POLITECNICO DI TORINO

Master's Degree in Mechatronic Engineering

Master's Degree Thesis

Global Mapping Algorithm for a Driverless Race Car

Supervisors

Prof. Nicola AMATI

Eng. Stefano FERACO

Candidate

Emanuele BERTRAND

Academic Year 2020/2021

Acknowledgment

I would like to express my gratitude to Prof. Nicola Amati, supervisor of this thesis, for allowing me to carry out this project and for the availability.

I would also like to thank Dr. Stefano Feraco for his professionalism and enthusiasm put for this project with all the advice, help and support given, and for his friendship that characterizes him.

Thanks to all the Squadra Corse Driverless team, in particular to the P&P division for the results obtained and for the time spent together.

A special thanks to my family, who supported me from the beginning of this journey with love and sacrifices allowing me to grow and reach this goal.

Thanks to my colleagues and friends of Politecnico, Giulia, Edoardo and in particular Tommaso, who accompanied me from the very first day, for having shared memorable moments inside and outside the university.

Finally, I want to thank Martina, Gaia, Davide, Mattia, Stefano and Thomas for having always been present and making me a better person every day.

Abstract

In autonomous driving systems and robotics, one of the main characteristics is the capability of the system to localize itself in the environment, to create a map and to optimize the navigation, which is the ability of getting the vehicle from place to place.
The Simultaneous Localization and Mapping (SLAM) is the branch of the autonomous system responsible to solve this problem. Fusing the data coming from sensors, it can provide a local and a global map, beside to correctly localize and orient the vehicle in it.
The aim of this thesis work is the deployment of a robust global mapping algorithm able to provide a precise and accurate map, of an unknown environment with cones, to the control system and allow a safe and optimal navigation.
The autonomous system is tested and experimentally validated both with simulations in ROS environment with Gazebo and in real time using an RC model of the SC19 electric racing vehicle developed by Squadra Corse PoliTO.
The sensors used for the perception pipeline are the ZED stereo camera and the Velodyne LIDAR, while for the localization the SBG Ellipse-N inertial navigation system is adopted.
The final aim of this thesis is to develop a reliable and efficient autonomous system that can be used by the SC19 vehicle to participate to the FSD (Formula Student Driverless) competition.

Summary

The idea of developing autonomous vehicles has always been a goal to be reached by the most important automotive industries in order to follow the main philosophy of continuous improvement and development of a product able to accomplish specific tasks, as the autonomous driving.

In the last few years, automotive industries' focus has been moved to self-driving vehicles to translate this idea into proper vehicles thanks to the technological developments and the experienced increasing interest all over the world.

Many Advanced Driving Assistance Systems (ADAS) are already present in most of the nowadays vehicles; however, fully autonomous vehicles are produced and adopted by only few companies even if they are considered to represent an important change in the traffic environment and the whole mobility in the next years thanks to the contribution of Artificial Intelligence.

This thesis work is related to the development and validation of a Global Mapping Algorithm for a driverless vehicle participating to the Formula Student Driverless (FSD) competition.

In autonomous driving systems and robotics, one of the main characteristics is the capability of the system to localize itself in the environment, to create a map and to optimize the navigation, which is the ability of getting the vehicle from place to place.

The Simultaneous Localization and Mapping (SLAM) is the branch of the autonomous system responsible to solve this problem. Fusing the data coming from sensors, it can provide a local and a global map, beside to correctly localize and orient the vehicle in it.

The aim of this thesis work is the deployment of a robust global mapping algorithm able to provide a precise and accurate map, of an unknown environment with cones, to the control system and allow a safe and optimal navigation.

First the theoretical background needed to understand the problem is presented, describing the software and hardware architectures adopted and the instruments used to develop the algorithm. An important section of the thesis is dedicated to the State of Art of the SLAM

process to study the techniques used in many fields of application and understand which the best approach is to follow and adapt to the competition and the rules related.
The overall autonomous system can be studied in its constituent sub-systems:

- Perception
- Localization
- Mapping
- Control and Trajectory Planning

The sensors used for the perception pipeline are the ZED stereo camera and the Velodyne LIDAR, while for the localization the SBG Ellipse-N inertial navigation system is adopted.
The final aim is to develop a reliable and efficient autonomous system that can be used by the SC19 vehicle to participate to the FSD racing competition.
After having described the main idea and functional aspect of the developed algorithm, the autonomous system is then tested and experimentally validated both with simulations in ROS environment with Gazebo and in real time using an RC model of the SC19 electric racing vehicle developed by Squadra Corse PoliTO.
Concerning the real-world validation sessions, the algorithm has been tested in different conditions and tracks to study the behavior when facing all the possible condition that can be found at the competition and tune the parameters in order to find an optimal solution which is the best compromise between performances and computational effort.
The tracks in which the vehicle has been driven are: a small-track made by straight sections and wide turning, a big-track, which consists of a longer path with narrower turning and few straight sections in order to check the robustness of the algorithm in different and hard driving conditions, and an acceleration track, namely one of the official tracks of the dynamic events of the race in which the position of the cones is known in advance and the most important aspect is that of testing the dynamic performances of the vehicle.
All the tracks tested have been reproduced at AeroClub in Torino with the official cones and following the official rules of the competition, in particular for what concern the color and distance of the cones, and the shape of the circuit; this, in fact, is made by two traces of cones,

one blues on the left and one yellow on the right, with four big orange cones to highlight the starting and ending point of each lap. The presence of the cones is a fundamental aspect in this kind of racing competition since they represent the landmarks that are perceived and used by the autonomous system of the vehicle to build the global map used to navigate in the track. The results obtained show a robust and accurate global map able to represent the surrounding environment based on landmarks, the cones, and so to create a reliable source to the Control pipeline for the safe driving of the vehicle in the first part of the competition, in which the track is not known a-priori and the system should create a global map and compute a local trajectory based on the perceived coned during the first lap.

After the completion of the first lap the system can pass to the control mode in which the mapping algorithm is used to update the position of the cones and decrease the associated covariance while the autonomous system provides a trajectory with a Model Predictive Control to drive as fast as possible the vehicle to obtain good dynamic performances for the competition.

As stated, the global map obtained with the algorithm developed is accurate and robust, highlighting the trace of the circuit and positioning almost every cone with high accuracy performing the tests in a race-kind environment that follows the official rules for a good validation stage.

The adoption of a RC model for Rapid Prototyping is a good solution, even if the tests on the official vehicle are fundamental for the final validation of the overall autonomous system.

The algorithm has been tested relying on the stereo camera only for the perception pipeline; however, some tests have already been carried out with the LiDAR in order to have a second perception source and make the system much more robust. In the future the global map will be based on the fusion of the information of these two sensors to obtain an even more accurate global map.

Contents

1 - Introduction

Since ancient times man has tried to use his resources to improve his life through various inventions and, with technological development, the progress made has increased exponentially without putting a brake on the continuous search for new technologies and possible improvements.

The evolution of vehicles has had the same history, starting in 1854 with the first internal combustion engine and the first car with Benz in 1883, going on with the implementation of the electric motor, until the last years where, with the advancement of robotics and electronic devices such as sensors, the development of driverless vehicles or vehicles that are self-driving without a driver, or with reduced tasks compared to the normal vehicles to which we are used to think about.

This type of technology could mark not only a technological breakthrough, but also a socio-economic one, implying safety and environmental impact that is bound to affect the years to come and technological goals with the focus that will be shifted on this topic.

Driverless vehicles are, in fact, one of several applications where autonomous systems can be adopted; industrial applications and in general automation are fields where this technology can grow fast and improve.

1.1 Autonomous Vehicle Development

The evolution process of autonomous cars is not as recent as it seems, it began in 1500 with Leonardo da Vinci's invention, which was a self-propelled vehicle moved through a mechanism formed by high tension springs. From this first basic idea, the evolution of this technology has grown mainly in the aerospace and automotive fields where, in the last few years, many companies like Ford, Mercedes and Tesla have set the attention on this topic, working to build autonomous vehicles for an important changing in the automotive world. The interest in autonomous vehicles reached also companies such as Google and nuTonomy who launched their own project, and nowadays many universities are studying and improving this system with collaborations and partnerships with transportation agencies and automotive companies to improve the technology [1].

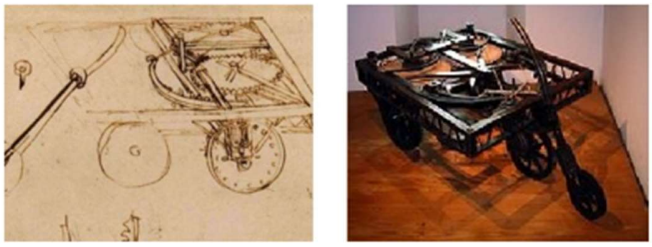


Figure 1.1: Da Vinci's Self-Propelled Cart

Environmental impact and technological development, together with the need to develop new skills and improve services have led to the progressive development of these vehicles that are beginning to become concrete realities.

As all new technologies, autonomous vehicles are thought to improve the life quality and so advantages are always larger than negative implications.

The first advantage is related to safety; in fact, most of the accidents are caused by human errors so driverless cars should lower the number of automobile accidents.

Public transports and traffic circulation could also benefit, since the number of vehicles travelling can increase and, with an efficient and smart communication between systems, it is possible to monitor the traffic and optimize the driving, also by reaching higher velocity

safely.

Finally, being electric vehicles, the environmental impact should be lowered, producing very low gas emissions that is expected to be 90 percent lower, than the actual emission, in 2030 [2].

In modern cars, many Advanced Driving Assistance Systems are already installed, allowing better vehicle's dynamic and passengers' comfort [3] [4].

However, fully autonomous systems are not ready yet due to technological limitations of the actual available solutions and to the limited trust of people on rely on a robot and on the Artificial Intelligence that is the fundamental base of driverless [5].

From the technical point of view, each vehicle can have a range of autonomous capabilities based on its equipment.

To promote a standard and common classification of autonomous systems, the Society of Automotive Engineers (SAE) International provides a taxonomy with detailed definitions for the level of driving automation.

SAE proposes a classification of six different levels in its official document SAE J3016, which is considered the industry standard for autonomous vehicles, aiming at developing an initial regulatory framework to guide manufacturers in the design, development and testing of Highly Automated Vehicles [6].

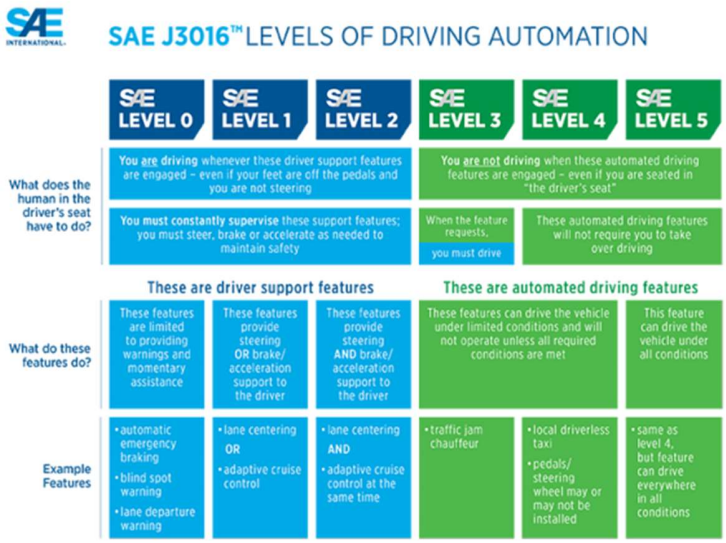


Figure 1.2: SAE Levels of Driving Automation

The six levels of driving automation are classified such that an increasing number means a greater autonomy of the system.
The classification is based on the level of responsibility required to the driver, which goes from full at Level 0 to un-necessary at Level 5. Most of nowadays car are classified at Level 2, but many models can reach Level 3 and 4 of automation, while at the state of art there is no model able to implement a full level of autonomy and research are focused mainly on this development, in order to deal with different environment conditions and traffic situation that can occur during a regular travel.
All the six levels are defined with respect to the role played by the main pillars of the Dynamic Driving Task: steering, acceleration and braking by monitoring the driving performances and the robustness.
The levels of driving automation are described as follow:

- Level 0 – No automation: the human driver is the one who fully control the dynamic performance of the car, even if there can be some assistant system for steering or speed control;

- Level 1 – Driver assistance: most of the dynamic driving task is controlled by the driver, the assistance system is in charge of control the steering and the velocity in well-known driving environment such as highway;

- Level 2 – Partial automation: the automated driving system is able to manage all the dynamic driving tasks by continuously keeping control of the lateral and longitudinal parameters, the driver should intervene at any moment if necessary;

- Level 3 – Conditional automation: the system is capable of taking full control of the car, the driver must intervene only in particular cases (e.g. accidents);

- Level 4 – High automation: vehicles can drive themselves without human intervention, completing autonomously a travel. Human intervention is not necessary

but systems like pedals and steering wheel are still present to allow driver's interaction;

- Level 5 – Full automation: the car should be able to take full control in any driving environment and condition, making the presence of humans just as a simple passenger with no duty in the driving framework.

The Driving-Assistance Systems (DAS) are fundamental components for describing the different levels of automation and correctly classify each vehicle and technology.
They are based on sensors measuring internal variables (proprioceptive sensors), such as acceleration and velocity to provide support information and optimize a safer driving.
The most adopted are:

- Anti-lock Braking System (ABS): used in wet conditions for a safe braking, preventing the wheels from locking [7];

- Cruise Control (CC): electronic device that allow the control and the adjustment of the speed of the car;

- Traction Control System (TCS): electronic system adopted to prevents the wheels from spinning during accelerations;

- Electronic Stability Control (ESC): adjust the engine power in order to control the vehicle's stability and, during braking actions, optimize the braking intensity on each wheel to stabilize the attitude.

In parallel to DAS, Advanced Driving-Assistance Systems (ADAS) are commonly installed on vehicles to provide deeper information to the autonomous system. These systems are based on sensors able to read and transmit information about the external environment (exteroceptive sensors).

Figure 1.3: Advance Driver Assistance Systems (ADAS)

- Autonomous Emergency Braking (AEB): improve safety by autonomously baking in critical situation or when the distance between vehicles is too close and prepare the overall car to the impact in order to reduce the internal damage [8]. The working principle is based on Blind Spot Detection (BSD), which allow to rely on a robust system able to recognize some particular situations, in circumstances where human cannot control;

- Adaptive Cruise Control (ACC): it represents a more robust and richer evolution of the Cruise Control, where the system can control and maintain the desired speed and is able to maintain a safe distance from other vehicles. This kind of technology is

classified as Level 1 in SAE level of driving automation [9];

- Lane Departure Warning (LDW): also known as Lane Keeping (LK), is installed on the vehicle to prevent accidents and guarantee safety. In works as a closed loop system, where first there is a warning when the car position is outside the established path, then the system intervenes to correct the position. This system, with the ACC, lead to classify the vehicle in the Level 2 of the SAE rule [10];

- Parking Assistance (PA): system able to find and calculate the space for a parking and control the steering maneuvers.

- Hazard Recognition (HR): system related to the safety of the car and the driver, based on the preliminary hazard analysis with Functional Failure Analysis to detect a risk and respond to it with the automatic control of the vehicle or by alarming the driver [11];

In general, beyond classification, an autonomous Driving System can be studied as a mobile robot with multi-field competences able to perceive the environment and react in a proper way to move itself from point to point. As a robot, the software and the hardware are fundamental components that work together to the creation of a full system. The main components at hardware level are:

- Sensors: used to perceive the environment and provide the input to the system;

- Actuators: devices used to translate the information coming from the Control Unit into mechanical actions.

They communicate with the Software level through the Processing Unit, which is an embedded processor that elaborates the inputs and derive the necessary outputs.

1.2 Formula Student Driverless (FSD)

Formula Student is a student competition organized by SAE international whose aim is that of create a competition of engineering from all around the world to design, fabricate and race a formula-style racing vehicle.
The program started in 1981 in the United States and from that day the level and the development of the competition is constantly growing.
The competition comprehends different categories based on the vehicle participating to the race:

- FS Combustion;
- FS Electric;
- FS Driverless

In particular, the mapping algorithm developed for this thesis work is part of the autonomous system of the SC19 car participating to the Formula Student Driverless competition.

Being an engineering competition, the Scoring is given by the sum of the following categories, according to the official rules [12]:

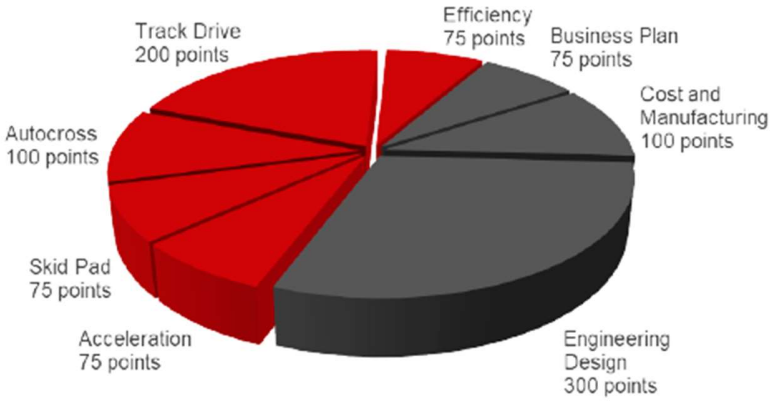


Figure 1.4: Scoring of FSD

The scoring can be divided in two main categories: the red part representing the Dynamic Events and the gray part representing the Static Events.

The overall total is of 1000 points and, for winning, all the categories are fundamental and not only the Dynamic Events; this creates a complete competition in each aspect of the developing of the vehicle.

For this reason, the Squadra Corse Driverless of Politecnico di Torino is composed by more than 25 people with different background divided into 5 divisions, each responsible of a specific part of the developing.

The final goal is to provide an opportunity for engineering students, from different fields, to investigate and grow new knowledge and apply them into the real world, leading to a more complete educational formation and experience on the field.

Starting from the theoretical background it is possible to apply with practical experience these competences and develop a complete race car.

Concerning the Dynamic events, a technical inspection must be passed, then the car is tested in all the tracks scheduled; in particular, the most challenging one is Track Drive where the vehicle must drive itself in a track delimited by cones, completing ten laps as fast as possible.

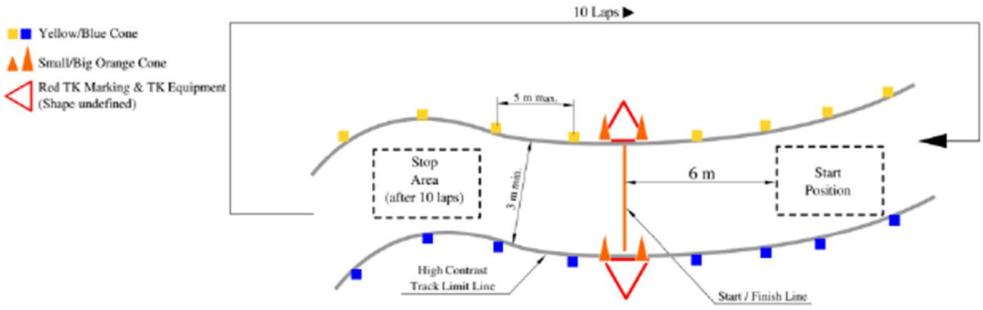


Figure 1.5: Track Drive at FSD

1.3 Thesis Outline

This thesis work is structured as follows:

1. Chapter 2 presents the theoretical background of the topics presented, with the focus on the software and the hardware used, in particular on SLAM techniques.

2. Chapter 3 is dedicated to the design and implementation of the mapping algorithm. Starting from the creation of a first local map and then with the global mapping algorithm that is used for the navigation.

3. Chapter 4 is related to the final results; in this part it is described the experimental validation process and the results obtained. The algorithm is first tested with a simulation in order to study the behavior of the car changing different parameters, and then is tested using a RC model for the validation of the algorithm in real time.

4. Chapter 5 reports the final considerations and possible improvement for future works.

2 – Theoretical Background

2.1 Robot Operating System (ROS)

Robot Operating System (ROS) is an open-source, meta-operating system for building robots. It is not an actual operating system, but a framework and set of tools that provide functionality of an operating system on a heterogeneous computer cluster. The applications are not limited to robots, but also on working with peripheral hardware thanks to the suitable tools. ROS provides functionality for hardware abstraction, device drivers, communication between processes over multiple machines and tools for testing and visualization [13].
The key feature of ROS is the way the software is run and the way it communicates, allowing to design complex software. ROS provides a way to connect a network of processes, or nodes, with a central hub. Nodes can be run on multiple devices, and they connect to that hub in various ways.
Defining publisher/subscriber connections with other nodes and providing requestable services are the main ways of creating the network used by ROS for building a complex system.
Despite the importance of reactivity and low latency in robot control, ROS itself is not a real-time operating system. It is possible, however, to integrate ROS with real-time code.
The main advantages of ROS with respect to another robotics software platform are:

- Thin: it is designed to be as thin as possible so that the code written can be used with other frameworks;
- ROS-agnostic libraries: possibility to write libraries with clean functional interfaces;
- Language independence: easy to implement with any programming language such as Python, C++, Lisp and Java;

- Easy testing: it integrates a built-in test framework called “Rostest” that makes it easy to bring up and tear down test fixtures;

- Scaling: appropriate for large runtime systems and large development processes [14].

ROS was designed such that users would be able to choose the best configuration of tools and libraries interacting with ROS’s core so that they can shift their software stacks to fit different robot and application area.
The philosophy on which ROS is based is a graph structure where nodes are connected to each other by edges called topics.
A node represents a single process and can pass messages to one another through topics that are named buses used for communication.
Messages are predefined structures in which data coming from a node is stored and can be of different types; to be sent to a topic, a node must publish to the unique and specified topic where another node can subscribe to receive messages.
Nodes are the core of ROS since they work also as a client/service model and take actions based on the information received from other nodes.
The whole working principle is made possible by a process called “ROS Master” by registering nodes to itself, setting up node-to-node communication for topics, and controlling parameters server updates. The Master sets up peer-to-peer communication between all node processes after the registration.
This decentralized architecture allows an optimized communication between devices leading to a lighter computation; this is fundamental when dealing with robots, where the system consists of a subset of networked computer hardware.

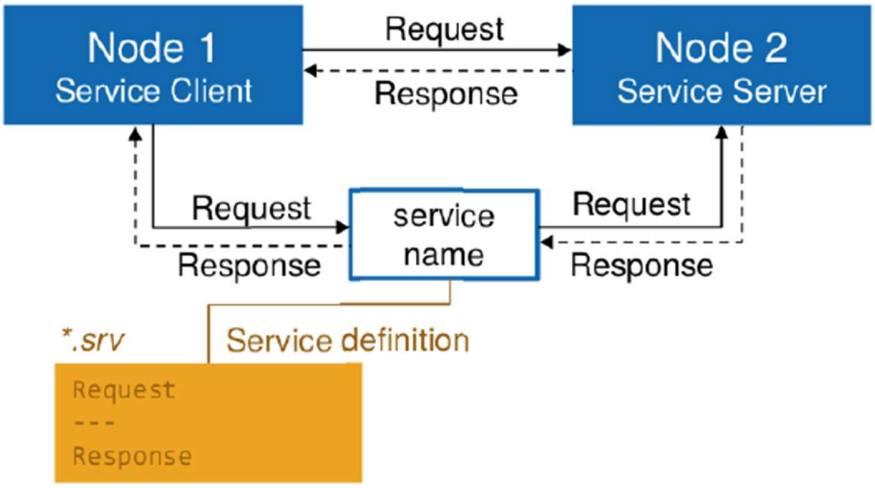


Figure 2.1: ROS Architecture

ROS is completed by a variety of tools which increase the capabilities of systems using ROS by simplifying and providing solutions to several common robotics problems.
The main tools provided are:

- Catkin: is the ROS build system. It is cross-platform, open-source, and language-independent;

- Rosbash: provides a suite of tools which increase the functionality of the bash shell, allowing users to use ROS package names in place of the file path where the package is located;

- Roslaunch: used to launch multiple nodes and setting parameters;

- Rosbag: is a command line used to record and playback ROS message data from topics;

- RVIZ: is a three-dimensional visualizer used to visualize robots, environments, and sensor data.

For this thesis work ROS is hosted by the Linux machine NVIDIA Xavier running Ubuntu 18.04, where the main stages of the data processing take place.
An important aspect to consider when dealing with different sensors' measurements is the synchronization. ROS messages carry the time stamp of the message in ROS time, which is associated to the CPU timing of the computer; this represents the time at which the message is processed. For these reasons different sensors can have different timestamps and so synchronization becomes a fundamental step in order to link all the reference frames together and guarantee a correct communication for the development of the mapping algorithm.

2.2 Software architecture

The final goal of having an optimal global mapping algorithm can be reached by developing a functional software architecture able to fuse all the useful information coming from different sensors and guarantee the right communication between these through ROS.

