device, a ROS node has been implemented to read and parse data from Kvaser, following the structure of the CAN messages.

5.2 Car sensorization

The first consideration when building an autonomous car, is to decide what sensors to use. The choices made were strongly constrained by the availability of the budget. Initially, sensors were borrowed from other projects. As the budget increased later on, the sensors were upgraded to higher quality and more robust versions. Detailed explanations of these upgrades are provided at the end of this chapter.

For top teams with substantial budgets and years of development experience,



Figure 5.2: Kvaser BlackBird v2

critical factors in selecting and positioning sensors in the car are aerodynamics and weight distribution or reduction. Figure 5.3 shows the car sensitization from one of the top teams in the category, KA-Racing, which integrated only a LiDAR on top of the nose of the car.



Figure 5.3: KIT24, KA-Racing

Here it is presented the list of the sensors used for the data acquisition campaign (fig 5.4):

- Velodyne VLP16 16-channels LIDAR
- Bumblebee2 RGB stereo camera

(a) Velodyne VLP16 LIDAR



(b) Bumblebee2 stereo camera



(c) XSens MTi IMU



(d) UBlox M8P GPS module

Figure 5.4: The complete set of sensors installed in the car.

- XSens MTi Inertial Measurements Unit (IMU)
- two M8P RTK-GPS modules from UBlox

In order to mount the sensors on the car, it is needed to design the support structure to securely integrate them on the car. Three main considerations should be taken into account while designing the support:

- Official Formula Student Regulations: These rules include regulations that must be followed in the car design, such as the requirement for an external envelope that must enclose the vehicle and all its components;
- Mechanical and Technical Constraints: The shape and design of the cars are already fixed and should not be altered. For instance, no holes can be made in the monocoque, and the shape of the main hoop must not be changed;
- Maximization of Effectiveness: The setup should be designed to maximize the performance of each sensor such as cameras, LiDAR, and IMU.

Considering the points mentioned above, the sensor should be placed within the car frame, should not obstruct the driver's sight, and should not be mounted on weak components. Additionally, sensor vibrations should be minimized, and the field of view for LiDAR and stereo cameras should be maximized. The optimal position for mounting the sensors is selected to be on the main hoop of the car, above the driver's head as shown in figure 5.5.

Based on this positioning, the support shown in Figure 5.6 has been designed. The plate for the stereo camera and IMU has been designed to be parallel to the ground, while the LiDAR is angled 5 degrees downward to maximize its field of view for detecting cones.

5.3 Dataset

To collect the data with this setup, a track has been made according to the layout shown in figure 5.7

The dataset has been collected using ROS Bags. Three bags have been recorded, with each bag containing data from the track being traced three laps.

Due to the substantial amount of information to record and the high dynamics of the situation, the primary focus is on real-time storage requirements.



Figure 5.5: Sensors support mounted on the main hoop

Minimizing the delay between data production and storage is crucial to prevent data loss during acquisition.

To ensure immediate data retrieval, every ROS node handling a sensor had its scheduler priority set to 99, indicating real-time processing. For efficient storage without any slowdown, an empty data folder was mounted in the Random Access Memory (RAM) and used as temporary storage. Additionally, to further reduce the delay in data saving, camera images were saved in greyscale instead of RGB.

5.4 Updated setup

As the final part of this chapter, a new sensor setup and embedded computing system have been designed and built for future improvements. This new setup is based on the experiences gained with the previous setup, mentioned in section 5.3.

As for the computing system, a custom build embedded computer has been

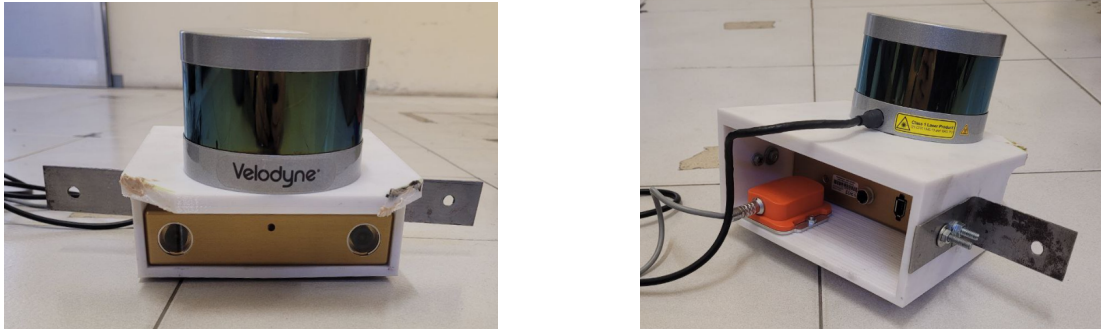


Figure 5.6: The support designed for mounting the LiDAR, stereo camera, and IMU on the main hoop.



Figure 5.7: Track layout

developed by us. This configuration includes:

- CPU: 12 core AMD Ryzen 9
- GPU: GeForce RTX 4060
- RAM: 96GB DDR5
- Operating System: Linux Ubuntu 20.04

The final assembly of this system is presented in figure 5.8.

In the context of sensor setup, we were not satisfied with the performance of the Velodyne VLP16 16-channel LiDAR (due to its low resolution) and the Bumblebee2 RGB stereo camera (due to its low-quality detection at long ranges). As a result, the LiDAR was replaced with an Ouster OS1 64-channel model, and a custom stereo camera was built by pairing two FLIR Blackfly S mono cameras.



Figure 5.8: Embedded computing system

All these sensors were assembled on a rigid support (fig 5.9) and mounted on the main hoop (fig 5.10), as before, because the results from the previous sensor positioning were satisfactory.

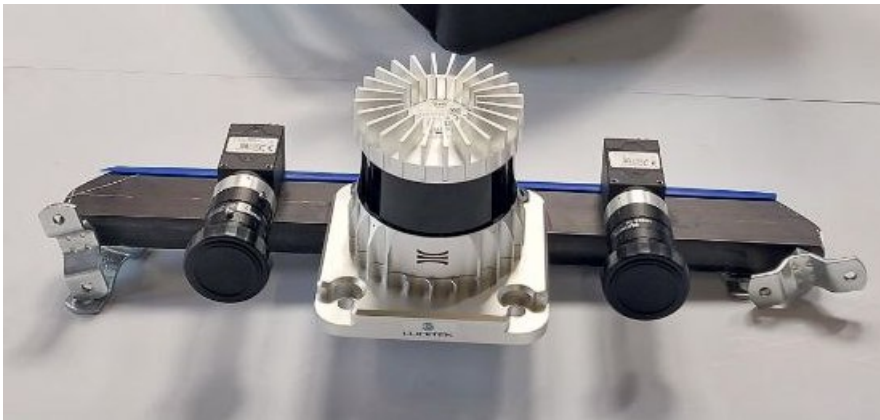


Figure 5.9: New sensors mounted on a rigid support

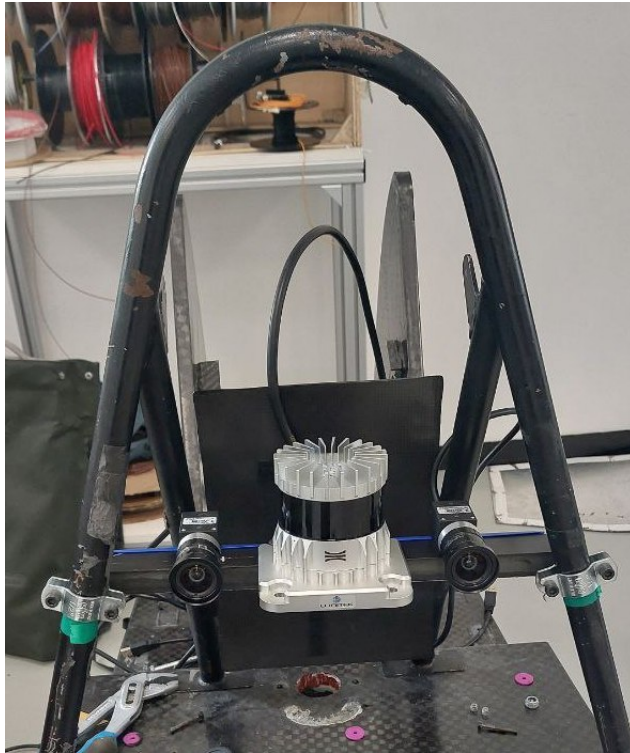


Figure 5.10: Positioning new sensors on the main hoop

Chapter 6

Front-end

The front-end in SLAM (Simultaneous Localization and Mapping) is responsible for processing raw sensor data to extract meaningful information that can be used for localization and mapping. It is a crucial component because it directly influences the quality and reliability of the features and measurements fed into the SLAM system.

In the context of Formula Student competitions, most teams use a combination of LiDAR and stereo cameras as their perception sensors. In this thesis, we focused on developing a perception module using stereo cameras due to the high cost and limited accessibility of LiDAR. The setup of the stereo camera has been discussed in details in chapter 5.3.

In the first part of this chapter, we will focus on image-based cone detection using the well-known YOLO object detector. Next, we will discuss the stereo matching module and introduce the experiments and results from this section. We will then address the limitations of the sensor and the approaches to solve them.

