UNIVERSITÀ
DEGLI STUDI
DI PADOVA

UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

MASTER'S DEGREE IN COMPUTER ENGINEERING

Design of a perception system for the Formula Student Driverless competition: from vehicle sensorization to SLAM

Supervisor:
PROF. ALBERTO PRETTO
Co-supervisor:
EMILIO OLIVASTRI, PHD CANDIDATE

Student:
ALESSANDRA TONIN
ID number:
2027136

Academic year 2022/2023

Acknowledgements

First of all, I wish to express my appreciation to my supervisor, Professor Alberto Pretto, for the trust and confidence he has bestowed upon me, and for giving me the opportunity to challenge myself with such an amazing project. I am extremely grateful for the chance to work under his supervision, in the interesting field of autonomous driving applied to racing scenarios.

Further, I would like to sincerely thank my co-supervisor Emilio Olivastri, PhD candidate, for his willingness to support and supervise my work and for the excellent advice he's given me throughout the entire project. Working with him has been very stimulating, and his belief in my abilities has been a constant source of motivation and inspiration. His precious help has been fundamental for the completion of this thesis.

"Go hard or go home"

Abstract

Formula Student Driverless is an international racing competition held among universities, where the vehicles must complete a set of trials without any human intervention.

Together with RaceUP, the Formula Student team of the University of Padova, this thesis represents the beginning of the project to build an autonomous prototype to compete in the Driverless Cup in the 2024 season.

Three important aspects of an autonomous system design will be tackled: vehicle sensorization, perception, and simultaneous localization and mapping (SLAM), with the main focus on the development of the last one.

The proposed approach for the back-end is based on the optimization of a factor graph, holding information about car poses and landmarks positions, by exploiting spatial and kinematic constraints between its vertices. The full back-end pipeline has been tested thoroughly, step by step, allowing to obtain satisfactory results on the different virtual tracks used for testing.

Using both modern and classical techniques, we can process information produced by the stereo camera and the LIDAR, to be able to localize the colored cones delimiting the track. The estimation of cones positions serves then as input for other important modules of the car, such as the control part and the SLAM pipeline.

Finally, a complete dataset has been acquired by properly sensorizing RaceUP's last year's car: having real data represents a helpful resource to make experiments and validate the system, even without the availability of the actual vehicle prototype.

Contents

Chapter 1

Introduction

In the recent years, autonomous driving is gaining increasing attention due to the numerous benefits it offers. It promises safer and less congested roads and a potential reduction of the environmental impact of human mobility. Furthermore, the social aspect must not be overlooked: having self-driving means of transportation could provide more freedom of movement to people unable to drive.

Many sectors could exploit the potential of self-driving vehicles: among the others, we have freight handling, agriculture, or daily traveling. However, social and regulatory issues are slowing down this paradigm shift in transportation: the technological complexity behind an autonomous machine makes the diffusion of these systems quite hard. In most countries, only a few big companies have been able to obtain permission to do tests or deploy their driverless vehicles on real roads: for this reason, autonomous driving is mostly applied to safer scenarios, like agriculture or racing, where interaction with people is minimal.

1.1 Problem statement

The design of a self-driving vehicle requires several different steps, going from hardware components arrangement to complex software development.

To be able to navigate independently from humans in unknown situations, an autonomous vehicle needs many different connected modules. Among the others, perception, localization, and mapping are the fundamental ones. Of course, the entire software stack would be useless without the aid of a set of sensors, properly mounted on the car.

Perception can be compared to human senses, since it involves the gathering of information from the surrounding environment: the capability to perceive the

nearby world is at the basis of autonomous navigation since this data is then used to understand, for example, if an obstacle is on the vehicle's trajectory or if the road is making a curve or being straight.

SLAM is the acronym for simultaneous localization and mapping, and it is probably the most important part of the self-driving pipeline: it aims at the creation of a map of the environment in which the vehicle is operating, while also localizing itself in the created map. So the vehicle is able to estimate its own pose with respect to the world, and the relative uncertainty of that estimate.

As briefly stated before, there are plenty of scenarios in which autonomous driving technologies can be applied. One of these is the car racing world: in this work, we will focus on a driverless racing event, part of the Formula Student competition. The opportunity to work on this project came from RaceUP [1], the Formula Student team of the University of Padova. For many years, they have been participating in both national-level and European-level races, but they still don't have an autonomous car to compete in driverless events.

In general, the design of a vehicle capable of participating in a high-level competition is not trivial: many different parts must be perfectly integrated to make this complex system work in a highly dynamic situation. Moreover, redundant modules should be provided, to guarantee fault recovery capabilities.

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 not available yet, so the entire development process must be carried out in simulation or by adapting the electric car from the last season to fit the driverless race constraints;

- the choice of sensors to be mounted on the vehicle is constrained to university funds, so their availability is not ensured in any moment of the design procedure;

- being this project the first concept of driverless vehicle at all, it is a learn-by-doing activity for everyone.

As Formula Student is a university competition that may not be widely recognized among individuals not directly involved in this field, the following section will provide an overview of this event. Specifically, it will focus on the details regarding the Driverless category.

1.2 Formula Student

Formula Student (also named Formula SAE, from the Society of Automotive Engineers, the original organizer) is a student competition founded in 1981 with the aim to design a prototype racing single-seater, meaning a high-performance Formula-style race car, tailored to participate in nationally and internationally recognized racing events. The main features of a Formula-style vehicle include being an open-wheeled, single-seat, and open cockpit, with four wheels that are not in a straight line [2]. Participating in such an activity is a complete challenge since students have to conceive, design, fabricate, develop, and compete with their cars.

Teams can be composed exclusively of university students, having different backgrounds to provide knowledge in all the required fields: necessary skills involve mechanical, electrical, and software engineering, but also economics, management, and marketing.

Each team has to build its own cars in compliance with a series of official rules defining precise characteristics of the chassis, but also constraints aimed at ensuring safety conditions during the races. Having regulations to follow also improves the problem-solving capabilities of future engineers, who rarely have total freedom in the development of their work projects.

Historically, Formula Student started with the development of a combustion vehicle with a traditional thermal motor, while in 2010 the electrical division has been introduced: from that year, every team also had to provide a fully electrically-powered prototype. In 2017, finally, the Driverless division has been started by the German Formula Student committee: the goal is to design a self-driving vehicle, able to complete all events without human driver assistance and without any kind of remote control.

From now on, everything related to Formula Student will implicitly refer to the driverless race car, since the thesis focuses on designing and developing part of an autonomous racing system. Moreover, the information given below is valid for the 2023 competition, as stated in the official documents [2] at the moment of writing.

A specific regulation, concerning both the car and the track, has been defined to guarantee the safety of all the participants.

According to the Formula Student Germany Competition Handbook 2023 [2], in all the configurations the tracks will be marked with colored cones (see figure 1.1): blue on the left side, yellow on the right one, small orange on the

Figure 1.1: Official Driverless competition cones.

exit and entry lanes and big orange before and after the start, at the finish and timekeeping points. Moreover, cones along the driving direction will be at most 5 meters far from each other, and specific or particular zones will be marked with colored paint.

Before starting the competition, each vehicle is officially checked for rule compliance through Technical Inspection. As an example, for the autonomous system, data sheets of all the perception sensors must be provided, together with the documents certifying that those sensors meet local legislation. Another important component that is verified to be working is the Remote Emergency System, a remote-controlled module mounted on the vehicles, that allows the activation of emergency behaviors. It is activated if at least one of the following occurs :

- 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

   Business Plan Presentation: each team has to convince potential investors or partners to be part of the project, by delivering a comprehensive business model representing a monetary profit opportunity for the group.

   Cost and Manufacturing: the purpose is to evaluate the ability of taking cost-content trade-off decisions while manufacturing the race car, including make or buy choices. In doing so, three Cost Report Documents must be submitted, containing detailed information about every material or component needed to construct the vehicle, and specifying the corresponding cost.

   Engineering Design: this event wants to evaluate the effort put into the engineering process to design the prototype. The team has to explain its design choices and highlight design features, concepts, or methods that add value to the vehicle.

2. DYNAMIC EVENTS

   Skidpad: an eight-shaped track, figure 1.2, 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 1.3, 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 according to the completion of the aforementioned laps and to the comparison between the elapsed time and the maximum allowed time.

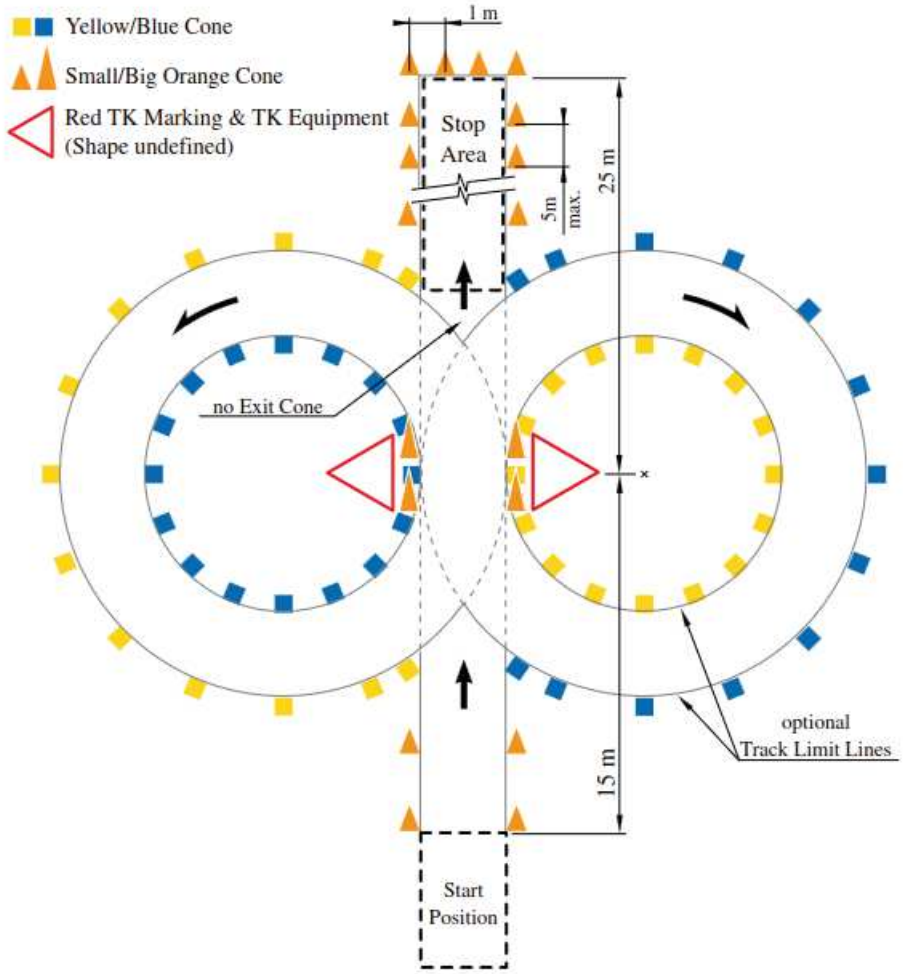


Figure 1.2: Skidpad track configuration.

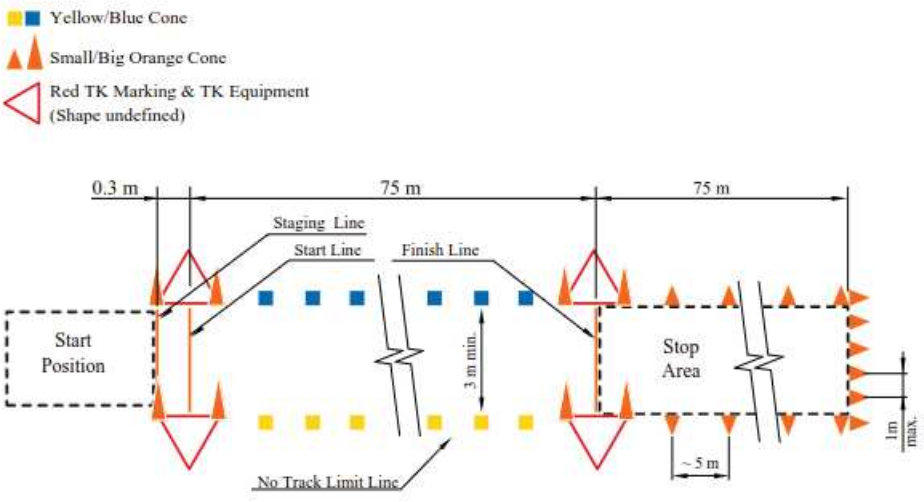


Figure 1.3: Acceleration track configuration.

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 work is to the develop first version of a simultaneous localization and mapping module for a Formula Student Driverless race car. Since this part of the system is strictly connected with other components of the autonomous stack, my work naturally expands into a deep study of vehicle sensorization and perception algorithms. The majority of this project has been developed relying on a simulation platform, described in chapter 4, except for the real sensors positioning and the data acquisition with the actual car.

Starting from literature research (see chapter 2) on scientific papers and articles related to autonomous racing scenarios, the state-of-the-art solutions adopted by other Formula Student teams have been retrieved and studied. Particular relevance has been given to the different sensor setups and to the different proposed approaches to the mapping problem.

As deeply explained in chapter 5, the final choice for the SLAM back-end system has been a graph-based approach, involving the mathematical optimization of spatial constraints linking vertices of a factor graph. The main idea is to jointly process odometry and sensors' observation data to provide an estimation of the current state of the vehicle and of the position of the surrounding landmarks. To make the algorithm more consistent with reality, an additional constraint on the allowed car motion has been added to the graph: considering that a Formula Student prototype is a non-holonomic system, restrictions on the translational motions must be imposed. Two different correction approaches have been compared, a global-only optimization, and a local and incremental one. Finally, a

nearest-neighbor data association algorithm has been analyzed, to make SLAM more robust against sensor noise and outliers. Tests have been conducted with Euclidean distance.

A bit of work has been made also concerning the front-end part of Simultaneous Localization and Mapping, aiming at developing a simple perception module exploiting both images and point clouds. The goal is to recognize the track-delimiting cones and compute their 3D position in space. The visual cone detection pipeline is based on a deep learning approach, implementing a fine-tuned version of YOLOv7 [3] to get the bounding boxes enclosing the aforementioned landmarks in the stereo camera images. The LIDAR detection pipeline, instead, is based on a classical segmentation approach, combined with Euclidean clustering, to spot cone shapes in the received point clouds. Some preliminary qualitative results will be exposed in chapter 6.

Finally, the third part of this work (chapter 7) will report the experimental part of the project, consisting of the acquisition of a real and complete dataset, after the proper sensorization of an official Formula Student car. Real sensors have been mounted on the prototype in a way as adherent as possible to the setup studied in simulation. Additionally, a set of sensor drivers have been developed to synchronize incoming data and to guarantee real-time storage of the received information.

Even if this may seem to be a common autonomous driving application, consisting in well-studied problems like perception and SLAM, the specific Formula Student scenario introduces some important challenges to cope with:

- careful and precise sensors data acquisition and processing is needed to deal with noise and errors introduced by the high speeds required during races;
- hard real-time constraints, with little or no delay at all between data reception and usage;
- sparseness of the environment, which is characterized only by small traffic cones delimiting the track, requiring specifically-tailored data processing algorithms;
- high uncertainty in data association, being that the cones have no discriminating features except for their color.

Some important theoretical background notions, useful to fully understand the proposed solution, are provided in chapter 3.

Chapter 2

Related work

Due to the competition format of Formula Student, it is not easy to find scientific literature specific to this application, since most of the teams are not willing to publish their work or to share their research with competitors. Furthermore, publicly available papers are often describing the first versions of the race cars, that were used in previous seasons.

However, it is useful to analyze these works to have an idea of the fundamental algorithms or setups that are currently used by other participants. Especially for new teams, which are just entering the competition, starting from simpler but effective systems is probably the best path to follow.

In [4][5][6] AMZ team from ETH Zurich, presents *Gotthard* and *Pilatus*, their cars used in seasons 2018-2021. In those papers, you can see the sensors setup remaining more or less the same across the years, except for an additional LIDAR added in 2019, placed on the main hoop together with the cameras. Regarding the perception module, both systems fuse together a camera-based cones detection pipeline and a LIDAR-based one. As visual pipeline both cars use a version of YOLO as detector, while the LIDAR is exploited also to recongnize color by means of intensity data analysis. If we consider the SLAM module, a substantial change happened from 2018 car to 2019 one: from a particle filter algorithm, FastSLAM 2.0 [7], a switch to graphSLAM has been made.

In [8], Austrian team TUW from Wien built a prototype without using a LIDAR, only with a planar laserscan and the other usual sensors, like camera, IMU and GPS. Even here, perception is based on a mixed approach, to exploit images acquired on the track and laserscan points. Perceived data is then used as input for an Extended Kalman Filter implementation, to build the map of the track.

In [9], KIT team from Germany presents a particular setup, where the sensor suite of the car includes 3 cameras and 4 laser scanners, placed all around the car. As before, perception is redundant, with two pipelines for camera and LIDAR data, and a CNN-based approach to detect bounding boxes of the cones. Regarding the SLAM pipeline, also here an Extended Kalman Filter is implemented, using (x, y, θ) as parameters. Data association exploits the famous Joint Compatibility Branch and Bound (JCBB) [10] algorithm.

Finally, MIT Driverless team in [9] uses a YOLO-based pipeline to detect cones, but does not provide much more information about the rest of the system.

Often, more precise and detailed information about the system adopted in a race car can be found directly on the website of the involved university or team. This is because most of the work done in this field is not published as research work.

To have a wider vision of possible approaches worthy to be further studied, it could be useful also to consult the numerous thesis that are written by university students about their prototypes. Of course, taking in mind that thesis are not published or officially reviewed works, so information needs to be carefully checked.

The main contribution of this thesis is the development of a simultaneous localization and mapping module based on graph optimization, to work under the specific conditions of Formula Student Driverless events. Minor development also regards a front-end perception system, trying to exploit redundant camera and LIDAR detection, as observed being the trend of all the other teams.

Chapter 3

Theoretical background

3.1 Perception

A simultaneous localization and mapping algorithm usually involves two steps, the front end, and the back end.

The front end deals with real-time data processing, extracting the most relevant information from the sensor readings that include, in our specific case, spatial relations and associations between landmarks, as explained in section 5.2.4.

In general, sensors are a crucial part of an autonomous robot, that can be considered as the senses of the system: they allow gathering information about the surrounding environment, that can be fused together to control the behavior of the vehicle. The most common sensors exploited to perceive the environment are cameras and LIDAR.

LIDAR stands for Light Detection And Ranging, and it belongs to the family of range-finder sensors. It emits laser beams, that hit obstacles within a certain distance: the time for the reflected light to return to the receiver is measured, and distance can be accurately estimated in this way, as depicted in figure 3.1.