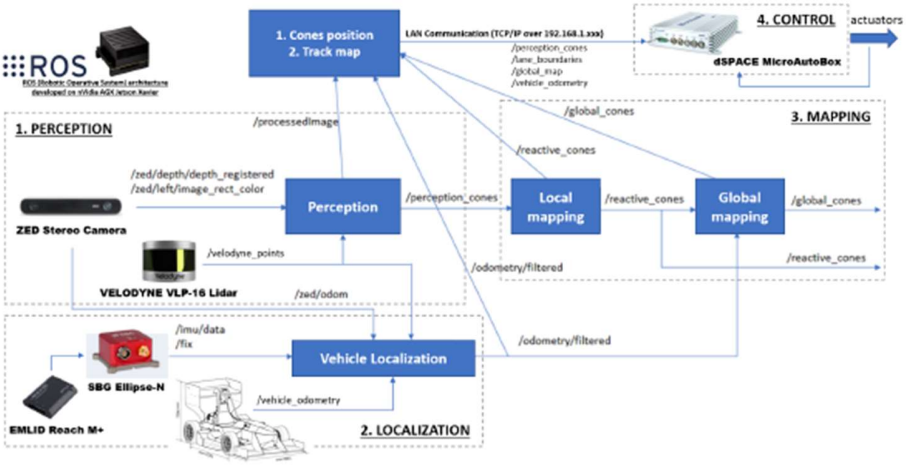


Figure 2.2: Software Architecture

As it is possible to see from the scheme, the architecture can be divided into three big categories: Perception, Localization and Mapping.

2.2.1 Perception

The perception pipeline is composed by two sources which provides different information of the same environment:

- ZED stereo camera: publishes information about the position and the color of the cones detected;
- Velodyne VLP-16 LiDAR: publishes information only about the position of the cones.

Both the sensors can detect cones and provide data useful to create a map of the environment independently, however if merged the resulting map is obtained as the combination of the information of the two sensors leading to a more detailed and much more robust map.
Raw data coming from the ZED are processed through the SDK software as follow:

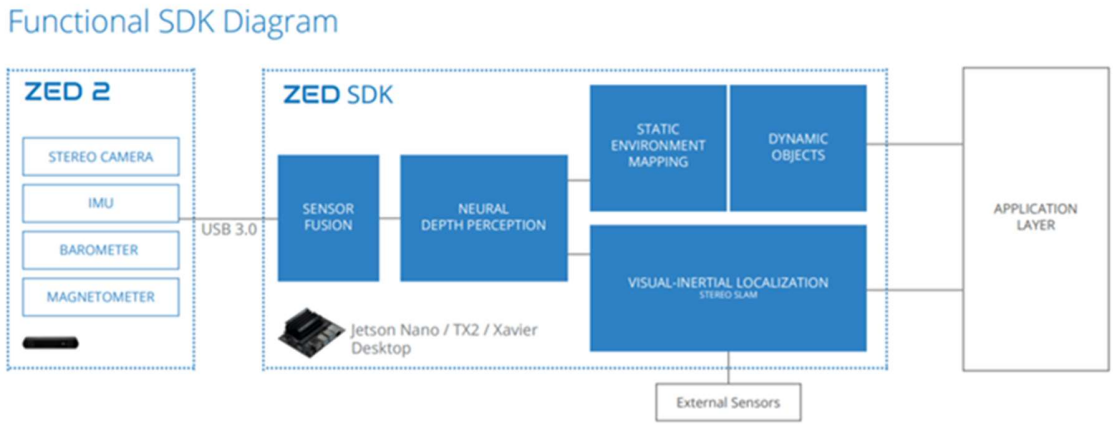


Figure 2.2: ZED SDK Software

The actual perception pipeline integrates data coming from the stereo-camera to perform the so-called visual perception, that is performed by an appropriate algorithm: the Single Shot Detector (SSD) Mobilenet V1 from Tensorflow models. The algorithm is based on an Artificial Neural Network that elaborates the images coming from the camera and recognize the cones along the path.
The process is performed by a ROS node that subscribes from three topics:

- /zed/left/image: contains information about the scene seen by the left lens of the camera;
- /zed/depth: contains information about the depth of the object in the environment through a depth map;
- /zed/odom: contains information about the odometry measured by the integrated sensors.

After calling back the SSD object detection, the node publishes the elaborated data over the topics:

- /processedImage: containing the same images taken as input with the bounding boxes and depth information of the cones detected;
- /perception_cones: with an array describing the local coordinated and the color of the cones in the 3D space, useful to create a local map of the environment.

The SSD is based on Artificial Neural Network (ANN), which is an interconnected group of neurons that uses a mathematical and computational model for information processing based on a connection approach to computation. In most cases an ANN is an adaptive system that changes its structure based on external or internal information that flows through the network. They are non-linear statistical data modeling and decision-making tools that can be used to model complex relationships between inputs and outputs or to find patterns in data [15].

Object Detection is a technique in Computer Vision that deals with detecting examples of semantic targets of a specific class in images and videos. It is a technique that works to locate

and identify objects in digital images and videos.

It specifically draws Bounding Boxes around the object which to locate where the objects are.

SSD does not resample pixels for bounding box hypothesis as other methods but eliminates bounding box proposals and the subsequent pixels and feature resampling. In this way to detect multiple object it is necessary only one single shot, against the two needed by approaches such as Regional Proposal Network, making the SSD very fast and reliable at the same time.

The algorithm is based on the Multi-Box theory, which uses the concept of Ground Truth; this separates observed or empirical evidence from inferred evidence. The SSD break the images coming from the camera into several segments and, for every segment, it constructs several bounding boxes. Then it checks each box on the image for an object of each class of interest the Neural Network is trained for and, at last, it compares the prediction with the Ground Truth. If there is any error after the comparison, then it is backpropagated through the network to help update the weights [16].

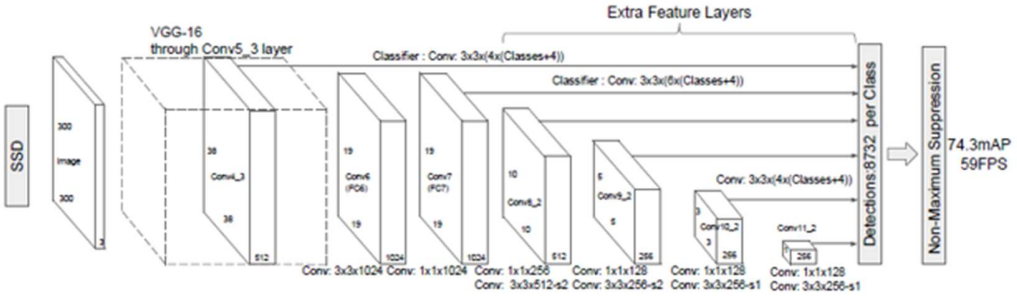


Figure 2.3: SSD Algorithm

The same process can be performed by other object detection algorithms such as YOLO, which seems to be more robust than the SSD but, at the same time, slower. A parallel thesis work is focused on the comparison of these two different algorithms to find out the best one to be implemented on the SC19 software architecture.

Concerning the LiDAR, the objects detected are represented as a set of points called “point cloud”. At a first stage, these represent all the object in the field of view of the LiDAR and so also the ground and any other kind of obstacles; this causes a doble drawback since the computational effort becomes huge and, most important, the information to be passed to the mapping algorithm is too dense and contains irrelevant data for the purpose of the algorithm. To this purpose, first the topic /velodyne_points is directly passed to the perception pipeline, then two operations are performed to filter as much as possible the data to keep and pass only the information needed by the pipeline to detect the cones.

The first operation consists in a pre-processing where an Adaptive Ground Removal Algorithm is used. It adapts to changes in the inclination of the ground using a regression-based approach to estimate the ground plane and distinguish between the ground and non-ground points, after which the ground points are discarded. The algorithm works by dividing the field of view of the LiDAR into angular segments and splitting each segment into radial bins; a line is then regressed through the lowermost points of all bins in a segment. All the points that are within a threshold distance to this line are classified as ground points and are removed [17].

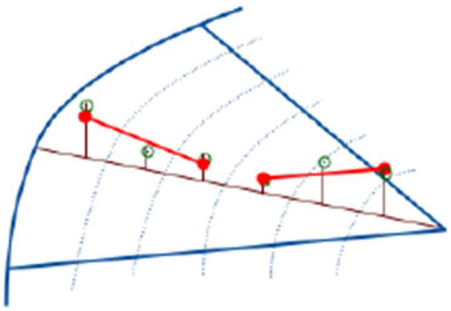


Figure 2.4: Ground Removal Algorithm

The second operation is the proper cone detection algorithm. After the application of the ground removal algorithm, in addition to those of the ground, a substantial amount of cone points is removed. This significantly reduces the already small number of return points that can be used to detect and identify cones. This is addressed by clustering the point cloud after ground removal using an Euclidean Distance-Based Clustering Algorithm.
In this way the output of the perception pipeline related to the LiDAR is a database containing all the clusters reconstructed and the position associated to each of them.

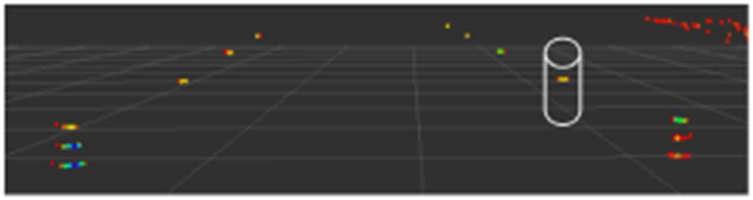


Figure 2.5: Adaptive Clustering Algorithm Working Principle

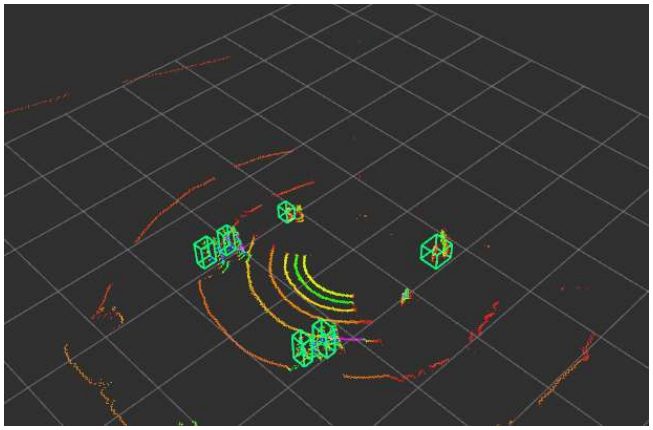


Figure 2.6: Adaptive Clustering Algorithm Results

2.2.2 Localization

The localization pipeline is a fundamental part of the software architecture since it directly deals with the Mapping and the Motion Planning pipelines and provides fundamental information for the development of the autonomous system. Through this component, in fact, it is possible to estimate the states of the vehicle with high accuracy and robustness while keeping trace of the trade-off between performance and requirements. To this purpose a decentralized system is adopted to lower the computation effort while maintaining a high accuracy in the estimation. This can be tackled, as for the Zed camera, with sensors equipped with embedded computers, which pre-process and communicate data to the central control unit for the final processing.
The Inertial Navigation System (INS) adopted, which is the SBG System Ellipse-N has been chosen to be able of real-time measurement pre-processing.

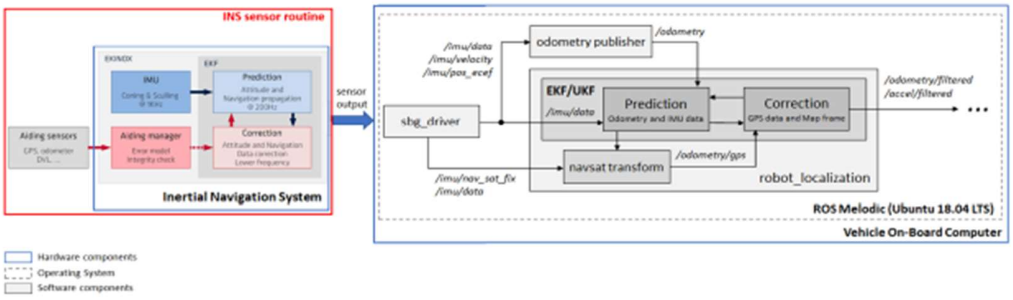


Figure 2.7: Localization Pipeline

In figure 2.8 it is possible to see how the decentralized architecture works and, in particular, how the INS and the central control unit communicate to obtain the final and common output. The INS sensor routine represents the pre-processing performed at sensor level with a first estimation of the states of the vehicle. The process starts with the calibration of the IMU, fundamental to filter the first data and create a solid base from which the sensor can start working; then the inputs read by the sensor are processed by a fusion routine with an Extended Kalman Filter (EKF), integrating also information coming from the GNSS module.

The output obtained at this stage is a complete set of the state measurements and raw data for post-processing.

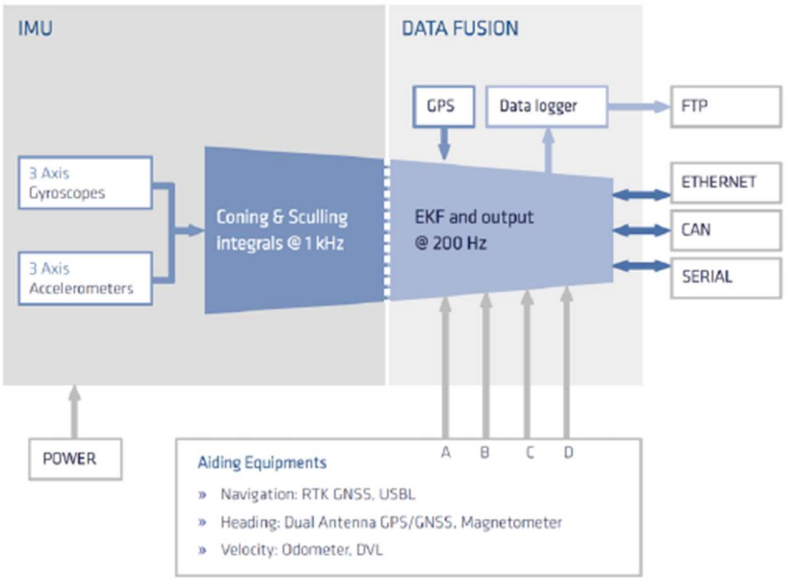


Figure 2.8: INS Internal Architecture

The post-processing is performed by means of three main components: the sbg_driver , the odometry_publisher and the robot_localization packages.

The sbg_driver node publishes the filtered sensor data at a frequency of 200 Hz that can also be tuned depending on the purposes.
The node publishes on four topics:/imu/data, /imu/velocity, /imu/pos_ecef, /imu/nav_sat_fix; they contain messages about the measures computed by the SBG in a common frame, which is the ECEF (Earth-Centered-Earth-Fixed), used as a basis for the transformation of all other frames related to other sensors.
A fundamental step is the initialization of the INS since this may be adapted for different systems and different environment. For this work the initialization presented in Appendix A.1 and A.2 has been carried out for testing the algorithm on the RC model and on the SC19.

The most relevant parameters are related to the accounting of the mechanical lever arms of the hardware and misalignment of the sensor. As it is possible to see these parameters are set to zero in the RC Model since this vehicle is very compact and so the approximation is acceptable. One of the most important position to take into account is the primary lever arm, which is the one related to the position of the GNSS antenna with respect to the SBG sensor; setting this parameter properly the transformation between data acquired from the two sensors is correct and so the fusing stage of position and velocity is made much more robust. Another import parameter is the motion profile, which is set to Automotive Mode in both cases, this is an important pre-defined parameters that makes assumption on the motion of the system on which the sensor is mounted and, consequently, the internal processing will make some a-priori assumption useful to make faster and more reliable the estimation process.

The odometry_publisher node takes information coming from the sbg_driver node and provides the right transformation between frames, that is published on the topic /odometry containing the pose and the velocity of the vehicle. The transformation provided by this node is fundamental since the INS and the GNSS acquire data in different coordinates and so to perform the filtering algorithm these data have to be consistent and must be referred to a common frame. Thanks to the synchronization of the messages with the ROS timestamp this transformation can be done in real time and so the estimation of the states can be performed online without need to process data after the acquisition, leading the autonomous system to work properly.

The robot_localization package completes the internal architecture of the localization pipeline.
The package is already available among ROS libraries and contains different non-linear state estimators for any general mobile robot to allow users to choose the best filter on the basis of the final goal of the robot [18].
In particular, it provides the Extended Kalman Filter (EKF) and the Unscented Kalman Filter (UKF), which are non-linear form of the Kalman Filter (KF) used for sensor fusion and state

estimation in different fields.
For this thesis work the sensor fusion is used to merge information coming from the INS and the GNSS, but, in general, both the estimators can fuse data from any number and any kind of sensors, providing a conform transformation between frames in which data are collected.
In this particular case, GNSS data are integrated with an external node called navsat_transform that is used to update the global positioning information of the vehicle.
Both the estimators, from the software point of view, work in the exact same way and so, at a first stage, the important aspect is not related to the choice of the algorithm but on the tuning of the parameters of both the filter to adapt them to the particular instance in which they should work.
For this purpose, it is important to select the right input data to be passed to the estimator, namely the ROS topics containing these messages.

As shown in Appendix A.3 and A.4, the input topics are /odometry and /imu/data, containing information coming from the INS raw data and the first pre-processing after the right transformation between frames provided by the robot_localization package.
The configuration file of the filter is important since it allows users to decide which information are useful and must be integrated in the estimation routine, saving memory and most of all the computational effort is lowered ensuring the right working of the overall autonomous system. For each input is then possible to decide which information are used and, in particular, since the system is adopted for a race car, the motion is imposed as a two-dimensional motion delating all the useless information about the speed and position along the z-axis and the roll and pitch of the vehicle.
The final output is published on the topic /odometry/filtered, containing the estimated states and a standard message with the transformation between frames called /tf that follows the ROS standard convention [19].

Kalman Filter

As stated, the localization stack is based on an estimation problem that is solved by fusing raw data coming from sensors to obtain a robust odometry able to describe the pose of the vehicle in the space and other important information about the navigation of the system.
The Kalman Filter is a recursive filter that estimates the states of a linear dynamic system starting from a collection of raw data affected by noise [20]. Noisy measurements are typical of sensor readings; therefore, sensor fusion is a common way to overcome this problem.
The algorithm is made by two processes: the prediction and the correction steps.
In the first step the filter estimates the states of the system, with some a-priori information, and the covariance associated. In the correction phase the estimations are updated thanks to the integration of data coming from sensors; this is made with a weighted average where the weights are chosen from the covariance, based on the reliability of the measurements. This process is repeated for a certain number of iterations until the system is working.
The power of the Kalman Filter comes from the information needed to perform the algorithm; with a complete knowledge of the system and the initial conditions, it is possible to compute the estimation of all the states over the time since the estimation of the previous time step is always known with respect to the actual time.

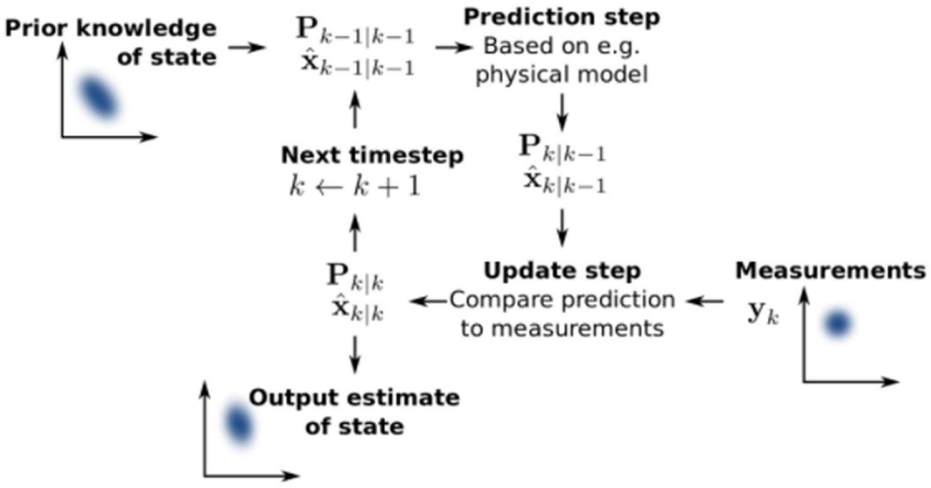


Figure 2.10: Kalman Filter

The theory behind the Kalman Filter is presented in the following equations:

$$x_k = F_x x_{k-1} + B_k u_k + w_k$$

Where:

- F_k: state transition model applied to the previous state x_{k-1};
- B_k: control-input model applied to the control vector u_k;
- w_k: process noise, assumed to be drawn from a zero mean multivariate normal distribution, *N*, with covariance, Q_k: $w_k \sim N(0, Q_k)$.

At a generic time *k*, a measurement z_k of the true state x_k is:

$$z_k = H_k x_k + v_k;$$

Where:

- H_k: observation model;
- v_k: observation noise, assumed to be zero mean Gaussian white noise with covariance R_k: $v_k \sim N(0, R_k)$.

Assuming to know that the initial state and the noises are mutually independent, the state of the filter is represented by two variables:

- $\hat{x}_{n|m}$: a posteriori state estimate at time *n* given observations up to and including *m*;
- $P_{k|k}$: a posteriori estimate covariance matrix.

The prediction phase, for both the state and the covariance, can be described as follow:

- Predicted state estimate: $\hat{x}_{k|k-1} = F_k \hat{x}_{k-1|k-1} + B_k u_k$;
- Predicted estimate covariance: $P_{k|k-1} = F_k P_{k-1|k-1} F_k^T + Q_k$;

Finally, the update stage can be described:

- Innovation: $\tilde{y}_k = z_k - H_k \hat{x}_{k|k-1}$;
- Innovation of the covariance: $S_k = H_k P_{k|k-1} H_k^T + R_k$;

- Updated state estimate: $\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k\tilde{y}_k$;
- Updated estimate covariance: $P_{k|k} = (I - K_k H_k)P_{k|k-1}$;
- Measurement post-fit residual: $\tilde{y}_{k|k} = z_k - H_k\hat{x}_{k|k}$;

In this thesis work, however, the vehicle can be modeled with the Bicycle Model which is a three degrees of freedom model that represents the vehicle in an easier way, allowing to study the dynamics associated with some simplification but maintaining a high level of accuracy. This model, even if simplified, is a non-linear system and so the assumptions made for the Kalman Filter do not hold.
A generic non-linear system can be describer with the following equations:

$$x_t = g(u_t, x_{t-1}) + \varepsilon_t$$
$$z_t = h(x_t) + \delta_t$$

Where x_t is the state to be estimated, u_t is the control input, z_t is the measured output, ε_t and δ_t are respectively the process and measurements noises.
A solution is obtained implementing the non-linear filtering algorithm provided by the robot_localization package: the EKF and the UKF.
These filters allow to estimate alle the 15 states characterizing the vehicle dynamics; and, assuming a two-dimensional motion as explained before, also the remaining states useful for the characterization of the overall algorithm:

$$x_t = \begin{bmatrix} x & y & \psi & v_x & v_y & \dot{\psi} & \dot{v}_x & \dot{v}_y \end{bmatrix}$$

Extended Kalman Filter

The Extended Kalman Filter represents an important solution in the filtering technique of non-linear system since it overcomes the problem by linearizing it. The linearization is not global but a local one, where the distribution is propagated through a Taylor expansion of the first order around a nominal value of the state.
Performing the linearization not only for the state but also for the input command and the output measurement, it is possible to obtain the complete information necessary to perform the estimation as for the standard Kalman Filter.
The linearization is performed first on the perturbation of all the components of the system as follow:

$$\delta x_k = x_k - \bar{x}_k$$
$$\delta z_k = z_k - \bar{z}_k$$
$$\delta u_k = u_k - \bar{u}_k = 0$$

And:

$$\delta x_t = \bar{G}_t \delta x_{t-1} + R_t$$
$$\delta z_t = \bar{H}_t \delta x_t + Q_t$$

Where $\bar{G}_t$ and $\bar{H}_t$ are the Jacobian of g and h with respect to x and u, and R_t, $Q_t$represents the Jacobian matrices of the non-linear functions with respect to the perturbations.
Introducing an approximation, allowed by some a-posteriori considerations, the system can be described as:

$$\bar{x_t} = G_t \hat{x}_{t|t-1}$$
$$\bar{z_t} = H_t \bar{x}_t$$

And the linearization can be performed according to the algorithm developed:

$$\hat{x}_{t|t-1} = g(u_t, \hat{x}_{t-1|t-1})$$
$$\hat{\Sigma}_{t|t-1} = G_t \Sigma_{t-1|t-1} {G_t}^T + R_t$$
$$K_t = \hat{\Sigma}_{t|t-1} {H_t}^T (H_t \hat{\Sigma}_{t|t-1} {H_t}^T + Q_t)^{-1}$$
$$\hat{x}_{t|t} = \hat{x}_{t|t-1} + K_t(z_t - h(\hat{x}_{t|t-1}))$$
$$\Sigma_{t|t} = (I - K_t H_t)\hat{\Sigma}_{t|t-1}$$

An important consideration must be done regarding the distributions of the random variables that, after the Taylor expansion, are no longer normal and so the properties exploited in the standard Kalman Filter about these variables are not valid with this algorithm.
During the tuning phase of the algorithm, it is important to understand which parameter affects the overall result and how to use it. For doing this, the two matrices R_t and Q_t are defined as the measurement noise covariance and the process noise covariance. The first one is the one used for tuning the algorithm with a trial-and-error process while the second is assumed to be constant; at the end the filter returns the estimation of the states $\hat{x}_{t|t}$ and $\Sigma_{t|t}$ [21].