6.1 Deep Vision-based cone detection

As the initial step in the pipeline, the system processes images captured by the cameras. In this context, a fine-tuned version of YOLOv7 [7]. YOLO is a cutting-edge real-time object detection algorithm known for its remarkable speed and accuracy. This version of YOLO is trained entirely on the MS COCO dataset without relying on pre-trained weights, allowing it to learn from a diverse set of object categories and scenarios.

In order to use the YOLOv7 model in the context of Formula Student com-

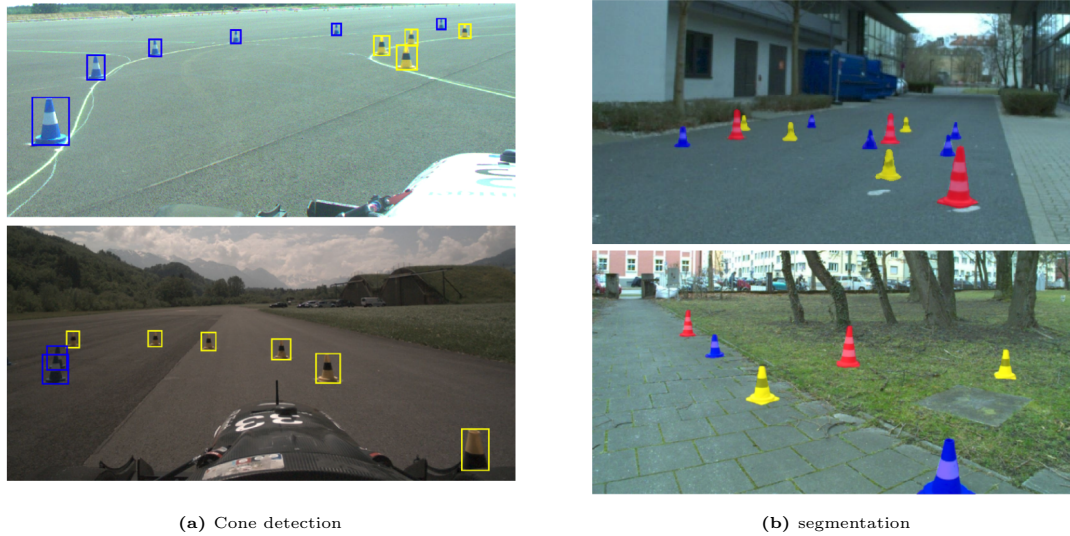
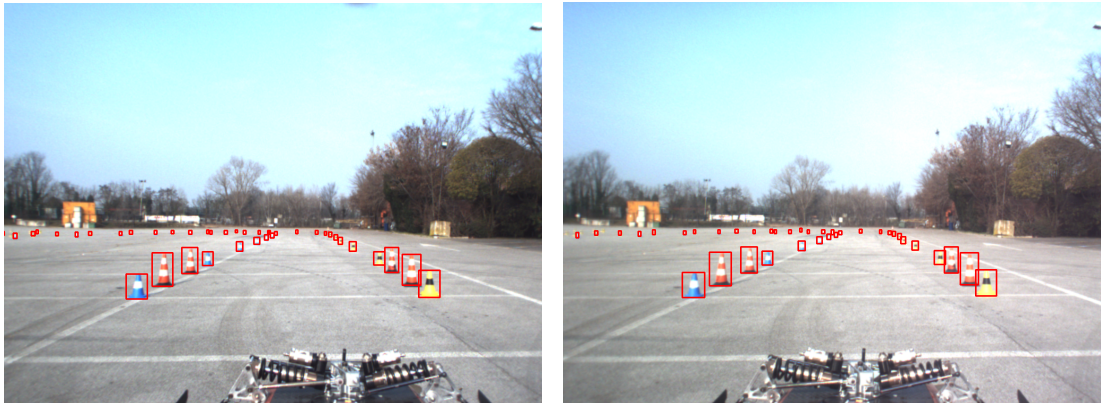


Figure 6.1: Example of data from FSOCO dataset

petition, we fine-tuned the general-purpose model with the FSOCO dataset [20]. The contributors to this dataset include several teams and the data includes images of real racing scenes, framed during driverless events or test sessions. Examples of this dataset are shown in figure 6.1. Annotations involve both bounding boxes and instance segmentation, provided for 11572 and 1517 images, respectively. The reasons for choosing this dataset include:

- it is a free publicly available dataset;
- it contains both real and simulated dataset, that are in the domain in which our detector needs to operate;
- the dataset is fully annotated
- images from different scenarios with different cameras from many different teams;
- they include different lighting conditions which is important as we don't know the weather condition during the competition.

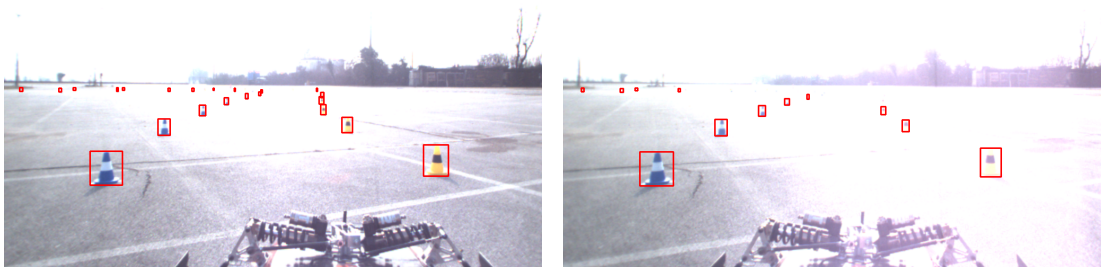
To qualitatively evaluate the performance of the trained model, we used a test set acquired by our setup (which has been explained in chapter 5). This test set includes challenging conditions such as direct sunlight, requiring the model to be highly robust for accurate detection. Our results show promising performance across all scenarios. The results are shown in figures 6.2 and 6.3 and it can be



(a) Left image.

(b) Right image.

Figure 6.2: Fine-tuned YOLO v7 performance in a normal scenario.



(a) Left image.

(b) Right image.

Figure 6.3: Fine-tuned YOLO v7 performance while there is direct sunlight.

seen that the detection module detects all the cones even in a scenario when there is over exposure of sunlight.

6.2 Stereo matching

Stereo matching is the process of finding correspondences between pixels in a pair of stereo images to estimate depth information. This process is fundamental in stereo vision systems, where two cameras capture images from slightly different viewpoints, mimicking human binocular vision. The goal of stereo matching is to determine the disparity (difference in pixel positions) between corresponding points in the left and right images, which can then be used to compute the depth of objects in the scene.

In this techniques, 3 steps should be performed:

- Rectification: Before matching, stereo images are rectified to align them



Figure 6.4: Green dots are centers estimated in the left image and red dots are centers estimated in the right image.

horizontally. This ensures that corresponding points lie on the same epipolar line, simplifying the matching process;

- Disparity calculation: Disparity is the horizontal shift between corresponding points in the left and right images. The greater the disparity, the closer the object is to the cameras;
- Depth calculation: Depth is inversely proportional to disparity. It can be calculated using the camera parameters using the formula: $z = (f * b)/d$ where f is the focal length and b is the baseline of the camera and d is the disparity.

In order to perform the stereo matching, we need to detect some common features in the left and right images. In our case, we should consider the cones as feature. We implemented two approaches to address this. In the first approach, only the centers of the bounding boxes are considered for matching, as shown in figure 6.4. At the first glance, this approach seems easy to implement, robust to underexposure and overexposure problems, and computationally fast. However, this approach has a fundamental issue: the center of the bounding box in the left image does not necessarily represent the same part of the cone in the right image. In other words, it can happen that the algorithm matches two points in the left and right image that are different.

To address the previous problem, a segmentation component has been added to the module to extract the features accurately. Using this segmentation, we consider the top point of each cone as the feature to be matched with the other



Figure 6.5: Top of each cone considered to be the features to be matched. Green dots are tips in the left image and red dots are tips in the right image.

image. Figure 6.5 shows the performance of our segmentation on returning the top point of each cone.

At this stage we will have the 2-dimensional image coordinates and the disparity related to each cone. Using a perspective transformation based on the calibration data we can project each point from the image frame into 3-dimensional space in the camera frame.

During our initial quantitative evaluation, a limitation of our sensor became apparent: the disparity we obtain is not fine-grained enough. As a result, even a small displacement of one pixel between the two cones leads to an error of almost half a meter in the final positioning. The way to address this issue is to use a stereo camera that is better suited for this type of task, as it has been discussed in chapter 5.3.

Chapter 7

Back-end building blocks

In the field of robot navigation, particularly in our Formula Student driverless scenario, having an accurate map and precise localization within that map is essential. When there is no prior knowledge of the environment and no external reference systems like GPS, SLAM (Simultaneous Localization and Mapping) has become the preferred solution for addressing the challenges of localization and mapping. As previously mentioned, SLAM is divided into two sub-problems: front-end and back-end. In Chapter 3, we discussed the front-end, and in this chapter, our focus will shift to the back-end.