The most important parameter to take into consideration, especially in a scenario like the Formula Student, where the environment does not contain many elements, is the number of channels. In other words, this corresponds to the number of laser-beams emitted by the sensor: a higher number of channels, a higher resolution of the produced scan, see an example in figure 3.2.

This sensor is one of the most used for perceiving obstacles or landmarks in the surrounding environment and put them on a map.

The other complementary technique is visual-based, meaning that perception is carried out using images produced by a camera. A camera is a device contain-

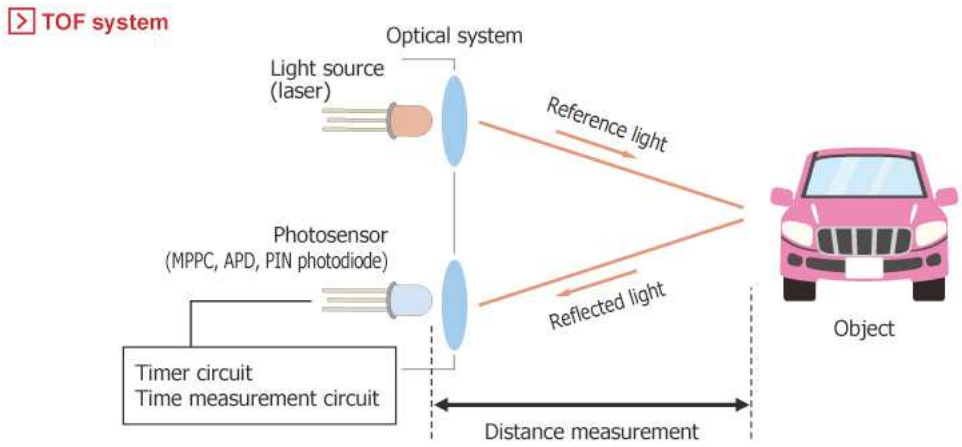


Figure 3.1: The principle behind LIDAR technology.

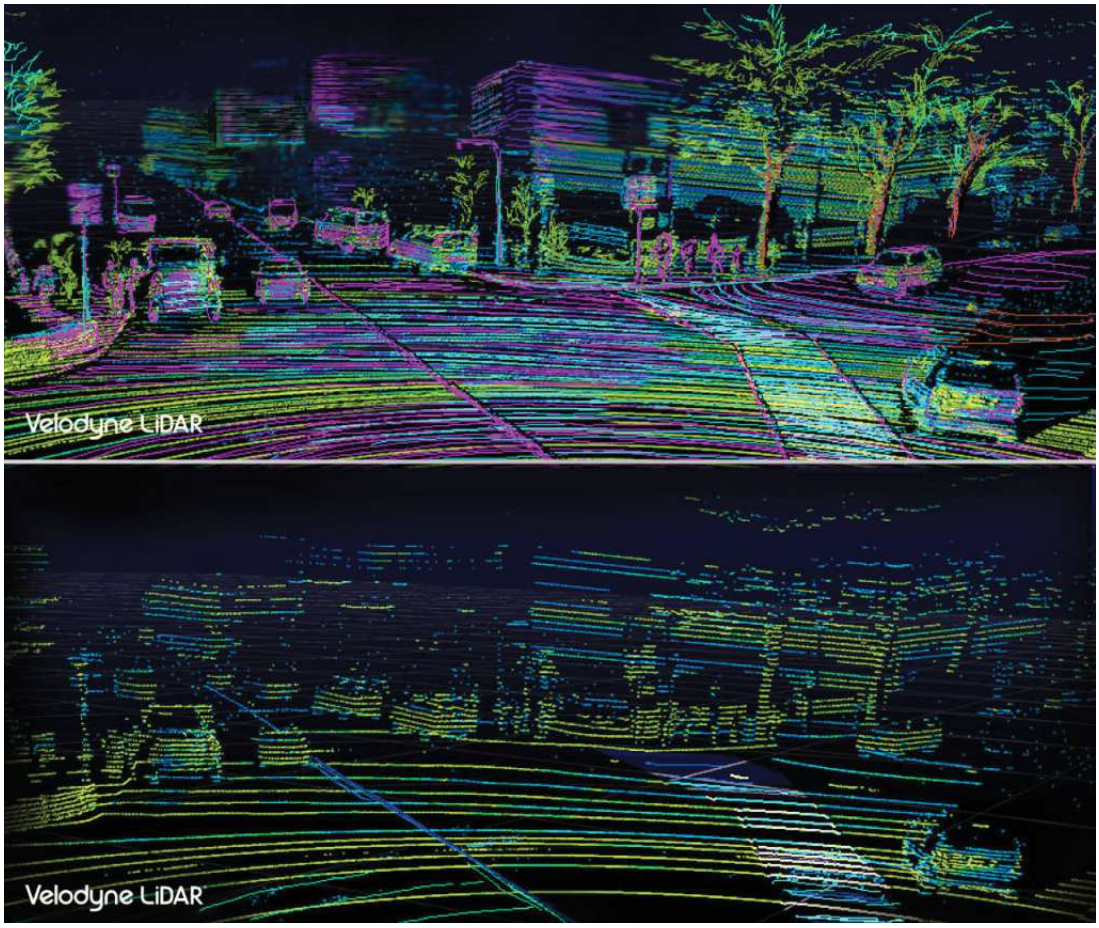


Figure 3.2: Two different resolutions sensors, from Velodyne.

ing an image sensor that converts light reflected by objects into current signals producing an image. The basic camera model is the pinhole camera model, depicted in figure 3.3. Light passing through the hole is captured by the sensor, and impressed in an image. A virtual image plane is then added to avoid having the image visualized upside-down.

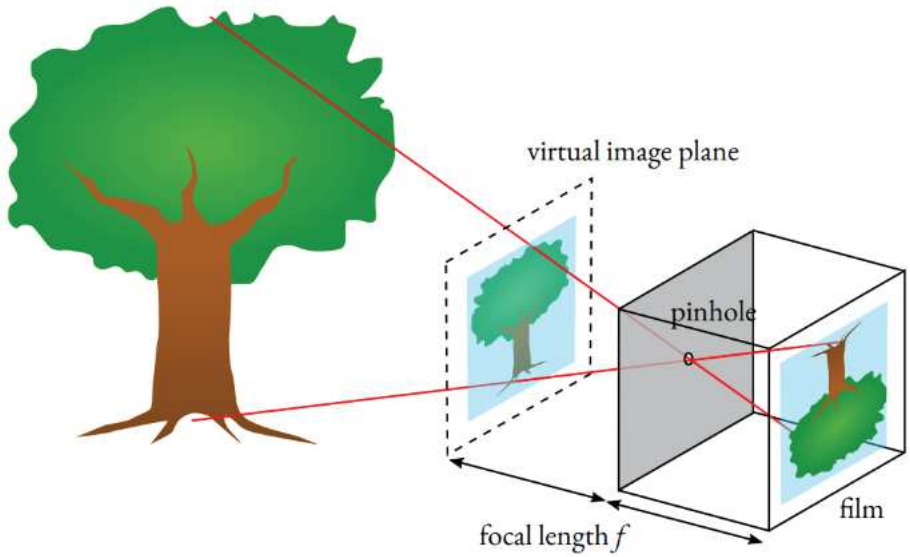


Figure 3.3: The pinhole camera model, with added virtual image plane.

In this work, we will deal with a stereo camera, so it is worth spending some words to introduce this setup.

A stereo system is composed of two or more usually identical cameras, rigidly mounted and framing the same scene from different points of view. Being known the rigid body transformation between the two sensors, it is possible to estimate the depth of a 3D point by using triangulation techniques.

Independently from the sensor, the first operations to do when gathering perceived data are denoising and undistortion, if necessary.

The second one is to extract and track relevant features, such as points, corners, lines, or entire regions, that will be used as landmarks for the robot's position estimation. Tracking these features means following them across individual consecutive frames, by establishing a correspondence between what is currently seen and what we expect to see. This is also known as data association, and it is of crucial importance to minimize the error of mapping algorithms.

Finally, by means of optimization, the robot's pose is estimated and the world map is refined. Correctly assigning data association constraints will allow the optimization to converge faster to an accurate estimate.

3.2 Simultaneous Localization And Mapping (SLAM)

SLAM stands for simultaneous localization and mapping. It is a widely studied robotic problem, that finds application whenever a system moves autonomously in an unknown environment: to reach its goal, the robot must be able to construct a precise model of the surrounding world and of the trajectory it is pursuing. Moreover, it must be able to localize itself inside the created map.

To achieve these goals, the robot exploits its sensors to observe the world and obtain information about the scene (e.g., landmarks). Landmarks will be then used to create the map and as reference points for the localization.

Being sensor measurements affected by noise, the robot trajectory and the created map must be described in a probabilistic way.

To formally define the problem [11], 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 the landmarks at time t.

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

$$P(\mathbf{x}_t, \mathbf{m}|\mathbf{z}_{0:t}, \mathbf{u}_{0:t}) = P(\mathbf{x}_t, |\mathbf{z}_{0:t}, \mathbf{u}_{0:t})P(\mathbf{m}|\mathbf{x}_{0:t}, \mathbf{z}_{0:t}), \tag{3.1}$$

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

The aforementioned probability distribution, obtained given the recorded observations and the control inputs until time t, together with the initial state of the vehicle, is the joint posterior density of the landmark locations and vehicle state at time t.

From 3.1, it is clear that simultaneous localization and mapping is composed by two sub-problem, the observation model and the motion model.

The former describes how the sensor measurements are mapped into the world, meaning the probability of making a certain observation, from a known vehicle and landmark locations:

$$P(\mathbf{z}_t, |\mathbf{x}_t, \mathbf{m}), \tag{3.2}$$

The latter represents how the state evolves in time, meaning how the previous position and control input lead to the new state :

$$P(\mathbf{x}_t, |\mathbf{x}_{t-1}, \mathbf{u}_t), \tag{3.3}$$

To summarize, the vehicle state and the world map will be continuously updated with the motion: new landmarks, perceived through sensors, will be tested for association with already mapped ones. In case of positive outcome, the two landmarks will be associated, otherwise the new one will be added to the map.

The current state-of-the-art relies on three main approaches to tackle the SLAM problem: Kalman filters, particles filters or graphs. Since this thesis focuses only on the third method, the mathematical formulation of graph-based SLAM will be formally stated below, according to [12].

The simultaneous localization and mapping problem can be modeled by means of a graph whose nodes represent the poses of the robot in different time moments, and edges impose constraints between them. Exploiting various sensors, observations of the environment are gathered and used to obtain such constraints.

Solving SLAM means creating a map by finding the spatial configuration of the graph vertices that is most compatible with the measurements represented by edges.

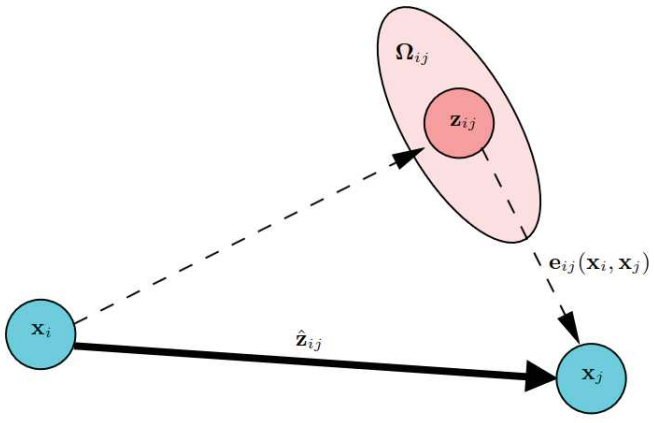


Figure 3.4: Mathematical configuration of the graph used to represent the SLAM problem.

A common graph configuration for the SLAM problem is depicted in figure 3.4. 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 3.4 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 :

$$\mathbf{e}_{ij}(\mathbf{x}_i, \mathbf{x}_j) = \hat{\mathbf{z}}_{ij} - \mathbf{z}_{ij}, \tag{3.4}$$

and it represents, in general, the difference between the expected measurement and the real measurement.

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

$$F(\mathbf{x}) = \sum_{(ij)\in C} \mathbf{e}_{ij}^T \Omega_{ij} \mathbf{e}_{ij} \tag{3.5}$$

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

$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmin}} F(\mathbf{x}) \tag{3.6}$$

Chapter 4

Simulation environment

4.1 Importance of simulation

A simulation is an imitation, over time, of how a process or a system might work in the real world. This is possible thanks to models of the involved process, reflecting its essential traits or behaviors. Any software development activity, in general, benefits greatly from having a controlled environment for testing and evaluating the different components.

Referring specifically to the main focus of this thesis, the simultaneous localization and mapping pipeline, a series of advantages given by the use of a simulator can be listed.

First of all, a virtual environment provides *ground truth data*, essential for proper benchmarking and algorithm evaluation. In SLAM, having such data means having a precise knowledge of the vehicle's real position and of landmarks' configuration. So, by comparing the estimated values with the actual ones, the accuracy and robustness of the proposed solution can be assessed.

Performance assessment is even more consistent if *repeatable experiments* can be carried out: the tested conditions can be precisely replicated thanks to saving and playing back simulated sensors data. In this way, evaluating the effects of the different components of the pipeline becomes simpler.

Sensor simulation is another crucial feature, which allows us to realistically deal with sensors modeling and calibration. By mimicking sensors' noise, distortion, parameter calibrations, and other aspects influencing SLAM performance, the behavior in the real world can be studied in advance with respect to the release time. Moreover, this is extremely useful in cases where the real platform is not available to make continuous or periodic tests.

Safety is a core issue when testing the algorithm in real systems. Especially when tackling problems such as SLAM, navigation, and control of an autonomous system, testing the pipeline in the real world can be dangerous, both for the vehicle and for the surroundings. Experiments with different conditions can be safely performed in the virtual world, without the risk of injuring people or damaging things. So, the amount of errors that could arise when testing in real-world scenario can be minimized through extensive testing in simulation.

Finally, *scalability* and *rapid testing* of many different environmental configurations cannot be underestimated. Having the possibility to validate the solutions on a variety of cases, without requiring any expensive and time-consuming physical installation, allows developers to perform deep testing and and evaluate the accuracy and adaptability of their algorithms.

To conclude, simulation represents a safe, efficient, and cost-effective instrument to develop and validate complex algorithms before releasing them into a real system.

4.2 EUFSSIM

As reported in section 1.2, Formula Student imposes a precisely-structured but challenging environment, where the track is delimited by colored cones and is designed according to specific rules. For this reason, without a realistic simulator, it would be impossible to reproduce scenarios suited for testing.

The Formula Student team of the University of Edinburgh developed and made freely available a simulation platform named EUFSSIM (Edinburgh University Formula Student Simulator) [13]. Of course, only a subset of the entire system has been made open-source. It is based on Gazebo and ROS2 and allows testing software on rule-compliant tracks being part of dynamic events in the competitions. The platform is modular and customizable, meaning that new tracks, new vehicle models, and new sensors can be added. Moreover, the environment is highly-configurable, allowing to change command modes, weather conditions, and vehicle dynamic/kinematic model. In figure 4.1, it can be seen an example of a simulated track, with the custom RaceUP car model inserted.

It is composed of a set of ROS packages and plugins exchanging information according to a state machine structure, designed in compliance with 2020 rules depicted in figure 4.2. Precisely analyzing such schema goes behind the purpose of this thesis, in few words it defines conditions to switch among different car states.

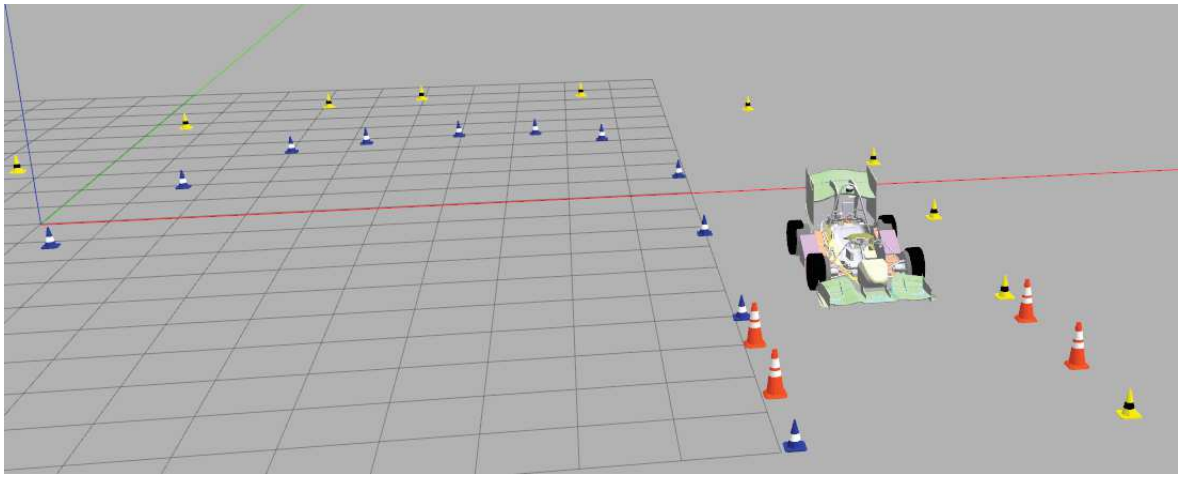


Figure 4.1: An example of simulated car and track.

For example, if the Emergency Brake System is activated, the Autonomous System enters the emergency state and the Ready To Drive is deactivated, together with the Tractive System.

A relevant feature for this thesis is the Manual Driving state. As the name says, it allows to manually drive the vehicle along the track through the provided Graphical User Interface (GUI) or a keyboard. For our pipeline, indeed, it is not strictly necessary having the vehicle running by itself: it is sufficient to get sensor data to perceive the environment and execute all the algorithms.

Coming back to the customization possibilities given by the tool, the Track Generator feature has been exploited to insert a new track into the simulation. This track, depicted in figure 4.3, has been specifically designed for the data acquisition performed with a real car, as will be deeply explained in chapter 7. This module works when an accurately formatted image, representing the layout of the new track, is given as input. This image is a png file, with specific pixels colors in RGBA format encoding the different elements of the track:

- the background of the starting image must be completely white;
- pixels representing yellow cones are encoded with magenta;
- pixels representing blue cones are encoded with blue;
- pixels representing small orange cones are encoded with orange;
- pixels representing big orange cones are encoded with dark orange;
- the pixel representing the car is encoded in green.