Unscented Kalman Filter

The second solution provided by the robot_localization package is the Unscented Kalman Filter. The final goal and the overall architecture of the algorithm is the same as the EKF and also the parameters configuration is very similar, allowing to have two different filters and so let the user to choose the best one according to the necessities and working conditions.
The adoption of this filter is also motivated by the fact that it has been proved to guarantee better performances with respect to the EKF leading to a more accurate estimation that is fundamental in the field of autonomous driving for safety and precision in the completion of

the job. The overall better result obtained is achieved by a different linearization that is based on the same Gaussian Random Variable distribution as for the EKF but, in this case, it is represented by the so called σ-points: a collection of 2n+1 points based on the original mean value and variance.

The Unscented transformation is based on some assumptions regarding the choice of the σ-points and the weights assigned to the filter as follow:

$$\chi_{t-1|t-1} = (\hat{x}_{t-1|t-1}, \hat{x}_{t-1|t-1} + \sqrt{(n+\lambda)\Sigma_{t-1|t-1}}, \hat{x}_{t-1|t-1} - \sqrt{(n+\lambda)\Sigma_{t-1|t-1}})$$

$$\hat{\Sigma}_{t|t-1} = \sum_{i=0}^{2n} {\omega_c}^{[i]} \left(\hat{\chi}^{*[i]}_{t|t-1} - \hat{x}_{t|t-1}\right)\left(\hat{\chi}^{*[i]}_{t|t-1} - \hat{x}_{t|t-1}\right)^T + R_t$$

$$S_{t|t-1} = \sum_{i=0}^{2n} {\omega_c}^{[i]} \left(\hat{Z}^{*[i]}_{t|t-1} - \hat{z}_{t|t-1}\right)\left(\hat{Z}^{*[i]}_{t|t-1} - \hat{z}_{t|t-1}\right)^T + Q_t$$

$$\hat{\Sigma}^{x,z}_{t|t-1} = \sum_{i=0}^{2n} {\omega_c}^{[i]} \left(\hat{\chi}^{*[i]}_{t|t-1} - \hat{x}_{t|t-1}\right)\left(\hat{\chi}^{*[i]}_{t|t-1} - \hat{x}_{t|t-1}\right)^T$$

$$\hat{x}_{t|t-1} = \sum_{i=0}^{2n} {\omega_m}^{[i]} \chi^{*[i]}_{t|t-1}$$

$$\hat{z}_{t|t-1} = \sum_{i=0}^{2n} {\omega_m}^{[i]} \hat{Z}^{*[i]}_{t|t-1}$$

$$\chi^{*}_{t|t-1} = g(u_t, \chi_{t-1|t-1})$$

$$\hat{Z}_{t|t-1} = \mathrm{h}(\hat{\chi}_{t|t-1})$$

$$K_t = \hat{\Sigma}_{t|t-1}^{x,z} {S_{t|t-1}}^{-1}$$

$$\hat{x}_{t|t} = \hat{x}_{t|t-1} + K_t(z_t - \hat{z}_{t|t-1})$$

$$\Sigma_{t|t} = \hat{\Sigma}_{t|t-1} - K_t S_{t|t-1} {K_t}^{-1}$$

Where χ represents the σ-points and ω_c, ω_m the weights defined as:

$$\chi^{[0]} = \mu \chi^{[i]} = \mu + (\sqrt{(n+\lambda)\Sigma}\,)_i$$

$$\chi^{[i]} = \mu - (\sqrt{(n+\lambda)\Sigma}\,)_{i-n}$$

$$\omega_m{}^{[0]} = \frac{\lambda}{n+\lambda}$$

$$\omega_c{}^{[0]} = \omega_m{}^{[0]} + (1 - \alpha^2 + \beta)$$

$$\omega_m{}^{[i]} = \omega_c{}^{[i]} = \frac{1}{2(n+\lambda)}$$

With α, β, κ, λ parameters that the user must choose according to the specification required.

After the transformation, the estimation has an accuracy of the third order of the Taylor expansion, which is two order higher than the one reached with the EKF.
At the end of the propagation, in this case, the normal distribution of the system is maintained also after the linearization and a better result is obtained even if the computational effort is higher than the other filter; a good solution is then to decide and tune properly the parameters of the two estimators and choose the best one evaluating the accuracy obtained and the computational effort of the computer since they are both fundamental in the realization of a good autonomous system [22].

2.2.3 Mapping

The mapping algorithm is the second fundamental component of the SLAM process, and the complete algorithm will be explained in detail in chapter 3.
Robotic mapping is related to computer vision and cartography. The goal for a robot is to be able to construct and use a map to localize itself and its reaching bases or beacons in it.
In general robots have two sources of information: the idiothetic, related to the use of the

dead reckoning methods that allow to obtain the absolute position of the robot, and the allothetic source that corresponds to the sensors.
The main drawbacks in the two sources are the cumulative error in the first one and the perceptual aliasing in the second; this last one is the perception of two different places recognized as the same and so leading to a series of errors in the creation of the map.
The map can be represented in two different ways:

- Metric framework: it considers a two-dimensional space where objects are placed with precise coordinates;

- Topological framework: it only considers places and relations between them; the resulting map represents the environment as a collection of nodes that are connected via arcs between them, corresponding to the paths.

In practice, metric maps are finer grained than topological ones. Higher resolution comes at a computational price, but it helps to solve various hard problems, such as the correspondence problem.
There exist many mapping algorithms and all of them are very different in both the information required and the final result provided: some algorithms are incremental, and hence can be run in real time, whereas others require multiple passes through the data, some algorithms require exact pose information to build a map, whereas others can do so
using odometry measurements; some algorithms are equipped to handle correspondence problems between data recorded at different points in time, whereas others require features to carry signatures that makes them uniquely identifiable.
An early representative of the former approach was Elfes and Moravec's important occupancy grid mapping algorithm, which represents maps by fine-grained grids that model the occupied and free space of the environment [23][24].
A second taxonomy of mapping algorithms is world-centric versus robot-centric. World-centric maps are represented in a global coordinate space. The entities in the map do not carry information about the sensor measurements that led to their discovery. Robot-centric maps,

in contrast, are described in measurement space. They describe the sensor measurements a robot would receive at different locations. At first glance, robot-centric maps might appear easier to build, since no 'translation' of robot measurements into world coordinates are needed. However, robot-centric maps suffer two disadvantages. First, it is often difficult to extrapolate from individual measurements to measurements at nearby, unexplored places. Second, if different places look alike, robot-centric approaches often face difficulties to disambiguate them, again due to the lack of an obvious geometry in measurement space. For these reasons, the dominant approaches to date generate world-centric maps.

All the sensors used for mapping the environment are subject to errors, often referred to as measurement noise. More importantly, most robot sensors are subject to strict range limitations. These range limitations make it necessary for a robot to navigate through its environment when building a map. The motion commands issued during environment exploration carry important information for building maps, since they convey information about the locations at which different sensor measurements were taken. Robot motion is also subject to errors, and the controls alone are therefore insufficient to determine a robot's pose relative to its environment.

A key challenge in robotic mapping arises from the nature of the measurement noise. Modeling problems, such as robotic mapping, are usually relatively easy to solve if the noise in different measurements is statistically independent. If this were the case, a robot could simply take more and more measurements to cancel out the effects of the noise. Unfortunately, in robotic mapping, the measurement errors are statistically dependent. This is because errors in control accumulate over time, and they affect the way future sensor measurements are interpreted. Avoid systematic errors is then fundamental to build maps successfully.

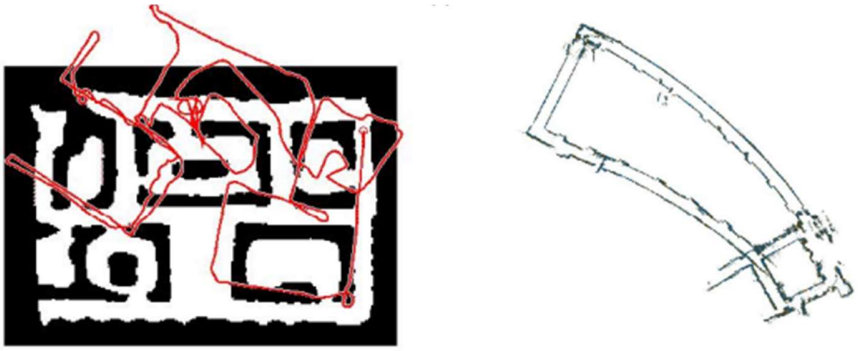


Figure 2.11: Typical Errors in Building Maps

Another aspect to take into consideration is the type of environment in which the robot is operating, in autonomous driving system scenarios the environment is dynamic and so the correspondence problem became very difficult to be solved and the computational effort increases its dimensionality leading to a complicated problem that is still not solved completely for many robots [25][26]. In FSD, however, this problem is not faced since the environment is expected to be time-invariant and structured.

2.3 Simultaneous Localization and Mapping (SLAM)

Since the 1990s, the field of robot mapping has been dominated by probabilistic techniques. A series of seminal papers by Smith, Self, and Cheeseman [27][28] introduced a powerful statistical framework for simultaneously solving the mapping problem and the induced problem of localizing the robot relative to its growing map. Since then, robotic mapping has commonly been referred to as SLAM or CML, which is short for Simultaneous Localization And Mapping [29], and Concurrent Mapping and Localization [30], respectively.
SLAM comprises the simultaneous estimation of the state of a robot equipped with on-board sensors and the construction of a model (the map) of the environment that the sensors are

perceiving. In simple instances, the robot state is described by its pose (position and orientation), although other quantities may be included in the state, such as robot velocity, sensor biases, and calibration parameters. The map, on the other hand, is a representation of aspects of interest describing the environment in which the robot operates.

The need to use a map of the environment is twofold. First, the map is often required to support other tasks; for instance, a map can inform path planning or provide an intuitive visualization for a human operator. Second, the map allows limiting the error committed in estimating the state of the robot. In the absence of a map, dead-reckoning would quickly drift over time; on the other hand, using a map the robot can "reset" its localization error by revisiting known areas (so-called loop closure). Therefore, SLAM finds applications in all scenarios in which a prior map is not available and needs to be built.

In general, the SLAM problem is defined as follow:

given the robot's controls $u_{1:T}=\{u_1, u_2, u_3,\ldots, u_T\}$ and the observations $z_{1:T}=\{z_1, z_2, z_3,\ldots, z_T\}$ the algorithm should provide the map environment *m* and the path of the robot $x_{0:T}=\{x_0, x_1, x_2,\ldots, x_T\}$.

The approach used is the probabilistic one in which the pose of the robot is never known exactly but it is assumed to be in a range with, if possible, a gaussian distribution that represents the uncertainties related to the robot's motion and observations.

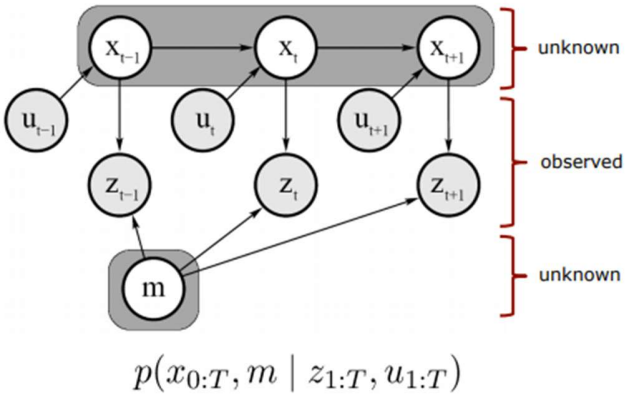


Figure 2.12: Graphical Model of the SLAM Process

2.3.1 State of Art

The key aspect of the SLAM is the loop closure; if this is sacrificed, SLAM reduces to odometry. In early applications, odometry was obtained by integrating wheel encoders. The pose estimate obtained from wheel odometry quickly drifts, making the estimate unusable after few meters [31]; this was one of the main thrusts behind the development of SLAM: the observation of external landmarks is useful to reduce the trajectory drift and possibly correct it [32]. However, more recent odometry algorithms are based on visual and inertial information and have very small drift.

The SLAM research done over the last decade has itself produced the visual-inertial odometry algorithms that currently represent the state-of-the-art; in this sense, visual-inertial navigation (VIN) is SLAM: VIN can be considered a reduced SLAM system, in which the loop closure module is disabled. More generally, SLAM has directly led to the study of sensor fusion under more challenging setups than previously considered in other literature.

By finding loop closures, the robot understands the real topology of the environment, and is able to find shortcuts between locations. The metric information makes place recognition much simpler and more robust; the metric reconstruction informs the robot about loop closure opportunities and allows discarding spurious loop closures. Therefore, while SLAM might be redundant in principle, it offers a natural defense against wrong data association and perceptual aliasing, where similarly looking scenes, corresponding to distinct locations in the environment, would deceive place recognition. In this sense, the SLAM map provides away to predict and validate future measurements: we believe that this mechanism is key to robust operation.

The maturity of the SLAM problem is evaluated on the basis of the following aspects:

- Robot: type of motion, available sensors, available computational resources;

- Environment: planar or three-dimensional, presence of landmarks, amount of dynamic elements, symmetry, risk of perceptual aliasing;

- Performance Requirements: desired accuracy in the estimation of the state of the robot, accuracy, and type of representation of the environment, success rate, estimation latency, maximum operation time, maximum size of the mapped area.

Current SLAM algorithms can be easily induced to fail when either the motion of the robot or the environment are too challenging; similarly, SLAM algorithms are often unable to face strict performance requirements.
Nowadays SLAM is entering in its third era, the robust-perception age, which is characterized by:

- Rubust Performance: operates with low failure rate; the system includes fail-safe mechanisms and has self-tuning capabilities in that it can adapt the selection of the system parameters to the scenario;

- High-Level Understanding: SLAM goes beyond basic geometry reconstruction to obtain a high-level understanding of the environment;

- Resource Awareness: the system is tailored to the available sensing and computational resources, and provides means to adjust the computation load depending on the available resources;

- Task-Driven Perception: SLAM is able to select relevant perceptual information and filter out irrelevant sensor data, in order to support the task, the robot has to perform; moreover, it produces adaptive map representations, whose complexity may vary depending on the task.

The architecture of a SLAM system includes two main components: the front-end and the back-end. The front-end abstracts sensor data into models that are amenable for estimation, while the back-end performs inference on the abstracted data produced by the front-end.

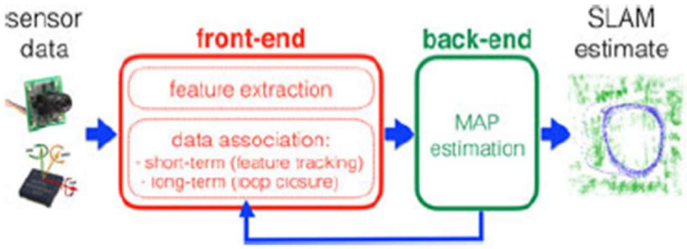


Figure 2.13: SLAM System

The current standard formulation of SLAM has its origins in the seminal paper of Lu and Milios [33]. Since then, numerous approaches have improved the efficiency and robustness of the optimization underlying the problem. All these approaches formulate SLAM as a maximum a posteriori estimation problem, and often use the formalism of factor graphs to reason about the interdependence among variables.

Assuming to estimate an unknown variable X; this X typically includes the trajectory of the robot and the position of landmarks in the environment. Given a set of measurements $Z = \{z_k : k = 1, \ldots, m\}$ such that each measurement can be expressed as a function of X: $z_k = h_k(X_k) + \varepsilon_k$, where $X_k \subseteq X$ is a subset of the variables, $h_k(\cdot)$ is the measurement or observation model, and ε_k is random measurement noise.

In MAP estimation, X is estimated by computing the assignment of variables X* that attains the maximum of the posterior p(X|Z):

$X^* = \text{argmax } p(X|Z) = \text{argmax } p(Z|X)p(X)$

where the equality follows from the Bayes theorem. In the equation, p(Z|X) is the likelihood of the measurements Z given the assignment X, and p(X) is a prior probability over X. The prior probability includes any prior knowledge about X; in case no prior knowledge is available, p(X) becomes a constant which is inconsequential and can be dropped from the optimization. In that case, MAP estimation reduces to maximum likelihood estimation. Note that, unlike Kalman filtering, MAP estimation does not require an explicit distinction between motion and observation model: both models are treated as factors and are seamlessly incorporated in the estimation process.

Moreover, it is worth noting that Kalman filtering, and MAP estimation return the same estimate in the linear Gaussian case, while this is not the case in general.

Assuming that the measurements Z are independent, the original problem can be interpreted in terms of inference over a factor graph and the variables correspond to nodes in the factor graph. The terms $p(z_k|X_k)$ and the prior $p(X)$ are called factors, they encode probabilistic constraints over a subset of nodes. A factor graph is a graphical model that encodes the dependence between the k-th factor and the corresponding variables X_k. A first advantage of the factor graph interpretation is that it enables an insightful visualization of the problem. A second advantage is generality: a factor graph can model complex inference problems with heterogeneous variables and factors, and arbitrary interconnections.

The key insight behind modern SLAM solvers is that the matrix appearing in the normal equations is sparse and its sparsity structure is dictated by the topology of the underlying factor graph. This enables the use of fast linear solvers [34]. Moreover, it allows designing incremental solvers, which update the estimate of X as new observations are acquired. Current SLAM libraries (e.g., GTSAM [35], g2o [36], Ceres [37], iSAM [34], and SLAM++ [38]) are able to solve problems with tens of thousands of variables in few seconds.

The SLAM formulation described so far is commonly referred to as maximum a posteriori estimation, factor graph optimization, graph-SLAM, full smoothing, or Smoothing And Mapping (SAM). A popular variation of this framework is the pose graph optimization, in which the variables to be estimated are poses sampled along the trajectory of the robot, and each factor imposes a constraint on a pair of poses.

MAP estimation has been proven to be more accurate and efficient than original approaches for SLAM based on nonlinear filtering.

Some SLAM systems based on EKF have been demonstrated to attain state-of-the-art performance. Excellent examples of EKF-based SLAM systems include the Multistate Constraint Kalman Filter of Mourikis and Roumeliotis [39], and the VIN systems of Kottas et al. [40] and Hesch et al. [41]. Not surprisingly, the performance mismatch between filtering and MAP estimation gets smaller when the linearization point for the EKF is accurate, when using sliding-window filters, and when potential sources of inconsistency in the EKF are taken care of.

As discussed, MAP estimation is usually performed on a preprocessed version of the sensor data. In this regard, it is often referred to as the SLAM back-end.

In practical robotics applications, it might be hard to directly write the sensor measurements as an analytic function of the state, as required in MAP estimation. For instance, if the raw sensor data is an image, it might be hard to express the intensity of each pixel as a function of the SLAM state; the same difficulty arises with simpler sensors. In both cases, the issue is connected with the fact that the system is not able to design a sufficiently general, yet tractable representation of the environment; even in the presence of such a general representation, it would be hard to write an analytic function that connects the measurements to the parameters of such a representation.

For this reason, before the SLAM back-end, it is common to have a module, the front-end, that extracts relevant features from the sensor data. For instance, in vision-based SLAM, the front-end extracts the pixel location of few distinguishable points in the environment; pixel observations of these points are now easy to model within the back end. The front-end is also in charge of associating each measurement to a specific landmark in the environment: this is the so-called data association. More abstractly, the data association module associates each measurement z_k with a subset of unknown variables X_k such that $z_k = h_k (X_k) + \varepsilon_k$. Finally, the front-end might also provide an initial guess for the variables in the nonlinear optimization. For instance, in feature-based monocular SLAM the front-end usually takes care of the landmark initialization, by triangulating the position of the landmark from multiple views.

The data association module in the front-end includes a short-term data association block and a long-term one. Short-term data association is responsible for associating corresponding features in consecutive sensor measurements; for instance, short-term data association would track the fact that 2 pixels measurements in consecutive frames are picturing the same 3-D point. On the other hand, long-term data association (or loop closure) is in charge of associating new measurements to older landmarks. Back-end usually feeds back information to the front-end, for example to support loop closure detection and validation.

The preprocessing that happens in the front end is sensor dependent, since the notion of feature changes depending on the input data stream we consider.

The problem of simultaneous localization and mapping has seen great progress over the last 30 years. Along the way, several important questions have been answered, while many new and interesting questions have been raised, with the development of new applications, new sensors, and new computational tools.

SLAM and related techniques, such as visual inertial odometry, are being increasingly deployed in a variety of real-world settings, from self-driving cars to mobile devices. SLAM techniques will be increasingly relied upon to provide reliable metric positioning in situations where infrastructure-based solutions, such as GPS, are unavailable or do not provide sufficient accuracy. One can envision cloud-based location-as-a-service capabilities coming online, and maps becoming commoditized, due to the value of positioning information for mobile devices and agents.

In some applications, such as self-driving cars, precision localization is often performed by matching current sensor data to a high-definition map of the environment that is created in advance. If the a priori map is accurate, then online SLAM is not required. Operations in highly dynamic environments, however, will require dynamic online map updates to deal with construction or major changes to road infrastructure. The distributed updating and maintenance of visual maps created by large fleets of autonomous vehicles is a compelling area for future work.

For many applications and environments, numerous major challenges and important questions remain open. To achieve truly robust perception and navigation for long-lived autonomous robots, more research in SLAM is needed. As an academic endeavor with important real-world implications, SLAM is not solved.

The unsolved questions involve four main aspects: robust performance, high-level understanding, resource awareness, and task-driven inference. From the perspective of robustness, the design of fail-safe self-tuning SLAM system is a formidable challenge with many aspects being largely unexplored. For long term autonomy, techniques to construct and maintain large-scale time-varying maps, as well as policies that define when to remember, update, or forget information, still need a large amount of fundamental research; similar problems arise, at a different scale, in severely resource-constrained robotic systems.

Another fundamental question regards the design of metric and semantic representations for

the environment. Despite the fact that the interaction with the environment is paramount for most applications of robotics, modern SLAM systems are not able to provide a tightly coupled high-level understanding of the geometry and the semantic of the surrounding world; the design of such representations must be task-driven and currently a tractable framework to link task to optimal representations is lacking. Developing such a framework will bring together the robotics and computer vision communities.

2.4 Control and Trajectory Planning

As shown in figure 2.2, the overall autonomous system comes complete with the Control pipeline. This takes as input the information coming from the perception, localization and mapping pipelines and use them to communicate directly with the actuators to translate commands into mechanical action, in particular in steering, acceleration and brake.
The on-board ECU of the vehicle is, in fact, responsible for Motion Control.
The ECU of the SC19 is a dSpace MicroAutoBox, a real-time system able to work as an ECU without user intervention.
The control algorithm adopted works as a general control system, it receives requests coming from the decision-making process and, after checking the feasibility, provides a torque to the motor consistent with the inputs.
A fundamental part of the Control pipeline is the Trajectory Planning algorithm; this is the last algorithm run by the autonomous system and deals with the decision-making of the path that the vehicle must follow in order to drive itself along the path safely and as fast as possible.
Starting from the map obtained with the mapping algorithm, the Trajectory Planning pipeline computes the best trajectory to be followed knowing the position of the cones and the dynamic allowed and forbidden by the model of the vehicle itself.
The planning method is based on the Rapidly-exploring Random Tree (RRT) algorithm using Dubins curves to compute in real-time a feasible trajectory that maximize the vehicle

longitudinal speed in autonomous driving with the adoption of a Model Predictive Control (MPC) algorithm, implemented to control the lateral and longitudinal dynamics and compute the acceleration/deceleration commands and the front wheel steering angle to follow the predicted trajectory [42].

To reach this final goal it is then necessary to have a robust and reliable map on which the Trajectory Planning algorithm can work and provide the best trajectory to the Control subsystem.