The back-end part of SLAM involves refining the robot's state using sensor information and creating a map that accurately represents the environment. As elaborated further in this chapter, one approach to formulate this problem is through a graph structure, where nodes represent the pose estimates or landmark locations and edges define the constraints between these poses and landmarks. This method is known as **graph-based SLAM** [17].

In this chapter we first start by introducing the SLAM problem as a graph structure and we talk about how the representation of the problem is handled in a graph. Then we present the algorithm implementation by mentioning the framework used, the optimization techniques, the motion models, and data association. And at the final part of the chapter we demonstrate the results obtained by this implementation.

7.1 Graph-Based SLAM

Graph-based Simultaneous Localization and Mapping (SLAM) systems rely on optimizable structures that encode the relationships between various state vari-

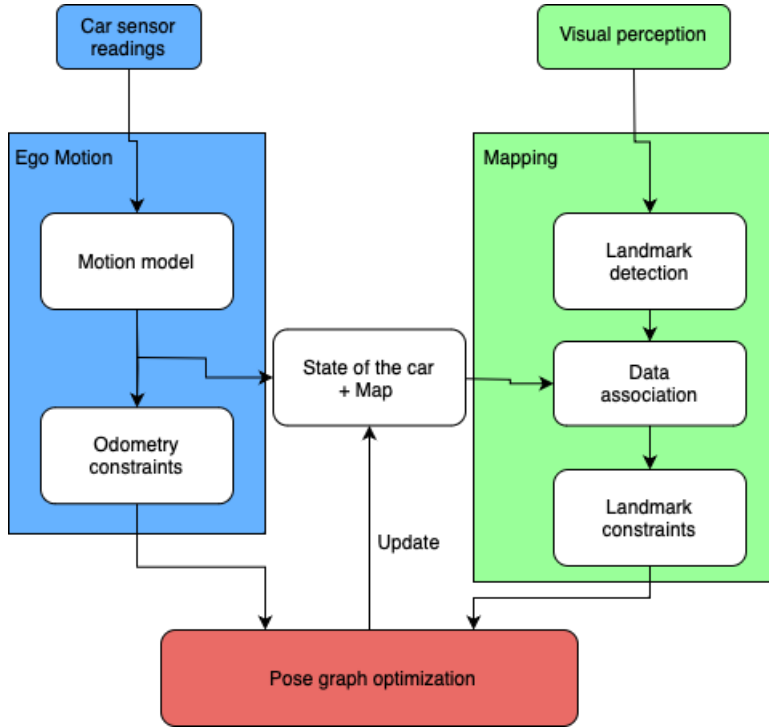


Figure 7.1: Detailed GraphSLAM architecture used to fuse landmark observations with car positions into a coherent map and pose within the map.

ables and measurements. The construction of these graphs is crucial for the accurate estimation of both the vehicle’s trajectory and the environment’s map.

In Figure 7.1, we illustrate a general scheme of how graph-based SLAM operates. In the ego-motion estimation part, sensor readings from wheel encoders and the steering sensor are provided to the motion model where the data is processed to find the state of the car and the odometry constraints. The resulting output from the motion model is then used to update the current state of the car. Simultaneously, in the mapping part, observations are given to the cone detection module (see Chapter 6) and then passed to the data association algorithm, which is responsible for matching new observations with previous ones. Next, based on both odometry and landmark constraints, the pose graph will be optimized and the localization and mapping will be updated.

This mapping and localization is formulated as a graph structure by nodes representing the vehicle poses (x, y, θ) , where θ is the orientation, and environmental landmarks (x, y) coordinates. Edges represent the transformation between two consecutive car poses x_i and x_j , as well as the transformation between each pose and the observed landmarks from that state. This graph structure, known as a *factor graph*, is depicted in Figure 7.2.

In more detail, the steps for processing the incoming data are as follows:

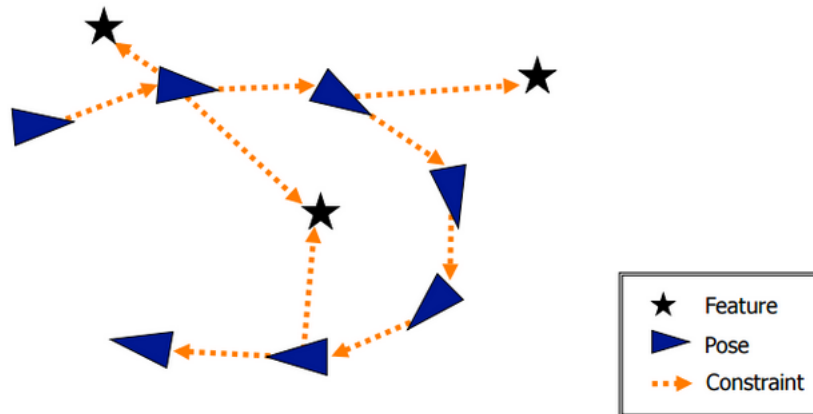


Figure 7.2: The factor graph structure used to formalize the back-end problem.

- **Odometry Information Processing:** a new pose node is created and inserted into the factor graph whenever a new car state is computed. The car state is characterized by its position and orientation at a specific time instant. Then, edges connecting two consecutive pose vertices are added to the graph. These edges are characterized by their relative spatial transformation.
- **Landmark Insertion:** at the beginning of the process, when there are no edges or nodes in the graph, all observed cones are directly inserted into the graph, and an edge is added between the initial pose and these observations. In the subsequent steps, for each new cone observation, the data association algorithm is called to determine whether the cone corresponds to an existing cone or not. Based on the output of the data association, an edge is added between the new observation and the current pose, or between the current pose and the associated existing cone.

In the transition from a simulated environment to the real world, we encountered several challenges. One significant challenge is dealing with noisy data from the front-end. As depicted in Figure 7.1, observations from the front-end are crucial because several other parts of the graph-based SLAM depend on them. Initially, in the simulator, ground truth data was used to populate the graph, ensuring that the system worked correctly in an ideal scenario. However, in the real world, uncertainties and noise should be managed by limiting the range of cone detection and using covariance matrices to handle uncertainties. Another problem we encountered was synchronizing information from the perception pipeline with the car's odometry data to estimate the vehicle's current pose and the po-

sitions of the observed cones. To ensure efficiency, incoming data is subject to discretizing continuous time signals into fixed time intervals or better known as time quantization. By aligning sensor readings to these fixed time intervals, it becomes easier to integrate and process the data in a consistent manner.

One other point to consider during the construction of the factor graph is the separation between data collection and graph construction:

- **Data Collection:** this step involves gathering sensor measurements of the perceived environment and storing them for later processing;
- **Graph Construction:** this involves converting the collected data into a mathematical formulation to build an optimizable structure representing the SLAM problem.

A simple approach is employed to manage data collection during the optimization phase, which alters the graph structure. Any incoming data received during this phase is held in a pending state until the optimization process is complete, thus avoiding concurrency issues that could result from simultaneous modifications to the graph structure.

7.2 Algorithm implementation

The estimated trajectory of a vehicle in SLAM often diverges from its actual path due to sensor noise and accumulated drift over time, as shown in figure 7.3. These discrepancies result in errors in the estimated positions of both the vehicle and the landmarks. Optimization in SLAM aims to minimize these estimation errors by establishing mathematical constraints between graph vertices and leveraging motion and measurement models. In this section we will talk about the implementation of this optimization.

Before delving into the specifics of the implemented SLAM algorithm, it's essential to introduce the library used for factor graph optimization. This will provide a foundational understanding of the tools and techniques employed in the optimization process.

7.2.1 g^2o framework

g^2o [19], is an open-source C++ framework for optimizing graph-based nonlinear error functions. g^2o has been designed to be easily extensible to a wide

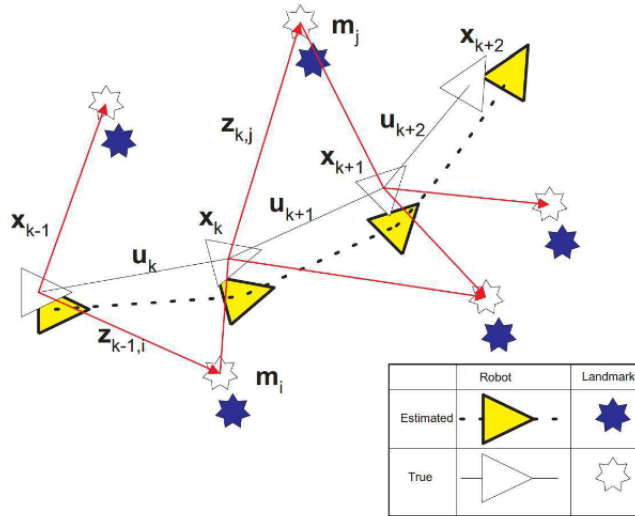


Figure 7.3: Why we need to optimize: landmarks being observed at different positions along the robot's trajectory [21].

range of problems and a new problem typically can be specified in a few lines of code. The current implementation provides solutions to several variants of SLAM and bundle adjustment. A wide range of problems in robotics as well as in computer-vision involve the minimization of a non-linear error function that can be represented as a graph. The overall goal in these problems is to find the configuration of parameters or state variables that maximally explain a set of measurements affected by Gaussian noise.