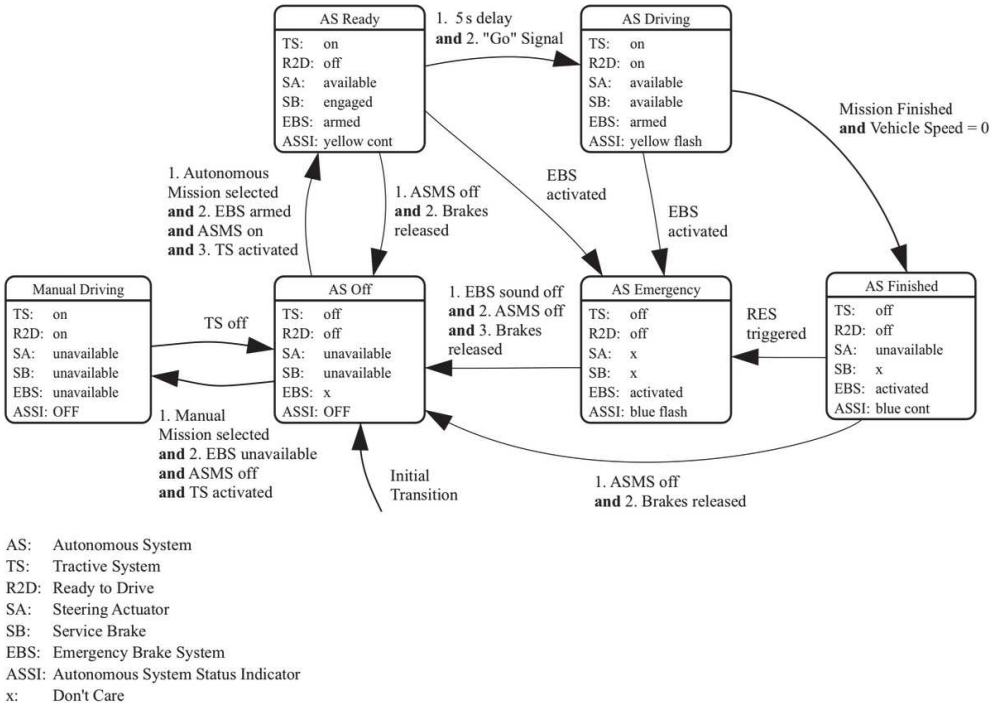


Figure 4.2: The current state machine on which EUFS simulator is based.

Regarding the green pixel, its alpha value is used for encoding the car heading, through conversion to an angle in radians.

The last important step to generate a new track is the computation of the scale pixel value, meaning the (0,0) metadata pixel encoding the dimension of the racetrack. Its value represents the conversion factor between pixels and meters, meaning how many square meters a pixel represents. The conversion is made by taking the pixel value in the range (1, 254), mapping it into a 4-digit base-254 number, and then applying linear interpolation to normalize it into the range (0.0001, 100) in square meters. This image is then used by the track generation module, which creates all the necessary files for the simulation of the environment.

The last modification made to the simulation regards the car model: our custom vehicle has been inserted in place of Edinburgh's team car, by adapting RaceUP's CAD model. Moreover, the sensors mounted on it have been aligned with our setup, in particular for what concerns the simulation of the LIDAR. More details on the sensorization of the car will be given in chapter 7.

Going more into technical details, the simulator can be utilized in two configurations, depending if the simulated perception is activated or not. In the first case, the output of the perception are not raw sensor measurements, but directly the processed results that one would have applying front-end algorithms,

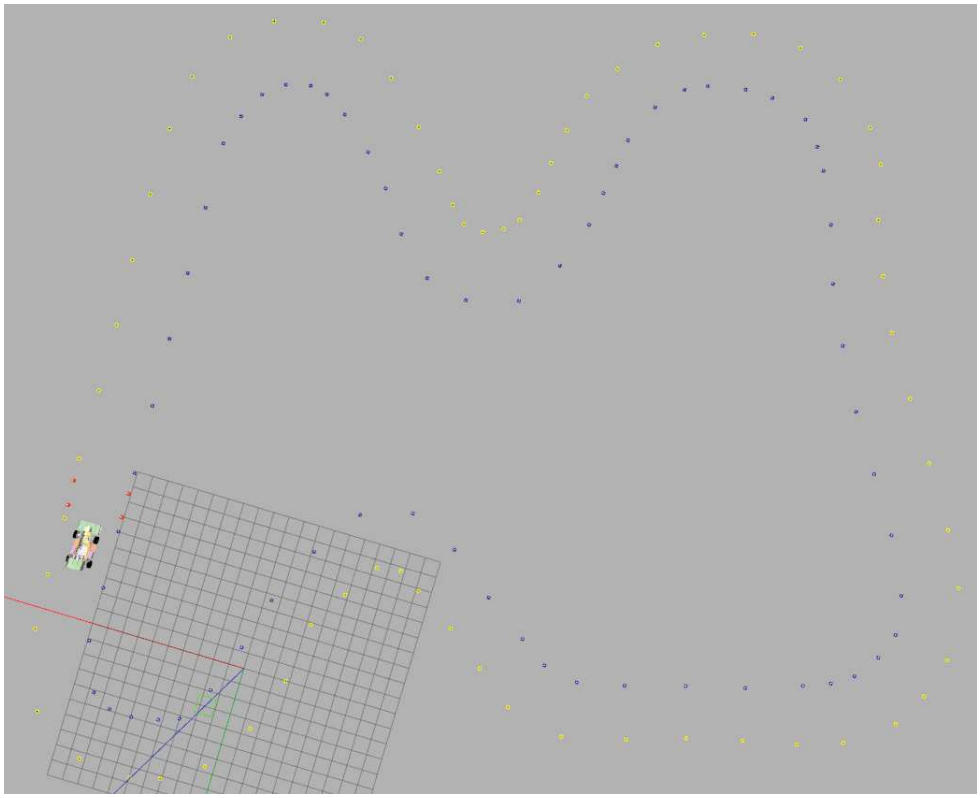


Figure 4.3: The custom track, inserted into the simulator.

like the coordinates of cones' positions. In the second case, raw sensor data is available, such as the LIDAR point clouds, to be processed by the user's perception pipeline. To clarify this concept, a simple example is presented. If a SLAM algorithm has to be tested, it only needs a set of coordinates representing the cones: in this case, the simulated perception is perfect since it directly provides such data. If, instead, a cone detection algorithm must be evaluated, the user can deactivate the simulated perception, in order to process raw data from camera and/or LIDAR.

Data publishing is handled by two plugins, one for cones-related information, and the other for vehicle data like kinematic state or transformations between reference frames.

The majority of this thesis has been developed by exploiting the simulated perception data, in particular cones positions and colors, and odometry data. About this last information, we started by using the already integrated odometry, meaning that the simulator gives directly the car pose as output. Then, to make the proposed solution more adherent to reality, we switched to directly integrating kinematic commands, as explained in section 5.2.3.

Chapter 5

GraphSLAM

As explained when reviewing the theoretical background in chapter 3, a SLAM module is composed of a front-end and a back-end. The main focus of my thesis is on the latter, so on the component which deals with the refinement of the robot's state given the measurements coming from different sensors, and the creation of a map that represents the surrounding environment.

In the latest years, graph-based back-ends for simultaneous localization and mapping have become more and more popular due to the advantages with respect to other available solutions. As highlighted in chapter 2, in recent years the tendency of the most competitive teams is to switch to a graph-based approach, confirming the more promising results of this algorithm. First of all, the graph-based formulation of the problem allows to exploit the natural sparseness of the SLAM problem, making fast problem-specific operation possible. Secondly, the graph-based SLAM solves the full SLAM problem, optimizing also past information. Keeping past information is fundamental for two reasons: it improves the state estimation, and it makes the optimization process more resilient to outliers. Finally, being a very general mathematical formulation, it is easy to extend to particular use cases.

5.1 Graph construction

As a side note before starting the discussion of this chapter, in this context *vertex* and *node* of a graph are used as synonyms.

There are many different types of graph-based optimizable structures, depending mainly on the information that they hold and on the construction technique.

The mathematical basis for all of them is the so-called *factor graph*.

It is a generic structure, a weighted graph that can hold different types of nodes and edges altogether, allowing the encoding of a wide set of constraints. Each node represents a state variable, for example, a robot position or a landmark in the environment, while edges represent measurements. Thanks to this generic structure, optimization is not limited to the vehicle's state, but can also be extended to the environment map.

A particular case of factor-graph is the pose graph, shown in figure 5.1: it contains only vertices representing the robot's state over time, and edges connecting pairs of such poses. When dealing with this type of representation, only the robot state can be optimized, since it is the only data included in the problem formulation.

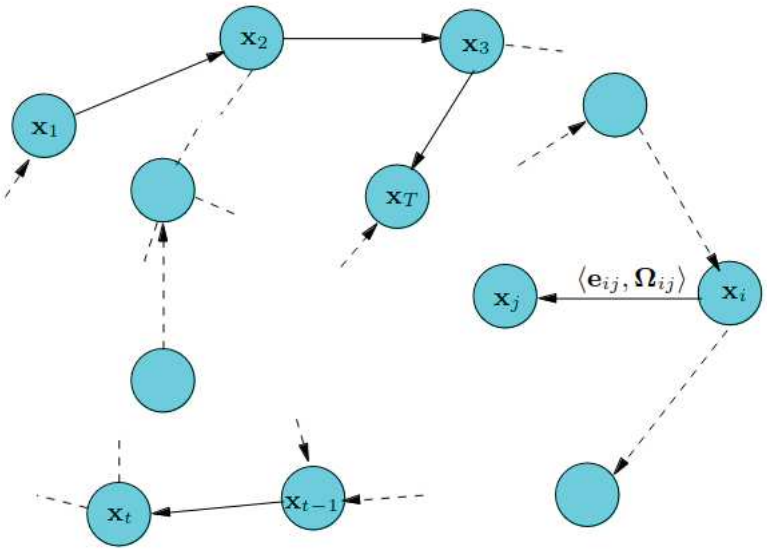


Figure 5.1: The pose graph structure.

In the case of SLAM, we are interested in both estimating the trajectory of the robot and building a map of the environment, which is usually expressed as a set of landmarks. Instead of having only vertices related to the state of the robot, now they are also used for representing landmarks. Similarly, we will impose landmark and odometry edges, holding their respective observations. The former connects a vehicle's pose with all the landmarks observed from that point in the map, while the latter connects two pose nodes, holding motion data as measurement. Such structure is depicted in figure 5.2.

Focusing on the Formula Student Driverless scenario, the factor graph is implemented following the structure reported in figure 3.4 and mathematically

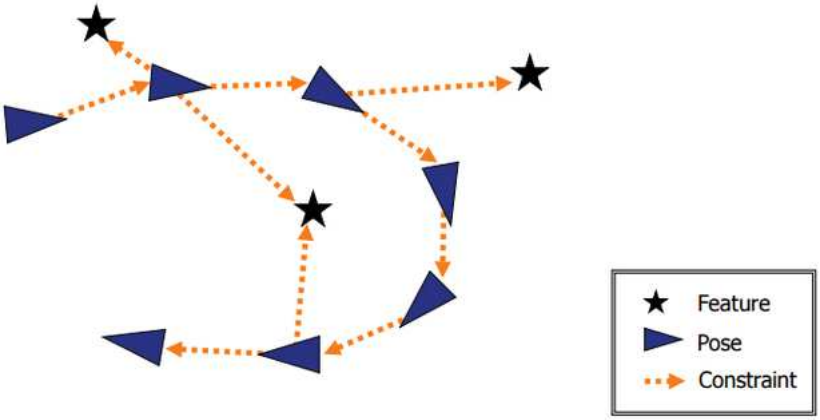


Figure 5.2: The factor graph structure.

described in chapter 3. Each landmark node represents a cone position, parameterized with (x, y) coordinates, while each pose vertex holds the car's pose described by (x, y, θ), where θ is an orientation angle denoting the heading of the vehicle.

Analyzing the graph edges, each of them encodes a spatial constraint between the two vertices it connects, and in particular, this implementation involves:

- odometry edges linking two consecutive car pose nodes x_i and x_j;
- odometry edges linking two non-consecutive car poses, also named loop closure constraints (see section 5.2.5);
- landmark edges, having as extremes a pose vertex and a landmark one.

Later, another custom type of edge will be added to the graph to make the motion more realistic, as explained in section 5.2.6.

Finally, measurements need to be assigned to each mathematical constraint: odometry edges are defined by the relative 2D rigid body transformation between the two poses, while landmark edges hold the relative observation of the landmark from the current position in the map.

A deeper explanation of the proposed implementation is now reported, giving more details about the actual graph construction in terms of processing and usage of incoming sensor data.

In the first experiment, ground truth data coming from the simulator has been used to populate the graph. Then, once ensured that the system was working at least in an ideal scenario, noisy data have been added by perturbing cones and

car positions with Gaussian noise. The graph construction pipeline is exactly the same in both cases, so it will be illustrated only once.

The basic idea is to synchronize information coming from the perception pipeline with the one coming from the odometry of the car, in order to compute an estimate of the current pose of the vehicle and an estimate of the positions of the cones observed at each time. For efficiency and processing time reasons, incoming data has been subject to time quantization according to a parameter that can be tuned depending on the available computational resources.

First, consider the odometry information processing, to deal with the portion of the graph encoding the trajectory of the vehicle. This part is very straightforward since a new pose node is created and inserted into the factor graph whenever a new car state can be computed: as a brief reminder, the car state is characterized by position and orientation in a certain time instant. Edges connecting two pose vertices are simply characterized by their relative spatial transformation.

Then, consider the insertion into the graph of landmark vertices and edges: perceived cones' positions and colors received from the front-end pipeline are converted into the map reference frame and incorporated into the world model.

An initialization procedure is performed only once when the graph is still empty. All the colored cones observed during the first scan are directly inserted as landmark nodes, without further controls. Contextually, a landmark edge connecting the initial pose with each observation is added, holding the relative coordinates of the perceived cone with respect to the car reference frame as measurement. This special step is done to avoid pointless checking for data association for these measurements, given that the vehicle is still in the initial map pose and there isn't any landmark yet.

If the graph is already initialized, whenever a new cone is perceived, the data association algorithm is run. By doing this, we detect if the vehicle is observing a new landmark or if it is just observing an existing one. Details about how data association is performed will be explained in section 5.2.4. Depending on the output of this algorithm, we can distinguish two different ways of modifying the graph:

- if the observed cone is associated with an existing one, then only a landmark edge between the current pose and the associated landmark vertex is added;
- if we are not able to associate it with existing landmarks, then a new cone node is created and added to the graph, together with the corresponding edge, as explained before.

The last thing worth noting about the construction of the factor graph is the decoupling between data collection and graph construction and modification. Data collection consists in gathering sensor measurements regarding the perceived environment and storing them for later use. Graph construction means putting this data into a mathematical formulation, to construct an optimizable structure representing the problem.

The idea is to have two separate threads, one that is continuously active in the foreground and the other one running in the background. The former is in charge of collecting sensors output and storing it immediately, without the risk of information loss, while the latter process the queue data in a safe way.

This feature is not fully implemented yet, but a simplified version is used to handle data collection during the optimization phase, which modifies the structure of the graph. For this reason, incoming data received when optimization is running is kept pending until it has been completed. In this way, concurrency issues that may arise from the simultaneous modification of the structure can be avoided.

5.2 Algorithm implementation

5.2.1 g^{2}o framework

Before starting with the details about the implemented SLAM algorithm, the library used for the factor graph optimization will be introduced.

To begin with, we provide an explanation for the crucial role of optimization in solving SLAM. As depicted in figure 5.3, the estimated trajectory of the vehicle does not align perfectly with the actual trajectory. This discrepancy arises due to errors introduced by factors such as sensor noise, and accumulated drift. Consequently, the estimated positions of landmarks also deviate from their actual locations. The purpose of optimization in SLAM is to tackle this issue. By establishing mathematical constraints between graph vertices, and leveraging the motion and measurement models, it becomes possible to minimize the estimation error.

g^{2}o [15] stands for "general graph optimization". It's an open-source state-of-the-art general optimization framework, largely exploited to solve many popular problems related to graph optimization, such as SLAM and bundle adjustment. The framework generality allows for easy extensibility, making it possible to define new types of problems, as long as they can be formulated as least squares

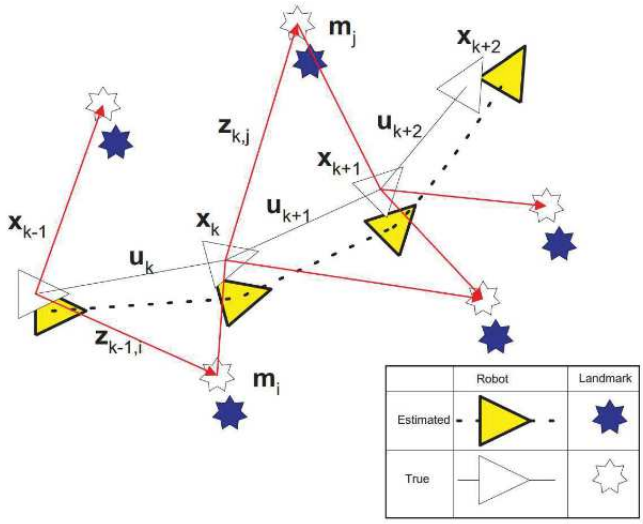


Figure 5.3: Why we need to optimize: landmarks being observed at different positions along the robot's trajectory [14].

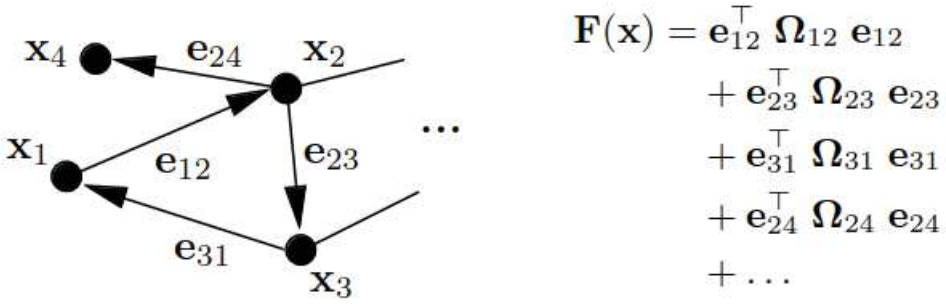


Figure 5.4: How to represent an objective function by a graph: each x_i is a parameter block, Ω_{ij} is the information matrix of the constraint relating parameters x_i and x_j, e_{ij} is a vector error function measuring how well x_i and x_j satisfy the constraint z_{ij}.

optimization problems. Being that these problems can be represented as graphs, as depicted in figure 5.4, defining new problems means that appropriate nodes and error functions need to be chosen or even defined if they don't exist already. Despite being general and extensible, another important advantage given by g^{2}o is its efficiency, obtained by exploiting the sparse connectivity of the graph and the particular structure recurring in the aforementioned problems.