2.5 SC19 Car

As mentioned before, the vehicle participating to the next FSD competition is the SC19, whose dimension are reported in figure 2.14:

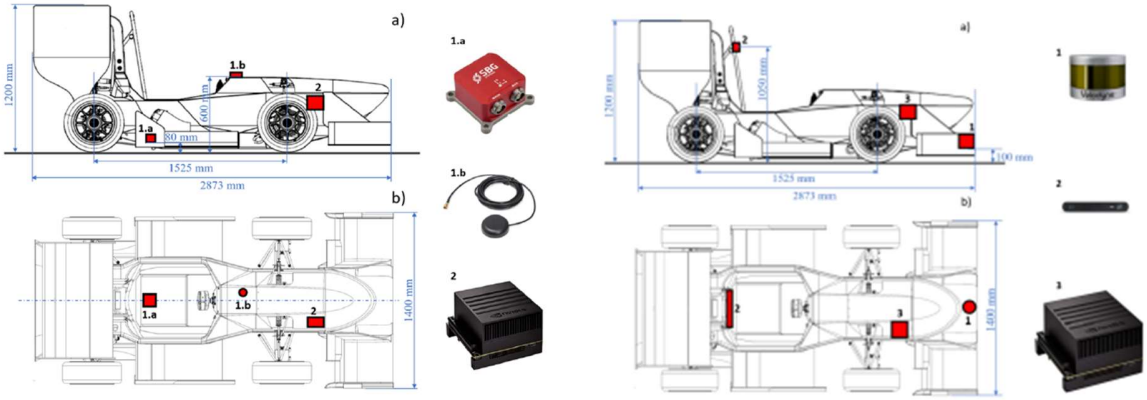


Figure 2.14: Hardware Configuration of the SC19

From the figure it is possible to see also the hardware layout related to sensors of the autonomous system. In particular, it possible to evaluate the lever arms related to the center of mass of the vehicle and all the sensors; in this way it is possible to relate all the different frames and the measurements coming from the sensors and create a functional Tf tree of the

autonomous system.
In the final configuration of the SC19 there should be one more sensor: the EMLID Reach M+, an additional GNSS module that uses the real-time kinematic positioning technique to estimate the position of the vehicle in motion. This will be used for the official race to improve the precision of the estimation of the position coming from satellite-measurements with the sensors fusion routine of the Kalman Filter; in this thesis work, however, this sensor is not implemented since the presence of one GNSS antenna is sufficient to develop the mapping algorithm and obtain robust results.
The same reasoning is done for the LiDAR, in fact for this work the sensor is not used and the mapping algorithm relies only on the camera information for perceiving the environment. In the final configuration the model chosen is a Velodyne LiDAR and is mounted on the front wing to allow the sensor to perceive the environment without obstacle; the LiDAR's working principle, in fact, is based on rotating mirrors emitting laser pulses and time returns of the wavelength to compute the distance of objects in the field of view, returning an accurate 3D representation of the environment as a dense pointcloud [43].

2.5.1 Sensors

The most important sensors used for developing the global mapping algorithm are the ZED 2 stereo camera and the SBG Ellipse-N IMU.

The ZED is a passive stereo camera that reproduces the way human vision works. Using its two "eyes", the ZED creates a three-dimensional map of the scene by comparing the displacement of pixels between the left and right images.
It perceives and understand the surrounding environment in 3D up to 20m distance, with increased accuracy in the close range. It captures two synchronized left and right videos of a scene and outputs a full resolution side-by-side color video on USB 3.0. This color video is used by the ZED software on the host machine to create a depth map of the scene, track the camera position and build a 3D map of the area.

The camera is also equipped with a Gyroscope, an Accelerometer, a Barometer, a Magnetometer and two temperature sensors that in this framework are not used, but provides useful information for other applications, that can be merged in the internal pre-processing of the SDK software, as explained in chapter 2.2.1 [44].

Figure 2.15: ZED Stereo Camera

The INS is a dead-reckoning navigation system which provides dynamic information and accurate vehicle's position over the time. The Ellipse -N adopted comes complete with a 64-bit microprocessor able to process the sensor fusion routine used by the localization pipeline. The sensor features also a dual-band antenna GNSS receiver that provides the orientation of the robot and the position through the satellite triangulation [45]. The adoption of the GNSS antenna is suitable for this application since the race is set in an outdoor environment and so the signal is needed by the antenna is always guaranteed; nevertheless, it is important to place it suitably on the car and so on a flat surface and without obstacle that can interfere with the signal. A different reasoning can be done for the SBG; this is mounted near the Center of Mass of the vehicle; however, this is not important since misalignments and lever arms can be taken into account through the initialization of the parameters as shown in chapter 2.2.2.

Figure 2.16: SBG Ellipse-N and GNSS Receiver

3 – Design and Implementation

Modeling the environment is the fundamental step of this thesis work and, in general, for any mapping algorithm for a robot or autonomous driving system.
As described in chapter 2.2.3, since in FSD the environment is structured and most important time invariant, the mapping algorithm chosen will provide a map based on landmarks which is a particular type of metric map that identify the position of certain characteristic objects in the environment and store it in a database. The adoption of such a map is related also to the fact that the autonomous system faces a large amount of data coming from sensors and it must run at the same time different pipeline algorithms; this map allows to reduce the memory occupied by the process by focusing the attention only on the landmarks and not on the whole environment.
The mapping algorithm adopted is based on the one published in GitHub: PerceptionAndSlam_KTHFSDV1718 [46] and specifically modified for the SC19 and for the RC model.
It is composed by two ROS nodes working together to build the global map used by the Control pipeline, and also a temporary map called Reactive Map used by the Trajectory Planning pipeline to compute in real time the path to be followed by the vehicle during the first lap of the race when the complete map is not available, and the environment is unknown.
The two nodes are: the Reactive Mapping node and the Global Mapping node and they are both presented in detail in the following description.

3.1 Reactive Mapping

The Reactive Mapping is ROS node written in Python 3.7 developed to create the first short-term map used to allow the autonomous system to compute a local trajectory when the vehicle is moving for the first time in the path. It is a map used for a rapid decision making in which the Control pipeline provides an instantaneous trajectory based only on the information coming from the real-time perceived cones and it does not rely on previous or future measurements.

Being the first node of the Mapping pipeline, the input of the node is the /perception_cones topic, which is the topic coming from the perception pipeline containing all the information about the cones perceived.

The data contained in this topic are a series of arrays of size 4N, where N represents the number of cones detected and recognized by the ZED. The four records of each array characterize each cone describing the position in the space with respect to the local camera frame, and the color assigned by the SSD algorithm. The shape of each array is then as follow: [x, y, z, color]; as it is possible to see the position is described in three-dimensional coordinates even if initial assumption of 2D motion, and the color is identified by a number coding the color identified.

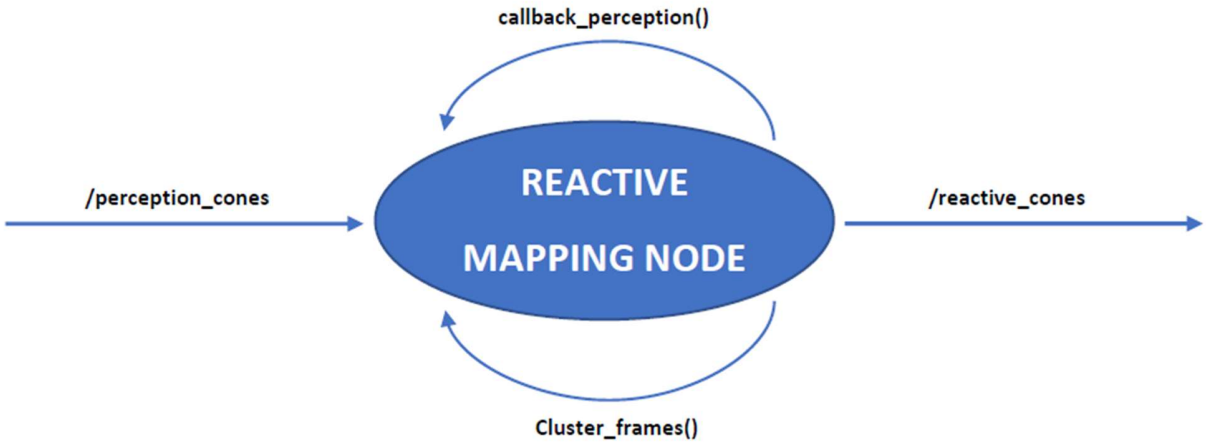


Figure 3.1: Reactive Mapping Node

At it is possible to see in figure 3.1, the node's working principle is based on two callback functions; functions that perform an internal algorithm every time a message is received in input until the input buffer is empty or the process is stopped.
The first function is the callback_perception(), this integrates the information coming from four successive frames to have a rich information on which the position of the cones can be consolidated. At the end of the process, the function provides every time a matrix of dimension 4Nx4 that is passed to the second part of the algorithm.
The second function is the cluster_frames() and it is called within the first function, so the input of this specific function is not directly the input message of the node but a pre-processed one, in particular it is 4Nx4 matrix with the data of the four successive frames.
This function identifies each cone from the matrix received in every frame and, by assigning them a cluster identification, computes the mutual distance between to understand if the same cone is present in more than one frame.
This is made assigning a threshold value that is estimated assuming a constant velocity of the vehicle of 3 m/s and an update frequency of 60 Hz, so the algorithm is able to derive the distance at which every single cone should be seen in the successive frame and if this distance is respected the function assign the same cluster identification to the cones.
At the end of this first process, the 4Nx4 matrix provides a series of cluster identification for each cone in the frames; this allow to generate a dictionary named clusted_dict containing, for each identification, the number of times the specific cone has been detected in the frame's matrix.
At this point the node processes each element of the cluster_dict and, if the number of times the element has been seen is grater or equal to three, the cone is legitimated, and the function computes the position and the variance of the cone based on the information coming from the different frames and the Perception pipeline output.
The choice of using four frames and augment the number of detections to three for the legitimation, comes from the experimental testing since it has been proved that with this configuration of parameters the Reactive Mapping stores the exact number of cones in the field of view. With the original configuration provided by [46], the results obtained are not that robust since with a lower number of hits needed the node tends to legitimate a greater

number of cones due to the assignation of different cluster identification to cones that are actually the same, and so this kind of error is not compensated with the original solution.

The output of the node is the topic /reactive_cones containing the information of the cones elaborated by the functions described. The data structure is coherent with the input one, in fact, it an array of size 4N where, in this case, the information contained are the results coming from this first process of the algorithm in the following structure: [x_centroid, y_centroid, variance, color]. The output contains condensed and useful information to be passed to the Global Mapping node and so the z-component of the position of the cones is neglected thanks to the initial assumption of 2D motion. The four records of the array are the updated position of the cones in the xy plane, the variance associated and finally the color of the cones that has been assigned from the perception pipeline, which is the same read in the input topic.
The code and the choice of the parameters associated to the Reactive Mapping Node is presented in detailed in Appendix A.5.
The reactive map obtained is related mostly to the driving of the first lap of the race, then, after the global map is built, the reactive map is discarded and used only to update the position of the cones if some of the previous measurements are wrong.

3.2 Global Mapping

The Global Mapping is the second fundamental ROS node written in Python 3.7; in this case the node provides a complete and global map of the circuit that can be passed to the Control pipeline not to generate an instantaneous trajectory but a global one of the whole path, if possible. For this reason, the control strategy adopted after the global map is acquired is a MPC where the path planning is computed with a horizon, and this is possible thanks to the already known position of the all the cones in the path and so the trajectory can be studied and computed for the whole path before the vehicle is actually in the instantaneous position.

From the second lap on, in fact, the control strategy passes from mapping to localization; this is a fundamental step since once the map is known with high accuracy the CPU of the system can stop processing the mapping algorithm and focus only on the localization, saving computation effort and improve the quality of the localization, leading to a more accurate control strategy and so better dynamic performances, fundamental for winning the race.

To build the global map the node needs two sources of information: the first is coming from the Perception pipeline and the second from the Localization one. Concerning the input related to the perception, the node subscribes from the topic /reactive_cones, which contains all the useful information of the perception pipeline already pre-processed and filtered, as explained in the previous description, and so ready to be used by the autonomous system to develop the map.

The second input comes directly from the Localization stack, in particular the node subscribes from the topic /odometry/filtered, which contains the pose and all the estimated states of the vehicle after the application of the cited Kalman Filter.

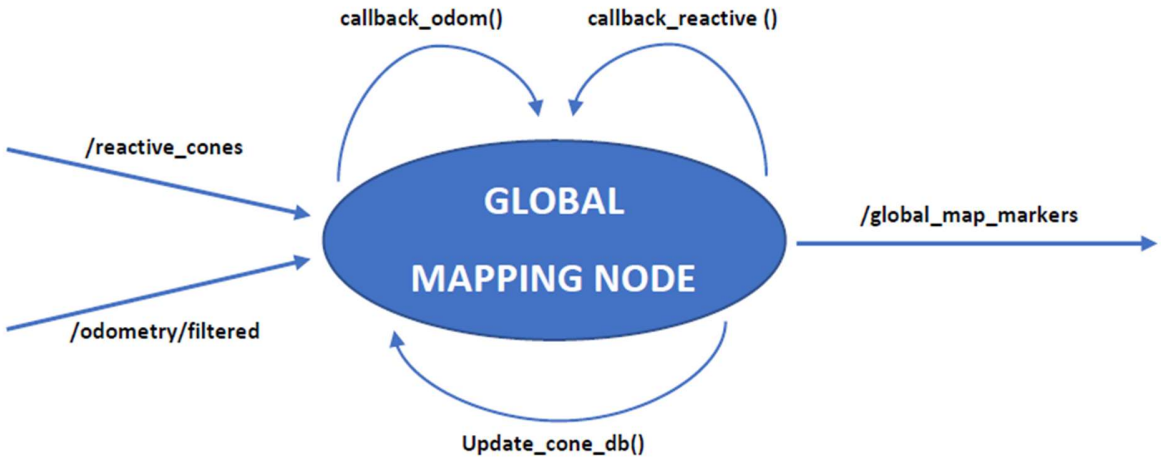


Figure 3.2: Global Mapping Node

As shown in figure 3.2, and similarly to the Reactive Mapping node, this node is based on three callback functions:

- callback_odom(): it uses the information coming from the topic /odometry/filtered to translate them from the local reference frame into the absolute one. The transformation includes the orientation of the vehicle expresses in quaternions, into the Euler configuration with the definition of Roll, Pitch and Yaw;
- callback_reactive(): the function receive the matrix coming from the Reactive Mapping node with the position and the color of the cones and reshape the array delating the useless information, such as the z component of the position of the vehicle, and save the filtered data into the variable self.reactive_cones with the following structure: [x, y, color];

- update_cone_db(): this function, differently from the others, is not called every time a message is read by the system, instead, it elaborates information with a constant frequency that is designed by the user and chosen by tuning the algorithm between result and performances. The function's working principle takes as input the information coming from the previous functions. The first operation consists in checking if a certain cone is in the operational field of view. The ZED can detect cones within a range of 20 meters and an angle of 70^{o}, however, the actual range used for the mapping algorithm is narrower; this is solution is adopted to reduce the computational effort of the algorithm and guarantee an optimal performance of the autonomous system. After this first control, the absolute position of the cone is computed in order to deal with the global positioning system, namely the reference one. At this stage the algorithm performs a second control; in this case the cycle is meant to check if a cone has been seen for the first time; if so, the cone is added to an internal database with a new identification number, otherwise the number of Hits associated to each cone in the database is incremented, the covariance value decreases, and the position of the cone is updated. This loop is the fundamental core of the mapping algorithm since it allows to update and consolidate the position of

each cone with high accuracy and repeatability if the process is performed for more than one lap during the race. The main idea is that if a cone is seen more times during the race, the covariance associated decreases proportionally since the algorithm can trust more the previous estimation made. The same reasoning is performed in the opposite situation, if a cone is detected with a frequency that does not correspond to the expected one or if it is not detected anymore, the covariance of the cones increases until it reaches a threshold value that delates the cone from the map, avoiding to store in the database an extra cone or one that should not be in the current position.

The choice of the parameters is the second fundamental step of the designing of the Global Mapping algorithm since small changes in the tuning of these bring to completely different results and it is important to study the behavior of all of them in the overall algorithm.
The most important parameters are related to the operational field of view ranges: these have been set to 65^{o} for the angular opening and between 1 meter and 4 meters for the minimum and maximum distances. This choice is consistent with the distance between the pair of cones at the race following the official rules. In this way only one couple of cones in processed at every iteration and the wide angular opening allow to detect cones also when the vehicle is turning.
Another important parameter is related to the value added to the database concerning the hits of every cone; this has been designed such that every time a cone is seen more times the number of hits is increased by seven units, whereas when it is not detected anymore the number of hits is decreased of one unit. The logic behind this choice is related to the fact that the vehicle speed is not constant as it is assumed to be in the algorithm and thus the node tends to delate cones that are in the right position. Adopting a non-proportional solution as the one described, this problem is overcome, and a more accurate map is obtained.
The last important parameter that has to be described, is related to the process of position update; for this purpose, the algorithm uses the Exponential Moving Average (EMA).
An exponential moving average is a type of Moving Average (MA), which is a calculation used to analyze data points by creating a series of averages of different subsets of the full data set, that places a greater weight and significance on the most recent data points.

Applied to the current function, it defines the new position of the cones giving a major weight to the first measurement rather than the latest. This is described by the parameter 'alpha', which is set to 0.9, meaning to trust more previous position computed in the first lap than those derived from the updating process. The parameter alpha is the so-called "degree of weighting decrease" and can assume a value between 0 and 1, where the higher is alpha the more weight is given to the older measurements, according to the following equation:

$$position_{updated} = (1 - \alpha) \times position_{new} + \alpha \times position_{old}$$

This choice is motivated by the fact that the Mapping algorithm is not an exact method, thus the continuous iterating process can lead to an error accumulation that can cause bad results and, in the worst case, the not closure of the loop at the end of the first lap.

The final output of the Global Mapping algorithm is published on the topic /global_map_markers. This is an array containing all the fundamental and final information for building the global map, in particular, it is composed by seven records as follows: [x, y, color, covariance, hits, inFOV, id]. The first two parameters are related to the computed global position of the cones with respect to the Map referral frame, whose origin coincides with the initial position of the vehicle at the starting of the autonomous system. Related to these parameters is the fourth record containing the covariance estimated by the algorithm; this assumes a graphical meaning when the map is visualized in Rviz as will be presented in chapter 4. The parameter inFOV is a flag variable that represents whether or not the cone detected is the operational field of view of the algorithm, according to the parameters described above.

Finally, the last two parameters of the output array are related to the number of times a cone has been detected and the associated identification number to characterize each cone. These parameters are not so important for the visualization of the map, but they are the fundamental information to be passed to the Control stack to identify each cone and derive the best trajectory for the navigation.

4 – Results and Discussions

In this chapter are presented and discussed the results obtained in two frameworks: the simulation and the experimental sessions.
Simulation is used to understand the working principle of the mapping pipeline and to check if the system is able to communicate with all its component in a proper way. The limitation coming from the low effectiveness of the simulated process is overcome with real experimental data. This is a fundamental step to validate the design developed especially when dealing with different sensors whose reference system could be different.
The results presented are obtained with an RC model simulating the SC19 in scale; the racing vehicle, in fact, requires to be prepared in advance and, due to the time required to get ready it is recommended to validate the autonomous system with other method and the most effective is a scaled version of the vehicle used for rapid prototyping; in this way many tests and experimental sessions can be carried out in a relative short time.
Adopting this kind of solution allows to tune the parameters of the algorithm fast and, at the same time, to test the results obtained with a similar architecture to the final one mounted on the SC19 for the race.

4.1 Simulation

The first step of the validation of the Global Mapping algorithm is the simulation.
This is a fundamental preliminary process that allows to understand the working principle of the whole system and whether or not there are errors in the coding, in the implementation of the algorithm or in the architecture developed with ROS, thus with the communication between nodes and pipelines for the realization of the global map.

To obtain useful results also with this first step, it is important to simulate the real condition in the best way, thus both the dynamic system and sensors, and the environment. To this purpose simulator provided by the Edinburgh University Formula Student team (EUFS) has been adopted. This simulator includes five different scenarios implemented in Gazebo, on which the algorithm can be tested, and they are conform to the rules of the FSD. The most important for the validation of the algorithm are the small track and the big track since they provide a race like path which is always the same and so it is possible to study the behavior of the algorithm with a great amount of data and tune the parameters according to the results obtained. The randomly generated track can be used for a final validation in a completely new environment, but it is not suggested to use it for the initial calibration. Finally the acceleration track and the ski pad simulates the path that the vehicle must face during the race; for these tracks can be done a different reasoning since they have a known geometry and thus also the control strategy used at the race is different, for this reason they are not so relevant in this thesis work, nevertheless they can be used as additional sources for the validation of the algorithm; in particular the acceleration track gives some interesting results in a straight path when the speed and the acceleration of the vehicle are higher.

Regarding the vehicle, this is simulated with a custom race car to allow the user to equip it with the sensors needed and, most important, with the right model. The importance of this choice is related to the fact that different sensors could provide different results, or the working principle is different, so it is fundamental to choose the right sensor between those available in the ros-melodic-robotnik-sensor package.

For the simulation the vehicle is equipped with both the perception sensors: the ZED stereo camera and the Velodyne VLP16 LiDAR, even if this last one is not used for this experiment, while concerning the localization stack the IMU is not adopted for the simulation. This choice is consistent with the above consideration about the model of the sensor, in fact, the SBG INS is not present in the library provided and so it is not substituted with any other sensor of the same type.

This choice leads to the impossibility to test and simulate the localization pipeline; however, this problem is overcome using the odometry source provided by the simulator: the Ground Truth Odometry of the simulators is an exact source of odometry that substitute the

Localization pipeline with the exact position of the vehicle in every moment. In this way running just the Mapping pipeline without the Localization one, the system has all the information needed to elaborate and build the global map. Relying on this information ensure the loop closure of the track with no troubles; the same cannot be obtained if another source of odometry is used to localize the vehicle; in fact, if instead of the Ground Truth it is used the odometry coming from the wheel encoder only, the system would diverge and the loop closure is not guaranteed to be achieved since any little slipping of the wheel leads to an error that is accumulated over time.

As mentioned in previous chapters, the system at this stage is not self-driven, since the goal of the mapping algorithm is not that of having a ready autonomous system, but one able to provide to the Control pipeline all the necessary information for the final task and be ready for the race. For the simulation the vehicle is driven through a controller with a suitable ROS node written in Python and reported in Appendix A.7 that allows to accelerate and steer the vehicle imposing a maximum speed to allow the response of the system in different dynamic conditions.

After all the measurement coming from the sensors are synchronized, the algorithm can be tested and the last fundamental part begins: the tuning of the parameters, which allow to optimize the algorithm and to understand the effect of each of them on the map obtained; understanding how to work on the final tuning when dealing with the real vehicle.

The map obtained is plotted using Rviz, a software provided by ROS that allow to show graphically sensor's measurements and post-processed data.