In this thesis, g^2o is employed to create, manage, and optimize the factor graph, which includes the 2D positions of cones and the 2D poses of the car.

$g2o$ offers different data types for the purpose of SLAM. The $g2o$ data types used for graph construction in this thesis include `g2o::VertexPointXY` which is used for the landmark nodes, `g2o::VertexSE2` which is use for the car 2D pose nodes, `g2o::EdgeSE2` which is used for odometry edges, and `g2o::EdgeSE2PointXY` which is used for observation edges.

7.2.2 Graph optimization

A graph in graph-based SLAM needs optimization to ensure that the estimated positions of the robot and the map features (landmarks) are as accurate and consistent with the sensor data as possible. Graph optimization may be necessary for various reasons, including:

- Measurement Uncertainty: Sensor measurements used to build the graph

(e.g., odometry, laser scans, camera images) are inherently noisy. Without optimization, the initial estimates of the robot's path and the map may be inaccurate due to this noise;

- **Accumulative Errors:** As the robot moves and continuously updates its position based on odometry or other relative measurements, small errors accumulate over time. This can lead to significant discrepancies between the estimated and actual positions if not corrected;
- **Loop Closure:** When the robot revisits a previously mapped area (a loop closure event), the new measurements can be used to correct past trajectory and map errors. However, integrating these new measurements into the existing map requires adjusting the previous estimates to ensure global consistency;
- **Nonlinear Relations:** The relationships between robot poses and landmarks are nonlinear. Optimization methods are designed to handle these nonlinearities effectively, finding the best estimates that minimize the overall error;
- **Global Consistency:** Ensuring that all the poses and landmarks in the map are consistent with each other and with all the measurements taken throughout the robot's journey requires a global optimization approach. This helps in reducing discrepancies across different parts of the map.

Two different strategies for graph optimization have been studied and tested in the simulator environment, as explored by Alessandra Tonin[6]. In this work, these strategies are adapted with minimal changes for real-world scenarios.

- **Global optimization:** In the global optimization approach, the optimization function is called only once, after completing one lap. This occurs when the initial indicator cones (big orange cones, as explained in section 1.2) are detected by the front-end system. The entire graph is optimized at this point, ensuring that all pose and landmark nodes are adjusted simultaneously to minimize overall estimation error;
- **Incremental optimization:** The incremental optimization approach performs local optimization at regular intervals, specifically after a certain number of pose nodes have been inserted into the graph. This technique focuses on optimizing a defined neighborhood of nodes, adjusting only the most recent

part of the trajectory and corresponding landmarks. After completing a lap, a global optimization is performed, similar to the global-only approach.

The Levenberg-Marquardt algorithm has been selected for both global and local optimizations. Also a block solver is coupled with the optimizer to resolve the linearized system. This choice is quite common for addressing non-linear optimization problems as it strikes a balance between the Gauss-Newton method, which is faster but less robust, and the gradient descent procedure, known for being more robust but slower.

When transitioning from a simulated environment to real-world application, the performance of the SLAM system exhibits notable differences due to two primary factors:

- **Increased Noise:** Real-world scenarios introduce significant noise in both landmark detection and odometry estimation. Sensor readings are subject to various inaccuracies and disturbances that are not present in a controlled simulator environment;
- **Computational Constraints:** In the real world, computational resources are limited. The detection module, which includes cone detection using YOLO and transforming the detections into the world frame, requires approximately 230 milliseconds to process each image, in our current computation system. This processing time introduces latency and affects the overall performance of the SLAM system.

Additionally, due to the inaccurate depth estimation for distant cones, many of the detections are discarded. As a consequence, the number of landmark nodes added to the graph at each timestamp is considerably fewer than the one of simulator so there would be fewer edges (constraints) between each position node and landmark nodes. As we can expect in this situation, local optimization will not perform the same way as the simulator and it would be more sensitive to noise. Therefore, this thesis will primarily focus on global optimization, with consideration for incremental optimization for future improvements.

7.2.3 Anti-Ackerman steering and Motion model

Ackermann steering geometry is designed to ensure that all wheels track correctly when a vehicle turns, meaning the inside wheel turns more sharply than the outside wheel during a turn to reduce tire scrubbing and improve handling stability.

However, in some racing setups or specialized vehicles, it is more common to use "anti-Ackerman" steering. We also utilized this steering system in the RaceUp SG-05 vehicle for our real-world experiments. Anti-Ackerman setups adjust the steering geometry so that the outside wheel turns more sharply than the inside wheel during a turn. This can be used to achieve specific handling characteristics, such as improved cornering grip or stability under certain conditions.

In Ackerman and anti-Ackerman geometry, the relationship between the steering angles δ_{inside} and $\delta_{outside}$ (steering angles of the inside and outside wheels, respectively) is given by:

$$\tan(\delta_{inside}) = \frac{L}{R_{inside}} \quad (7.1)$$

$$\tan(\delta_{outside}) = \frac{L}{R_{outside}} \quad (7.2)$$

where:

- L : Wheelbase of the vehicle (distance between the front and rear axles);
- R_{inside} : Radius of the turn for the inside wheel;
- $R_{outside}$: Radius of the turn for the outside wheel.

In Ackerman $R_{inside} < R_{outside}$ while for the anti-Ackerman $R_{inside} > R_{outside}$ meaning that in the anti-Ackerman the outside wheel turns more sharply than the inside wheel.

In this thesis the system has been specifically designed for RaceUp SG-05 racing car. The data coming from the steering sensor is transformed into final wheel angle based on the following formula:

$$R_{car} = 1163 * 0.01273 * \delta_{steering} * \pi / 18 \quad (7.3)$$

where:

- R_{car} is the turning radius of the car;
- $\delta_{steering}$ is the steering wheel angle from sensor readings.

Now that we have established the steering system model, we can integrate it into the overall motion model of the vehicle:

$$\begin{aligned}
\dot{x} &= v * \cos \theta \\
\dot{y} &= v * \sin \theta \\
\dot{v} &= a \\
\dot{\delta} &= \phi \\
\dot{\theta} &= v/W * \tan \delta
\end{aligned} \tag{7.4}$$

where x, y are the vehicle 2D position coordinates in map frame, v is the linear velocity, θ is the vehicle orientation angle, a is the linear acceleration, δ is the steering angle, ϕ is the steering angle velocity, and W is the wheelbase.

Finally, the new state at time $t + 1$ can be computed, updating the current state at time t according to the kinematic model of the system:

$$\begin{aligned}
x_{t+1} &= x_t + \dot{x} * dt \\
y_{t+1} &= y_t + \dot{y} * dt \\
\theta_{t+1} &= \theta_t + \dot{\theta} * dt \\
v_{t+1} &= v_t + \dot{v} * dt \\
\delta_{t+1} &= \delta_t + \dot{\delta} * dt
\end{aligned} \tag{7.5}$$

where dt is the time passed between sensor readings.

One important point regarding the velocity v is that calculating this value accurately is complex, as relying solely on wheel encoders is insufficient due to wheel slips. However, in this work, a simple averaging of the speeds from the four wheels has been used for velocity estimation. In the future, this part can be improved by integrating GPS and IMU data for a more accurate velocity estimation.

7.2.4 Data association

In many real-world applications where SLAM techniques are employed, landmarks are not identifiable, and the total number of landmarks cannot be known a priori. Therefore, a process—typically probabilistic—is required to associate observations with existing landmarks in the map or create new ones when observations do not match any of the existing landmarks. This process, known as data association, is one of the most challenging problems in SLAM or localization. Effective data association involves matching sensor observations with corresponding landmarks in the environment, correctly associating measurements with the

correct state, initializing new tracks, and detecting and rejecting spurious measurements. Successful data association is crucial for maintaining an accurate and consistent map and for correctly estimating the robot's trajectory.

In the simulator environment, data association can be handled more easily due to the existence of ground truth data. However, in the real-world environment, it remains a complex problem due to reasons such as:

- Noisy Sensor Measurements: Sensor noise can lead to incorrect matches between observations and landmarks;
- Ambiguity: Similar features or repetitive patterns in the environment can cause confusion in matching observations to landmarks;
- As the number of landmarks and observations grows, the complexity of finding correct associations increases.

Approaches to data association methods can be broadly classified into Bayesian and non-Bayesian approaches.

Bayesian approaches use probabilistic models to represent the uncertainty in sensor measurements and landmark position. They typically involve updating belief distributions over possible associations based on new observations. These methods include Particle filters and Kalman filters which are commonly used methods in SLAM.

On the other side we have non-Bayesian methods which do not explicitly model uncertainties probabilities. Instead, they rely on deterministic criteria to match observations to landmarks. These methods are often simpler and computationally more efficient. One common non-Bayesian approach is Nearest Neighbor(NN) filtering.

