POLITECNICO DI TORINO

Master of Science in Mechatronic Engineering

Master of Science Thesis

Obstacles Detection and Global Mapping Algorithm for an Autonomous Racecar

Supervisors: Prof. Nicola Amati
Dr. Feraco Stefano

Candidate: Santacroce Giovanni
265331

April 2021

Acknowledgment

I wish to extend my sincere gratitude to Prof. Nicola Amati for offering me the opportunity to work in the innovative field of the autonomous driving and to Dr. Stefano Feraco for his kind and experienced support.

A special thanks also to my teammate Sebastian Fernandez for his kind availability and for his indispensable work.

The most special thanks go to my parents, my sister and to my grandparents who have always supported me.

Finally, I am thankful also to my great friend Giuseppe for his sincere and beautiful friendship, and to all my friends in Lucera and in Turin for all the fun moments spent together.

Abstract

Nowadays, mapping is one of the most active research fields in robotics. In this thesis it is faced the global mapping issue related to the SLAM (Simultaneous Localization and Mapping) process of an autonomous vehicle which will participate to the FSD (Formula Student Driverless) competition. The mapping process together with the localization one allows to perform car navigation, which is the ability of getting the vehicle from place to place. Thus, SLAM allows to answer to three fundamental questions for a driverless vehicle: "Where am I?", "Where am I going?" and "How do I get there?". So, in this thesis it is implemented and tested a global mapping algorithm that uses the data coming from a stereo camera to build the map of a racetrack traveled by the driverless racecar. To do this, it is used a ROS Melodic node written in Python 3.7. And all the software stuff is installed on an Nvidia Jetson Xavier computer mounted on the PoliTO SC19 electric racecar. As far as the algorithm testing is concerned, this has been performed through a computer simulation using the EUFS (Edinburgh University Formula Student) simulator and through an on-track testing in order to verify the right functioning of software and hardware together.

Table of Contents

Chapter 1 - Introduction

1.1 Evolution of Driver-Less Vehicles

An autonomous car, driverless car or self-drive car is a vehicle that is able to move safety within a certain environment with little or no human input. To do this an autonomous vehicle must have the capacity of sensing the surrounding environment thanks to a variety of sensors like radar, laser scan, camera, sonar, GPS and IMU. The dream of self-propelled cars goes back to the Middle Age, centuries before the invention of the car. In fact, one of the first idea regarding the autonomous vehicle were found in some Leonardo Da Vinci sketches. The first self-drive prototypes date back to the 1920s. Although this kind of cars are named driverless, they relied heavily on specific external inputs. The most famous of these prototypes was the radio controlled "American Wonder", a 1926 Chandler equipped with a transmitting antenna on the tonneau. To work this autonomous car needs of a person within another car that follow it and sent out radio impulses which were caught by Chandler's transmitting antenna to guide the small electric motor of the car [1]. In the 1935 a radio-controlled electric car, propelled by electromagnetic fields provided by circuits embedded in the roadway, was presented in the Norman Bel Geddes's Futurama exhibit sponsored by General Motors at the World's Fair. From approximately 1980 to 2003, university research centers, sometimes in partnership with transportation agencies and automotive companies, undertook basic studies of autonomous transportation. Two main technology concepts emerged from this work. The first one had as objective the development of automated highway system where the vehicles are strictly dependent on the highway infrastructure. Instead, the second one had as task the designing of autonomous vehicles that depend little, if at all, on highway infrastructure. In the early 1980s, a team led by Ernst Dickmanns at Bundeswehr University Munich in Germany developed a vision-guided vehicle that navigated at speeds of 100 kilometers per hour without traffic. From 2003 to 2007, the U.S. Defense Advanced Research Projects Agency (DARPA) held three "Grand Challenges" for driverless vehicles that gave a great impetus to the development of new autonomous technology solutions [2]. During these last

years, most of the big companies like Mercedes Benz, Audi, BMW, Tesla, Hyundai, etc. have begun developing or forming partnerships around autonomous technology. In particular the companies that are playing a crucial role in the autonomous technology innovation are Tesla, Google and nuTonomy [1].

Figure 1: Evolution of Driverless car [1]

1.2 Driverless Vehicle Impact

The potential impact that the driverless vehicle utilization have is very large. It could be both positive and negative implications. In this paragraph will be illustrated some of the most significative positive implications of the self-drive car deployment. The first advance that we can obtain adopting the autonomous cars is the improvement of public safety. In fact, about the 90 % of automobile accidents are caused by human errors. So, with the self-drive cars the number of automobile accidents could be strongly lowered. Another important advantage

that can be obtained is the improvement of the elderly, disabled and youth mobility. Moreover, the usage of driverless car could have positive effects also in the traffic circulation increasing road capacity and throughput. The ability to constantly monitor surrounding traffic and respond with finely tuned braking and acceleration adjustments should enable autonomous vehicles to travel safely at higher speeds and with reduced headway (space) between each vehicle. Research indicates that the platooning of autonomous vehicles could increase lane capacity (vehicles per lane per hour) by up to 500 percent. And finally, driverless car can give an important contribution for reducing emissions. For instance, a self-driving, electric taxi in 2030 would produce 90 percent lower greenhouse gas emissions (GHG) than a 2014 gasoline-powered privately owned vehicle, and 63 to 82 percent fewer GHG emissions than a 2030 privately owned vehicle with a hybrid engine [3].