In the following picture are reported the results obtained for the small track and the big track on which the algorithm can be validated more robustly:

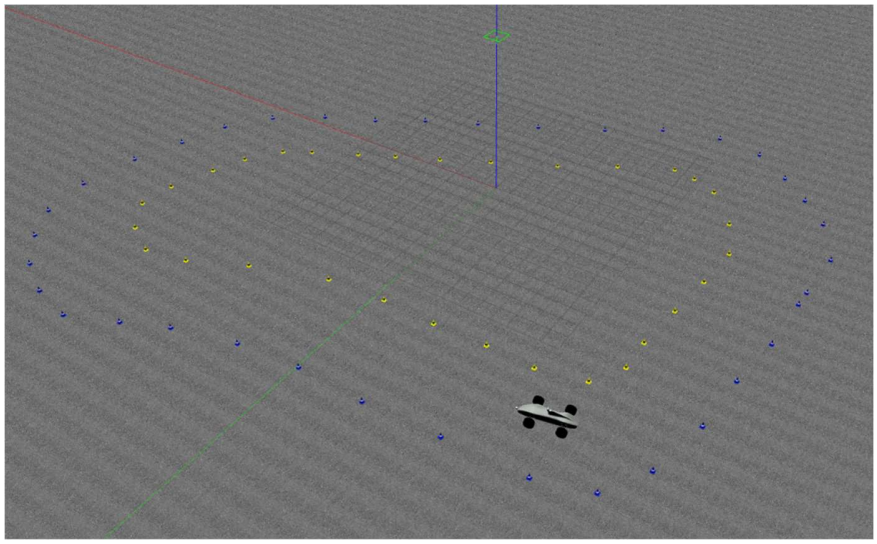


Figure 4.1: Small Track Circuit on Gazebo

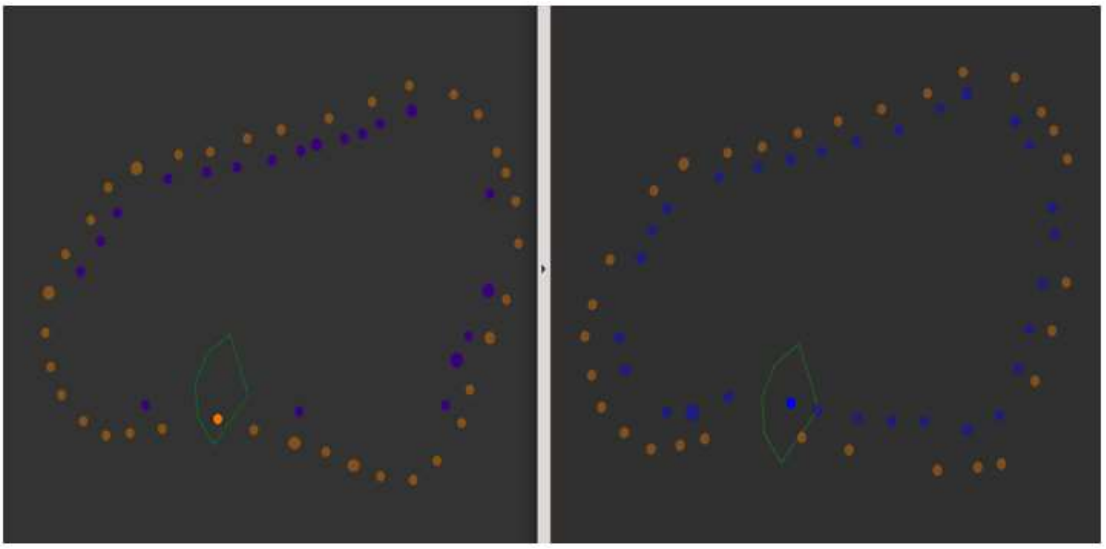


Figure 4.2: Small Track's Map in Rviz

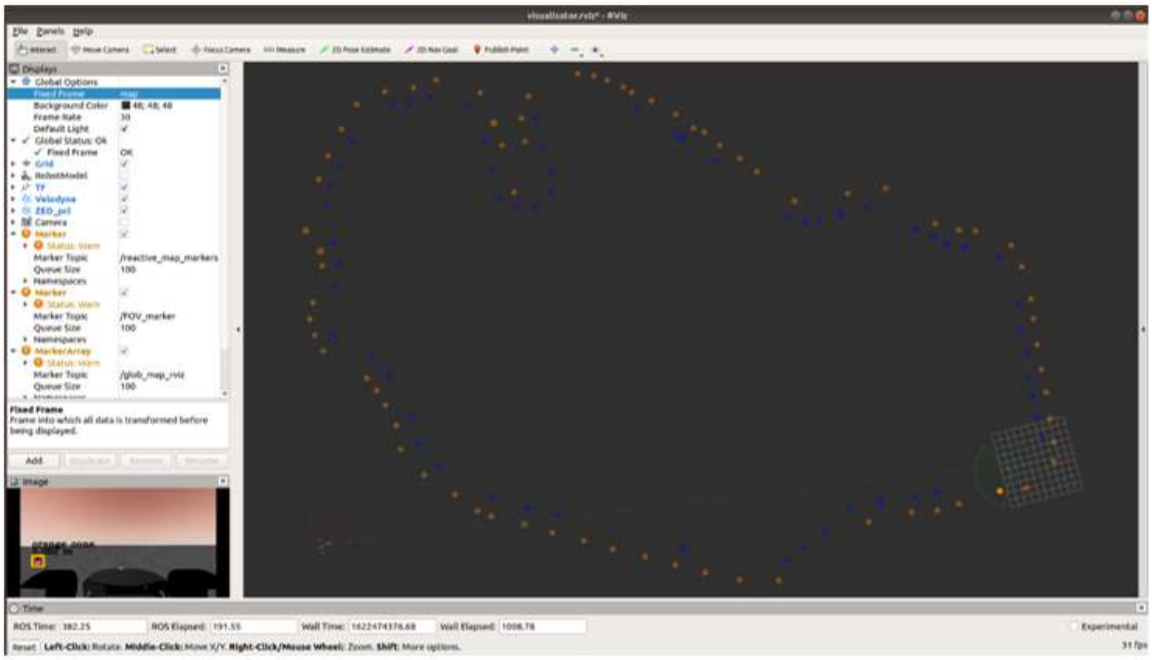


Figure 4.3: Big Track's Map in Rviz with Configuration

As it is possible to see in figure 4.2 and 4.3 the geometry of the circuits is perfectly mapped and this is a first and important result obtained; the Ground Truth helps to reach this goal, in fact, the loop closure is not only reached but also ensured in these conditions.

Focusing the attention of figure 4.2, the image presents two maps; these are the same map recorded with the same parameters but with two different driving conditions. In the left image the path followed is shifted on the left with respect to the imaginary central line between each couple of cones, whereas in the right image the driving is shift to the right of the same imaginary line. These simulations allow to understand how the perception pipeline actually works in different conditions. During straight sections the results are the same, while when facing a turn, the different driving conditions lead to two different maps, where it is possible to see how the cones nearer to the vehicle, and so to the camera, are detected and saved correctly, while cones on the opposite side are cut from the field of view and so they are not saved in the global map.

This problem cannot be solved in the simulation but, in the real word testing, it is possible to

adjust the operational field of view of the ZED such that when facing a turn, the system is able to detect cones in advance and, when during the turning operation the field of view is cut, the algorithm can store the information of those cones in advance and plot them in the map.

In general, both the two driving conditions generate a similar global map; the position of the cones plotted is the same of the simulated environment and also the covariance associated is very low, meaning that the algorithm sets the cone's position with high accuracy and reliability. Concerning the number of cones stored, the algorithm recognized almost all the cones, except those in the turning as described; the final result is quite good even if not sufficient to pass the map to the Control pipeline. The presence of blind areas can cause a bad trajectory planning and, in worst cases, the crash of the vehicle against a cone or the exit from the track. To overcome this problem real tests must be performed to avoid the limitations of the simulation and if the same problems are still present it is possible to develop a code to "automatically" place the missing cones, knowing the position of all the other cones with the help of a maximum likelihood estimator or other kind of estimator to have a final map dense of cones without holes.

After driving the vehicle for one lap, as in the small track, the global map of the big track is obtained as shown in figure 4.3. In the figure is possible to see the panel of parameters of Rviz, with whom it is possible to select which topic output the user wants to see and how to plot the different results on screen. In the bottom left corner, it is possible to see also the output of the topic /processedImage; this topic is one of the outputs of the Perception pipeline and shown what the camera is perceiving, in this way all the algorithms can be changed properly to obtain the best image result. The topic also provides the bounding boxes and the labels associated to each cone detected so that the user can understand how the perception algorithm is working and tune the parameters of the ZED such that the turnings are faced rightly as explained for the small track.

The overall map presents the same characteristics of the one obtained for the small track: the cones in the database are set properly, however, some cones are missing, and these cannot be perceived with only one lap; a solution could be to drive the vehicle for two laps moving the center of the trajectory from left to right to detect all the cones.

Even though some cones are missing, the geometry of the track is perfectly recognized and also the cones of very narrow turnings are stored properly and with low covariance.
The length of the track allows to validate the algorithm over the time and the result obtained leads to a robust and reliable Mapping Algorithm also for a long time travelling; the time to perform one lap is, in fact, of about 5 minutes, which is a relative long time if compared to the time estimated to perform the first lap at the official race.
Concerning the driving parameters is worth explaining the choice of the maximum speed and the update frequency of the perception algorithm.
For the velocity the limit has been set to 3 m/s which is a good compromise between speed performance and time allowed to the algorithm to perceive the cones. For the first lap, in fact, it is necessary to limit the speed and allow the system to work properly since it is fundamental to have a precise global map, otherwise it may be needed one more lap at low velocity to map the entire track properly leading to a loss of time for winning the competition.
A parameter related to this purpose is the update frequency of the perception algorithm; this may be thought as if the lenses of the camera are normally closed and are opened with the update frequency to perceive the surrounding environment. Knowing this consideration, it is possible to tune it, finding the best compromise between performances and computational effort. Setting the parameter to the highest value means to have the lenses always open and ready to detect cones without any chance to miss one of them, on the other hand this is very expensive from the computational point of view since the system is always working with the camera and must still work to guarantee the right functionality of the overall system.
Choose a law value of the update frequency is not recommended since the effect obtained is a map full of cones that does not represents the real map since the perception algorithm recognize different cones as the same in different positions and as a consequence, due to the structure of the Reactive Mapping node, the algorithm plots the same cone in different places because of the delay between the camera frame and the Ground Truth provided by the simulator, leading to the so called “trail effect”.
The results shown in the figures are obtained after many simulations and the final value chosen is set to 60Hz, which is a reasonable value to have good performances while saving computational effort.

4.2 Experimental Theory

For the experimental session, in every field of application, it is important to study the theory correlated and the setup to prepare the testing session in advance and without hitches and troubles.

For a good validation of the algorithm is important also to study which dynamic characteristics the user want to check and the condition in which the experiment takes place. The setup that must be prepared in advance is related mainly to two categories: the sensor setup and the track setup.

The sensor setup is related to the initialization of the INS and the ZED in order to have them ready to acquire data and allow the pipelines in which they are involved to perform properly.

In chapter 2.2.2 is presented the initialization of the INS sensor through the sbg_driver node to make some a-priori considerations about the hardware setup and the dynamic and driving conditions, in particular it is assumed to deal with a two-dimensional motion with low values of slip angles due to the medium velocity reached by the vehicle and, to simplify the filter computation, it is assumed that the x-axis of the associated moving frame is always pointing in the forward direction.

Related to the SBG, a fundamental step is the initial calibration once the pipeline is launched. To reach a sub-meter accuracy in the estimation of the position of the vehicle, it is necessary to leave some time to the sensor to calibrate the data measured by the SBG and by the GNSS antenna.

The stages of such calibration are three: Vertical Gyro Mode, Attitude and Heading Reference System Mode and Full Navigation Mode.

In the first stage the measurements rely only on the INS thus the information obtained is poor and not sufficient to correctly localize the vehicle. In the second stage the GNSS signal is acquired but the resulting measurements are not synchronized and so not acceptable. In the final stage the synchronization of the two sensors allows to obtain an optimal measurement to estimate the pose of the car accurately.

As shown in figure 4.4 the final configuration is achieved when the reference frame is set to

UTC | SYNC and so the sensor is fully calibrated and also the GNSS signal is accounted in the measurements.

During the race the time allowed to prepare the vehicle is not enough to perform a series of tests and check periodically the value registered, a solution can be obtained optimizing the time allowed and calibrate the sensor when the vehicle is moved from the boxes area to the starting point on the track.

For the testing sessions with the RC model some problem raised with the calibration since the approximation made for the lever arms and the orientation of the sensor are reasonable, but the time required for the initial calibration increases and this is not suitable for the race event.

To speed up the calibration process it has been proved that driving the model with high accelerations and covering a wide path with a lot of turning helps the calibration stage to reach the Full Navigation Mode. A similar reasoning can be done for the GNSS, in fact, driving the car in a wide and free environment allow a fast detection of the satellites and so the sensor can quickly calibrate the INS signal to the GNSS one.

Even taking into account these considerations, sometimes the calibration was still difficult to be reached and, in particular, the sensor used to read a constant acceleration along the z-axis, while it should be stable at zero since the dynamic associated is expected to be a two-dimensional one. To overcome this problem, it is possible to move the vehicle up and down leading to a real measurement of the z-component of the acceleration, in this way once the system is placed on the ground the SBG is able to understand that the velocity along z is zero and therefore it calibrates itself and it is ready to perform the experiment.

All these problems are due to the approximation done for the RC model and for the dynamic associated; regarding the SC19 car the system is well designed and the dynamic of the vehicle allows to reach higher velocities and much more detectable movements of the system with respect to the RC model, as a consequence the calibration stage for the competition vehicle is faster and easier to complete in order to have the system and the sensors ready for the competition.

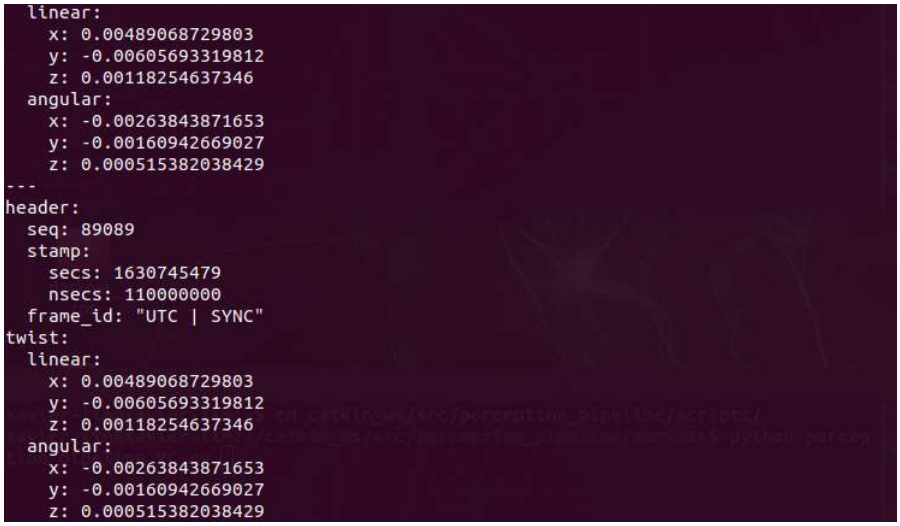


Figure 4.4: SBG Calibration

Concerning the ZED initialization, the only operation needed is a calibrate the camera by focusing it on a predefined pattern that allow a fast and automatic calibration once for all the testing sessions.

A final and important theoretical consideration must be done about the localization stack, in particular about the passage form the simulation to the real word testing.

This passage is necessary to also test the localization pipeline that, as explained, cannot be tested with the simulation. When dealing with real world data, there is no certain information about specific measurement, thus the system cannot rely anymore on the exact information coming from the Ground Truth provided by the simulator.

In real world testing it is then possible to validate the whole system, including the localization stack with the equipped INS sensor and the communication between all the different pipelines of the vehicle.

The passage from the Ground Truth information to the /odometry/filtered topic is then not only obliged but also fundamental to validate the Global Mapping algorithm in a race kind environment.

4.3 RC Model

As explained, the mapping algorithm is tested on the RC model used for rapid prototyping and obtain fast result without spend a lot of time in preparing the SC19 for the testing.
This solution allows to collect an important amount of data with real testing in a relative short time, thus real results are obtained and can be studied to develop a better algorithm that can actually be implemented on the real autonomous system for the race.
The RC model is an electric remote-controlled car used to study only the mapping pipeline. Being remote-controlled the experiment cannot test the whole autonomous system, the pipelines interested are those related to the Global Mapping algorithm, thus the Perception and Localization stacks are involved in the testing while, concerning the Control and Trajectory Planning pipeline, real test must be carried out on the SC19 and preliminary studied can be done only with simulations or with a different car model equipped with controlled actuators able to follow instruction coming from the autonomous system.
Being a model of the SC19, the RC car is equipped with the same sensors used for the race; in this way the Global Mapping algorithm developed can be fully tested taking into account the scale of the model with respect to the race car.

Figure 4.5: RC Model - First Configuration

As it is possible to see in figure 4.5, the model is equipped with the ZED stereo camera, the SBG Ellipse -N INS and the GNSS antenna.
Alle the sensors are directly connected to the NVIDIA Xavier running Ubuntu 18.04 with the complete autonomous system.
The hardware configuration comes complete with a battery, mounted on a 3D designed cart, that supply the Xavier with a suitable DC-DC to provide the right voltage to the system. Finally, a USB hub is mounted on top of the structure to allow a fast connection between all the sensor and the computer.

After some experimental tests, some problems arise with the structure of the whole model since the hardware configuration studied and presented above is not perfect. This is mainly due to the oscillations coming from the dynamic of the car and the weight not well distributed on the model, which results to have a Center of Mass that does not coincide with the geometric center of the vehicle.
For these reasons a new solution has been implemented to take into account the mentioned consideration.
The plastic body of the model has been removed and a suitable structure has been designed to allow all the sensors to be in a fixed position and with a spatial distribution that allows a good stability to the whole system.
The structure has been designed using SOLIDWORKS and the final result is shown in figure 4.6.

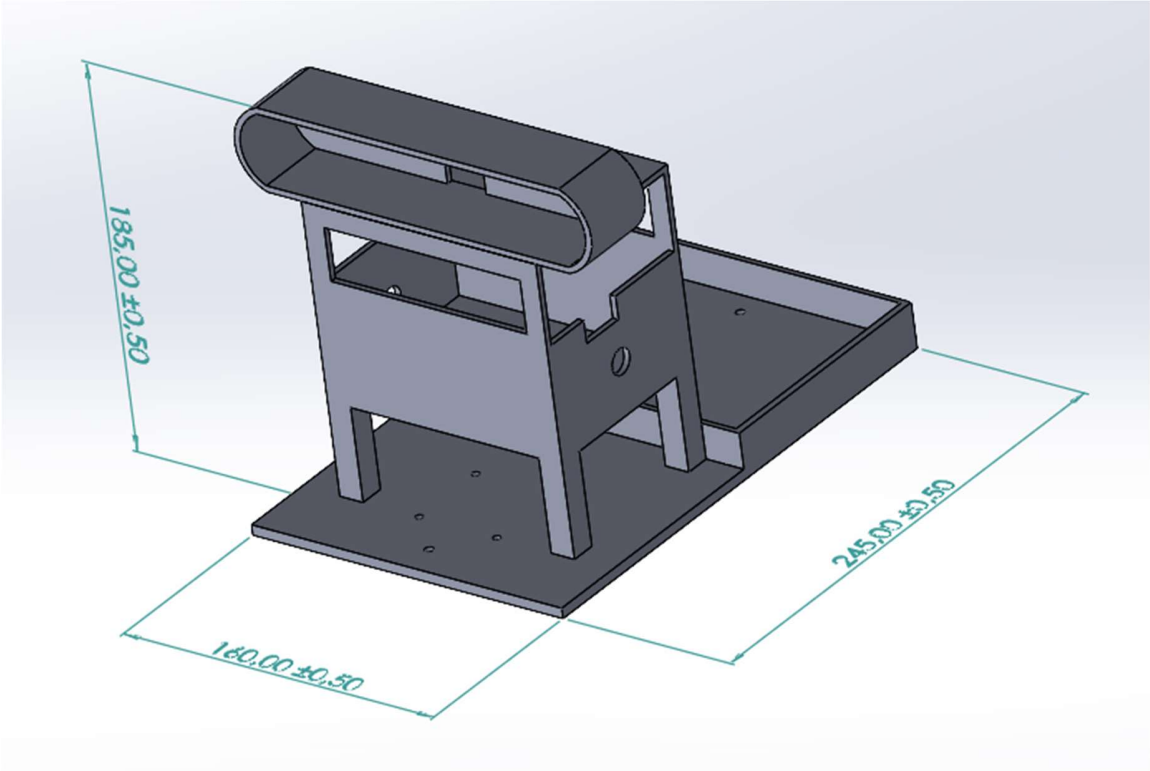


Figure 4.6: 3D Structure for the RC Model

The structure works as a support for all the sensors and for the NVIDIA Xavier itself. Being designed as a unique part, it is compact and robust at the same time and the flatness of all the surfaces allow a faster calibration of all the parameters of the sensors differently from the initial configuration, where the plastic body's shapes were not suitable for mounting all the sensor with precision as they are expected to be.

The structure designed, suitably transformed in a STL file, has been 3D printed and the final result, with all the sensor mounted, is shown in figure 4.7.

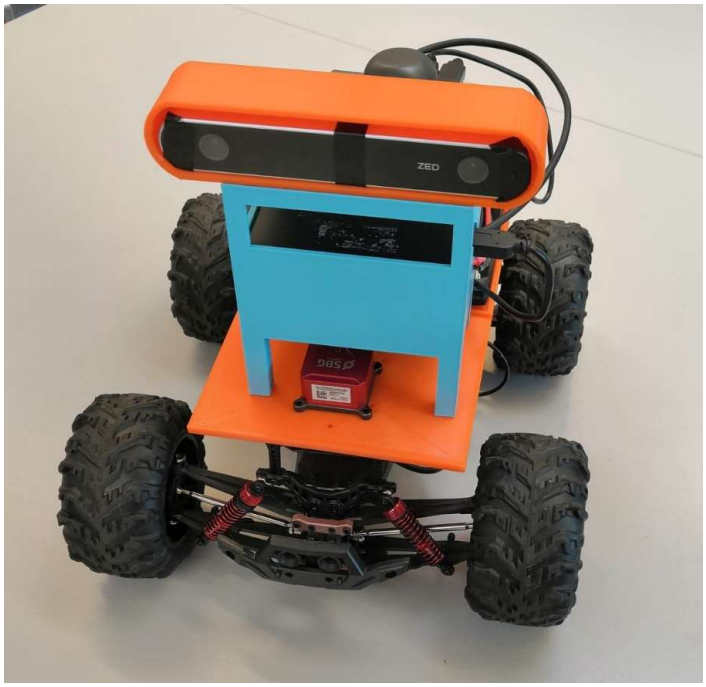


Figure 4.7: RC Model - Final Configuration

As it is possible to see all the sensors are well mounted on the structure, which guarantees a robust support to the whole vehicle.
The sensor configuration reflects the one mounted on the SC19, allowing to have a scaled version of the official vehicle as expected.
Moreover, the structure has been designed to allow the ZED to be in a higher position than the initial configuration, in this way the perception algorithm can perform better since the field of view of the camera is wider and, in part, compensates the limitation given by the small size of the model.

4.4 Validation and Results

The result obtained with the real system to validate the algorithm developed are presented in this section.

The testing sessions described are all performed at the AeroClub in Torino, where the team can work in a dedicated hangar and the experiments can be carried out in a wide environment with all the necessary instrumentation conform to the rule of the competition.

For testing the algorithm it is necessary only one lap of the track but, for this thesis work, it is important to check and validate every pipeline of the autonomous system involved in the Mapping algorithm; for this reasons the vehicle has been driven for a greater number of laps in order to control the localization stack and to have a double check of the performance achieved by the global mapping node, evaluating the results of the same track in different laps and, consequently, it is possible to evaluate the update process of the algorithm to restore the position of the cones when they are perceived for the second time on and eventually the removal of some of them when they are recognized as a previous error of the algorithm itself.

As stated in the previous section the preparation of the experimental setup is fundamental to save time and to avoid errors during the validation and recording of the algorithm to study the results offline and evaluate the behavior in real time. This operation can be performed thanks to Rosbag: a ROS package that provides a command-line tool for working with bags, which are file formats for storing ROS message data. These files allow to store, process, analyze and visualize the data recorded online using a computation graph or offline to record big dataset and, when the experimental session is finished, visualize the results recorded and work on them to obtain the final results or understand what and when something goes wrong and fix it in the original algorithm.

The main drawback related to bag files is the dimension of the records. Rosbag records topics and so a huge amount of messages that are transmitted over them; when recording a lot of topics the dimension of the final file is huge and when they are played to be studied some information are lost due the amount of data the bag should manage during the recording stage. For this reason, the topics recorded are only the necessary ones for the final validation of the algorithm and so all the topics related to the perception pipeline are discarded because they

are the heaviest ones since they transmit images, and the final bag file would be over 10 Gigabytes. This choice must be taken into account when the final considerations are carried out since the absence of the scene seen by the autonomous system is an important missing information to understand when the mapping algorithm is working well. To overcome this problem the same experiment can be performed twice and in the second case the missing topics can be recorded alone and be compared with those recorded during the first experiment to have a complete dataset on which the user can study the information recorded.

Figure 4.8: /processedImage Output Topic of a Real Experiment

In image 4.8 it is possible to see the output of the topic /processedImage of a real experiment with the RC model. The picture is similar to that of image 4.3, in fact the software used for the visualization is Rviz, which is the same used for the simulation. In this case it is possible to see exactly what the perception pipeline is perceiving through the stereo camera and how the SSD network is working. Overlapping this image to the mapping algorithm it is possible to understand which cone is detected from the perception pipeline and the correspondence in the map plotted; in this way one can compare the two information and determine whether or not the mapping algorithm is performing well its task assuming that the perception pipeline is detecting all the cones as will be discussed.

Concerning the bag files recorded for the validation of the mapping algorithm, two different recordings have been tested: one for the online computation of the global map and one for an offline procedure to reduce the weight of the bag files recorded.
In the first configuration the topic recorded are: /perception_cones and /odometry/filtered. These topics are the outputs of the perception and the localization pipelines respectively; thus, there is no need of additional offline computation since once the topics have been recorded it is possible to play the bag file and, by launching the two mapping nodes, the global map is obtained.