In the particular context of this thesis, g^{2}o is used for the creation, handling, and optimization of the factor graph holding 2D positions of the traffic cones and 2D poses of the car. Going into implementation details, the following data types are used for the construction of the graph:

- `g2o::VertexPointXY` are landmark nodes described using the following state variable $\mathbf{x}_{l_j} = [x, y]$;

- `g2o::VertexSE2` are the nodes that are going to be used to represent the car 2D poses. It encodes the state variable $\mathbf{x}_{r_i} = [x, y, \theta]$;
- `g2o::EdgeSE2` for odometry edges, that connect two nodes $\mathbf{x}_{r_i}$ and $\mathbf{x}_{r_j}$. The observation $\mathbf{Z}_{ij}$ is an isometry that encodes the transformation from pose $\mathbf{x}_{r_i}$ to $\mathbf{x}_{r_j}$.
  The error that describes the edges is formulated as follows :

$$\mathbf{e}_{ij}(\mathbf{x}_{r_i}, \mathbf{x}_{r_j}) = (\mathbf{X}_{r_j}^{-1}\mathbf{X}_{r_i})\mathbf{Z}_{ij}, \tag{5.1}$$

  where the uppercase bold notation represents the transformation matrix encoding the pose of the robot;
- `g2o::EdgeSE2PointXY` for observation edges, connecting a pose node $\mathbf{x}_{r_i}$ to a landmark node $\mathbf{x}_{l_j}$. The observation

$$\mathbf{z}_{ij} = [x_l, y_l]$$

  is the relative position of the cone with respect to the car. The error that describes this kind of edge is formulated as follows :

$$\mathbf{e}_{ij}(\mathbf{x}_{r_i}, \mathbf{x}_{l_j}) = \mathbf{X}_{r_i}^{-1} \begin{bmatrix} x_{l_j} \\ 1 \end{bmatrix} - \mathbf{z}_{ij}. \tag{5.2}$$

As a brief reminder, in mathematics $SE(2)$ denotes a special Euclidean group of dimension 2. It is a group of direct Euclidean isometries, that is transformations in the Euclidean space that preserve the distance between any two points and the handedness of figures. More formally, given two spaces X and Y, where the metrics d_X and d_Y are defined, a map $f : X \to Y$ is called *isometry* if, for any a, $b \in X$ it holds

$$d_X(a, b) = d_Y(f(a), f(b)) \tag{5.3}$$

For this reason, only translations and rotations are allowed in this group, not reflections [16].

5.2.2 Graph optimization

Optimizing a factor graph means finding the configuration of robot and landmark positions that is most consistent with the observations in the edges. In this work, two different ways of executing graph optimization have been studied and tested.

The first one involves performing only a global optimization once every a certain amount of time, or when loop closure has been detected. Global optimization means that the whole graph is optimized to find the node's configuration minimizing the error function over the entire map. So, measurements and constraints provided by all the edges are considered in the computation of the error.

The second technique is an incremental procedure, meaning that local optimization is run once every certain number of pose nodes has been inserted into the graph. Local optimization means that only a small subset of the graph is optimized at each time, finding the configuration of nodes that best satisfies the constraints in a precisely defined neighborhood. In this way, just the last part of the map is optimized, without taking into consideration the entire trajectory since the beginning of the motion. In addition, a periodic global optimization is performed, as in the first case.

The main advantage of the second approach is to detect and correct errors locally, reducing their impact on the integration along with the vehicle's motion. Once the optimizer has corrected a portion of the graph, all the nodes composing it are set as fixed: this means that those vertices will be no more involved in the subsequent local optimizations, and their values will be kept unchanged until the next global optimization is run.

Also from a conceptual point of view, local optimization of relative poses is more suitable to the scenario we are dealing with, since measurements and car states are available in real-time.

In both cases, Levenberg-Marquardt has been select as optimization algorithm, in combination with a block solver to resolve the linearized system. This choice is pretty common when dealing with non-linear optimization problems since it is a trade-off between the Gauss-Newton method [17], faster but less robust, and the gradient descent procedure [18], which is more robust but slower.

Qualitative and quantitative results obtained from both approaches will be exposed in section 5.3.

5.2.3 Motion model

In the specific context of this thesis, and following the Ackermann motion model [19], the variables exploited for motion integration are:

- the position of the vehicle, represented by x and y coordinates;
- the orientation of the vehicle, represented by the angle θ;

- the linear velocity;

- the steering angle.

Together with the car state, we need to retrieve some important kinematic parameters, fundamental to perform motion integration: car wheelbase, linear acceleration, and steering angle velocity, that is the steering rate of the autonomous system.

Once the car state and all the necessary parameters are available, odometry data can be integrated according to the following equations:

$$\begin{aligned} \dot{x} &= v * \cos\theta \\ \dot{y} &= v * \sin\theta \\ \dot{v} &= a \\ \dot{\delta} &= \phi \\ \dot{\theta} &= v/W * \tan\delta \end{aligned} \tag{5.4}$$

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:

$$\begin{aligned} x_{t+1} &= x_t + \dot{x} * dt \\ y_{t+1} &= y_t + \dot{y} * dt \\ \theta_{t+1} &= \theta_t + \dot{\theta} * dt \\ v_{t+1} &= v_t + \dot{v} * dt \\ \delta_{t+1} &= \delta_t + \dot{\delta} * dt \end{aligned} \tag{5.5}$$

where dt is the time passed between sensor readings.

The odometry edges' measurements are computed according to the kinematic model aforementioned. The implementation of this model will allow a more direct transition from the simulator, which was already providing motion integration using the Ackermann model, to a real application.

5.2.4 Data association

Data association tries to solve a correspondence problem. Considering the SLAM scenario, it is the process of relating observations coming from the sensors with elements of the environment. It plays a key role in estimation pipelines, strongly influencing the quality of the convergence of the problem's result. If the association is not correct, indeed, the error of the system will explode, making the optimization diverge.

Despite being crucial for the performance of the entire pipeline, many difficulties are encountered when trying to solve it: detection uncertainty, occlusion, ambiguities, multiple targets, and so on.

Two main approaches are usually applied to tackle the data association problem, Bayesian or Non-Bayesian. The former computes a full probability distribution from priors, posterior beliefs, and observations, the latter computes a maximum likelihood estimate from a set of possible solutions.

What we have chosen to apply in this work is Nearest Neighbor filtering, which belongs to the non-Bayesian family. The idea is to compute the distance between the expected measurement, computed from the current estimates of landmarks and poses, and the received one. In our specific scenario, when a cone is observed, it is matched against all the landmark nodes already in the graph. It is then associated with the nearest one if their distance is below a certain threshold.

This approach is quite simple to implement, but it may integrate some wrong measurements and produce overconfident estimates. Despite that, it works pretty well if the current estimates are accurate enough. Combined with an incremental local optimization of the graph, results could improve a lot.

From a theoretical point of view, Mahalanobis distance is the more appropriate metric to use since it considers position, uncertainty, and correlations. However, in this thesis, we tested the data association using the Euclidean distance, which takes into account only the position. As a future step, it would be interesting to make a comparison between the use of the Euclidean distance and the use of the Mahalanobis distance for the association.

5.2.5 Loop closure

Loop closure is probably the most crucial task when performing simultaneous localization and mapping: it has the purpose to identify previously visited locations and use them to correct the accumulated drift on the trajectory. If a loop

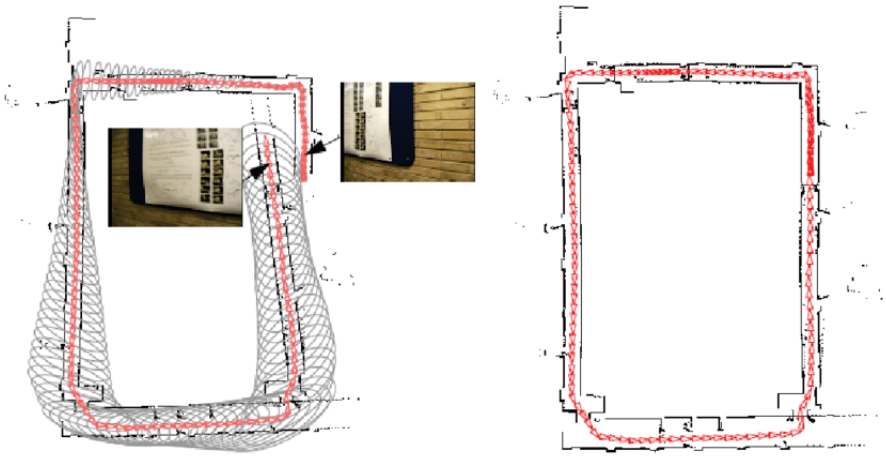


Figure 5.5: Example of loop closure.

closure has been detected, the current pose estimate $\mathbf{x}_{t_i}$ will be constrained to a previous pose $\mathbf{x}_{t_j}$, where $t_j < t_i$. The constraint consists on adding spatial relation between the two poses, due to the fact that they are in the same location. Thanks to this additional constraint, the drift accumulated from t_j to t_i will be corrected as shown in figure 5.5. Practically, it consists on adding to the graph an odometry edge between $\mathbf{x}_{t_i}$ and $\mathbf{x}_{t_j}$, which has as a measurement the relative transformation provided by the loop closure module.

In general scenarios, loop closure is achieved by exploiting camera-based methods or LIDAR-based ones.

In vision-based techniques, distinctive features, like corners and edges, can be extracted from camera images exploiting algorithms like SIFT [20], SURF [21], and ORB [22]. Descriptors of these features can then be matched against the ones in the previous images, taking as reference a database of features that has being built over time. If correspondence goes above a certain threshold, then the two images can be considered framing the same place. For example, using a Bag Of Words [23] approach, matching the descriptors to visual words of a fixed-size vocabulary, the autonomous vehicle is able to recognize already visited places.

Similarly, point clouds coming from a LIDAR sensor can be aligned with a set of previously captured point clouds, that represents a candidate for loop closure. If the alignment with the minimum score is successful, meaning that the aforementioned score is under a defined threshold, a loop is detected. Iterative Closest Point Algorithm [24] is a typical algorithm used to perform point cloud alignment.

Unfortunately, in our particular scenario, none of these methods gave satisfac-

tory results: the presence of large untextured areas, the ambiguity of the cones, and the extreme sparsity of the point clouds caused by the low-resolution LIDAR made impossible the successful extraction and matching of relevant features using classic techniques.

For this reason, in my thesis, the first version of loop closure detector has been implemented in a very simple and effective way: exploiting the prior knowledge that we have about the Formula Student racing environment. Since we know that big orange cones are used to mark the start of the track, once we detect them as explained in section 5.2.5, an odometry edge between the current pose and the initial pose of the graph is added. The proposed solution works under the assumption that the big orange cones are positioned only at the starting line of the track. A future improvement could be making the module more resilient to outliers, and to dynamic events that could break our working assumption.

5.2.6 Additional motion constraint

An autonomous platform, whether robot or vehicle, is said to be holonomic if it can freely move in any direction in space, meaning that the controllable degrees of freedom are equal to the total degrees of freedom. This particular condition can be realized thanks to special types of wheels, such as omnidirectional and Mecanum ones, depicted in figure 5.6a and 5.6b respectively.

A Formula Student single-seater is a non-holonomic vehicle. As a consequence, some movements such as the translations along the perpendicular to the direction of motion, are not allowed. The odometry constraint that we enforce on the motion is more general, and it does not consider the banned movements.

A non-holonomic vehicle, indeed, moves along a circumference whose center is called Instantaneous Centre of Rotation (ICR), as you can see in figure 5.7. The ICR is defined as the point in the robot frame that instantaneously does not move in relation to the robot [25].

Therefore, the trajectory of the race car can be decomposed into a sequence of consecutive displacements around this fixed point and can be parameterized on the radius of curvature ρ and the traveled angle ω. Formally, ρ is the vector connecting the center of the rear axis of the car with the ICR, and it is collinear with the posterior axle itself. ω, instead, is the angle between the vector ρ and the vector connecting the ICR with the center of the front axle.

To enforce motion around an ICR constraint, a custom type of edge named `g2o::Edge2ICR` has been implemented and defined using Eq(5.9) and added to

(a) Omnidirectional wheel. **(b)** Mecanum wheel

Figure 5.6: Examples of wheels for holonomic motion.

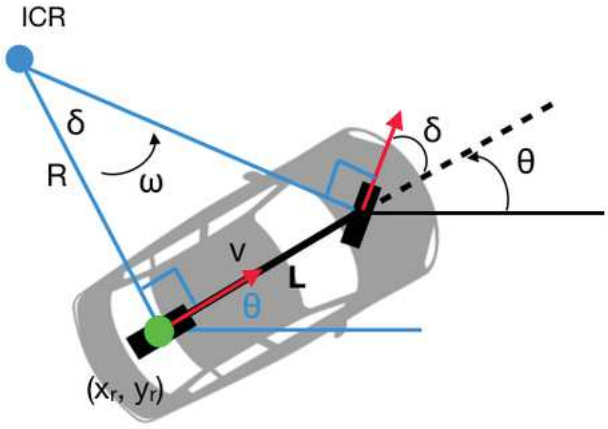


Figure 5.7: Simple representation of ICR.

the graph. In this way when optimizing, the solutions that have infeasible car motions will be discarded.

The mathematical derivation of the constraint, added between consecutive pairs of poses at time t and $t+1$, is as follows:

- compute the relative motion between the two poses

$$\begin{aligned} dx &= x_{t+1} - x_t \\ dy &= y_{t+1} - y_t \\ d\theta &= \theta_{t+1} - \theta_t \end{aligned} \tag{5.6}$$

- compute the motion magnitude

$$\rho = \sqrt{dx * dx + dy * dy} \tag{5.7}$$

- compute the expected motion

$$\begin{aligned} \dot{dx} &= \rho * \cos\frac{d\theta}{2} \\ \dot{dy} &= \rho * \sin\frac{d\theta}{2} \end{aligned} \tag{5.8}$$

- compute the residual

$$\mathbf{e}_{icr}(x_t, x_{t+1}) = \left[\dot{dx} - dx, \dot{dy} - dy\right]' \tag{5.9}$$

Finally, the resulting constraint can be expressed as

$$\mathbf{e}_{icr}(x_t, x_{t+1})^T \Omega \mathbf{e}_{icr}(x_t, x_{t+1}) \tag{5.10}$$

where $\Omega = \begin{bmatrix} 1/\rho & 0 \\ 0 & 1/\rho \end{bmatrix}$ is the $2x2$ diagonal information matrix.

5.3 Algorithm evaluation

Evaluation of the proposed implementation can be performed both qualitatively and quantitatively. The idea is to compare the output of the simultaneous localization and mapping algorithm with the ground truth provided by the simulator. More than comparing the obtained values with the performance reached by other

algorithms, the goal of this evaluation is to show the improvements deriving from the addition of the different blocks, such as the incremental graph optimization. A true comparison with other benchmarks, by the way, is not actually possible, since no other implementations of SLAM modules specific to the Formula Student Driverless scenario are available.

In the first part of this section, metrics used to evaluate the proposed algorithm are illustrated. In the second part, instead, qualitative and quantitative results arising from the various experiments will be shown.

5.3.1 Metrics

It is necessary to establish a clear distinction between the evaluation of the acquired trajectory and the evaluation of the acquired map concerning the detection and mapping of cones.

To check the quality of the cones' positions estimated, the following metrics are computed:

- ratio between the number of mapped cones and the actual number of cones in the considered track. This metric will give a first idea about how well the data association algorithm performed. If the mapped cones are too much with respect to the ground truth ones, it means that new landmarks are generated, even when they should be associated with already existing ones. The possible reasons for this behavior to occur are the following: the error metric between a new candidate landmark and existing ones is too high with respect to a threshold, the corresponding threshold is constraining too tightly the error, or the chosen error metric for the data association inadequately represents the problem.

- percentage of True Positive, True Negative, False Positive, and False Negative associations, computed as the ratio between the number of occurrences of each event and the total number of association tests performed during the mapping phase.

It is worth spending a few words to deepen the four metrics related to data association. To compute these values, association tests executed with ideal and estimated data are compared.

- True Positive(TP): if the association using ideal and estimated information links the new measurement to the same existing cone.

- True Negative(TN): if the association using ideal and estimated information both agree that the new measurement represents a new cone.

- False Positive(FP): if the association using ideal and estimated information links the new measurement to a different existing cone. Or also when the association using ideal information deems that the received measurement corresponds to a new cone, while instead the one using estimated information wrongly associates it to an existing cone.

- False Negative(FN): if the association using ideal information is able to find a correspondence, while the association using estimated information deems that the measured cone is actually a new landmark.

Regarding the estimated trajectory, instead, two state-of-the-art metrics widely used to evaluate SLAM benchmarks are considered: Absolute Pose Error, also known as Absolute Translation Error, and Relative Pose Error [26].

Absolute Position Error, or APE, measures the error of the vehicle position with respect to the ground truth trajectory. Before the computation, the two trajectories must be aligned one with the other, since they can be specified in arbitrary coordinate frames. With this metric, global consistency is tested, by checking the absolute error between two trajectories.

Relative Pose Error, or RPE, measures the error of the transformation between two consecutive robot poses, with respect to the given ground truth relative transformation. The goal is to check the local consistency of the motion, without caring about global measurements.

More formally, given a sequence of poses composing the estimated trajectory $\mathbf{P}_1, ..., \mathbf{P}_n$ and the ground truth one $\mathbf{Q}_1, ..., \mathbf{Q}_n$, the relative pose error at time step i, over the time interval Δ, is defined as :

$$\mathbf{RPE}_i = (\mathbf{Q}_i^{-1}\mathbf{Q}_{i+\Delta})^{-1}(\mathbf{P}_i^{-1}\mathbf{P}_{i+\Delta}) \tag{5.11}$$

Instead, given an alignment procedure that produces the rigid-body transformation $\mathbf{S}$, to map the estimated trajectory $\mathbf{P}_{1:n}$ into the ground truth trajectory $\mathbf{Q}_{1:n}$, the absolute trajectory error at time step i is

$$\mathbf{APE}_i = \mathbf{Q}_i^{-1}\mathbf{S}\mathbf{P}_i \tag{5.12}$$

5.3.2 Experiments and results

Four experiments have been executed to test our proposed SLAM pipeline. For each of them, qualitative results on both the obtained map and on the estimated trajectory are presented. In the end, also two tables with quantitative metrics will be analyzed, to incrementally compare the results of the different configurations.

For the graph images, green dots represent the discretized poses of the vehicle. Red dots, instead, represent the mapped cones, expressed in the map reference frame. The orange, blue and yellow markers, finally, are placed on the ground truth positions of the cones in the track.

The first configuration that has been implemented includes motion integration according to Ackermann model, 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, by exploiting the ideal ground truth coordinates provided by the simulator.