1.3 SAE Autonomous Vehicle Standard Levels

Society of Automotive Engineers International (SAE) is a standardization body in the aerospace, automotive and vehicle industries. Its headquarters is in Troy, Michigan (USA). This society was born in 1905 and its main task is which to promote the usage of standard rules in the field of automotive industry [4]. It provides a taxonomy with detailed definitions for six levels of driving automation, ranging from no driving automation (level 0) to full driving automation (level 5), in the context of motor vehicles. The levels apply to the driving automation feature(s) that are engaged in any given instance of on-road (publicly accessible roadways) operation of an equipped vehicle. This means that although a given vehicle may be equipped with a driving automation system that can deliver multiple driving automation features that perform at different levels, the level of driving automation exhibited in any given instance is determined by the feature(s) that are engaged. The levels of driving automation are defined with respect to the specific role played by each of the three primary actors in performance of the DDT (Dynamic Driving Task) and DDT fallback. They are the steering, acceleration, braking control, the monitoring of the driving environment and the fallback performance. Active safety systems, such as electronic stability control and automated emergency braking, and certain types of driver assistance systems, such as lane keeping

assistance, are excluded from the scope of this driving automation taxonomy [5]. So, specifically the SAE levels are:

Level 0: No driving automation. This level is characterized by the 100% of human presence. Acceleration, braking, and steering are constantly control by a human driver even if they are supported by safety intervention systems. This level also included the automated emergency braking.

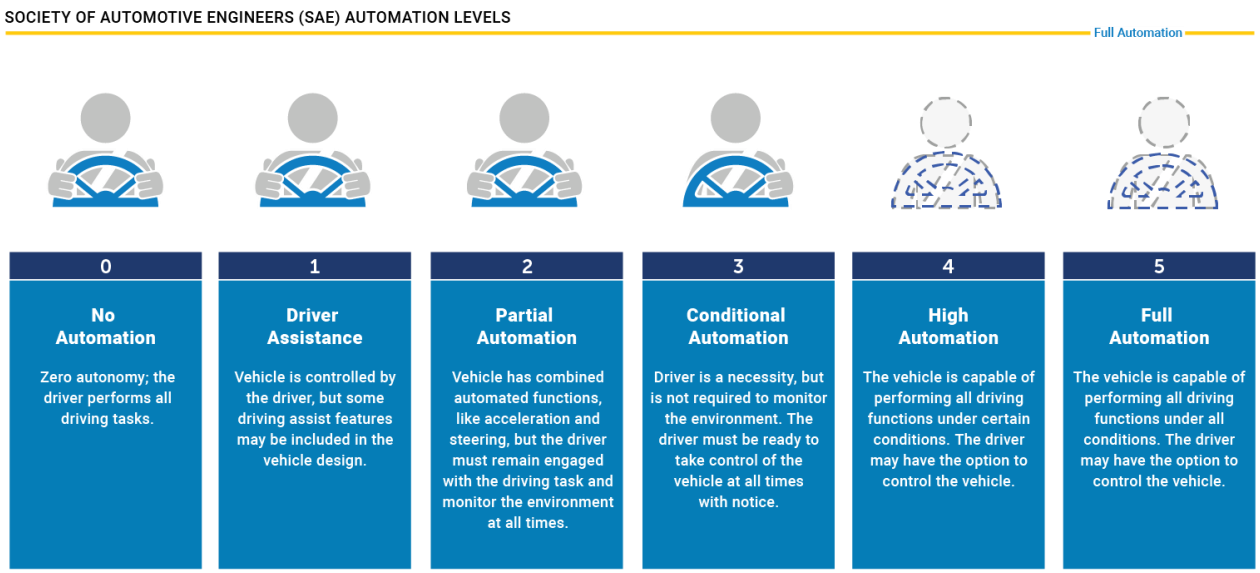


Figure 2: SAE automation levels [6]

Level 1: Driver Assistance. The computer never controls steering, accelerating and braking simultaneously. The best examples of this kind of controls are the adaptive cruise control and the parking assistance.

Level 2: Partial Automation. The driver can take his hands off the steering wheel. At this level, there are set-up options in which the car can control both pedals and the steering wheel at the same time, but only under certain circumstances. During this time, the driver has to pay attention and if it is necessary, intervene.