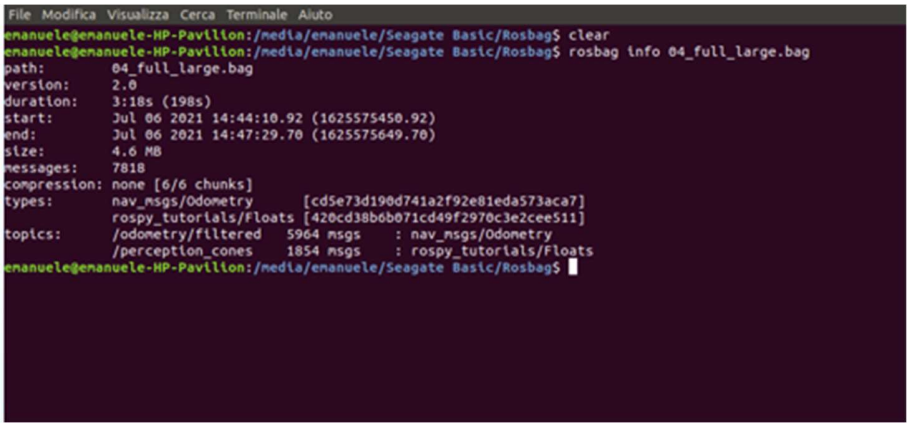


Figure 4.9: Rosbag File for the Online Computation of the Global Mapping Algorithm

For the offline computation, instead, the recorded topics are not the output of the two pipelines, but they are: /perception_cones, /imu/data, /imu/nav_sat_fix, /imu/pos_ecef, /imu/utc_ref, /imu_velocity.
As explained before the output of the perception pipeline is fundamental and must be recorded the information about the cones' detection only to avoid problem of high dimension of the bag file. In this case the second important information for the mapping pipeline, which is the localization's measurements, are recorded differently with respect to the online stage; in fact, the topics recorded are the inputs of the pipeline instead of the output. This choice has been adopted to allow the study of the two different Kalman Filters with the same dataset,

in this way the number of files recorded is half of the initial number due to the possibility of computing the localization stack offline. To validate the global mapping algorithm is then necessary to play the bag file containing the five topics recorded, at this point the user can decide which filter wants to adopt and finally the Reactive Mapping and the Global Mapping nodes can be run to obtain the global map as for the online stage.

The main difference between the two approaches is related to the ductility of the offline one to be able to adapt to different conditions; the main drawback is related to the computational effort required for recording five topics in a single bag file, since the amount of data coming from the IMU is huge and, as stated before, some messages could be lost in the recording, leading to some synchronization errors or in general in the overall mapping algorithm due to the corrupted input received.

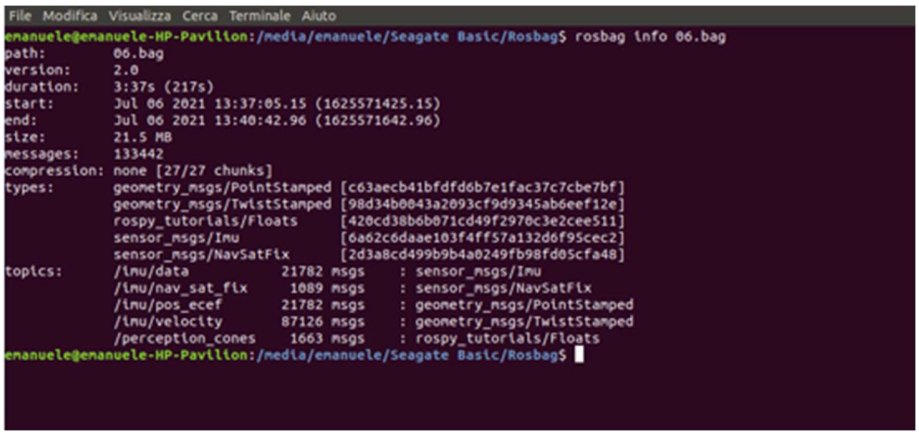


Figure 4.9: Rosbag File for the Offline Computation of the Global Mapping Algorithm

To complete the procedure a suitable launch life has been developed in order to launch with a single command all the nodes necessary to obtain the global map.

In the following picture it is shown the script developed:

Apri

RC.launch

~/CATKIN_AND_SIM_WS_2.0/src/perc_slam_launch/launch

```
<launch>
  <!-- Rigid transform robot -->
  <node pkg="tf" type="static_transform_publisher" name="per_slam_tf_1" args="0 0 0 0 0 0 1 base_link zed_center 100" />
  <node pkg="tf" type="static_transform_publisher" name="per_slam_tf_2" args="0 0 0 0 0 0 1 map odom 100" />

  <!-- Reactive Mapping -->
  <node pkg="reactive_mapping" type="reactive_mapping_bag.py" name="reactive_mapping" clear_params="true"  />

  <!-- Global Mapping -->
  <node pkg="global_mapping" type="global_mapping_RC.py" name="global_mapping" clear_params="true" output="screen" />

  <!-- RVIZ -->
  <node name="rviz_visualizer" pkg="rviz_visualizer" type="rviz_visualizer.py" output="screen" />
  <node name="rviz" pkg="rviz" type="rviz" args="-d $(find zed_display_rviz)/rviz/zed.rviz" />

</launch>
```

Figure 4.10: RC Launch File

The file runs the two mapping nodes and Rviz to plot the map on screen; moreover, in the first lines of the file there are two important command to provide the right transformation between the frames of the system.

The package static_transform_publisher allow to relate two frames with a static transformation that, in general, can be a roto-translation where the arguments passed at the end of each line corresponds to the values of translation and rotations along the axes. In this particular case the transformations created relate the frames base_link-zed_center and map-odom such that the frames are coupled and overlapped as a unique frame.

At this stage it is possible to validate the Global Mapping Algorithm in all its aspects and check if the system developed is consistent to the rules of the competition and provides a detailed and accurate map to the control subsystem.

The validation has been carried out with three different circuits similar to those considered for the simulation stage and with a shape of the tracks studied to test the system in an environment conform to those that can be found at the race.

Small Track

The first track on which the algorithm is tested is a small track in which there are straight sections and wide turning in order to validate the algorithm in different conditions that can encounter during the race.
The experimental session has been repeated twice for the testing the EKF and the UKF in the same track and find the best one comparing the performances achieved.
In general, the global map obtained represents a good base to be passed to the Control pipeline to estimate a local trajectory to be followed since the limits of the map are well separated and a central line can be plotted by the Trajectory Planner.
The main problem related to the results is related to the outer cones, in fact, most of them are not detected, especially when turning. This is due to the orientation of the stereo camera whose field of view is cut during these operations and so when driving it is necessary to increase the radius of curvature to detect the cones.
Driving the car in the opposite direction does not change the result as shown in the following map obtained, leading to a problem of field of view and not to the perception pipeline nor to the mapping one.
In straight sections, in fact, the placement of the cones is the right one and the information contained in the map are more accurate and precise.

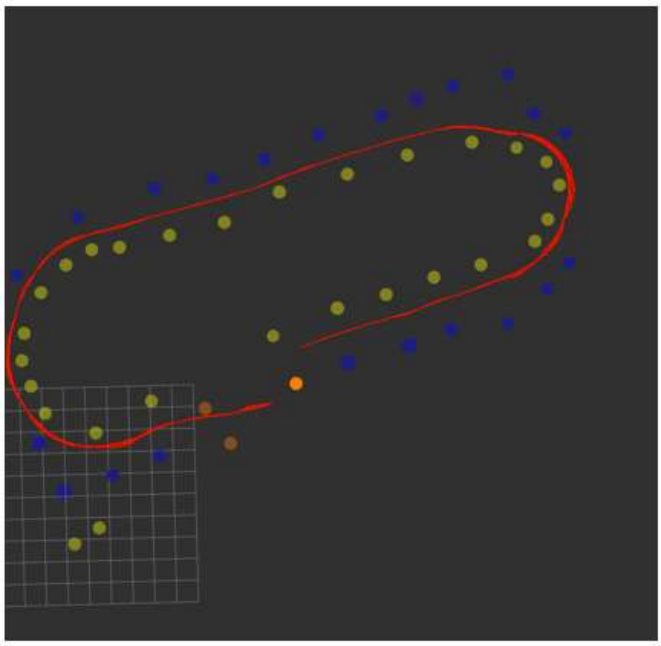


Figure 4.12: Small Track's Global Map with UKF

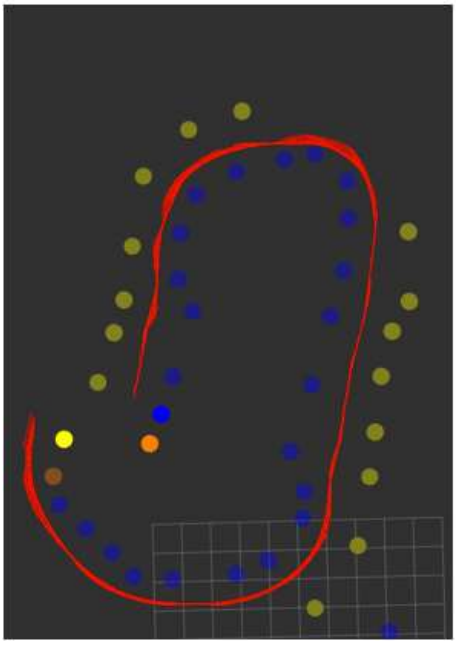


Figure 4.13: Small Track's Global Map with EKF

Considering the localization stack, it is possible to note that the loop closure is not achieved perfectly; this is due to the fact that the algorithm is validated following the rules of the competition about the dimension of the track, thus if compared with respect to the dimension of the vehicle are quite big and when driving the vehicle in the track it is almost impossible to follow a perfect ideal central trajectory and, as a consequence, the initial and the final arrival point do not coincide. However, the distance between these two points is very little even in the global map, in fact, scaling the map with the due conversion factor it is possible to estimate this distance as less than 1 meter, which is also consistent with the sensibility of the IMU sensor and, in general, an acceptable result for a safe driving of the car.

Concerning the two maps obtained, one can highlight the fact that in the second image this distance is bigger: this, in fact, is more than one meter and is due to the choice of the Kalman Filter adopted in the estimation of the pose of the vehicle. In the first case the filter adopted is the UKF, while on the second case the EKF has been chose as estimator; the explanation of such difference will be described later in a dedicated analysis.

Big Track

In the big track the algorithm is tested in a longer path with narrower turning and few straight sections in order to check the robustness of the algorithm in different conditions and hard driving conditions.

The big track represents the most important validation test since the algorithm must deal with an unknown a-priori track and consequently it should be able to provides a global map in the most different driving conditions.

The experimental session carried out is based on the UKF for the localization stack and all the rules of the race have been followed to validate a robust system. Moreover, the driving trajectory kept has been the most central one, simulating the real driving conditions and avoiding to follow the best possible one to facilitate the detection of the perception pipeline.

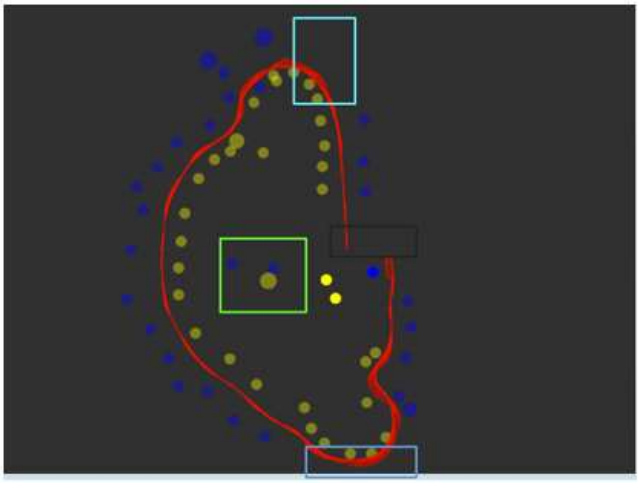


Figure 4.14: Big Track's Global Map

Figure 4.15: Big Track's Real Circuit

The map obtained is very good, the plot is precise and rich in information allowing the computation of an optimal trajectory to be followed during the first lap of the competition in each sanction of the track.

To better understand the map, three crucial areas have been selected to describe the results obtained:

- green area: in this section three cones are highlighted since their position seem to be not motivated and source of possible error; they represent effective errors if considered in the global mapping since they are the images of other cones that are not placed in the center of the track as it is possible to see in the image representing the real path build at Aeroclub, however their presence is justified by the fact that the vehicle has been brought by hands to the initial points of the track from the working station where all the algorithms and the preparatory stages are carried out, and in this passage the ZED detected some cones that are placed in the wrong position since the motion was corrupted by the walking of the transportation. The presence of such cones should then be ignored since in race-kind conditions the vehicle is not transported and the whole autonomous system can be turned on whenever is needed and no preparatory stages are necessary;

- black area: this area represents the loop closure; as stated above the discrepancy between the initial and the final position is less than 1 meter which is an acceptable result if compared to the dimension of the vehicle and of the track;

- blue area: as highlighted in the picture, the blue sections are two and correspond to the tightest curves of the track. Here is possible to see how the internal trace of cones is perfectly mapped, whereas the outer trace is almost empty even if for a small section of the track; the cause is the same stated for the small track and in particular to the cut of the field of view of the sensor when facing the turning.

In general, after testing the algorithm in the two closed tracks, it is possible to find out that the system is working well but some limits are evident. In particular the most important one is related to the height of the ZED with respect to the cones; this is placed at 30 cm and so it is possible to detect only few couples of cones in each frame.

In the SC19 the ZED is placed at 1 m from the ground; this solution allows a wider field of view leading to a more precise sensor's estimation.
The height of the camera is one of the most important parameters for allowing a good perception stage in all conditions. When facing a curve, having the camera at a high level let the algorithm to obtain a much more precise map since the height of the sensor with respect to the landmarks of the track increase the operation field of view, finding a good solution for the perception of the inner trace of cones that otherwise cannot be sensed and, consequently, be plotted on the global map as in the case of the RC model.

Acceleration Track

The last track on which the algorithm has been validated is the acceleration track, namely one of the official tracks of the dynamic events of the race.
The path to be followed is always composed by cones but in this case the position of them is known in advance and the main purpose is that of checking the dynamic performances of the vehicle rather than the perception pipeline.
To be consistent with the purpose of the track, the acceleration and the velocity registered during the driving are much larger than those kept for the small and the big track.
As explained above during the first lap of the race it is important to drive the car with low speed to allow the autonomous system to perceive all the cones and place them with high accuracy in the global map of the circuit; however, in this experimental session it has been tested the Global Mapping Algorithm also in these conditions of relative high velocity and acceleration to understand the accuracy and the repeatability of the developed algorithm.
The choice of testing the algorithm in harder conditions for the overall autonomous system is derived from the necessity to study the behavior in the worst condition and understand how good the map obtained is, setting the dynamic parameters to the limit in order to complete the first lap in the shortest time.
The straight shape of the track facilitates the job of both the perception and the mapping pipeline, but it is important also to understand in which part of the unknown track the vehicle

can accelerate relying on the fact that the global map will be accurate.
The result obtained confirm the fact that even with higher velocity the algorithm works well if the field of view is wide and straight. The absence of turning allows the algorithm to place the cones in advance with high accuracy since in all the frames considered for the placement all the cones are perfectly visible and the system can compute the right depth with accuracy, without any risk to change one cone of a straight area with one of a turning in the other direction.

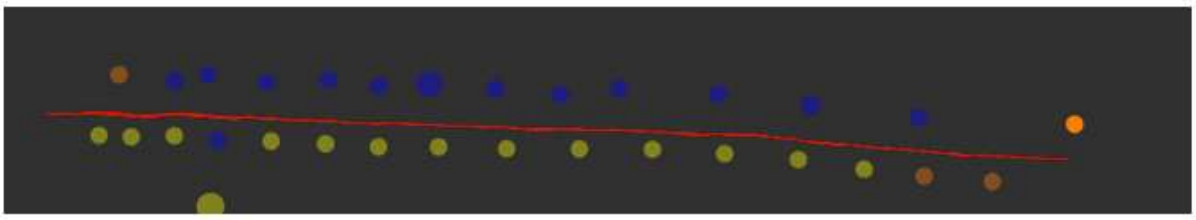


Figure 4.16: Acceleration Track's Global Map

Considering the global map obtained it is possible to highlight the accuracy of the algorithm: 31 cones over a total of 34 have been placed, and only one color has been detected wrong at the beginning of the track. The three cones missing are at the end of the path where, in particular, the orange cones that signal the end are very close and the algorithm perceived them as just one cone.

A final consideration can be done about the differences between the EKF and UKF on the final result of the mapping algorithm.
As explained in chapter 2.2.2 the two filters provide good results in term of localization, nevertheless, theoretically, it has been proved that the UKF should provide better results in term of accuracy and associated covariance.
All the testing sessions have been carried out with both the filters to understand which one is the best to adopt to the final configuration.

In the following it is possible to see a representative comparison of the two filters in two different tracks:

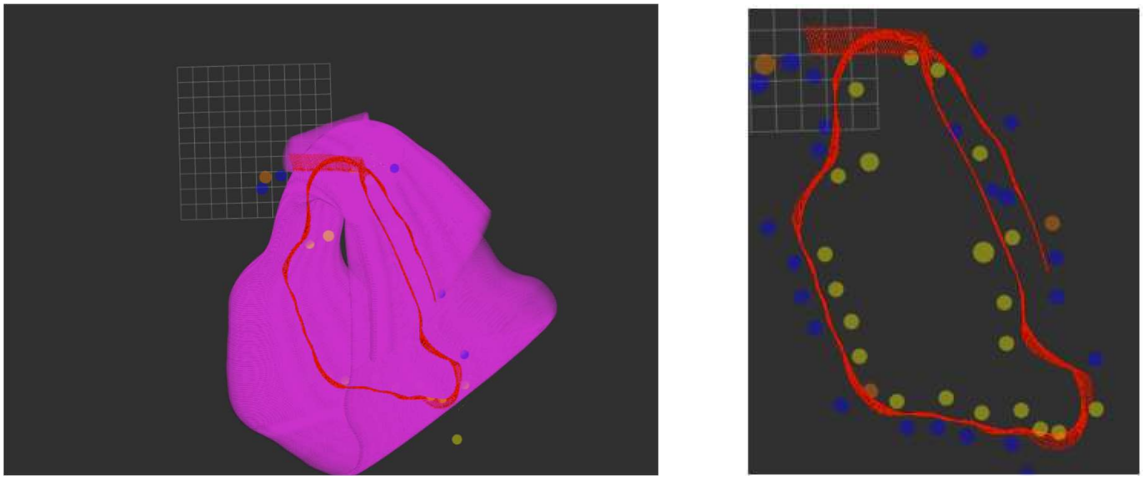


Figure 4.17: Global Map and Covariance with EKF

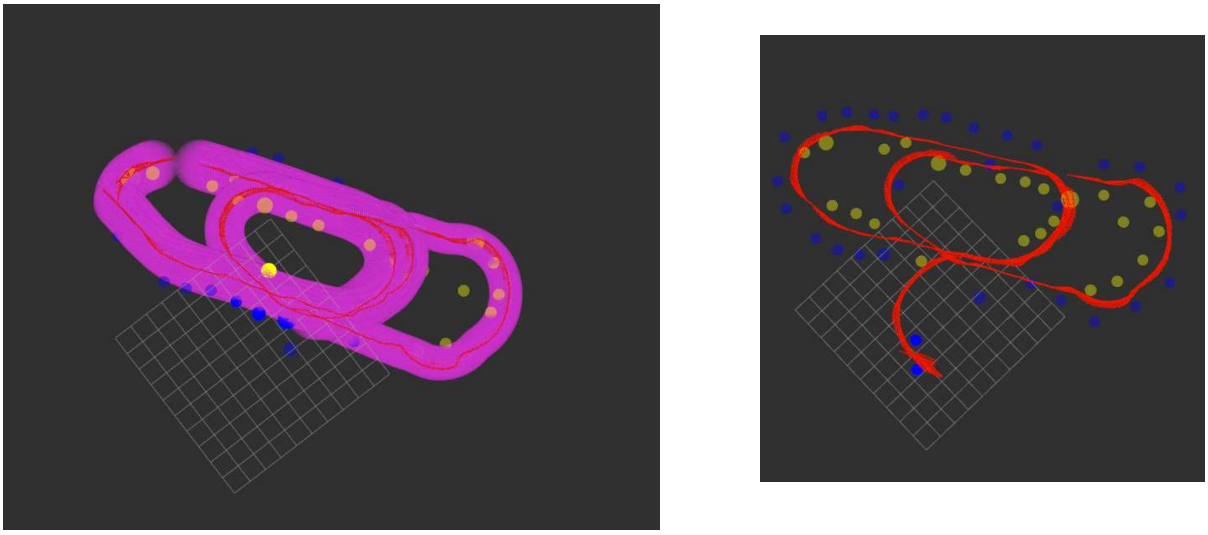


Figure 4.18: Global Map and Covariance with UKF

As shown, the UKF provides better results with respect to the EKF in the same conditions as

expected. The computational effort, which is the weak point of the UKF, is not a problem since the system is able to perform correctly in both the two configurations without problem related to the memory or to the computation effort dedicated to the Localization pipeline at the expense of the Mapping stack.

The first consideration can be done about the covariance associated to the filters; in the UKF case this is constant and small all over the track and this represents a good result since it is a sign that the pose of the vehicle has been estimated well and with high accuracy. Concerning the EKF, the covariance is a first big problem, in fact, this shows high values from the beginning, and it increases over the time. Having big values of covariance is not a good compromise since the pose of the vehicle can be any in the cloud of the covariance and this may lead to a not closure of the loop or to hit some cones during the autonomous driving. In this case the loop closure has always been reached quite well but the map obtained shows some relevant troubles.

The mapping algorithm based on the EKF, in fact, provides a final global map that is bad and poor in information: only the initial area of the track has been detected well, after that only few cones have been placed and most of them are not in the correct position; moreover, orange cones characterizing the beginning of the track are placed in different positions after the closure of the first lap. These considerations bring to the choice of the UKF as the main filter for the localization pipeline.

The choice of testing the filter in a non-standard track as the one shown in figure 4.18 is due to the fact that for all the other race-kind tracks the results obtained, in terms of localization, are very good and so testing the overall algorithm in a difficult track as the one presented is a useful experimental session for an additional validation of both the Localization and the Mapping pipelines; moreover the shape of the track is similar to that of the Skid Pad which is one of the a-priori known track of the dynamic event at the official competition.

In this case the map, as described above, is a good representation of the real track and the Trajectory Planning algorithm can develop a suitable path to be followed to complete the first loop safely and with high precision. In this case all the area of the track are well described by the cones and the position is the correct one leading to a functional global map with a low level of uncertainty.

5 – Conclusions and Future Works

SLAM represents one of the most important aspect in the development of autonomous vehicles, both in real-world applications and for racing competition. It is a bridge-process between the Perception and the Control pipelines and allow the autonomous system to compute at the same time the pose estimation of the vehicle making the system fully controllable and observable, and develop a global map of the surrounding environment for deriving the right reference signal to control the dynamic of the vehicle.

In this thesis work the development and the validation of the Global Mapping Algorithm has been carried out first in the simulation environment with Gazebo to make a preliminary study of the working system and to fix errors and parameters in order to have a first functional version of the algorithm in ideal condition. The second and fundamental step is that of validating the algorithm in the real world with the real sensors; this stage allow to obtain as a final result a working mapping algorithm consistent with the official rules of the FSD and able to provide to the autonomous system a reliable map of the a-priori unknown environment.

The overall algorithm developed for the Mapping pipeline has been tested in numerous experimental sessions to study the performance achieved in different conditions and with different software architecture solutions in order to derive the best one for the global map representation.

The global maps obtained in the different tracks tested are accurate representations of the real circuits and thus they can be passed to the Control pipeline to derive the local trajectory that the vehicle should follow to drive itself in the map during the first lap of the race, which is the one in which the autonomous system perceives the environment and develops a global map that is stored to allow the Control subsystem to drive the vehicle safely and as fast as possible for the remaining ten laps.

Future works related to the mapping pipeline are strictly linked to the perception stage too, in fact a possible solution for the improvement of the algorithm developed is that of adopt the LiDAR sensor to perceive the surrounding environment and fuse the potentialities of the stereo camera as discussed in the thesis work and those of the LiDAR to obtain a full global map whose robustness and accuracy allow to describe perfectly the landmark-based track.
Some tests have already been performed on the RC model, but the battery used to supply the system and all the sensors does not provide enough power and, as a consequence, it was possible only to test the LiDAR independently from the stereo camera; the main drawback is related to the limit of the sensor that cannot detect the color of the cones and so the Mapping algorithm could not be tested. However, the perception stage has been performed very well and the reliability of the LiDAR has been proved even on the RC model leading to a robust pipeline for the perception which can increase the overall capability of the autonomous system to provide a detailed global map.
The LiDAR, in fact, can detect cones in a very big range and with a higher accuracy than the ZED, setting a good solution for the detection of cones when the vehicle is turning, which has been detected as the biggest issue in this validation thesis work.
On the SC19 project the equipment of the Velodyne LiDAR is already taken into account, which should be positioned on the front wing of the vehicle to guarantee a wide field of view to the sensor and perceive as much cones as possible at each frame.
Fusing the two perception pipelines in a single global map could be the final step to guarantee a robust Path Planning and thus a safe conclusion of the first lap at the race competition, which is the most important goal to be reached with this thesis work.