In the context on this project, the selected approach is Nearest Neighbor(NN) filtering. The idea behind this approach is to match each observation to the closest landmark based on the chosen distance metric, in our case Euclidean distance. In our scenario, when a cone is detected, it is compared with all the existing landmark nodes in the graph. It is then linked to the closest one, provided their distance falls below a specified threshold.

Despite its simplicity and efficiency, Nearest Neighbor filtering has several disadvantages. For instance, in environments with many similar feature, NN filtering can make incorrect matches. Also this method does not handle uncertainties explicitly, making it less robust in dynamic or highly uncertain environments.

To propose an improvement for the future and addressing the limitations of the Nearest Neighbor filtering, a more sophisticated approach can be using Mahalanobis distance.

7.2.5 Loop closure

Loop closure is a critical component in Simultaneous Localization and Mapping because it significantly enhances the accuracy and consistency of the generated map and the estimated trajectory of the robot. The key reasons for the importance of loop closure are:

- as a robot navigates through its environment, small error in odometry and sensor reading accumulate over time, leading to drift in the estimated trajectory and landmark positions. Loop closure helps to correct these accumulated errors by recognizing when the robot return to a previously visited location, as shown in figure 7.4;
- ensuring the the map remains consistent is crucial for long-term navigation. Loop closure adjusts entire map to align newly detected features with the previously mapped features, ensuring that overlapping regions are correctly matched and integrated;
- by recognizing previously visited locations, the robot can re-localize itself within the map more accurately, reducing the uncertainty in its position. This is particularly useful in large-scale or feature-sparse environments where odometry alone may not suffice for accurate localization.

From a theoretical point of view, loop closure involves the following steps:

- Step 1: The system identifies that the robot has returned to a previously visited area. This is often achieved using feature matching techniques where current observations are compared with stored observations to find correspondences;
- Step 2: A loop closure is detected and a new constraint is added to the SLAM problem. This constraint is formulated as an edge in the factor graph, connecting the current pose to a previous pose, which in our case is the initial position of the car;

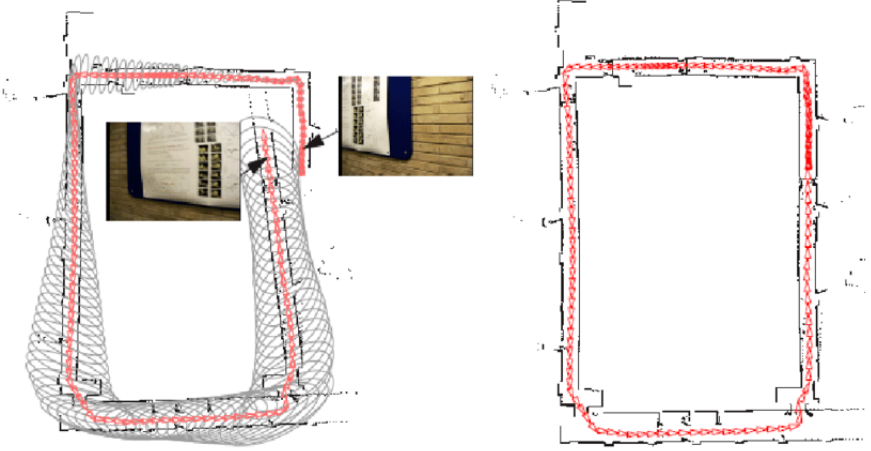


Figure 7.4: Example of loop closure.

- Step 3: Optimize the full system by incorporating the loop closure constraint into the existing graph of the poses and landmarks. This is done by Levenberg-Marquardt algorithm. The goal is to minimize the overall error in the graph considering the new loop closure constraint and the existing constraints (e.g., odometry and landmark observations).

Mathematically, the loop closure constraint can be represented as:

$$e_{ij}(x_i, x_j) = (X_j^{-1}, X_i)Z_{ij} \quad (7.6)$$

where:

- x_i and x_j are the poses involved in the loop closure
- Z_{ij} is the measured relative transformation between these poses
- X_i and X_j are the transformation matrices representing the poses
- the error function e_{ij} quantifies the difference between the predicted and measured transformations.

Moving from theory to practise, in the scenario of Formula Student Driverless competitions, we have challenges matching the distinctive features like corners and edges using algorithms such as [22], SURF [23], and ORB [24]. The reason for this is the presence of large untextured areas and the ambiguity of the cones.

As the solution, we perform a simple approach, made possible by the information we have about the layout of the competitions. Since it is established that large orange cones mark the beginning of the track, once we identify them

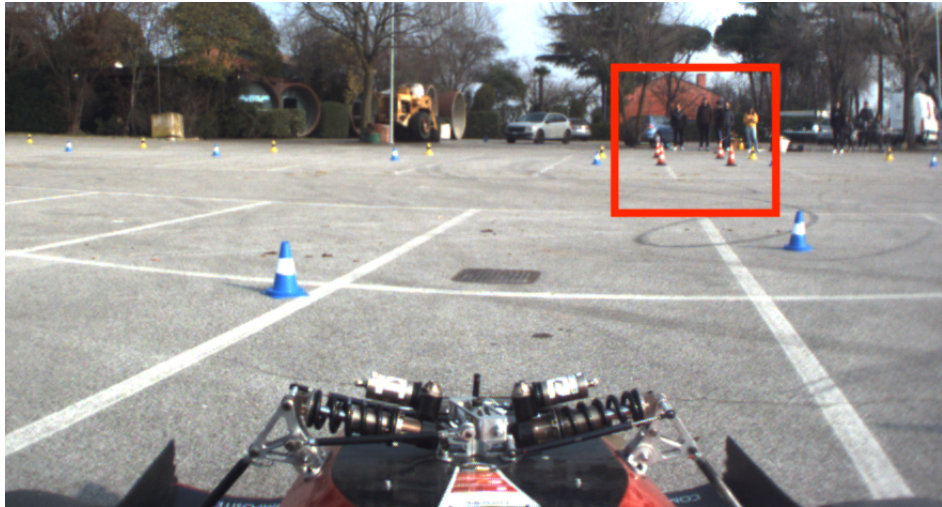


Figure 7.5: Example of a scenario where big orange cones are observed but optimization should not be called.

an odometry edge is added between the current pose and the initial pose of the graph. This approach operates under the assumption that the large orange cones are exclusively located at the track’s starting line.

This method works perfectly in a simulated environment because the process is repeatable and consistent each time the car completes a lap, allowing us to know exactly when to add the final edge and perform loop closure and optimization. However, in the real world, this remains challenging because it is uncertain that the car will be in the same place every time the large orange cones are observed. The solution is to make data association more robust. Whenever the data association algorithm detects previously observed large orange cones, we perform loop closure. With our current data association algorithm, this requires adding more conditions for loop closure, such as ensuring the orange cones are within a certain distance from the car or verifying that a specific pattern of four big orange cones is observed by the front-end. An example of such scenario that may mislead the system to loop closure is presented in the figure 7.5.

7.2.6 ICR motion constraint

The Instantaneous Center of Rotation (ICR) motion constraint refers to the point around which a vehicle is momentarily rotating. For wheeled robots, particularly those with non-holonomic constraints (i.e., they cannot move sideways) as shown in figure 7.6, this concept helps with understanding the expected motion of the robot [25]. By incorporating the ICR motion constraint into SLAM, we can enhance the accuracy of the robot’s trajectory estimation by enforcing realistic

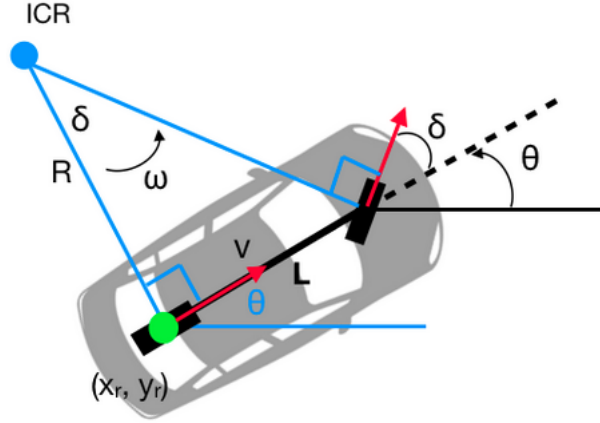


Figure 7.6: Simple representation of ICR.

motion constraints.

In the context of Formula Student racing cars, the ICR constraint is particularly important due to the high-speed dynamics and precision required in racing scenarios. These cars exhibit non-holonomic motion, meaning they predominantly move forward and backward, with limited lateral motion. The ICR motion constraint ensures that the vehicle's motion model accurately reflects its physical capabilities, leading to more precise trajectory and map estimates.

The trajectory of the race car can be decomposed into a sequence of consecutive displacements around the ICR. This trajectory can be parameterized by the radius of curvature ρ and the traveled angle ω . Radius of curvature ρ is the vector connecting the center of the rear axle of the car with the ICR. It is collinear with the rear axle itself. The traveled angle ω is the angle between the vector ρ and the vector connecting the ICR with the center of the front axle.