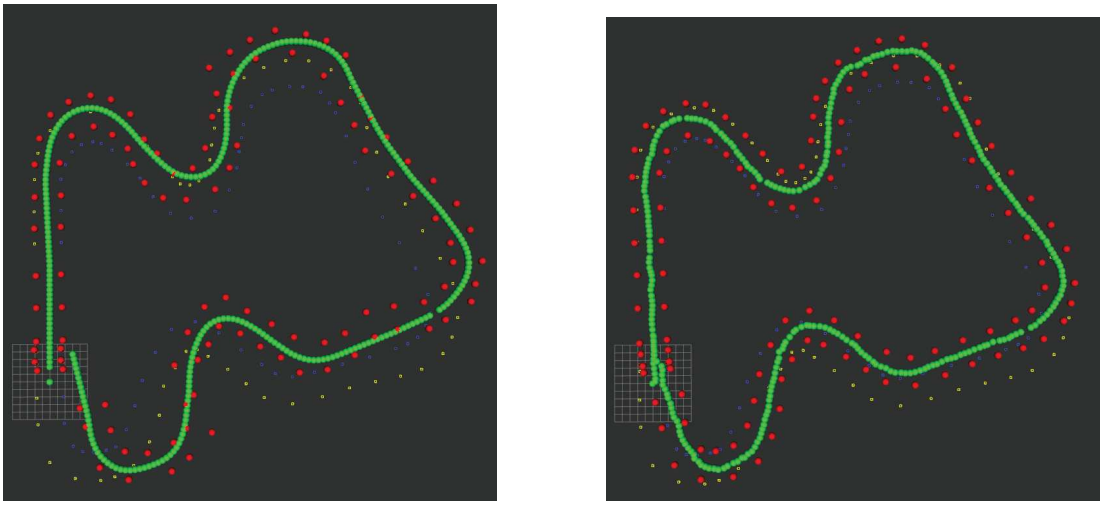
Starting from qualitative results, a visualization of the graph produced by the mapping module is reported in figure 5.8.

As expected, it is easily noticeable how optimization improves the map. This is due to the constraints given by the data association and the loop closure algorithm, which allows to reduce the drift that was accumulated in the trajectory. In fact, the start and the end point of the trajectories meet after the optimization procedure converges.

The superimposition with the ground truth track is not particularly relevant from a map-quality point of view, since the misalignment is due to a small difference between the two reference frames. The important aspect is that the shape of the trajectory is preserved, indicating successful convergence.

By exploiting [27], visual evaluation charts for trajectories have been generated. In the top left corner of figure 5.9, a visualization of RPE is depicted: the dashed line is the ground truth trajectory, while the colored one is the estimated vehicle's path. Here, the two trajectories have not been aligned, coherently with the metric's definition. According to the legend, the more bluish the line, the lower the RPE. As expected, it is low along the entire trajectory, except from the last corner where the error is slightly higher. This is due to the fact that there is a small synchronization issue at the beginning of the simulation.

In the top right corner, instead, APE is shown. The results show that data integration (motion and landmarks) is handled correctly since the correct shape of the track is retrieved. It can be seen from the fact that the estimated trajectory



(a) Before optimization

(b) After optimization

Figure 5.8: EXP1: map and optimization results

is almost perfectly aligned with the reference one in figure 5.9a. In table 5.1 are reported the different quantitative metrics, that later will be commented on to confirm the correct execution of the SLAM pipeline.

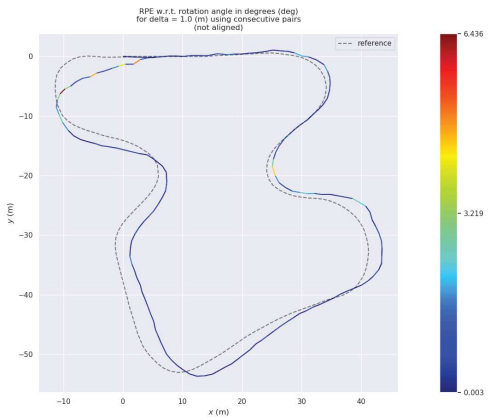
Lastly, in the bottom part of the figure, a comparison between real and optimized values of the components of the state x, y, and θ is reported. This small variation between the compared values further confirms our claims.

The second configuration that has been implemented adds the ICR constraint with respect to the scenario of experiment 1, and is depicted in figure 5.10. As a brief reminder, ICR is an additional constraint added between pairs of pose nodes to ensure the executability of the estimated motion for a non-holonomic vehicle.

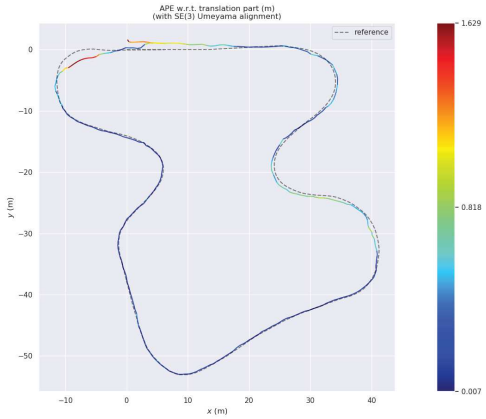
By looking at the obtained map, results are comparable with respect to the previous case: the optimization still works well, and the loop is correctly closed.

Analyzing the metrics, ICR is not particularly influent on the relative pose error, and results on the relative consistency are comparable with the previous case. The absolute pose error, instead, gives better performance adding this additional constraint, reducing the error of about 20 centimeters in mean. Implying that the constraint has a positive impact on the global consistency of the map.

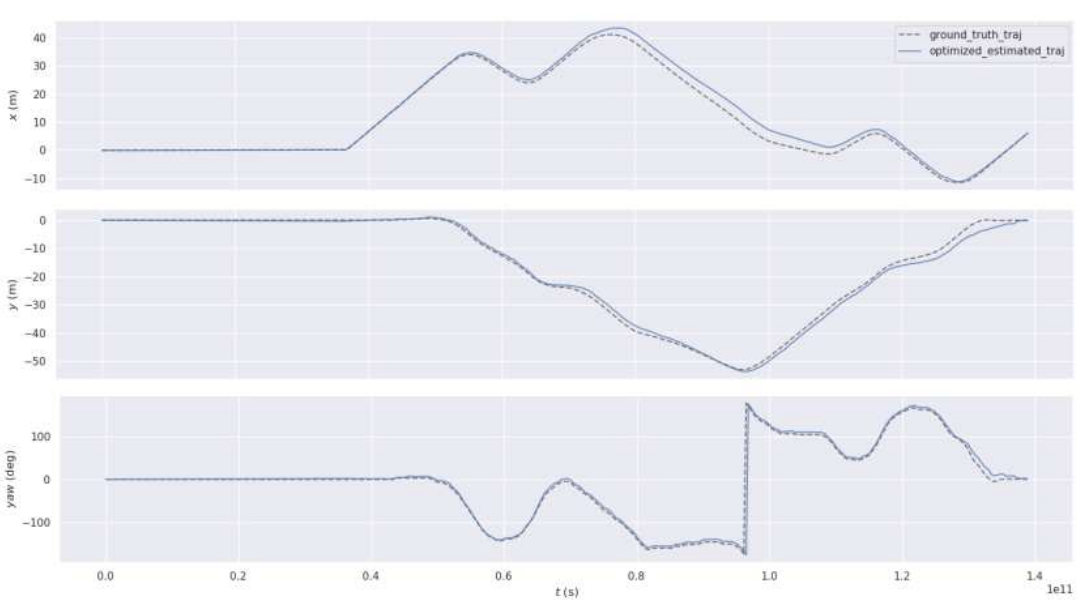
The third configuration that has been implemented adds the incremental optimization with respect to the scenario of experiment 2. This means that the graph is no more optimized only globally, at the end of the lap, when loop closure is detected. Now, once every a fixed number of new pose nodes have been inserted, the local optimization is performed on the active portion of the graph, that is composed mainly by the latest nodes. The method follows a sliding-window style



(a) RPE w.r.t. rotation angle (in deg) for delta=1.0 (m) using consecutive pairs

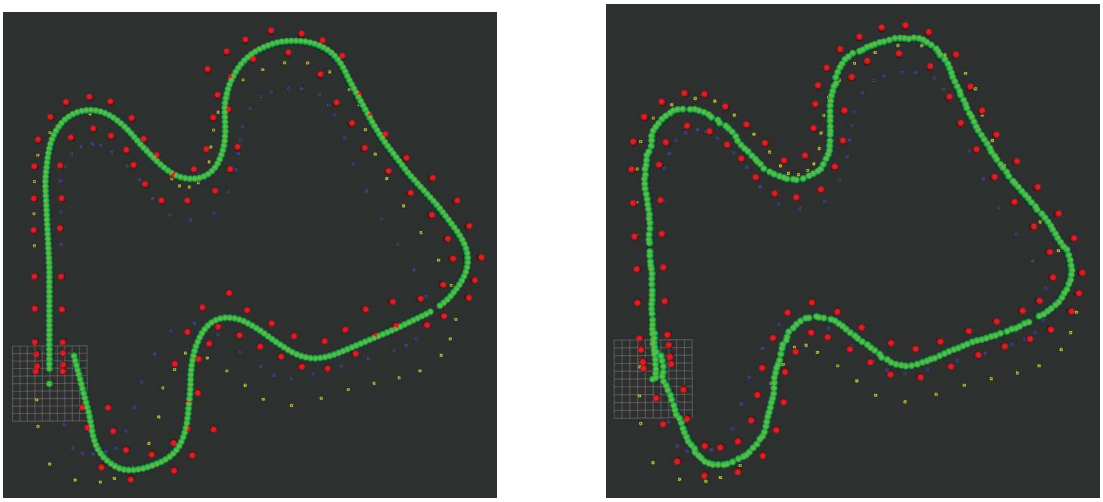


(b) APE w.r.t. translation part (m) (with SE(3) Umeyama alignment)



(c) Trajectory

Figure 5.9: EXP1: trajectory results



(a) Before optimization

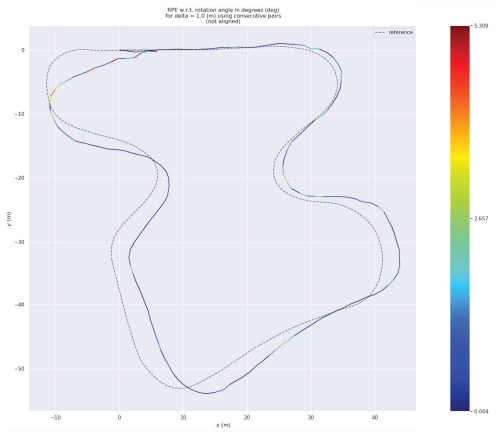
(b) After optimization

Figure 5.10: EXP2: map and optimization results

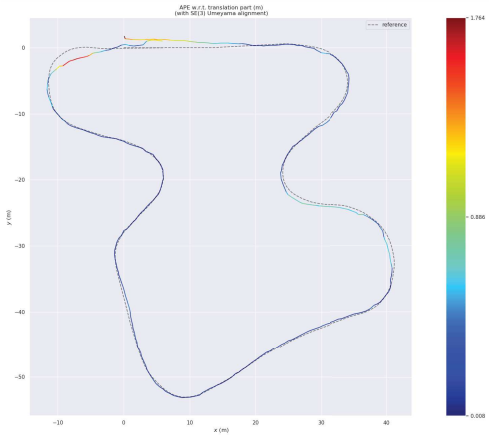
approach. As soon as the latest set of nodes is optimized, they are then set to fixed, in order to exclude them from the future optimizations, reducing time and complexity of the problem to solve. With this incremental approach, error is corrected periodically, allowing the global optimization to converge faster due to being closer to the optimum value.

As expected, both qualitative and quantitative results show a huge improvement on the performance of the simultaneous localization and mapping algorithm, resulting in lower trajectory estimation error, and higher quality of the created map. In particular, observing the graph in 5.12b, an almost perfect estimation of the cones has been performed in the majority of the track. The high quality results are also noticeable in 5.13a and 5.13b, where the last corner estimation reports higher fidelity of the trajectory with respect to the previous cases.

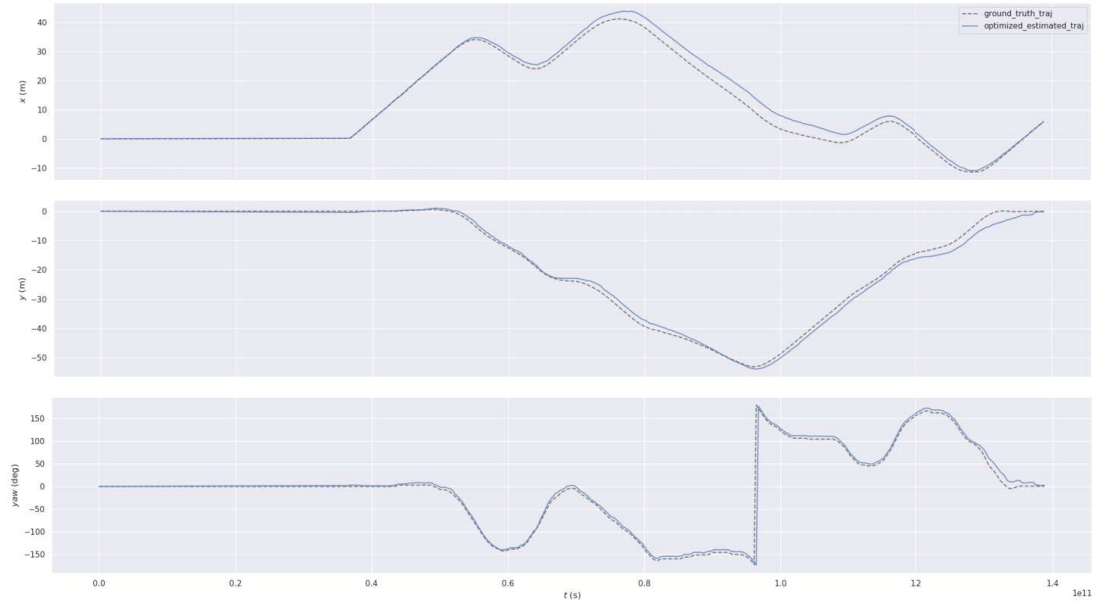
The fourth configuration that has been implemented replaces the ideal data association with the real one, computed with Euclidean distance. This means that observed cones are tested for association with already mapped ones, by comparing the estimated coordinates, and no more the ground truth data. As expected, here the performance of the system gets worse results, because we add an error in the association of observations and landmarks, introducing local minimas in the solution space. This can be immediately noticed by looking at the optimized graph in 5.14b, where an anomaly remains in the last section of the track even after the optimization steps. This behaviour confirms that Euclidean distance is not the most appropriate technique to perform such association, since it does not consider important aspects like the correlation between the different



(a) RPE w.r.t. rotation angle (in deg) for delta=1.0 (m) using consecutive pairs

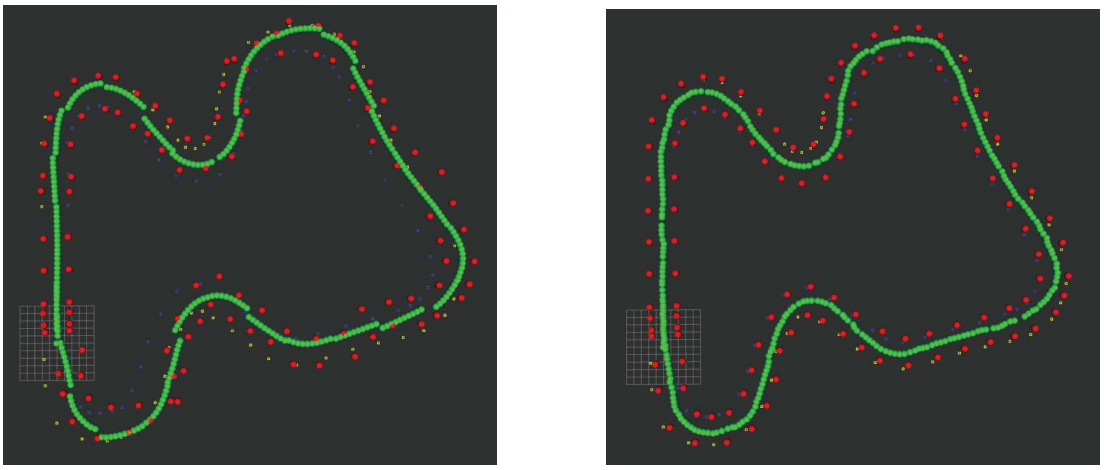


(b) APE w.r.t. translation part (m) (with SE(3) Umeyama alignment)



(c) Trajectory

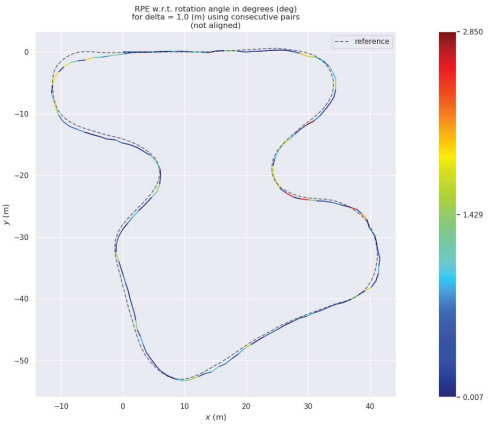
Figure 5.11: EXP2: trajectory results



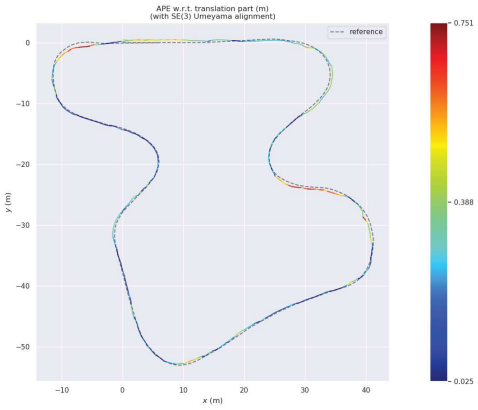
(a) Before optimization

(b) After optimization

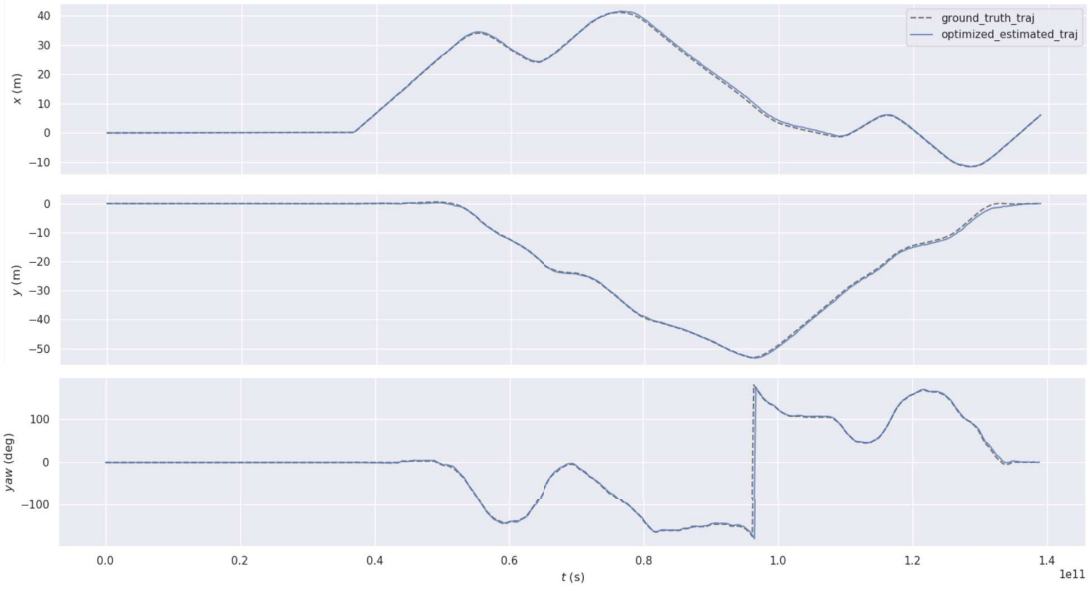
Figure 5.12: EXP3: map and optimization results