Appendix A – Codes and Configurations

A.1 RC Model INS Parameters

```yaml
# Configuration file for SBG device through an Uart interface.

# Configuration of the device with ROS.
confWithRos: true

# Uart configuration
uartConf:
  # Port Name
  portName: "/dev/ttyUSB0"

  # Baude rate (4800 ,9600 ,19200 ,38400 ,115200 [default],230400 ,460800 ,921600)
  baudRate: 921600

  # Port Id
  # 0 PORT_A: Main communication interface. Full duplex.
  # 1 PORT_B: Auxiliary input interface for RTCM
  # 2 PORT_C: Auxiliary communication interface. Full duplex.
  # 3 PORT_D: Auxiliary input interface
  # 4 PORT_E: Auxiliary input/output interface
  portID: 0

# Sensor Parameters
sensorParameters:
  # Initial latitude (°)
  initLat: 45.086909
  # Initial longitude (°)
  initLong: 7.609471
  # Initial altitude (above WGS84 ellipsoid) (m)
  initAlt: 281.85
  # Year at startup
  year: 2020
  # month in year at startup
  month: 12
  # day in month at startup
  day: 07

  # Montion profile ID
  # 1 GENERAL_PURPOSE Should be used as a default when other profiles do not apply
  # 2 AUTOMOTIVE      Dedicated to car applications
  # 3 MARINE          Used in marine and underwater applications
  # 4 AIRPLANE        For fixed wings aircraft
  # 5 HELICOPTER      For rotary wing aircraft
  motionProfile: 2

# IMU_ALIGNMENT_LEVER_ARM
imuAlignementLeverArm:
  # IMU X axis direction in vehicle frame
  # 0 ALIGNMENT_FORWARD   IMU Axis is turned in vehicle's forward direction
  # 1 ALIGNMENT_BACKWARD  IMU Axis is turned in vehicle's backward direction
  # 2 ALIGNMENT_LEFT      IMU Axis is turned in vehicle's left direction
  # 3 ALIGNMENT_RIGHT     IMU Axis is turned in vehicle's right direction
  # 4 ALIGNMENT_UP        IMU Axis is turned in vehicle's up direction
  # 5 ALIGNMENT_DOWN      IMU Axis is turned in vehicle's down direction
  axisDirectionX: 0
  # IMU Y axis direction in vehicle frame
```

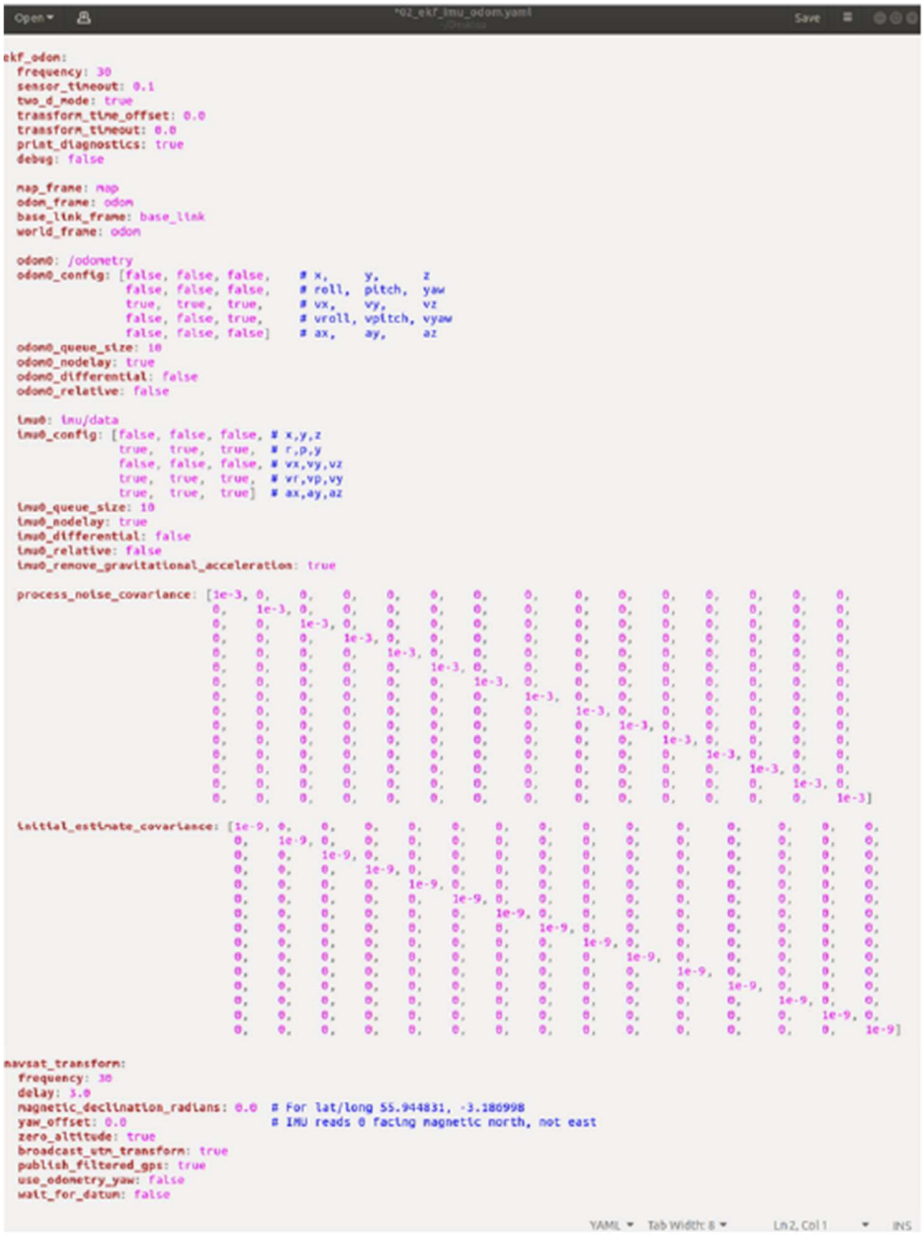
```
  axisDirectionX: 0
  # IMU Y axis direction in vehicle frame
  # 0 ALIGNMENT_FORWARD   IMU Axis is turned in vehicle's forward direction
  # 1 ALIGNMENT_BACKWARD  IMU Axis is turned in vehicle's backward direction
  # 2 ALIGNMENT_LEFT      IMU Axis is turned in vehicle's left direction
  # 3 ALIGNMENT_RIGHT     IMU Axis is turned in vehicle's right direction
  # 4 ALIGNMENT_UP        IMU Axis is turned in vehicle's up direction
  # 5 ALIGNMENT_DOWN      IMU Axis is turned in vehicle's down direction
  axisDirectionY: 3
  # Residual roll error after axis alignment rad
  misRoll: 0
  # Residual pitch error after axis alignment rad
  misPitch: 0
  # Residual yaw error after axis alignment rad
  misYaw: 0
  # X Primary lever arm in IMU X axis (once IMU alignment is applied) m
  leverArmX: 0
  # Y Primary lever arm in IMU Y axis (once IMU alignment is applied) m
  leverArmY: 0
  # Z Primary lever arm in IMU Z axis (once IMU alignment is applied) m
  leverArmZ: 0

# AIDING_ASSIGNMENT
# Note: GNSS1 module configuration can only be set to an external port on Ellipse-E version.
# Ellipse-N users must set this module to MODULE_INTERNAL. On the other hand, rtcmModule is only
# available for Ellipse-N users. This module must be set to MODULE_DISABLED for other users.
aidingAssignment:
  # GNSS module port assignment:
  # 255 Module is disabled
  # 1 Module connected on PORT_B
  # 2 Module connected on PORT_C
  # 3 Module connected on PORT_D
  # 5 Module is connected internally
  gnss1ModulePortAssignment: 5
  # GNSS module sync assignment:
  # 0 Module is disabled
  # 1 Synchronization is done using SYNC_IN_A pin
  # 2 Synchronization is done using SYNC_IN_B pin
  # 3 Synchronization is done using SYNC_IN_C pin
  # 4 Synchronization is done using SYNC_IN_D pin
  # 5 Synchronization is internal
  # 6 Synchronization is done using SYNC_OUT_A pin
  # 7 Synchronization is done using SYNC_OUT_B pin
  gnss1ModuleSyncAssignment: 5
  # RTCM input port assignment for Ellipse-N DGPS
  rtcmPortAssignment: 255
  # Odometer module pin assignment
  # 0 Odometer is disabled
  # 1 Odometer connected only to ODO_A (unidirectional).
  # 2 Odometer connected to both ODO_A (signal A) and ODO_B (Signal B or direction) for bidirectional odometer.
  odometerPinAssignment: 0

magnetometer:
  # Magnetometer model ID
  # 201 Should be used in most applications
```

A.2 SC19 INS Parameters

```yaml
# Configuration file for SBG device through an Uart interface.

# Configuration of the device with ROS.
confWithRos: true

# Uart configuration
uartConf:
  # Port Name
  portName: "/dev/ttyUSB0"

  # Baude rate (4800 ,9600 ,19200 ,38400 ,115200 [default],230400 ,460800 ,921600)
  baudRate: 921600

  # Port Id
  # 0 PORT_A: Main communication interface. Full duplex.
  # 1 PORT_B: Auxiliary input interface for RTCM
  # 2 PORT_C: Auxiliary communication interface. Full duplex.
  # 3 PORT_D: Auxiliary input interface
  # 4 PORT_E: Auxiliary input/output interface
  portID: 0

# Sensor Parameters
sensorParameters:
  # Initial latitude (°)
  initLat: 45.086909
  # Initial longitude (°)
  initLong: 7.609471
  # Initial altitude (above WGS84 ellipsoid) (m)
  initAlt: 281.85
  # Year at startup
  year: 2020
  # month in year at startup
  month: 12
  # day in month at startup
  day: 07

  # Montion profile ID
  # 1 GENERAL_PURPOSE Should be used as a default when other profiles do not apply
  # 2 AUTOMOTIVE      Dedicated to car applications
  # 3 MARINE          Used in marine and underwater applications
  # 4 AIRPLANE        For fixed wings aircraft
  # 5 HELICOPTER      For rotary wing aircraft
  motionProfile: 2

# IMU_ALIGNMENT_LEVER_ARM
imuAlignementLeverArm:
  # IMU X axis direction in vehicle frame
  # 0 ALIGNMENT_FORWARD   IMU Axis is turned in vehicle's forward direction
  # 1 ALIGNMENT_BACKWARD  IMU Axis is turned in vehicle's backward direction
  # 2 ALIGNMENT_LEFT      IMU Axis is turned in vehicle's left direction
  # 3 ALIGNMENT_RIGHT     IMU Axis is turned in vehicle's right direction
  # 4 ALIGNMENT_UP        IMU Axis is turned in vehicle's up direction
  # 5 ALIGNMENT_DOWN      IMU Axis is turned in vehicle's down direction
  axisDirectionX: 2
  # IMU Y axis direction in vehicle frame
```

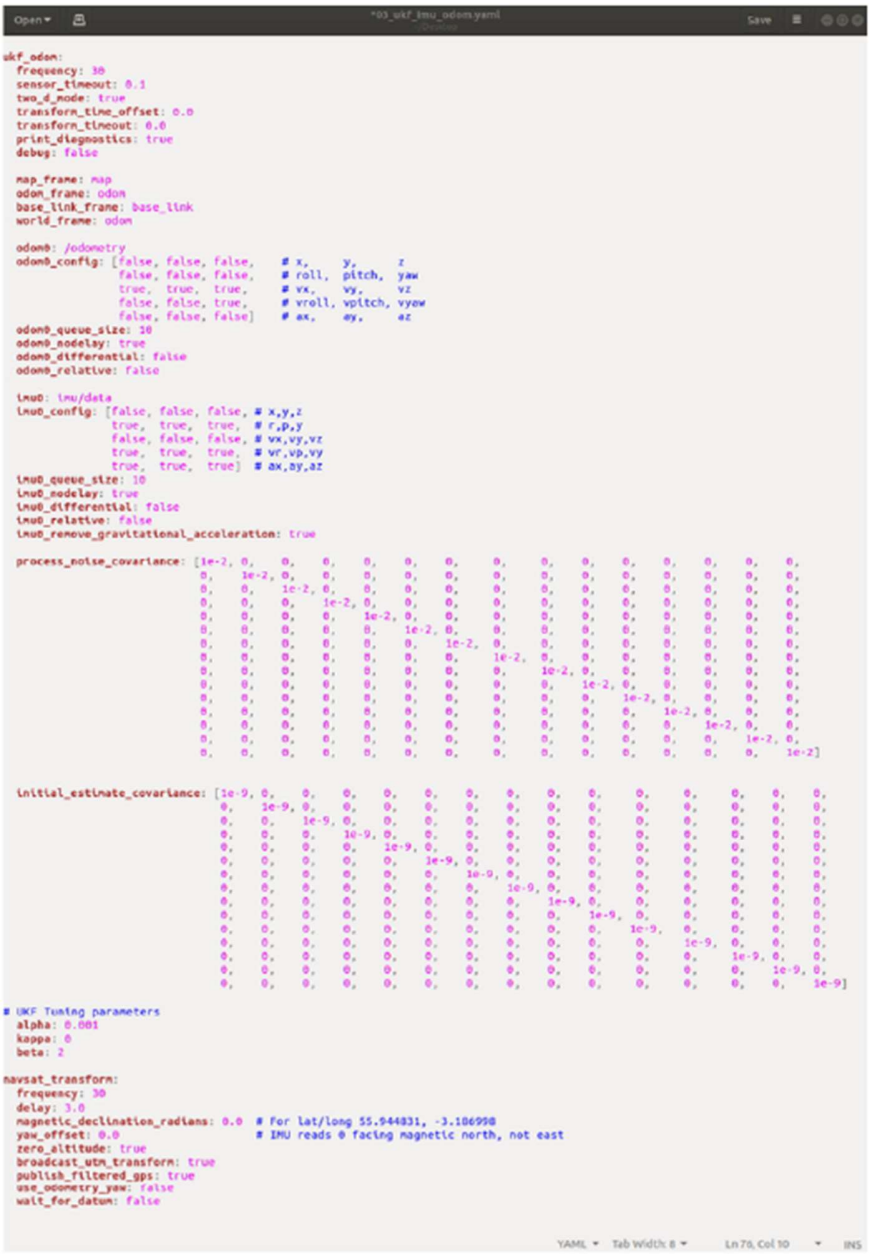
```
  axisDirectionX: 2
  # IMU Y axis direction in vehicle frame
  # 0 ALIGNMENT_FORWARD   IMU Axis is turned in vehicle's forward direction
  # 1 ALIGNMENT_BACKWARD  IMU Axis is turned in vehicle's backward direction
  # 2 ALIGNMENT_LEFT      IMU Axis is turned in vehicle's left direction
  # 3 ALIGNMENT_RIGHT     IMU Axis is turned in vehicle's right direction
  # 4 ALIGNMENT_UP        IMU Axis is turned in vehicle's up direction
  # 5 ALIGNMENT_DOWN      IMU Axis is turned in vehicle's down direction
  axisDirectionY: 0
  # Residual roll error after axis alignment rad
  misRoll: 0
  # Residual pitch error after axis alignment rad
  misPitch: 0
  # Residual yaw error after axis alignment rad
  misYaw: 0
  # X Primary lever arm in IMU X axis (once IMU alignment is applied) m
  leverArmX: -0.65
  # Y Primary lever arm in IMU Y axis (once IMU alignment is applied) m
  leverArmY: 0.06
  # Z Primary lever arm in IMU Z axis (once IMU alignment is applied) m
  leverArmZ: 0.00

# AIDING_ASSIGNMENT
# Note: GNSS1 module configuration can only be set to an external port on Ellipse-E version.
# Ellipse-N users must set this module to MODULE_INTERNAL. On the other hand, rtcmModule is only
# available for Ellipse-N users. This module must be set to MODULE_DISABLED for other users.
aidingAssignment:
  # GNSS module port assignment:
  # 255 Module is disabled
  # 1 Module connected on PORT_B
  # 2 Module connected on PORT_C
  # 3 Module connected on PORT_D
  # 5 Module is connected internally
  gnss1ModulePortAssignment: 5
  # GNSS module sync assignment:
  # 0 Module is disabled
  # 1 Synchronization is done using SYNC_IN_A pin
  # 2 Synchronization is done using SYNC_IN_B pin
  # 3 Synchronization is done using SYNC_IN_C pin
  # 4 Synchronization is done using SYNC_IN_D pin
  # 5 Synchronization is internal
  # 6 Synchronization is done using SYNC_OUT_A pin
  # 7 Synchronization is done using SYNC_OUT_B pin
  gnss1ModuleSyncAssignment: 5
  # RTCM input port assignment for Ellipse-N DGPS
  rtcmPortAssignment: 255
  # Odometer module pin assignment
  # 0 Odometer is disabled
  # 1 Odometer connected only to ODO_A (unidirectional).
  # 2 Odometer connected to both ODO_A (signal A) and ODO_B (Signal B or direction) for bidirectional odometer.
  odometerPinAssignment: 0

magnetometer:
  # Magnetometer model ID
  # 201 Should be used in most applications
```

A.3 EKF Parameters Configuration



```
ekf_odom:
  frequency: 30
  sensor_timeout: 0.1
  two_d_mode: true
  transform_time_offset: 0.0
  transform_timeout: 0.0
  print_diagnostics: true
  debug: false

  map_frame: map
  odom_frame: odom
  base_link_frame: base_link
  world_frame: odom

  odom0: /odometry
  odom0_config: [false, false, false,    # x,     y,     z
                 false, false, false,    # roll,  pitch, yaw
                 true,  true,  true,     # vx,    vy,    vz
                 false, false, true,     # vroll, vpitch, vyaw
                 false, false, false]    # ax,    ay,    az
  odom0_queue_size: 10
  odom0_nodelay: true
  odom0_differential: false
  odom0_relative: false

  imu0: imu/data
  imu0_config: [false, false, false, # x,y,z
                true,  true,  true,  # r,p,y
                false, false, false, # vx,vy,vz
                true,  true,  true,  # vr,vp,vy
                true,  true,  true]  # ax,ay,az
  imu0_queue_size: 10
  imu0_nodelay: true
  imu0_differential: false
  imu0_relative: false
  imu0_remove_gravitational_acceleration: true

  process_noise_covariance: [1e-3, 0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
                             0,    1e-3, 0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
                             0,    0,    1e-3, 0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
                             0,    0,    0,    1e-3, 0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
                             0,    0,    0,    0,    1e-3, 0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
                             0,    0,    0,    0,    0,    1e-3, 0,    0,    0,    0,    0,    0,    0,    0,    0,
                             0,    0,    0,    0,    0,    0,    1e-3, 0,    0,    0,    0,    0,    0,    0,    0,
                             0,    0,    0,    0,    0,    0,    0,    1e-3, 0,    0,    0,    0,    0,    0,    0,
                             0,    0,    0,    0,    0,    0,    0,    0,    1e-3, 0,    0,    0,    0,    0,    0,
                             0,    0,    0,    0,    0,    0,    0,    0,    0,    1e-3, 0,    0,    0,    0,    0,
                             0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    1e-3, 0,    0,    0,    0,
                             0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    1e-3, 0,    0,    0,
                             0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    1e-3, 0,    0,
                             0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    1e-3, 0,
                             0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    1e-3]

  initial_estimate_covariance: [1e-9, 0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
                                0,    1e-9, 0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
                                0,    0,    1e-9, 0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
                                0,    0,    0,    1e-9, 0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
                                0,    0,    0,    0,    1e-9, 0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
                                0,    0,    0,    0,    0,    1e-9, 0,    0,    0,    0,    0,    0,    0,    0,    0,
                                0,    0,    0,    0,    0,    0,    1e-9, 0,    0,    0,    0,    0,    0,    0,    0,
                                0,    0,    0,    0,    0,    0,    0,    1e-9, 0,    0,    0,    0,    0,    0,    0,
                                0,    0,    0,    0,    0,    0,    0,    0,    1e-9, 0,    0,    0,    0,    0,    0,
                                0,    0,    0,    0,    0,    0,    0,    0,    0,    1e-9, 0,    0,    0,    0,    0,
                                0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    1e-9, 0,    0,    0,    0,
                                0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    1e-9, 0,    0,    0,
                                0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    1e-9, 0,    0,
                                0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    1e-9, 0,
                                0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    1e-9]

navsat_transform:
  frequency: 30
  delay: 3.0
  magnetic_declination_radians: 0.0  # For lat/long 55.944831, -3.186998
  yaw_offset: 0.0                    # IMU reads 0 facing magnetic north, not east
  zero_altitude: true
  broadcast_utm_transform: true
  publish_filtered_gps: true
  use_odometry_yaw: false
  wait_for_datum: false
```

A.4 UKF Parameters Configuration



```
Open ▾   *03_ukf_imu_odom.yaml                                   Save

ukf_odom:
  frequency: 30
  sensor_timeout: 0.1
  two_d_mode: true
  transform_time_offset: 0.0
  transform_timeout: 0.0
  print_diagnostics: true
  debug: false

  map_frame: map
  odom_frame: odom
  base_link_frame: base_link
  world_frame: odom

  odom0: /odometry
  odom0_config: [false, false, false,    # x,     y,     z
                 false, false, false,    # roll,  pitch, yaw
                 true,  true,  true,     # vx,    vy,    vz
                 false, false, true,     # vroll, vpitch, vyaw
                 false, false, false]    # ax,    ay,    az
  odom0_queue_size: 10
  odom0_nodelay: true
  odom0_differential: false
  odom0_relative: false

  imu0: imu/data
  imu0_config: [false, false, false, # x,y,z
                true,  true,  true,  # r,p,y
                false, false, false, # vx,vy,vz
                true,  true,  true,  # vr,vp,vy
                true,  true,  true]  # ax,ay,az
  imu0_queue_size: 10
  imu0_nodelay: true
  imu0_differential: false
  imu0_relative: false
  imu0_remove_gravitational_acceleration: true

  process_noise_covariance: [1e-2, 0,    0,    0,    0,    0,    0,     0,     0,    0,    0,    0,    0,    0,    0,
                             0,    1e-2, 0,    0,    0,    0,    0,     0,     0,    0,    0,    0,    0,    0,    0,
                             0,    0,    1e-2, 0,    0,    0,    0,     0,     0,    0,    0,    0,    0,    0,    0,
                             0,    0,    0,    1e-2, 0,    0,    0,     0,     0,    0,    0,    0,    0,    0,    0,
                             0,    0,    0,    0,    1e-2, 0,    0,     0,     0,    0,    0,    0,    0,    0,    0,
                             0,    0,    0,    0,    0,    1e-2, 0,     0,     0,    0,    0,    0,    0,    0,    0,
                             0,    0,    0,    0,    0,    0,    1e-2,  0,     0,    0,    0,    0,    0,    0,    0,
                             0,    0,    0,    0,    0,    0,    0,     1e-2,  0,    0,    0,    0,    0,    0,    0,
                             0,    0,    0,    0,    0,    0,    0,     0,     1e-2, 0,    0,    0,    0,    0,    0,
                             0,    0,    0,    0,    0,    0,    0,     0,     0,    1e-2, 0,    0,    0,    0,    0,
                             0,    0,    0,    0,    0,    0,    0,     0,     0,    0,    1e-2, 0,    0,    0,    0,
                             0,    0,    0,    0,    0,    0,    0,     0,     0,    0,    0,    1e-2, 0,    0,    0,
                             0,    0,    0,    0,    0,    0,    0,     0,     0,    0,    0,    0,    1e-2, 0,    0,
                             0,    0,    0,    0,    0,    0,    0,     0,     0,    0,    0,    0,    0,    1e-2, 0,
                             0,    0,    0,    0,    0,    0,    0,     0,     0,    0,    0,    0,    0,    0,    1e-2]


  initial_estimate_covariance: [1e-9, 0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
                                0,    1e-9, 0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
                                0,    0,    1e-9, 0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
                                0,    0,    0,    1e-9, 0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
                                0,    0,    0,    0,    1e-9, 0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
                                0,    0,    0,    0,    0,    1e-9, 0,    0,    0,    0,    0,    0,    0,    0,    0,
                                0,    0,    0,    0,    0,    0,    1e-9, 0,    0,    0,    0,    0,    0,    0,    0,
                                0,    0,    0,    0,    0,    0,    0,    1e-9, 0,    0,    0,    0,    0,    0,    0,
                                0,    0,    0,    0,    0,    0,    0,    0,    1e-9, 0,    0,    0,    0,    0,    0,
                                0,    0,    0,    0,    0,    0,    0,    0,    0,    1e-9, 0,    0,    0,    0,    0,
                                0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    1e-9, 0,    0,    0,    0,
                                0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    1e-9, 0,    0,    0,
                                0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    1e-9, 0,    0,
                                0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    1e-9, 0,
                                0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    1e-9]

# UKF Tuning parameters
  alpha: 0.001
  kappa: 0
  beta: 2

navsat_transform:
  frequency: 30
  delay: 3.0
  magnetic_declination_radians: 0.0  # For lat/long 55.944831, -3.186998
  yaw_offset: 0.0                    # IMU reads 0 facing magnetic north, not east
  zero_altitude: true
  broadcast_utm_transform: true
  publish_filtered_gps: true
  use_odometry_yaw: false
  wait_for_datum: false

YAML ▾   Tab Width: 8 ▾   Ln 79, Col 10   ▾   INS
```