From the implementation point of view, an edge has been defined base on the equation 7.7 and added to the graph.

$$\mathbf{e}_{icr}(x_t, x_{t+1}) = [\dot{d}x - dx, \dot{d}y - dy]' \quad (7.7)$$

where:

- dx and dy are the computed relative motion between the two poses;
- $\dot{d}x$ and $\dot{d}y$ are the expected motion values;

In this way, while optimizing the graph the solutions that are infeasible based on this constraint will be discarded.

7.3 Evaluation on simulator and real world

In this section, we present some results comparing the same SLAM implementation in both simulated and real-world environments. We will first discuss the results achieved in the simulator by Alessandra Tonin [6], and then compare them with the real-world performance. The evaluation is conducted on the same track layout in both the simulator and real-world scenarios.

Unfortunately, due to the competitive nature of the Formula Student competition, there are no available SLAM module implementations specific to the Formula Student Driverless scenario. As a result, there are no other benchmarks available for comparison. However, we will use the results from the simulator as a benchmark for this thesis.

In the first part, we will discuss the approach for obtaining ground truth data, which is crucial for both qualitative and quantitative evaluations. Following this, we will briefly review the results from the simulator and then present the results from the real-world implementation. Finally, we will explore possible approaches to achieving similar results in the real world as those obtained in the simulator.

7.3.1 Ground Truth

Ground truth refers to the accurate, real-world measurements of the environment and the robot's trajectory, serving as a benchmark for comparison with the SLAM system's estimates.

In the simulator environment, all necessary ground truth data is provided by the simulator. This includes the ground truth positions of the cones in the world frame and the car's trajectories.

However in the real world we need to find a solution for obtaining the ground truth cones and car trajectories. The approach for the cone ground truth is to use a drone and take aerial photos from the track. Then the camera on the drone have been calibrated by taking pictures of a checkerboard and then using softwares such as AutoCalib [26] and Kalibr [27] [28] [29]. In addition, four cones have been put on the four corners of the track as reference points. The exact distance between these cones are known and by exploiting this information we can get the ground truth cones from the aerial images. In the last step, all the cones in the image have been annotated manually and then transformed from the image frame to the camera frame using the information from the calibration of the camera. Figure 7.7 shows the acquired aerial image.



Figure 7.7: On the left, aerial view of our track and on the left the annotations of the cones.

For trajectory ground truth, the idea is to use a DGPS. Differential GPS (DGPS) involves using two GPS receivers: one stationary reference receiver and one mobile receiver (such as on the car). The stationary receiver is placed at a known fixed location, while the mobile receiver is attached to the moving vehicle. The reference GPS receiver is positioned at a precisely known location. Since its position is fixed and known, any deviations in its GPS readings are due to satellite signal errors. The reference station calculates the error in its position by comparing the GPS-received position with its known fixed position. It then broadcasts these error corrections to the mobile receiver. And finally the mobile receiver uses the correction data from the reference station to adjust its own position readings, significantly reducing the overall error.

However, this method has not been used for two reasons:

- according to the rules of Formula Student competition, it is forbidden to send any signals to the car. In this type of DGPS, a signal will be continuously sent to the car for position correction so as a consequence it will not be possible to use this technique;
- the cost for this sensor is high and due to the reason that it is not possible to use it in the competitions, the team prefers to not spend the resources on this tool;

7.3.2 Experiments and results

In this section, we will present the results from both the simulator and real-world implementations. The results from the simulator serve as a benchmark for the real-world implementation, with the goal being to improve the real-world

performance to match the simulator’s results. It is important to note that, to ensure realistic comparisons, the same kinematic model of the car and the same track layout were used in both the simulator and real-world implementations. In addition, the car sensor positions are the same in both simulator and real-world experiment.

Before presenting the results, it is important to mention that in the images, green dots represent the discretized poses of the vehicle, while red dots represent the mapped cones, expressed in the map reference frame.

Simulator results

There are two primary reasons for presenting the results from the simulator. First, it ensures that the algorithm implementations work correctly in the ideal conditions of the simulator, without random noises and unwanted changes, and with the possibility for repetition. The ultimate goal of presenting the simulator results is to establish a benchmark for the real-world implementation. This benchmark allows us to understand the capabilities of our graph-SLAM implementation, guiding our efforts to improve the real-world implementation to achieve similar results. It is important to mention that all the results achieved in the simulator were obtained by Alessandra Tonin [6].

In this experiment, the implementation includes motion integration as explained in section 6.2, Instantaneous Center of Rotation (ICR) as an additional constraint, ideal data association, loop closure, and global-only graph optimization. Ideal data association means that observed cones are tested for association with already mapped ones using the ideal ground truth coordinates provided by the simulator. Additionally, global-only graph optimization indicates that the optimization is performed only after a complete lap is traced by the car.

Starting with the first experiment, global-only optimization, the results are shown in the figure 7.8. The improvement of mapping and localization is clear after the loop closure. Main part of this improvement is due to the loop closure algorithm, which allows to reduce the drift that was accumulated in the trajectory. The overlay with the ground truth track isn’t very important for evaluating map quality because the misalignment is due to a small difference in reference frames. What’s important is that the trajectory’s shape is maintained, showing successful convergence.

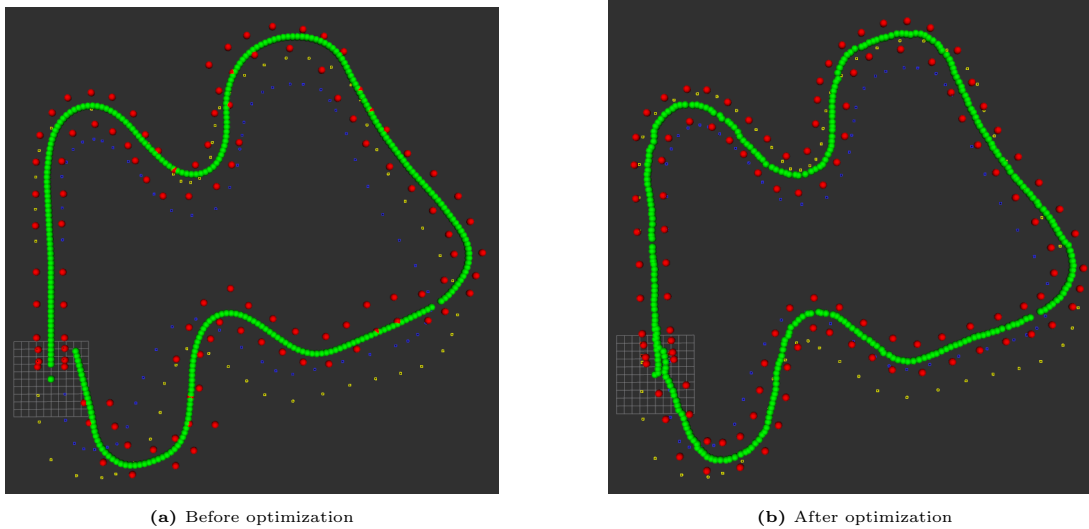


Figure 7.8: EXP1: map and optimization results

Real world results

In this section, we will present the results from our real-world tests. It is important to note that the core algorithms implemented in both the simulator and real-world scenarios are the same, allowing for a meaningful comparison.

Starting with the initial experiment, we perform the localization and mapping without any optimization, loop closure, or data association. As you can see in figure 7.9, the mapping is very noisy and contains many redundant cones. However, the key point is that the cones follow the trajectory of the car.

In the next experiment, shown in Figure 7.10, data association is applied. As before, the green dots indicate the estimated poses of the car, while the red dots show the estimated cones' position while performing data association. It is easily noticeable that the data association significantly improves the quality of the mapping by removing the redundant cones from the map.

The following and final step is to perform global optimization. Figure 7.11 shows the localization and mapping after the loop closure. Comparing Figure 7.10 (before loop closure) and Figure 7.11 (after loop closure), it is clear that the trajectory has improved and the loop has been closed. However, the mapping is not perfectly aligned with the ground truth (points in blue) due to the low quality of the front-end performance. To support this claim, we conduct two experiments.