(a) RPE w.r.t. rotation angle (in deg) for delta=1.0 (m) using consecutive pairs

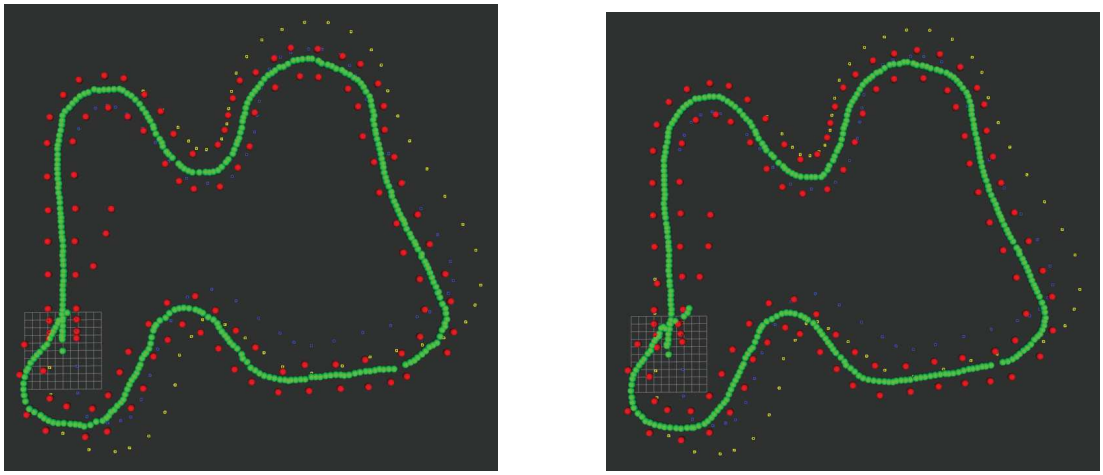


(b) APE w.r.t. translation part (m) (with SE(3) Umeyama alignment)



(c) Trajectory

Figure 5.13: EXP3: trajectory results



(a) Before optimization

(b) After optimization

Figure 5.14: EXP4: map and optimization results

		EXPERIMENT_1	*EXPERIMENT_2*	*EXPERIMENT_3*	*EXPERIMENT_4*
RPE [m]	***max***	6.435884	5.309379	**2.850375**	10.299785
	min	**0.002899**	0.003774	0.006694	0.004104
	mean	0.819888	0.871201	**0.729945**	0.875245
	rmse	1.293808	1.301008	**0.952769**	1.450978
	std	1.000861	0.966246	**0.612330**	1.157273
APE [m]	***max***	1.629269	1.764260	**0.751472**	5.490941
	min	**0.007230**	0.007921	0.025071	0.113987
	mean	0.421073	0.401338	**0.288106**	1.076611
	rmse	0.545399	0.536665	**0.329692**	1.399580
	std	0.346638	0.356282	**0.160288**	0.894278

Table 5.1: Comparison of trajectory metrics in the four experiments.

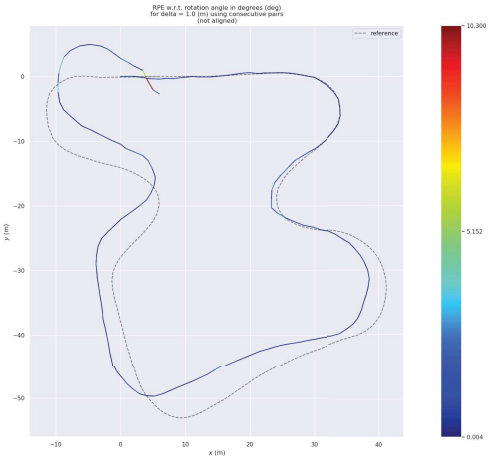
elements and how the system evolves.

In tables 5.2 and 5.1, numeric data about the aforementioned results are reported, confirming what has emerged from the plots. In the first one, metrics about trajectory evaluation are listed, while the second one contains cones-related values. For each evaluation criterion, the best value among the four experiments is highlighted in bold.

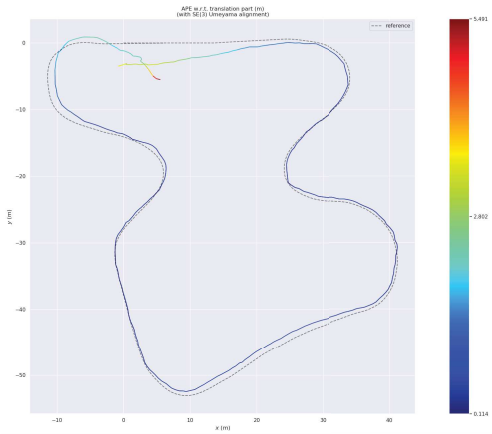
Now the final setup, the one used in experiment 4, will be tested on two other virtual tracks taken from the simulator, denoted as TRACK_2 and TRACK_3. Both qualitative and quantitative results on TRACK_2 show that our approach can generalize quite well on sufficiently-constrained environments, so it is not

	EXPERIMENT_1	*EXPERIMENT_2*	*EXPERIMENT_3*	*EXPERIMENT_4*
Mapped/GroundTruth	113/134	113/134	113/134	**115/134**
True Positive (TP)	80.11%	81.26%	**87.78%**	83.11%
True Negative (TN)	3.11%	3.18%	**2.88%**	3.13%
False Positive (FP)	11.06%	10.11%	**8.16%**	13.49%
False Negative (FN)	5.72%	5.45%	1.18%	**0.26%**

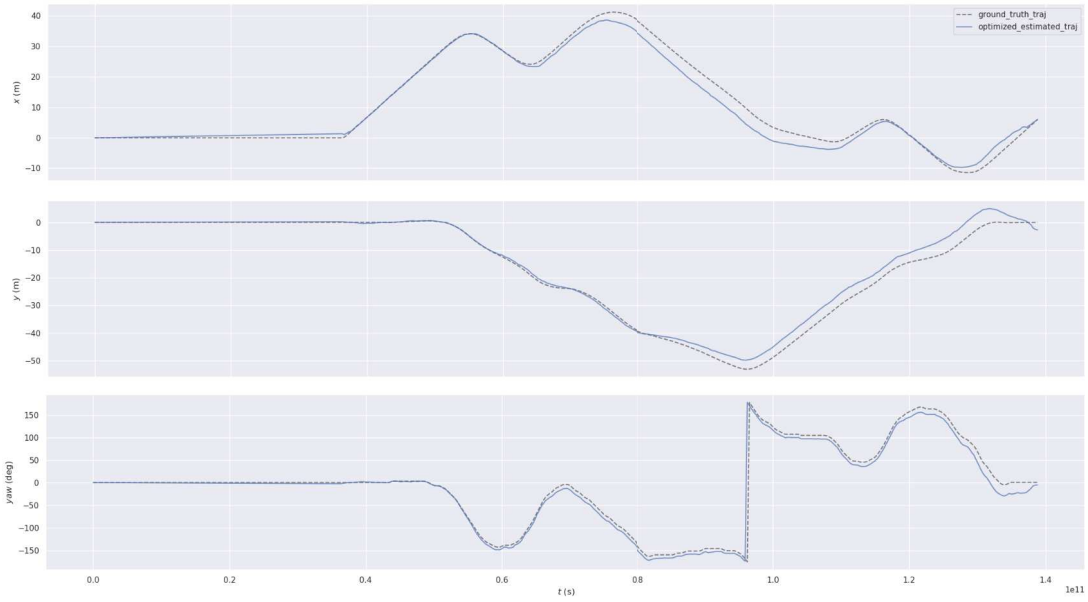
Table 5.2: Comparison of cones metrics in the four experiments.



(a) RPE w.r.t. rotation angle (in deg) for delta=1.0 (m) using consecutive pairs



(b) APE w.r.t. translation part (m) (with SE(3) Umeyama alignment)



(c) Trajectory

Figure 5.15: EXP4: trajectory results

dependent on the track shape. In TRACK_3, instead, results are quite worse, and also the loop closure has not been correctly handled.

In both tracks, all the limitations of our data association algorithm arise:

- in TRACK_2, we map a number of cones that is lower than the actual one, meaning that we have a lot of false positive associations;
- in TRACK_3, we map a number of cones that is slightly higher than the actual one, but still acceptable.

Analyzing the resulting trajectories, for TRACK_2 we obtain satisfactory results both regarding the absolute pose error and the relative pose error, meaning that the algorithm performs quite well in this track configuration. For TRACK_3, there is a bigger misalignment with respect to the ground truth trajectory, and also the loop closure has not been correctly handled. The hypothesis is that there are not enough constraints in the graph for the optimizer, so the shape is not retrieved as well as in the previous cases.

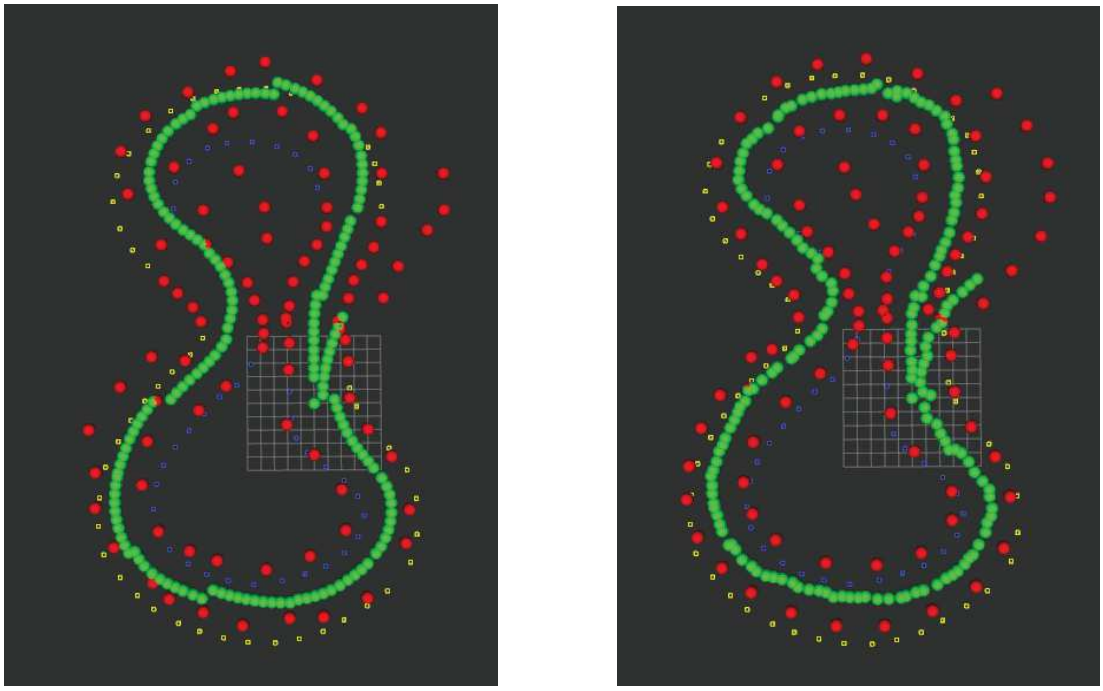
Performing a more general analysis, these results confirm what we actually stated at the beginning of this work: data association is fundamental to have a robust pipeline. The back-end alone is not able to correct errors deriving from wrong associations, and the front-end is an essential requirement to have a system working correctly.

		TRACK_2	*TRACK_3*
RPE [m]	***max***	7.559016	5.070571
	min	0.052654	0.036058
	mean	1.859469	1.025821
	rmse	2.344713	1.470736
	std	1.428305	1.053924
APE [m]	***max***	3.333464	6.223878
	min	0.115991	0.043243
	mean	0.800667	1.728495
	rmse	0.933397	2.017538
	std	0.479752	1.040560

Table 5.3: Comparison of trajectory metrics for TRACK_2 and TRACK_3

	TRACK_2	*TRACK_3*
Mapped/GroundTruth	92/122	73/71
True Positive (TP)	60.70%	74.82%
True Negative (TN)	0.89%	2.51%
False Positive (FP)	37.74%	21.67%
False Negative (FN)	0.67%	1.00 %

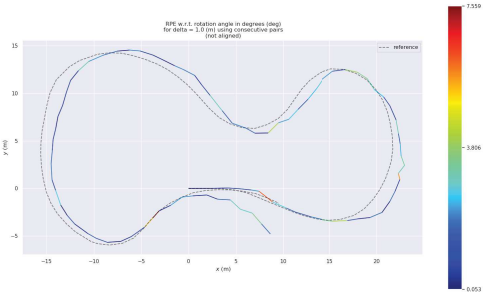
Table 5.4: Comparison of cones metrics for TRACK_2 and TRACK_3



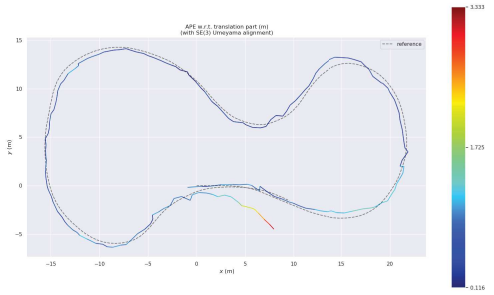
(a) Before optimization

(b) After optimization

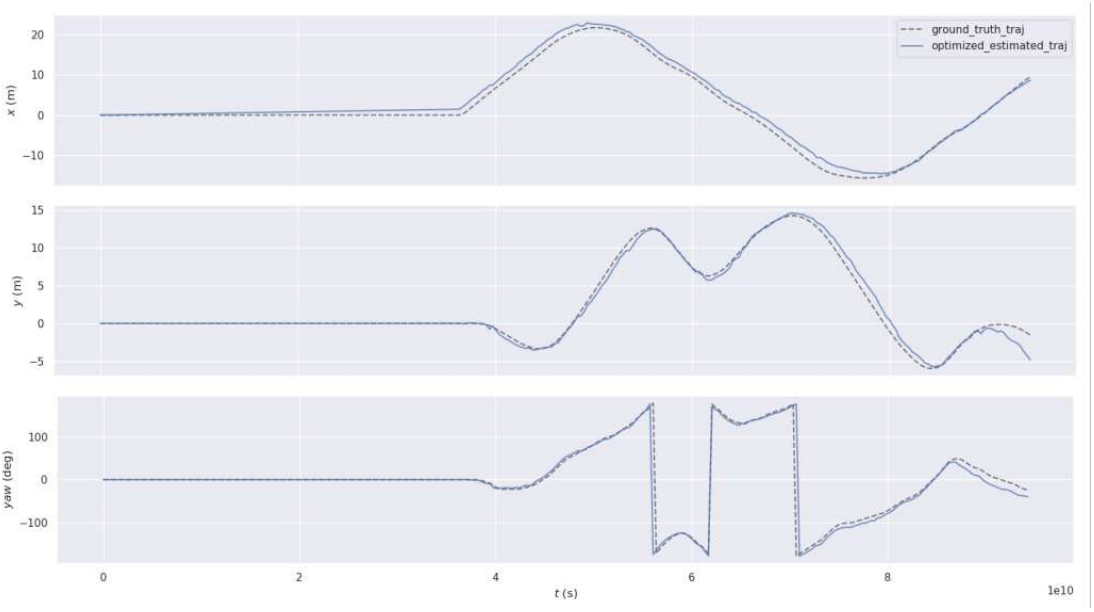
Figure 5.16: TRACK_2: map and optimization results



(a) RPE w.r.t. rotation angle (in deg) for delta=1.0 (m) using consecutive pairs

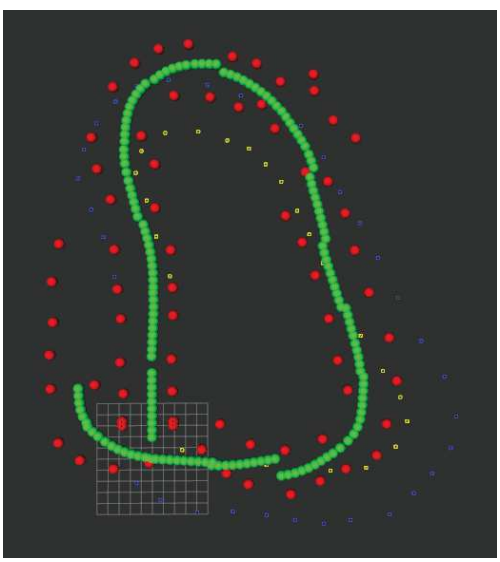


(b) APE w.r.t. translation part (m) (with SE(3) Umeyama alignment)