A.5 Reactive Mapping Node

```python
#!/usr/bin/env python
import rospy
import numpy as np
from std_msgs.msg import Float64MultiArray
from geometry_msgs.msg import Pose
from nav_msgs.msg import Odometry
from rospy.numpy_msg import numpy_msg
from rospy_tutorials.msg import Floats
import copy

class Reactive_Mapping_Node():

    def __init__(self):
        # Global variables
        self.acc_frames = 4
        self.perc_cluster = 0.75
        self.rad_cluster = 0.5

        self.bias_left_lense = 0.06

        self.delta = 99/1e4

        # -----

        self.integrated = np.array([])
        self.int_counter = 0

        # -----

        self.main()

    def callback_perception(self, msg):
        last_frame_left = msg.data.reshape(int(msg.data.shape[0]/4),4)  #reshape the image read into a vector with (x,y,z,color)
        last_frame = copy.copy(last_frame_left)
        last_frame[:,1] = last_frame[:,1] + float(self.bias_left_lense)
        print(last_frame)

        # add frame till have 3 frames to compute
        if self.integrated.shape[0] == 0:
            self.integrated = last_frame
        else:
            self.integrated = np.vstack((self.integrated, last_frame))

        self.int_counter = self.int_counter + 1

        if self.int_counter == self.acc_frames:
            centroids = self.cluster_frames()  #when reach 3 frames call the cluster_frames function to comput the centroid
            self.publish_reactive_map(centroids)
            self.int_counter = 0
            self.integrated = np.array([])

    def cluster_frames(self):

        points = self.integrated[:,0:2]  #contains the coordinates of the cones detected

        dist_matrix = np.around(np.linalg.norm(points - points[:,None], axis = -1) , decimals=3) #contains mutual distances

        cluster_id = np.zeros((points.shape[0],1), dtype=int)

        for i in range(points.shape[0]):
            neighbours = np.argwhere(dist_matrix[i,:] < self.rad_cluster)
            #if the distance between two cones in dist_matrix is < rad_cluster it means that the cone decected is the same in the different frames
            #assignation of an ID to each cone (if seen more time assign the same ID)
            if cluster_id[i] == 0:
                cluster_id[i] = np.amax(cluster_id)+1

                for n in neighbours:
                    if cluster_id[n] == 0:
                        cluster_id[n] = cluster_id[i]

            else:
```

```
                for n in neighbours:
                    if cluster_id[n] == 0:
                        cluster_id[n] = cluster_id[i]

        unique, counts = np.unique(cluster_id, return_counts=True)
        cluster_dict = dict(zip(unique,counts))

        cluster_centroid = []

        #for each ID count the number of views, if >=3 then comput the centroid
        for k in cluster_dict.keys():
            n_points = cluster_dict[k]
            if n_points >= int(round(self.acc_frames * self.perc_cluster)):
                indexes = np.argwhere(cluster_id==k)[:,0]
                p_clusters = points[indexes,:]
                cent = np.mean(p_clusters, axis=0)
                covar = np.var(p_clusters, axis=0)
                covar = np.sqrt(covar[0]**2+covar[1]**2)
                color_points = self.integrated[indexes,3]
                uniq, count = np.unique(color_points, return_counts=True)
                color = uniq[np.argmax(count)]
                cent = np.hstack((cent, covar, color))
                cluster_centroid.append(cent)

        return cluster_centroid

    def publish_reactive_map(self, reactive):
        reactive = np.array(reactive, dtype=np.float32)
        cone_publisher = rospy.Publisher('/reactive_cones', numpy_msg(Floats), queue_size=10)
        cone_publisher.publish(reactive.flatten())
        for r in reactive:
            print(np.around(r[0],2), np.around(r[1],2), np.around(r[2],4), int(r[3]))
        print("----------")


    def main(self):
        rospy.init_node('reactive_mapping_node')
        rospy.Subscriber('/perception_cones', numpy_msg(Floats), self.callback_perception)
        #rospy.Subscriber('/perc', numpy_msg(Floats), self.callback_perception)
        rospy.spin()

Reactive_Mapping_Node()

~/CATKIN_AND_SIM_WS_2.0/src/reactive_mapping/scripts/reactive_mapping.py*  67:37
```

A.6 Global Mapping Node

```python
#!/usr/bin/env python
import rospy
import numpy as np
from std_msgs.msg import Float64MultiArray
from geometry_msgs.msg import Pose
from nav_msgs.msg import Odometry
from rospy.numpy_msg import numpy_msg
from rospy_tutorials.msg import Floats
from geometry_msgs.msg import PointStamped
import tf
import time
import copy

class Global_Mapping_Node():

    def __init__(self):
        # Global variables
        self.alpha = 0.9     #for the EMA (Exponential Moving Average), closer to 1 = rely more on preavious measures
        self.max_hits = 100
        self.min_hits = -50
        self.R_max_cov = 1.2   #max cov radius
        self.R_min_cov = 0.8
        self.freq_output = 10
        self.delta_cov = self.R_max_cov * (self.max_hits - 1) / self.max_hits**2
        # -----

        self.cone_db = np.array([])
        #self.listener = tf.TransformListener()
        self.dist_FOV_max = 4.0
        self.dist_FOV_min = 1.0
        self.last_odom = Odometry()
        self.bias_left_lense = -0.06  #distance between ZED lenses
        self.freq_update_db = 60
        self.add_hit = 5
        self.sub_hit = 2
        self.angle_FOV = 65  #max=69

        self.transformed_cones = []

        self.legit_cone_hits  = 20
        self.freq_clean_db = 1

        # -----

        self.id_cone = 0
```

```
        self.main()

    def callback_odom(self, msg):
        self.last_odom = msg
        self.T = np.array((msg.pose.pose.position.x , msg.pose.pose.position.y)).reshape(1,2) #vehicle pose (x,y)
        (r, p, y) = tf.transformations.euler_from_quaternion([msg.pose.pose.orientation.x, msg.pose.pose.orientation.y, msg.pose.pose.orientation.z, msg.pose.pose.orientation.w])
        self.theta = y    #yaw

        self.R_plus = np.array([[np.cos(self.theta), -np.sin(self.theta)],[np.sin(self.theta), np.cos(self.theta)]]).reshape(2,2)      #update the max covariance
        self.R_minus = np.array([[np.cos(-self.theta), -np.sin(-self.theta)],[np.sin(-self.theta), np.cos(-self.theta)]]).reshape(2,2)   #update the min covariance

        self.pose = np.array([msg.pose.pose.position.x, msg.pose.pose.position.y, y], dtype=np.float32)  #(x,y,teta)

    def callback_reactive(self, msg):
        last_reactive = msg.data.reshape(int(msg.data.shape[0]/4),4)  #reshape the message in a more suitable form
        last_reactive = np.delete(last_reactive, 2, axis=1)   #delete the z component (useless)
        self.reactive_cones = copy.deepcopy(last_reactive) #here there is (x,y,color)

    def publish_2_icp(self, react, db, pose):
        react = copy.deepcopy(react)
        db = copy.deepcopy(db)
        pose = copy.deepcopy(pose)

        if db.shape[0] != 1:
            db = db.squeeze()
        elif  db.shape[0] == 1:
            db = db.squeeze()
            db = db.reshape(1,2)

        if db.shape[0] != react.shape[0]:
            return

        data = np.zeros((1+db.shape[0]+react.shape[0]))


    def update_cone_db(self, event):

        # Get reactive cones only in FOV
        try:
            reactive_cones = self.reactive_cones
            R_plus = self.R_plus
            R_minus = self.R_minus
            T = self.T
        except:
            print("No ready yet")
            return

        reactive_in_fov_index = []

        # Check cones on reactive which are also in the FOV (if so add them to reactive_in_fov_index)
        for i in range(reactive_cones.shape[0]):
            c = reactive_cones[i,:]

            angle = (np.arctan2(c[1],c[0]) * 180 / np.pi)
            dist = np.sqrt(c[1]**2 + c[0]**2)

            if dist <= self.dist_FOV_max and angle <= self.angle_FOV and angle >= -self.angle_FOV:
                reactive_in_fov_index.append(i)

        # [x,y,color]
        reactive_in_fov_local = copy.deepcopy(reactive_cones[reactive_in_fov_index,:])

        # [x,y,color,cov,hits,inFOV, id]
        reactive_in_fov_local = np.hstack((reactive_in_fov_local, np.zeros((reactive_in_fov_local.shape[0], 4))))
        # Assign max covariance to cones
        reactive_in_fov_local[:,3] = self.R_max_cov

        reactive_in_fov_map = copy.deepcopy(reactive_in_fov_local)

        #multiply the first 2 elements of each row and add T
        if reactive_in_fov_map.shape[0] > 1:
            reactive_in_fov_map[:,0:2] = np.matmul(reactive_in_fov_map[:,0:2] , R_minus)
            reactive_in_fov_map[:,0:2] = reactive_in_fov_map[:,0:2] + T
        elif reactive_in_fov_map.shape[0] == 1:
            reactive_in_fov_map.reshape(1,7)
            reactive_in_fov_map[0, 0:2] = np.matmul(reactive_in_fov_map[0, 0:2] , R_minus)
            reactive_in_fov_map[0, 0:2] = reactive_in_fov_map[0, 0:2] + T
```

```
124
125
126         # If db empty, add all reactive cones
127         if self.cone_db.shape[0] == 0:
128             print("Empty db, adding new cones")
129             self.cone_db = copy.deepcopy(reactive_in_fov_map)
130             self.output_4_rviz(self.cone_db.astype(np.float32)) #publish on Rviz
131             return
132         else:
133             print("Cones in db: "+str(self.cone_db.shape[0]))
134
135         # If cones in db, add hits and ignore those too close to a seen cone
136         reactive_in_db = []
137         db_in_reactive = []
138         for i in range(reactive_in_fov_map.shape[0]):
139             reactive = reactive_in_fov_map[i]
140             dist_db = np.linalg.norm(self.cone_db[:,0:2]-reactive[0:2] ,axis=1)  #compute the distance between each cone in reactive_in_fov_map
141             if np.amin(dist_db) <= self.R_max_cov:
142                 reactive_in_db.append(i)  #if the distance is less than R_max_cov add the cone to reactive_in_db
143                 db_in_reactive.append(np.argmin(dist_db))
144         reactive_in_fov_map_new = np.delete(reactive_in_fov_map, reactive_in_db, axis=0)
145
146         # Data for ICP
147         dataICP = copy.deepcopy(self.pose).reshape(1,3)
148         for i in range(len(reactive_in_db)):
149             i_react = reactive_in_db[i]
150             i_db = db_in_reactive[i]
151             react = np.array([reactive_in_fov_map[i_react,0], reactive_in_fov_map[i_react,1],0]).reshape(1,3)
152             db = np.array([self.cone_db[i_db,0],self.cone_db[i_db,1],0]).reshape(1,3)
153
154             dataICP = np.vstack((dataICP, db, react))
155
156         # Send to ICP
157         if len(reactive_in_db) > 3:
158             icp_publisher = rospy.Publisher('/icp_input', numpy_msg(Floats), queue_size=10)
159             icp_publisher.publish(dataICP.flatten())
160
161
162
163
164         # Add hits to db cones seen in the reactive map
165         self.cone_db[db_in_reactive,4] = self.cone_db[db_in_reactive,4] + self.add_hit
166         self.cone_db[np.argwhere(self.cone_db[:,4] > self.max_hits),4] = self.max_hits
167         self.cone_db[np.argwhere(self.cone_db[:,4] < self.min_hits),4] = self.min_hits
168         # Update position with moving average
169         self.cone_db[db_in_reactive,0:2] = (1-self.alpha) * reactive_in_fov_map[reactive_in_db,0:2] + (self.alpha) * self.cone_db[db_in_reactive,0:2]
170         # Update covariance
171         self.cone_db[:,3] = self.R_max_cov - self.delta_cov * self.cone_db[:,4]
172         self.cone_db[np.argwhere(self.cone_db[:,3] > self.R_max_cov),3] = self.R_max_cov
173         self.cone_db[np.argwhere(self.cone_db[:,3] < self.R_min_cov),3] = self.R_min_cov
174
175         # Add the new cones to the db
176         self.cone_db = np.vstack((self.cone_db, reactive_in_fov_map_new))
177
```

```
        # Give id to cones with id == 0
        no_id_index = np.argwhere(self.cone_db[:,6] == 0)
        for id_cone in no_id_index:
            self.cone_db[id_cone,6] = self.id_cone
            self.id_cone = self.id_cone + 1


        # Calculate cones in db and FOV
        db_local = copy.deepcopy(self.cone_db)

        if db_local.shape[0] > 1:
            db_local[:,0:2] = db_local[:,0:2] - T
            db_local[:,0:2] = np.matmul(db_local[:,0:2] , R_plus)
        elif db_local.shape[0] == 1:
            db_local.reshape(1,7)
            db_local[0, 0:2] = db_local[0, 0:2] - T
            db_local[0, 0:2] = np.matmul(db_local[0, 0:2] , R_plus)

        db_local_distance = np.linalg.norm(db_local[:,0:2], axis=1)
        db_local_in_distance_index = np.argwhere(np.logical_and(db_local_distance <= self.dist_FOV_max, db_local_distance >= self.dist_FOV_min))

        db_local_in_angle_index = []

        for index_cone in db_local_in_distance_index:
            x = db_local[index_cone,0]
            y = db_local[index_cone,1]
            angle = np.arctan2(y,x) * 180 / np.pi

            if angle <= self.angle_FOV and angle >= -self.angle_FOV:
               db_local_in_angle_index.append(index_cone)

        self.cone_db[db_local_in_angle_index,5] = 1

        # if the cone isn't in db_in_reactive: hits down till deleta if < min_hits
        for index in db_local_in_angle_index:
            if index not in db_in_reactive[:]:
                self.cone_db[index,4] = self.cone_db[index,4] - self.sub_hit

        self.cone_db = np.delete(self.cone_db, np.argwhere(self.cone_db[:,4] <= self.min_hits), axis=0)

        self.output_4_rviz(self.cone_db.astype(np.float32))


        self.cone_db[:,5] = 0

    def output_4_rviz(self, rviz_cones):
        pub_rviz = rospy.Publisher('/global_map_markers', numpy_msg(Floats), queue_size=10)
        pub_rviz.publish(rviz_cones.flatten())


        #rospy.init_node('mapping_node')
    def main(self):
        rospy.init_node('mapping_node')
        rospy.Subscriber('/reactive_cones', numpy_msg(Floats), self.callback_reactive)
        #rospy.Subscriber('/ground_truth/state_raw', Odometry, self.callback_odom)
        rospy.Subscriber('/odometry/filtered', Odometry, self.callback_odom)
        #rospy.Subscriber('/robot_control/odom', Odometry, self.callback_odom)
        # rospy.Timer(rospy.Duration(1.0/self.freq_output), self.output_4_nav)
        rospy.Timer(rospy.Duration(1.0/self.freq_update_db), self.update_cone_db)
        # rospy.Timer(rospy.Duration(1.0/self.freq_clean_db), self.clean_cone_db)
        rospy.spin()

Global_Mapping_Node()

~/CATKIN_AND_SIM_WS_2.0/src/global_mapping/scripts/global_mapping.py*  21:47
```

A.7 Eufs_joy Node

```python
#!/usr/bin/env python

import math
from math import sin, cos, pi
import numpy as np
import rospy
import tf
from nav_msgs.msg import Odometry
from ackermann_msgs.msg import AckermannDriveStamped
from sensor_msgs.msg import Joy
from geometry_msgs.msg import Point, Pose, Quaternion, Twist, Vector3
from scipy.interpolate import interp1d

class Agent(object):
    def __init__(self):
        self.drive_pub = rospy.Publisher('/robot_control/command', AckermannDriveStamped, queue_size=50)
        self.joy_sub = rospy.Subscriber('/joy', Joy, self.joy_callback, queue_size=1)
        # intial pose
        self.x = 0
        self.y = 0
        self.th = 0
        self.radius = 20
        # speed set
        self.vx = self.vy = 0
        # tangential speed
        self.velocity = 5
        # angular speed
        self.vth = self.velocity/self.radius

        self.current_time = rospy.Time.now()
        self.last_time = rospy.Time.now()

        self.r = rospy.Rate(1.0)


    def joy_callback(self, joy_msg):
        mj = interp1d([-1,1],[2,0])
        ms = interp1d([-1,1],[-0.7,0.7])
        print(mj(joy_msg.axes[5]), ms(joy_msg.axes[0]))
        # set the Ackermann command
        drive = AckermannDriveStamped()
        drive.drive.speed = mj(joy_msg.axes[5])
        drive.drive.steering_angle = ms(joy_msg.axes[0])
        # publish driving message
        self.drive_pub.publish(drive)



if __name__ == '__main__':
    rospy.init_node('odom_publisher')
    dummy_agent = Agent()
    rospy.spin()
```

~/CATKIN_AND_SIM_WS_2.0/src/joystick/script/eufs_joy.py 1:1

List of Figures

Bibliography

[1] P. S. P. a. N. Madarász, «Self-driving cars – the human side Working paper».

[2] L. Isaac, «Driving towards driverless: a guide for government agencies».

[3] A. Ziebinski, R. Cupek, D. Grzechca, and L. Chruszczyk, «Review of advanced driver assistance systems (ADAS) », In AIP Conference Proceedings, 2017.

[4] A. Bonfitto, S. Feraco, A. Tonoli, N. Amati, «Combined regression and classification artificial neural networks for sideslip angle estimation and road condition identification», Vehicle System Dynamics, 2020.

[5] K. Kaur, G. Rampersad, «Trust in driverless cars: Investigating key factors influencing the adoption of driverless cars», Journal of Engineering and Technology Management, 2018.

[6] On-Road Automated Driving (ORAD) committee, «Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles J3016_201806», SAE International, 2018.

[7] Z. Wei, G. Xuexun, «An ABS control strategy for commercial vehicle», IEEE/ASME Transactions on Mechatronics, 2014.

[8] W. Hulshof, I. Knight, A. Edwards, M. Avery, C. Grover, «Autonomous emergency braking test results», Proceedings of the 23rd International Technical Conference on the Enhanced Safety of Vehicles (ESV), National Highway Traffic Safety Administration Washington, DC., 2013.

[9] G. Marsden, M. McDonald, M. Brackstone, «Towards an understanding of adaptive cruise control», Transportation Research Part C: Emerging Technologies, 2001.

[10] C. R. Jung, C. R. Kelber, «A lane departure warning system based on a linear-parabolic lane model», IEEE Intelligent Vehicles Symposium, IEEE, 2004.

[11] D. Kritzinger, «Functional Hazard Analysis», in Aircraft System Safety, 2017.

[12] Formula Student Germany. Formula Student Rules 2021, 2021.

[13] Mo. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Ng. «Ros: an open-source robot operating system», 2009.

[14] «ROS.org», [Online]. Available: ROS/Concepts - ROS Wiki.

[15] Y. S. Park, S. Lek, «Artificial Neural Network, Model Types», in Developments in Environmental Modelling, 2016.

[16] W. L. e. al., «SSD: Single Shot MultiBox Detector».

[17] M. Himmelsbach, F. v. Hundelshausen, H. Wuensche, «Fast segmentation of 3d point clouds for ground vehicles», IEEE Intelligent Vehicles Symposium, 2010.

[18] «ROS.org, The robot localization package for ros», [Online]. Available: robot_localization - ROS Wiki, 2020.

[19] «ROS.org, Documentation», https://www.ros.org/reps/rep-0105.html.

[20] «Wikipedia (2021), Kalman Filter», [Online]. Available: https://en.wikipedia.org/wiki/Kalman_filter.

[21] G. Welch, G. Bishop, «An Introduction to the Kalman Filter», Technical report, 1995.

[22] S. J. Julier, J. K. Uhlmann, «Unscented filtering and nonlinear estimation», Proceedings of the IEEE, 2004.

[23] H. a. E. A. Moravec, «High resolution maps from wide angle sonar», in Proc. of the IEEE International Conference Robotics and Automation, St. Louis, 1985.

[24] L. Matthies, A. Elfes, «Integration of sonar and stereo range data using a grid-based representation», In International Conference on Robotics and Automation, 1988.

[25] J. Borenstein, B. Everett, L. Feng, «Navigating Mobile Robots: Systems and Techniques», A. K. Peters, Ltd., Wellesley, MA, 1996.

[26] D. Fox, W. Burgard, S. Thrun, «Markov localization for mobile robots in dynamic environments», Journal of Artificial Intelligence Research, 1999.

[27] R. a. S. P. Cheeseman, «On the representation and estimation of spatial uncertainty», International Journal of Robotics, 1986.

[28] G. J. a. N. E., «Solving computational and memory requirements of feature-based simultaneous localization and mapping algorithms», IEEE Transactions on Robotics and Automation, 2003.

[29] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, J. J.

Leonard, «Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age», IEEE Transactions on Robotics, 2016.

[30] J. J. Leonard, «Large-Scale Concurrent Mapping and Localization», MIT Dept. of Ocean Engineering, Cambridge, MA 02139, U.S.A, 2001.

[31] A. Kelly, «Mobile Robotics: Mathematics, Models, and Methods», Cambridge, U.K., Cambridge Univ. Press, 2013.

[32] P. Newman, J. J. Leonard, J. D. Tardos, J. Neira, «Explore and Return: Experimental validation of Real-Time concurrent mapping and localization», in Proc. IEEE Int. Conf. Robot. Autom., 2002.

[33] F. Lu and E. Milios, «Globally consistent range scan alignment for environment mapping», Auton. Robots, vol. 4, no. 4, 1997.

[34] M. Kaess, A. Ranganathan, F. Dellaert, «iSAM: Incremental smoothing and mapping», IEEE Trans. Robot., vol. 24, no. 6, 2008.

[35] F. Dellaert, «Factor graphs andGTSAM:Ahands-on introduction», Georgia Institute of Technology, Atlanta, GA, USA, Tech. Rep. GT-RIMCP& R-2012-002, 2012.

[36] R. Kummerle, G. Grisetti, H. Strasdat, K. Konolige, W. Burgard, «g2o: A general framework for graph optimization», in Proc. IEEE Int. Conf. Robot. Autom., 2011.

[37] S. Agarwal et al., «Google. Ceres Solver», 2016. [Online]. Available: http://ceres-solver.org.

[38] R. Salas-Moreno, R. Newcombe, H. Strasdat, P. Kelly, A. Davison, «SLAM++: Simultaneous localisation and mapping at the level of objects», in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2013.

[39] A. I. Mourikis and S. I. Roumeliotis, «A Multi-state constraint Kalman filter for vision- aided inertial navigation», in Proc. IEEE Int. Conf. Robot. Autom. (ICRA), 2007.

[40] D. G. Kottas, J. A. Hesch, S. L. Bowman, and S. I. Roumeliotis, «On the consistency of vision-aided inertial navigation», in Proc. Int. Symp. Exp. Robot., 2012.

[41] J. A. Hesch, D. G. Kottas, S. L. Bowman, and S. I. Roumeliotis, «Camera-IMU-based localization: Observability analysis and consistency improvement», Int. J. Robot. Res., 2014.

[42] E. Tramacere, S. Luciani, S. Feraco, S. Circosta, I. Khan, A. Bonfitto, N. Amati, «Local trajectory planning for autonomous racing vehicles based on the Rapidly-Exploring Random Tree algorithm», Proceedings of the ASME 2021 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, Torino, 2021.

[43] «Velodyne website», [Online]. Available: https://velodynelidar.com/

[44] «The stereolabs website», [Online]. Available: https://www.stereolabs.com/zed/.

[45] «SBG system website», [Online]. Available: https://www.sbg-systems.com/.

[46] «PerceptionAndSlam_KTHFSDV1718 package», [Online]. Available: https://github.com/isothiazol/PerceptionAndSlam_KTHFSDV1718.