The estimates of the cones positions depend on many different factors. Due to a non-perfect calibration and a low-resolution camera, the landmark constraints produced by the front-end made generated this misalignment. In fact, using only

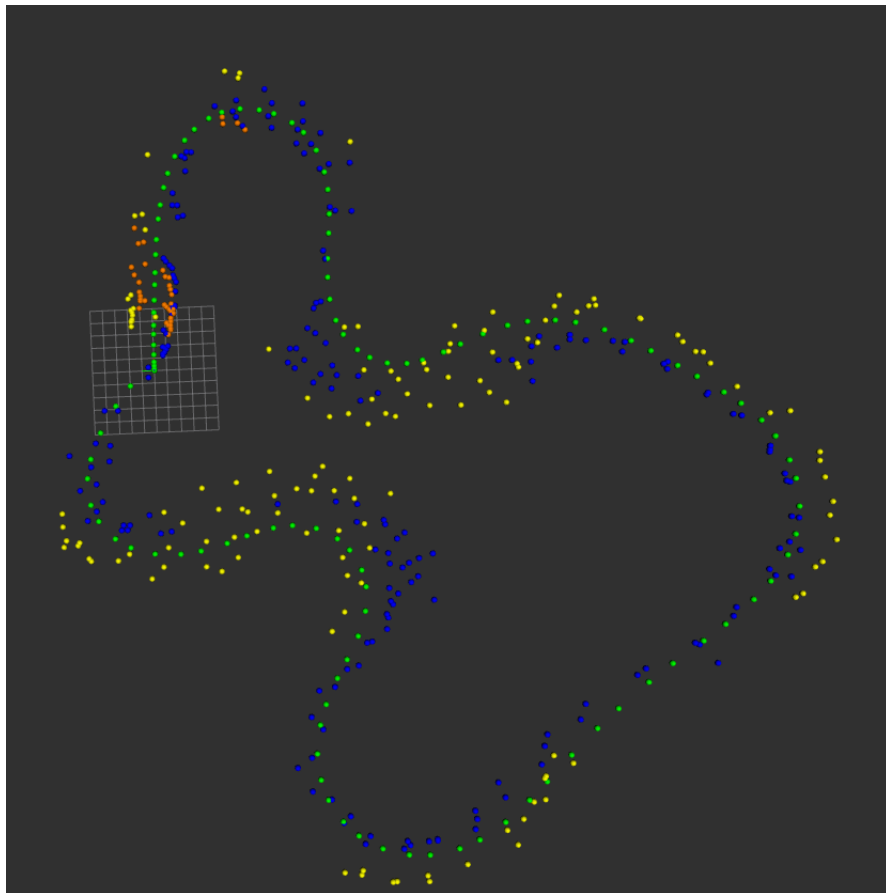


Figure 7.9: Localization (green dots) and mapping of the landmark detections (blue, yellow, and orange dots) without any data association. The presentation is in the world frame.

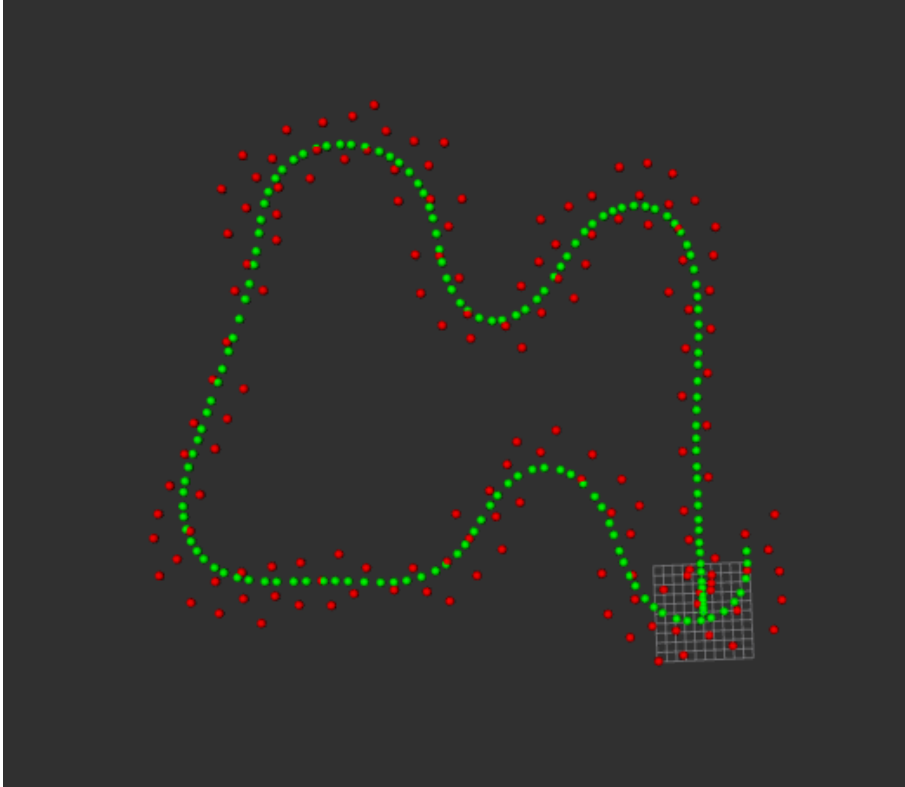


Figure 7.10: localizations(green dots) and landmark detections(red dots) with data association applied. It presentation is in the world frame

the pose and the loop closure proved to be a more effective solution. In the first experiment, we remove the edges between the landmarks and car poses, thereby ignoring their constraints on the graph optimization. As it can be seen in the figure 7.12, the loop is perfectly closed showing that low quality of the front-end was responsible for noisy results.

In the second experiment, we replace the ideal data association in the simulator with a real data association. This means that, instead of using the ground truth cone positions as the reference for data association, we use the cones detected by the detection module to choose the reference for each data association. As expected, this introduces more inaccuracy in the mapping, but it provides a good example of how it affects the quality of loop closure and optimization. As it is clear in the figure 7.13, the mapping and localization quality decreases significantly compared to the result from 7.8. This demonstrates that improving the front-end is essential to obtain both robust and accurate estimates of the car's poses and the map.

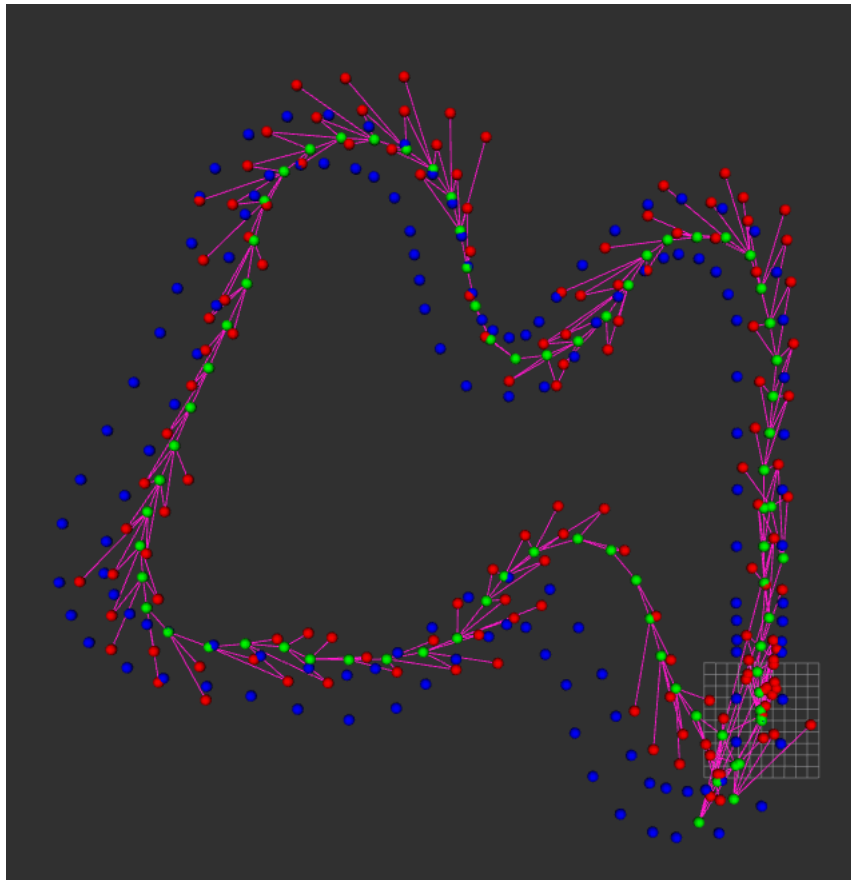
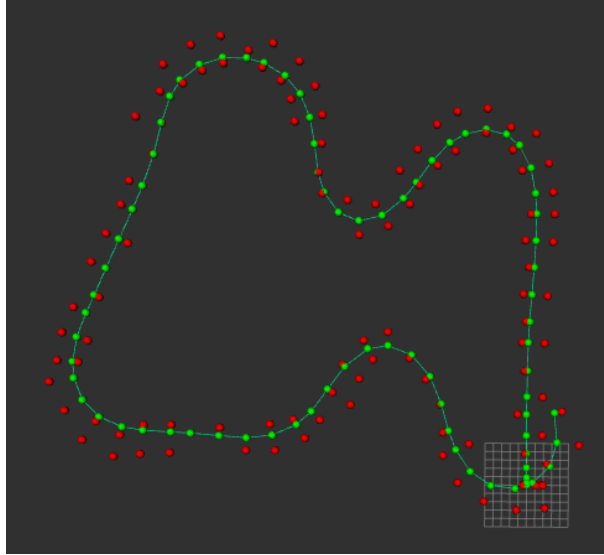
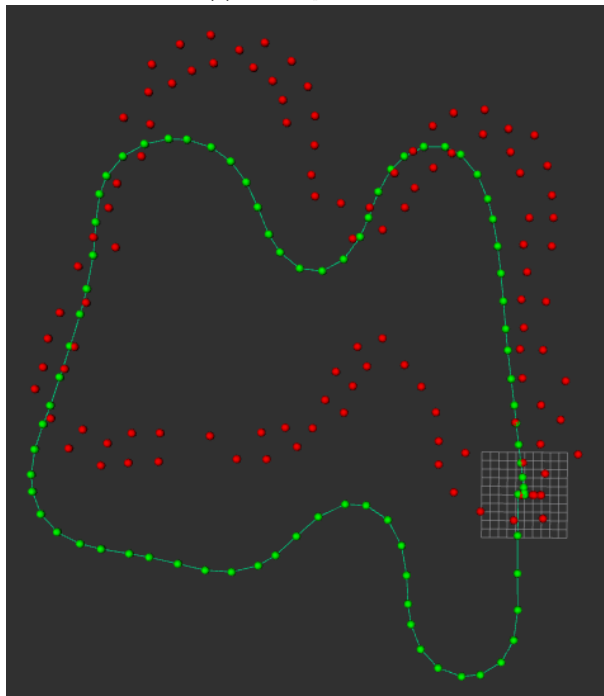


Figure 7.11: Global optimization performed with loop closure after finishing one lap. Blue dots represent the ground truth cone positions, green dots the localizations, and red dots the landmark detections.