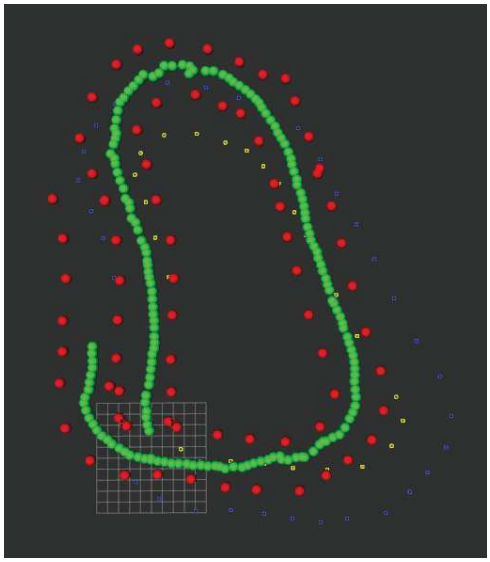


(c) Trajectory

Figure 5.17: TRACK_2: trajectory results

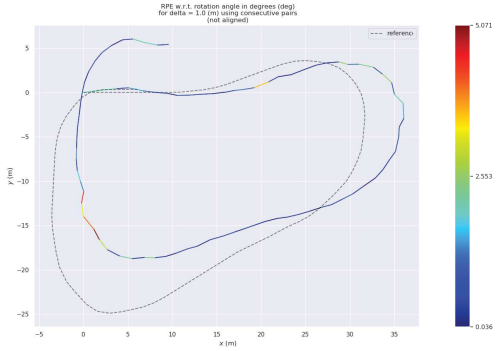


(a) Before optimization

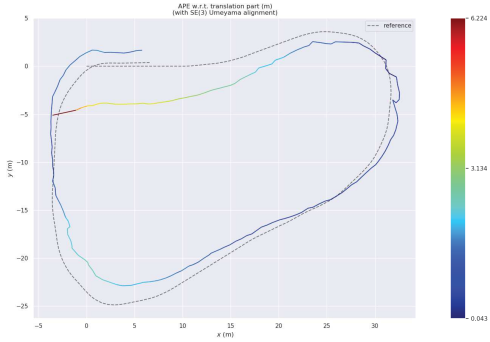


(b) After optimization

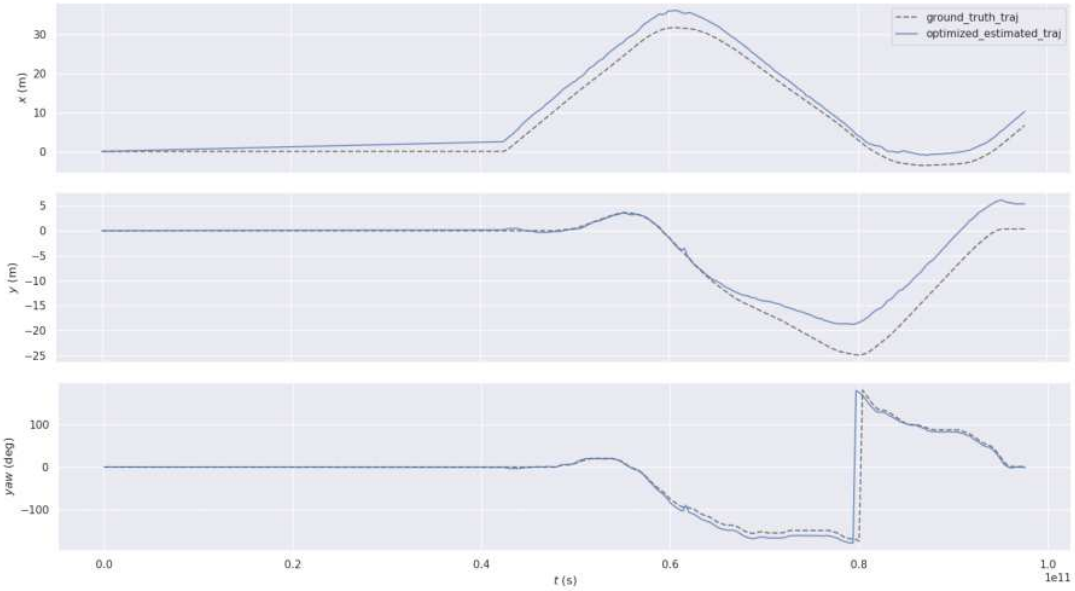
Figure 5.18: TRACK_3: map and optimization results



(a) RPE w.r.t. rotation angle (in deg) for delta=1.0 (m) using consecutive pairs



(b) APE w.r.t. translation part (m) (with SE(3) Umeyama alignment)



(c) Trajectory

Figure 5.19: TRACK_3: trajectory results

Chapter 6

Perception building blocks

Perception is a fundamental task for autonomous driving and usually provides input data to the localization and mapping algorithms. Developing a full front end for the SLAM module goes beyond the purpose of this thesis, but some smaller blocks have been tackled, putting the bases for the actual perception pipeline. Following the trend that emerged from other teams, the idea of fusing together LIDAR-based and camera-based detections has been pursued.

The first section of this chapter will focus on LIDAR data processing, to detect cones from the point clouds taking inspiration from MUR Motorsport team [28]. For this part, raw sensor data produced by the simulator will be used. The last section, instead, will deal with image-based cones detection, exploiting the well-known YOLO object detector.

In both cases, the results are partial and only qualitative, since these modules are just a draft and are not completely implemented and tested.

6.1 LIDAR-based cones detection

Starting from raw point clouds produced by the LIDAR, the objective is to end up having the detection of cones, in particular the cylindrical volume in which each of them is enclosed.

When processing points, however, a problem arises: the LIDAR in our possession has only 16-channels, so its resolution is too low to be able to detect a cone using the setup described in 7.1. Each of them, indeed, is hit by too few scanlines, resulting in an insufficient number of points.

For this reason, the first step of the detection algorithm is point cloud accumulation over time: this means that, before processing points, a fixed number of

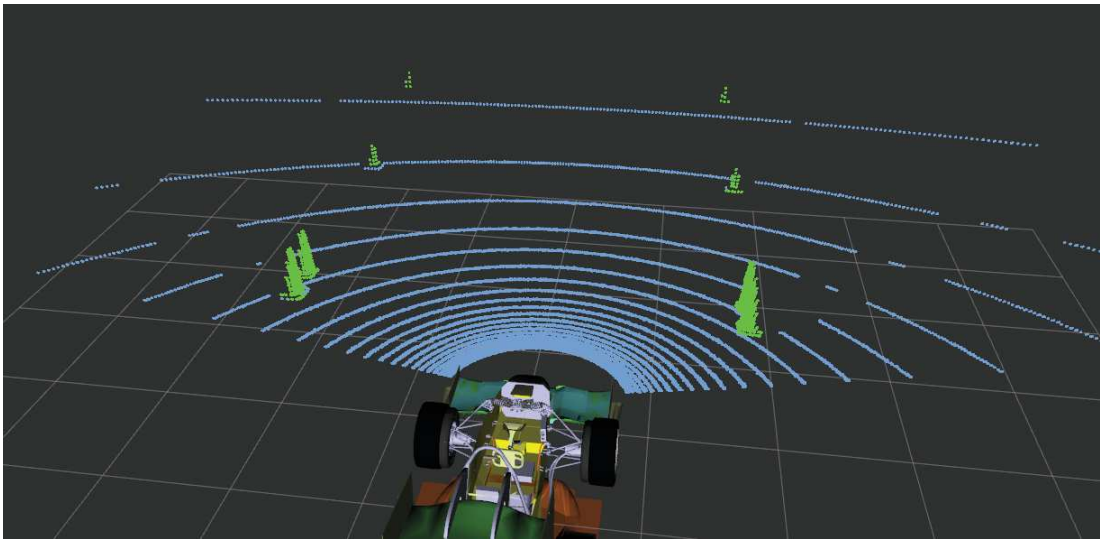


Figure 6.1: Qualitative results of the ground-plane segmentation step.

clouds are accumulated and integrated together, in order to have a denser cloud. Accumulation is executed by considering the motion of the vehicle over time, in order to cover a bigger part of the cones' surface with LIDAR scans.

The second step is to perform segmentation of the point cloud, that will allow to distinguish the actual cones from the ground plane. To satisfy real-time requirements, the segmentation has been performed according to Himmelsbach et al. [29]. Two smaller sub-problems are addressed: local ground plane estimation and 2D labeling of connected components. It is a fundamental operation for reducing the computational complexity of the process. Point clouds are not considered as a whole: the xy-plane is represented as a circle of infinite radius, split into a certain number of segments. This gives an ordering of points with respect to their angular component, and we can denote the set of points contained in a segment S as P_S. To discretize also the range component of the points, all the elements in P_S are then mapped into one of the many bins, each one covering a certain range.

Then, for each segment, line extraction is performed, based on conditions such as maximum slope and maximum distance from the previous line. Any point within a designated distance from a line is classified as ground, resulting in a complete labeling of a scan's points.

Qualitative results of this step are depicted in 6.1.

After ground removal, filtering and clustering of the remaining data are performed to remove potential outliers and reconstruct the cylindrical volume occupied by a cone. Cylindrical volume reconstruction is a process needed to restore

some points belonging to the traffic cones but accidentally removed during the ground removal step. If these points are inside a cylinder-shaped volume, they are added back to the corresponding cluster. So, using Euclidean clustering based on Kd-Tree [30], all the non-ground points are grouped together, according to some thresholds like the expected number of points. By exploiting the known dimensions of the cones, the number of expected points in a certain cluster can be computed as:

$$E(d) = \frac{1}{2} E_v E_h = \frac{1}{2} \frac{h_c}{2d \tan(\frac{r_v}{2})} \frac{w_c}{2d \tan(\frac{r_h}{2})} \tag{6.1}$$

where:

- d is the distance to the cluster, being $\sqrt{x^2 + y^2 + z^2}$ with (x, y, z) centroid of the cone;
- r_v is the vertical resolution of the LIDAR;
- r_h is the horizontal resolution of the LIDAR;
- w_c is the width of the cone;
- h_c is the height of the cone.

Once all the clusters have been retrieved and filtering has been applied, their centroids can be computed and used as an estimate of the cones' positions.

The high-level pseudocode for this approach is reported here:

Algorithm 1 Cones detection from point clouds

Input: Raw LIDAR point clouds
Output: 3D positions of cones
accumulate n point clouds over time
perform cloud segmentation
perform points labeling (ground, non-ground)
remove non-ground points
perform Euclidean clustering
perform cylindrical volume reconstruction
compute each cluster's centroid to have estimated cone position

A limit of this approach is observed when cones are occluded one by each other, cases in which the algorithm is not able to detect all the cones because the LIDAR beams cannot hit hidden objects.

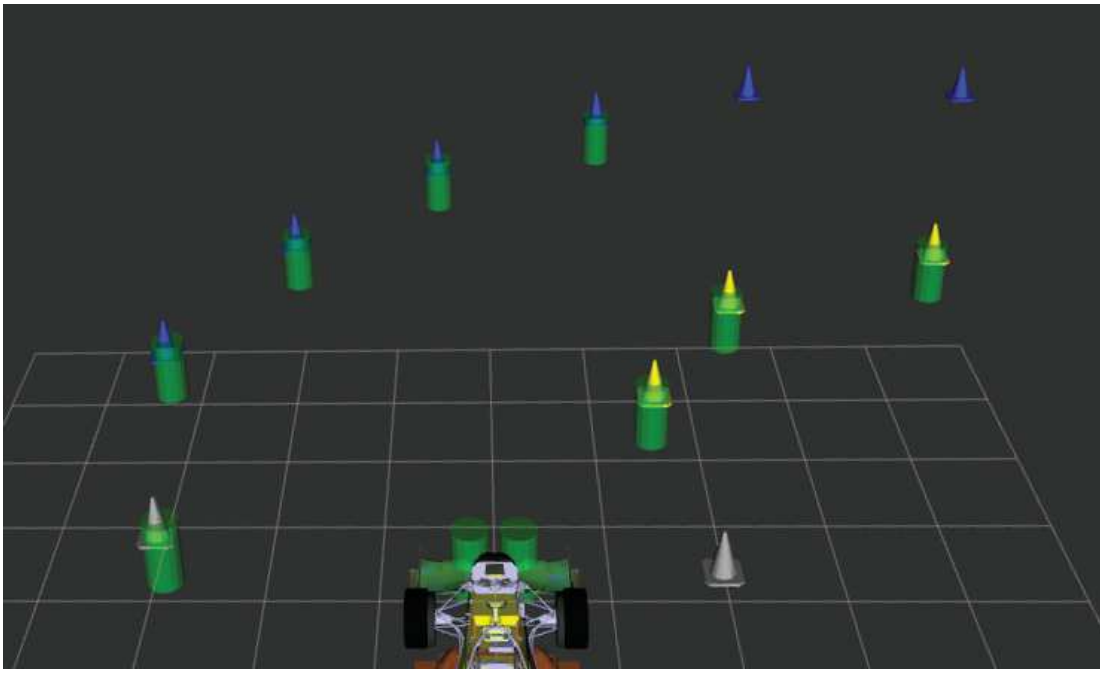


Figure 6.2: Qualitative results of cylindrical volume reconstruction step.

An example showing the computed cylinders can be seen in figure 6.2. As you can notice, two spurious detections appear in that image: this is caused by the LIDAR hitting the structure of the front wing of the car, and should be removed with future developments of this pipeline.

Anyway, the proposed solution gives satisfactory results, as shown in figure 6.3: a complete track lap has been completed here, and the majority of the ground truth cones, depicted in blue and yellow, have been correctly detected.

A further improvement of the proposed solution would come from a point cloud pre-processing step, to remove distortion caused by the velocity of the motion.

6.2 Camera-based cones detection

To process stereo images acquired with RaceUP's single-seater, a deep learning approach has been selected. More specifically, real-time detection of cones delimiting the track is performed by exploiting YOLOv7 [3]. The goal is to detect cones, together with their color, and provide a bounding box enclosing each of the detected landmarks.

YOLO is a state-of-the-art real-time object detector, trained entirely on MS COCO dataset [31], without pre-trained weights. With respect to previous ver-

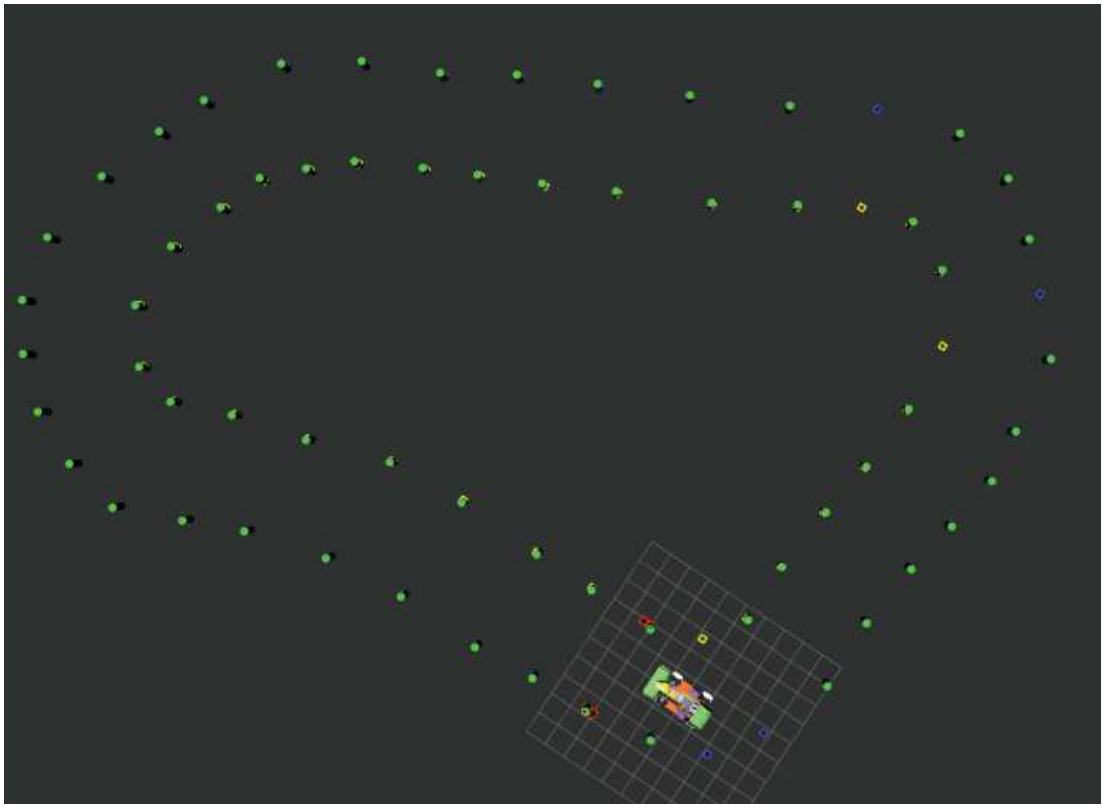


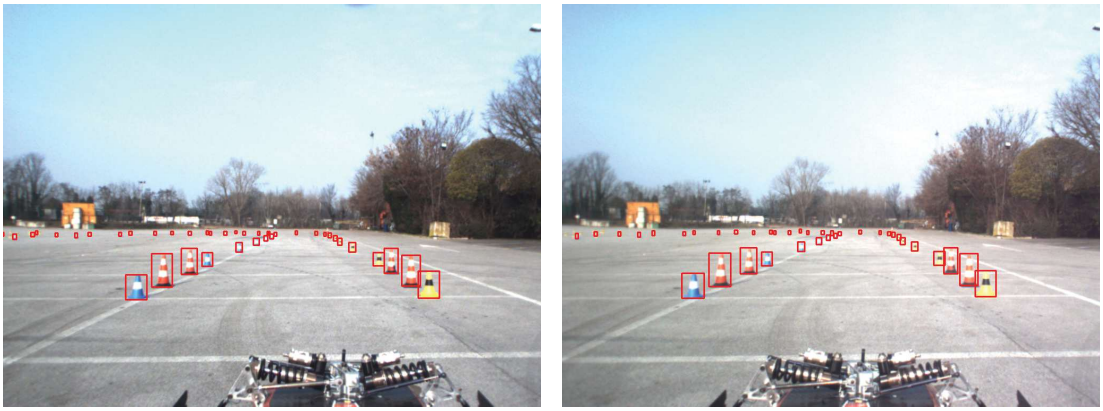
Figure 6.3: Qualitative results of the complete detection pipeline on a lap.

sions, it highly reduces the number of parameters of the model and the number of computations. This algorithm optimizes both the architecture and the training phase, introducing some modules called *bag-of-freebies*, to improve detection accuracy without increasing the inference cost.

The general-purpose model has been fine-tuned on the FSOCO dataset [32], to detect the different official cones used in Formula Student events. Many teams participating in this competition have contributed to the release of such dataset, which contains images of real racing scenes, framed during driverless events or test sessions. Annotations involve both bounding boxes and instance segmentation, provided for 11572 and 1517 images, respectively.

The choice of performing the fine-tuning on this dataset has been dictated by different reasons:

- FSOCO in the only Formula Student-specific dataset available;
- many teams contributed to this, providing a huge variety of testing and event sites;
- images come from a variety of sensor setups, meaning that the detection is



(a) Left image. **(b)** Right image.

Figure 6.4: YOLO cones detection from stereo images.

not sensor-dependant;

- images have been framed in diverse lighting conditions, to tackle even challenging scenarios.

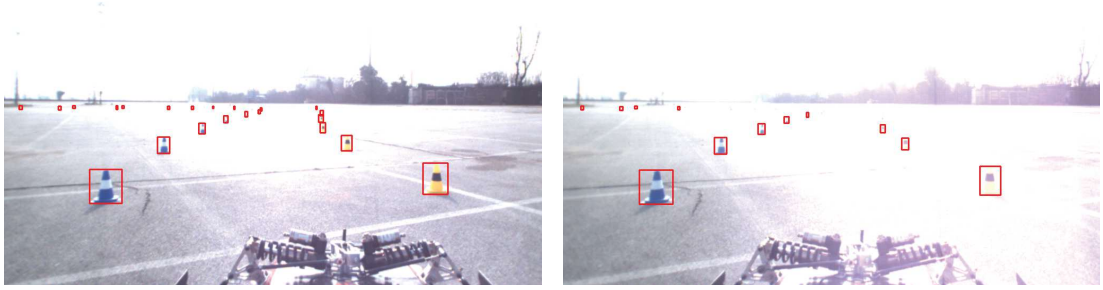
The performance reached by the YOLO model on our stereo camera images is beyond our expectations, especially in frames with high underexposure/overexposure problems, as can be seen in figures 6.4 and 6.5.

After the completion of the detection phase, a simple 3D pose estimation algorithm is run, to retrieve the three-dimensional cones coordinates in the real world, starting from their pixel coordinates.

Exploiting the bounding boxes produced by YOLO, stereo matching is performed. So, for each bounding box in the left image, the center is computed, to average the variance of the drawn rectangle. Perform a search along the epipolar line, evaluating the class of the detection, its confidence score, and the size of the bounding box. This last check is used to discard cones that are too far since we want to process only the ones nearby the car. Then, the disparity d between the centers of the matched bounding boxes is computed. This value allows to retrieve the depth of each point using the formula $z = (f * b)/d$ where f is the focal length and b is the baseline of the camera.

Now that the 2-dimensional image coordinates and depth of each cone have been retrieved, a perspective transformation is used in order to project each point in the 3-dimensional space.

When doing some first quantitative evaluation of the results, a limit of our sensor emerged: the disparity that we get is not fine-grained enough, so even a small displacement of one pixel between the two cones leads to an error of almost



(a) Left image. **(b)** Right image.

Figure 6.5: YOLO cones detection from challenging stereo images.

half a meter on the final positioning. The simpler way to solve this issue is to use a stereo camera more suited for this kind of task.

The high-level pseudo-code for the described algorithm is stated below:

Algorithm 2 Basic 3D pose estimation algorithm

Input: YOLO bounding boxes enclosing cones
Output: 3D positions of cones
  for each bounding box in the left image **do**
    compute the center of the rectangle
    search along the epipolar line for a bounding box in the right image
    if correspondence is found **then**
      compute disparity between the two
      compute the depth
      use a perspective transformation to reproject in 3D
    end if
  end for

Chapter 7

Real data acquisition

In order to test the proposed SLAM and perception in a real scenario, a real dataset is needed. The last part of the thesis will be focused on how to prepare a data acquisition campaign. It was all made possible thanks to the RaceUP team, which has made available SG-e 05 (fig. 7.1), their electric single-seater from the last season.

Figure 7.1: SG-e 05 car from RaceUP team, used for real data acquisition.

The vehicle is a traditional electrical car used in dynamic events, so it is not an autonomous prototype. For this reason, it has been properly sensorized to be as similar as possible to the self-driving vehicle that will compete in the events. Since this car has not been built having in mind the driverless category, sensor placement must be adapted to the actual structure of the car.

In the next sections, an accurate explanation of the entire dataset acquisition process, starting from the choice of sensors to use, will be given.

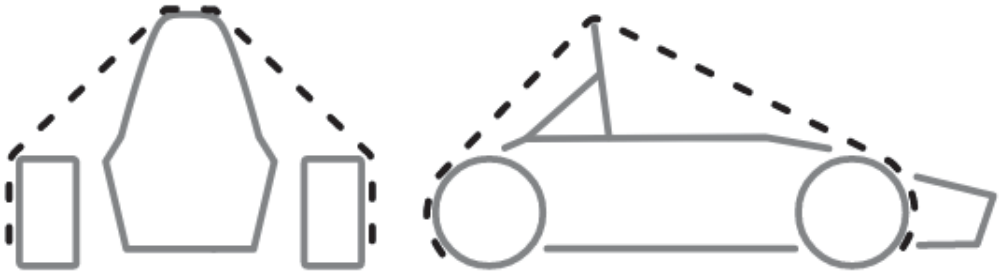


Figure 7.2: The surface envelope defined on the FSG rules: sensors cannot be placed outside this space.

7.1 Car sensorization

Car sensorization is a crucial task, it doesn't only include the process of deciding where and how to place the sensors in the car, but also which sensors to use. The goal is to try to obtain the most suited configuration for a real racing situation. An example of such sensors is depicted in figure 7.3: a Velodyne VLP16 16-channels LIDAR, a Bumblebee2 RGB stereo camera, an XSens MTi Inertial Measurements Unit (IMU), and two M8P RTK-GPS modules from UBlox.

The choice of sensors to use has been heavily constrained by the current availability of resources, both regarding RaceUP funds and university funds. For this reason, all the sensors have been borrowed from other projects, so they will not be the ones actually installed on the prototype for the next year's events. In other words, they have been exploited for proof of concept and the data acquisition campaign, but the actual car will mount a different sensor set.

When allocating space in the car for the sensors, many constraints need to be taken into account. Among the others, the most important are:

- the official Formula Student regulation, defining specific restrictions such as the external envelope that needs to enclose the vehicle and all its components (see fig. 7.2);

- mechanical and technical constraints, meaning that the shape and the design of the car are already fixed and they were not meant for this scenario. For example, sensors cannot occlude the driver's sight, they cannot be placed too close to some other components or above too weak elements;

- maximization of their effectiveness, meaning maximization of the field of view, minimization of avoidable noise like vibrations, and so on.

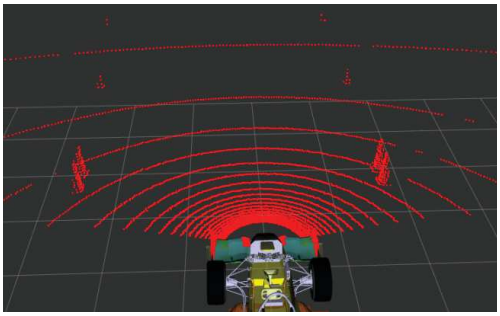
(a) Velodyne VLP16 LIDAR

(b) Bumblebee2 stereo camera

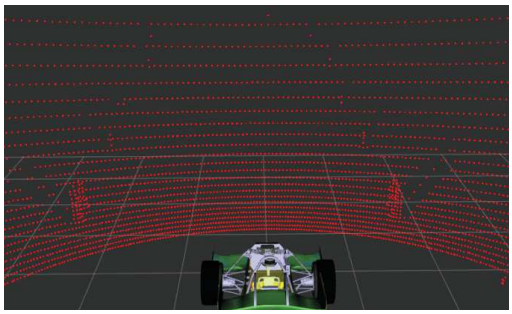
(c) XSens MTi IMU

(d) UBlox M8P GPS module

Figure 7.3: The complete set of sensors installed in the car.



(a) Scan with LIDAR on the front wing.



(b) Scan with LIDAR on main hoop, tilted.

Figure 7.4: Qualitative evaluation of different LIDAR placements.

The most difficult components to position were the LIDAR and the camera, being the most important sensors among the set and also the most sensitive to noise or to external interference.

To qualitatively evaluate their possible positioning, the EUFS simulator has been exploited, as depicted in figure 7.4. Thanks to one of the features of the simulator, we are not only able to simulate the raw sensor, but we are free to place the sensors into different positions, whenever necessary.

By looking at the other teams' cars, the possible placements have been restricted to a choice among two configurations. The first one puts the stereo camera on the main hoop of the car, just above the driver's head, and the LIDAR in the front wing, nearby the ground. The second one places the two sensors together on the main hoop, in a unique support enclosing both of them.

The camera positioning has been almost a forced choice, due to the nature of this sensor but also because all other vehicles follow this trend. For the LIDAR, instead, there is a bit more freedom, having a 360 degrees field of view. Placing it on the front wing results in a better detection of the base of the cones, since it is closer to ground level and so more laser beams hit the targets. Placing it on the main hoop gives the advantage of almost completely removing the lateral field of view occlusion, which persists instead on the front wing due to the structure of the wing itself.

The optimal placement of the LIDAR sensor has been evaluated using the visualization of the point clouds described in section 4, and a simple cone extraction algorithm. In the end, also due to mechanical constraints difficult to remove, the placement on the top of the main hoop has been selected.

By the way, the common point that emerged from this study has been the inclination. To maximize the field of view of the sensor, meaning to maximize the number of laser beams actually hitting the asphalt or the cones instead of the

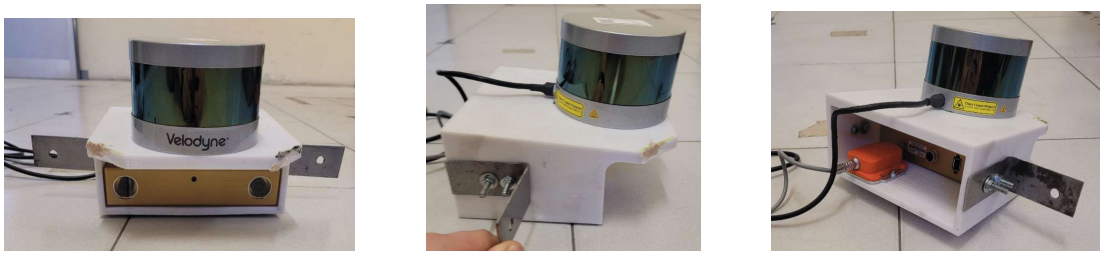


Figure 7.5: Front, side, and back view of the developed sensors support.

horizon, a 5 degrees pitch angle was the ideal solution.

For this reason, a sketch of the support to hold those sensors has been designed as a two layers component, having one inclined plane for the Velodyne and one plain support for the stereo camera as shown in figure 7.5.

The current positioning is temporary, meaning that it has been designed to adapt to the current electric car, and tries to imitate as faithfully as possible a driverless set-up. The final setup surely will result from a deeper study of aerodynamic efficiency, vibrations reduction, and stability of the entire structure. The sensor's support was made movable so it could be removed during non-driverless events, to have the minimum collision-free space between the driver's helmet and the sensors, required by the regulation.

7.2 Dataset

The scope of mounting the sensors on the vehicle was to be able to collect a Formula Student autonomous race scenario dataset.

After the car was ready, the next steps were: to create a custom track, made to be compliant with the autonomous category rules, and to collect as much data as possible. Colored cones have been placed according to the shape depicted in figure 7.6.

To have a ground truth reference for the data, four spurious cones have been placed on the corners of a square enclosing the track: the resulting square has been measured, to have an idea about the size of the track. This information was later exploited for estimating the ground truth position of the cones from an aerial image, containing the entire track, acquired from a drone.

With the same drone, also images of a checkerboard have been acquired to perform the calibration of its camera. Both individual and pairwise calibration of the other sensors, instead, has not been made in advance: using tools such as

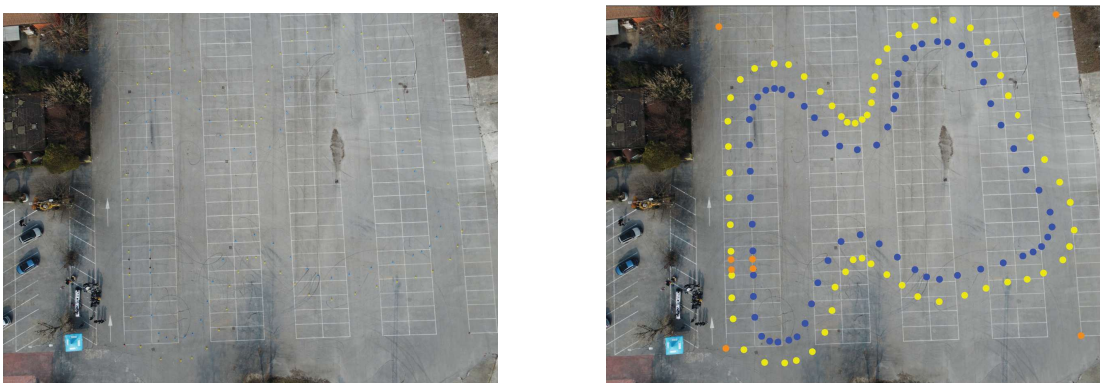


Figure 7.6: Aerial view of our track. The cones on the right image have been enhanced, for better shape understanding.

AutoCalib [33] and Kalibr [34] [35] [36], it will be performed later on.

From a software point of view, a fundamental step for the preparation of the actual data acquisition is the development of ROS2-wrapped stand-alone drivers to perfectly integrate each sensor with the system. Drivers are low-level software components used to connect and communicate with the hardware of the sensor, to retrieve data, and then save it.

Due to the amount of information to record, and the high dynamicity of the situation, the main focus is on the real-time storage requirements. Minimization of the delay between the data production and storage, indeed, is crucial to minimize data loss during the acquisition.

To get data as soon as available, every ROS node handling a sensor got its scheduler priority set to 99, meaning real-time. To save it without any slowdown, an empty data folder has been mounted in the Random Access Memory (RAM) and used as temporary storage. Always to reduce the delay in data saving, camera images have been saved in greyscale instead of RGB.

For the actual dataset acquisition, three ROS bags have been registered during the day. Each bag consists of three laps around the track, the first one at low speed, and the other two at higher velocity. In the last lap, some cones have been voluntarily hit, to introduce also some noisy data.

Moreover, other data has been recorded by driving on the same track, but in the opposite direction, meaning both in clockwise and counter-clockwise way.

The acquired dataset, being a real set of sensor measurements, is expected to contain some challenges. In particular, due to the weather conditions of the acquisition day, there has been a noticeable camera exposure change along the track. Since our sensor doesn't have the possibility to auto-regulate gain and

exposure time according to the environmental light, underexposure/overexposure is experienced when changing the direction of motion, for example on corners.

For the moment, usage of the acquired dataset is limited to two situations:

- cones detection with YOLO, as will be explained in section 6.1;
- parsing of data retrieved from the CAN line of the vehicle, concerning the part holding kinematic information of the car, like speed, acceleration, and steering.

Finally, the acquired dataset has been made available to the RaceUP team for further testing of perception modules, such as the front end of the SLAM pipeline.

Chapter 8

Conclusion and future works

In this thesis we tackled many problems inside a challenging autonomous racing scenario: the SLAM problem, the perception pipeline, the sensorization of a real vehicle, and the acquisition of a dataset.

We started by explaining key concepts about SLAM and its solution based on graph structures. We then presented the challenges that arise by referring specifically to the Formula Student case, highlighting the limits of commonly used approaches.

We then focused on the core of the work, the SLAM module based on factor graph optimization, which was used for obtaining a refined estimate of the car's state and a map of the surrounding environment.

In conclusion, the reported experimental results are satisfactory, and demonstrate the effectiveness of the proposed system, providing also useful information about strengths, limitations, and applicability of the analyzed techniques

Of course, there is still a margin for improvement and future research in this work: enhancing the robustness and reliability of the whole perception system, optimizing computational efficiency, and addressing complex scenarios, such as adverse weather conditions, remain key challenges that warrant further investigation. Moreover, making data association more robust by replacing Euclidean distance with the Mahalanobis one could be a good choice, to deal with uncertainty and correlation of the considered values. Finally, it would be interesting to investigate the actual real-time capabilities of the proposed solution, since it should run on a real vehicle participating in a competition.

References

[1] Raceup, 2023.

[2] Formula Student Germany. Fsg competition handbook 2023 and fsg rules 2023, 2023.

[3] 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.

[4] 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.

[5] Juraj Kabzan, Miguel de la Iglesia Valls, Victor Reijgwart, Hubertus Franciscus Cornelis Hendrikx, 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.

[6] 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.

[7] Michael Montemerlo, Sebastian Thrun, Daphne Koller, and Ben Wegbreit. Fastslam 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. *Proc. IJCAI Int. Joint Conf. Artif. Intell.*, 06 2003.

[8] Marcel Zeilinger, Raphael Hauk, Markus Bader, and Alexander Hofmann. Design of an autonomous race car for the formula student driverless (fsd). 05 2017.

[9] 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.

[10] 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.

[11] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part i. *IEEE Robotics Automation Magazine*, 13(2):99–110, 2006.

[12] 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.

[13] University of Edinburgh. Eufs simulator.

[14] Hugh Durrant-whyte and Tim Bailey. Simultaneous localization and mapping: Part i. *Robotics Automation Magazine, IEEE*, 13:99 – 110, 1995.

[15] 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.

[16] Jose-Luis Blanco. A tutorial on SE(3) transformation parameterizations and on-manifold optimization. Technical Report 012010, University of Malaga, 2010.

[17] Ferris M.C. A Gauss Burke, J.V. Newton method for convex composite optimization. mathematical programming. *Robotics Automation Magazine, IEEE*, 13:179–194, 07 2006.

[18] Sebastian Ruder. An overview of gradient descent optimization algorithms. *ArXiv*, abs/1609.04747, 2016.

[19] Helmut Hlavacs. A 2d car physics model based on ackermann steering. 2006.

[20] Tony Lindeberg. *Scale Invariant Feature Transform*, volume 7. 05 2012.

[21] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. volume 3951, pages 404–417, 07 2006.

[22] 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.

[23] Wisam Qader, Musa M. Ameen, and Bilal Ahmed. An overview of bag of words;importance, implementation, applications, and challenges. pages 200–204, 06 2019.

[24] Paul Besl and H.D. McKay. A method for registration of 3-d shapes. ieee trans pattern anal mach intell. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 14:239–256, 03 1992.

[25] Lionel Clavien, 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] Jrgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. A benchmark for the evaluation of rgb-d slam systems. pages 573–580, 10 2012.

[27] Michael Gruppo. evo, a python package for evaluation of odometry and slam.

[28] MUR Motorsport. Real-time cone detection with lidar.

[29] M. Himmelsbach, Felix v. Hundelshausen, and H.-J. Wuensche. Fast segmentation of 3d point clouds for ground vehicles. In *2010 IEEE Intelligent Vehicles Symposium*, pages 560–565, 2010.

[30] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, sep 1975.

[31] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2015.

[32] 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.

[33] Aditya Vaishampayan. Autocalib.

[34] 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.

[35] 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.

[36] Jérôme Maye, Paul Furgale, and Roland Siegwart. Self-supervised calibration for robotic systems. In *2013 IEEE Intelligent Vehicles Symposium (IV)*, pages 473–480, 2013.