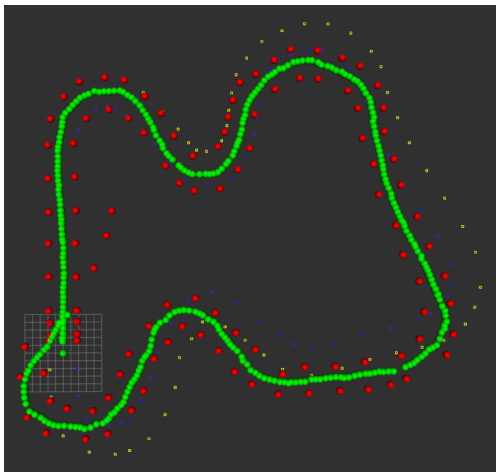


(a) Before optimization

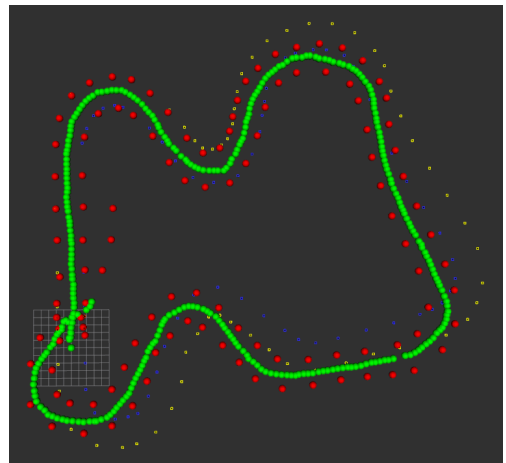


(b) After optimization

Figure 7.12: Performing loop closure without landmark edges



(a) Before optimization



(b) After optimization

Figure 7.13: Realistic data association map and optimization results in simulator environment

Chapter 8

Conclusion and future works

In this thesis, we outlined the process of implementing a localization and mapping system for a Formula Student racing car. We discussed the challenges and requirements for transitioning this algorithm from a virtual environment to the real world by detailing the sensorization of the car and the integration of these sensors onto the actual vehicle. Additionally, we described the dataset acquired to facilitate this transition from the simulator environment to the real world. We then explained the techniques used in the visual perception component and presented the results from the detection module. Finally, we presented the graph-SLAM as the back-end block for the SLAM problem, reviewed the implementation details, and compared the results achieved in the simulator as a benchmark with the results obtained using real-world data.

This thesis was the beginning and an initial step toward the long journey of developing a self-driving car. There are many details and topics that can be added or improved upon in future work. As the first step, the current software pipeline should be updated with the new sensor setup. This updated setup would enhance the robustness and accuracy of detection by allowing improvements in the perception module. Next, the LiDAR should be integrated into the system by fusing the point clouds from the LiDAR with the detections from the camera. Finally, data association can be improved by replacing the Euclidean distance with the Mahalanobis distance.

References

- [1] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part i. *IEEE Robotics Automation Magazine*, 13(2):99–110, 2006.
- [2] Race UP Team. <https://raceup.it>.
- [3] FSEast FSG. Competition handbook 2024.
- [4] Formula Student Germany. <https://www.formulastudent.de/fsg/>.
- [5] AMZ Team. <https://www.amzracing.ch/en>.
- [6] Alessandra Tonin. Design of a perception system for the formula student driverless competition: from vehicle sensorization to slam. *Master thesis*, 2023.
- [7] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-yuan Liao. Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. 07 2022.
- [8] University of Edinburgh. Eufs simulator.
- [9] et al. N. Jun, H. Jibin. Autonomous driving system design for formula student driverless racecar. pages 1 – 6, 2018.
- [10] Sherif Nekkah, Josua Janus, Mario Boxheimer, Lars Ohnemus, Stefan Hirsch, Benjamin Schmidt, Yuchen Liu, David Borb’ely, Florian Keck, Katharina Bachmann, and Lukasz Bleszynski. The autonomous racing software stack of the kit19d. *ArXiv*, abs/2010.02828, 2020.
- [11] J. Neira and J.D. Tardos. Data association in stochastic mapping using the joint compatibility test. *IEEE Transactions on Robotics and Automation*, 17(6):890–897, 2001.
- [12] Marcel Zeilinger, Raphael Hauk, Markus Bader, and Alexander Hofmann. Design of an autonomous race car for the formula student driverless (fsd). 05 2017.

- [13] Sirish Srinivasan, Inkyu Sa, Alex Zyner, Victor Reijgwart, Miguel Valls, and Roland Siegwart. End-to-end velocity estimation for autonomous racing. *IEEE Robotics and Automation Letters*, PP:1–1, 08 2020.
- [14] Juraj Kabzan, Miguel de la Iglesia Valls, Victor Reijgwart, Hubertus Franciscus Cornelis Hendriks, Claas Ehmke, Manish Prajapat, Andreas Bühler, Nikhil Bharadwaj Gosala, Mehak Gupta, Ramya Sivanesan, Ankit Dhall, Eugenio Chisari, Napat Karnchanachari, Sonja Brits, Manuel Dangel, Inkyu Sa, Renaud Dubé, Abel Gawel, Mark Pfeiffer, Alexander Liniger, John Lygeros, and Roland Siegwart. AMZ driverless: The full autonomous racing system. *CoRR*, abs/1905.05150, 2019.
- [15] Leiv Andresen, Adrian Brandemuehl, Alex Honger, Niclas Vodisch, Hermann Blum, Victor Reijgwart, Lukas Bernreiter, Lukas Schaupp, Jen Chung, Mathias Bürki, Martin Oswald, Roland Siegwart, and Abel Gawel. Accurate mapping and planning for autonomous racing. pages 4743–4749, 10 2020.
- [16] Michael Montemerlo, Sebastian Thrun, Daphne Koller, and Ben Wegbreit. Fast-slam 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. *Proc. IJCAI Int. Joint Conf. Artif. Intell.*, 06 2003.
- [17] Giorgio Grisetti, Rainer Kümmerle, Cyrill Stachniss, and Wolfram Burgard. A tutorial on graph-based slam. *IEEE Transactions on Intelligent Transportation Systems Magazine*, 2:31–43, 12 2010.
- [18] Juan D. Tardós Raúl Mur-Artal, J. M. M. Montiel. Orb-slam: A versatile and accurate monocular slam system. pages 1147 – 1163, 10 2015.
- [19] Rainer Kümmerle, Giorgio Grisetti, Hauke Strasdat, Kurt Konolige, and Wolfram Burgard. G2o: A general framework for graph optimization. In *2011 IEEE International Conference on Robotics and Automation*, pages 3607–3613, 2011.
- [20] Niclas Vödisch, David Dodel, and Michael Schötz. Fsoco: The formula student objects in context dataset. *SAE International Journal of Connected and Automated Vehicles*, 5(12-05-01-0003), 2022.
- [21] Hugh Durrant-whyte and Tim Bailey. Simultaneous localization and mapping: Part i. *Robotics Automation Magazine, IEEE*, 13:99 – 110, 1995.
- [22] Tony Lindeberg. *Scale Invariant Feature Transform*, volume 7. 05 2012.
- [23] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. volume 3951, pages 404–417, 07 2006.

- [24] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *2011 International Conference on Computer Vision*, pages 2564–2571, 2011.
- [25] Lionel Clavier, Michel Lauria, and François Michaud. Instantaneous centre of rotation estimation of an omnidirectional mobile robot. In *2010 IEEE International Conference on Robotics and Automation*, pages 5435–5440. IEEE, 2010.
- [26] Aditya Vaishampayan. Autocalib.
- [27] Joern Rehder, Janosch Nikolic, Thomas Schneider, Timo Hinzmann, and Roland Siegwart. Extending kalibr: Calibrating the extrinsics of multiple imus and of individual axes. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, page 4304–4311. IEEE Press, 2016.
- [28] Paul Furgale, Joern Rehder, and Roland Siegwart. Unified temporal and spatial calibration for multi-sensor systems. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1280–1286, 2013.
- [29] Jérôme Maye, Paul Furgale, and Roland Siegwart. Self-supervised calibration for robotic systems. In *2013 IEEE Intelligent Vehicles Symposium (IV)*, pages 473–480, 2013.