Level 3: Conditional Automation. The car has a certain mode that can take full responsibility for driving in certain circumstances, but the driver must take the control back when the

system asks. At this level, the car can decide when to change lanes and how to respond to dynamic events on the road and it uses the human driver as a backup system.
Level 4: High Driving Automation. The vehicle can drive itself under suitable circumstances, and it does not need human intervention. If the car meets something that it cannot handle, it will ask for human help, but it will not endanger passengers if there is no human response.
Level 5: Full Driving Automation. All driving tasks are performed by the computer on any road under any circumstances, whether there is a human on board or not [1].

1.4 Few Words About FSD Competition

Formula Student Germany (FSG) is an international engineering competition aimed at teams of university students which have the objective of designing, fabricating, and competing with small, formula style, race cars. The competition is split into three classes: internal combustion engine vehicle, electric vehicle, driverless vehicle (which can be either electric or combustion engine vehicle). Contrary to the other two classes, the FSD rules allow to use a team's existing race car. Thus, we are trying to turn the PoliTo SC19 electric race car into a driverless one, that will nevertheless have to comply the requirements of its appropriate class. All this will come equipping the car with the appropriate sensors, actuators and computing hardware and software. Independently of which class the team participates the competition consists of two disciplines a static and a dynamic one. As far as static discipline is concerned this consists in the presentation of a cost report and a business plan. Then to participate to the dynamic event a technical inspection must be passed to ensure the car is safe and respects the competition requirements. On the other hand, regarding dynamic discipline, the vehicle must run along a track delimited by cones (5 km over 10 laps) as fast as possible without the help of human racer pilots or remote-control systems. Since a car's total score is comprised of two disciplines, so the fastest car does not necessarily win the competition [7].

Figure 3: The PoliTo SC19 electric race car during an experimental session;

1.5 The SLAM Process

The SLAM (Simultaneous Localization and Mapping) or CML (Concurrent Mapping and Localization) is a process through which a mobile robot, moving into a certain environment, is able to derive a map from its perception and simultaneously computes its own position within this map. In order to build a map from the environment, the robot must be equipped with sensors that allow it to perceive and obtain measurements of the elements from the surrounding world. These sensors are classified into exteroceptive and proprioceptive. Exteroceptive sensors are for example: sonars, range lasers, cameras, and global positioning systems (GPS). While proprioceptive are, for instance: encoders, accelerometers, and gyroscopes. Generally, the exteroceptive sensors are noisy and have limited range capabilities. In particular, sonars, range lasers and cameras allow only local views of the environment. Sonars and range lasers allow to gather very accurate and dense information of the external environment, but they have the following drawbacks: they are not useful for recognizing objects and in highly cluttered environments; therefore, they are very heavy and expensive. Instead, as far as the GPS sensor is concerned, it does not work well in narrow streets, under water, in other planets or in certain indoor environments. On the other hand, regarding proprioceptive sensors, these allow obtaining an incremental estimate of the vehicle's movements by means of dead-reckoning navigation method (also known as deduced-reckoning), but due to their inherent noise they are not sufficient to have an accurate estimation of the car's position all the time, since errors are cumulative [8]. In our case the

mobile robot is the formula student car that moves along the racetrack delimited by traffic cones; hence, the cones are the landmarks of our map. These cones are detected and recognized thanks to a stereo camera and a LIDAR. While the position of the car is computed thanks to a GPS and an IMU. Since the racetrack can be considered a nearly planar environment, the vehicle pose is described by three variables, two for the car position (x_{car}, y_{car}) and one for the car orientation (yaw angle φ), while the cones can be described by two coordinates (x_{cone}, y_{cone}). Thus, the SLAM is an estimation problem which usually use an Extended Kalman Filter to provide estimates of vehicle and landmarks positions, and which is characterized with respect many other estimation problems by three exceptional features: the error accumulation, the high dimensionality, and the non-linearity [9].

1.6 The Error Accumulation Problem

As far as the error accumulation is concerned, let us image that our vehicle is moving in a known racetrack using an a-priori map. In this case the uncertainty of the car's position remains bounded. This because the detection of two landmarks (two cones) reduces the uncertainty to the landmarks' uncertainty plus the observation one. But this is not our case, in fact, during the race the vehicle is moving along an unknown circuit and the uncertainty of its pose in absolute (global) coordinates will get arbitrarily large because the odometry error accumulates overtime. So, the uncertainty of car pose increases the uncertainty of the absolute position of all cones that are in the global map, acting as a rigid-body transformation. Thus, under these conditions, we can say that relations are enough accurate while the absolute position are highly uncertain especially if we are mapping big environments. This cause the building of not-consistent map of the environment and one of the main problems of the error accumulation is the not closed loop. Let us consider our car moving along the racetrack. During the first lap the vehicle computes a first global map. Nevertheless, this map must be updated periodically so when the car starts the second lap, if the car does not re-identify any landmarks the loop is not close as we can observe in the following pictures:

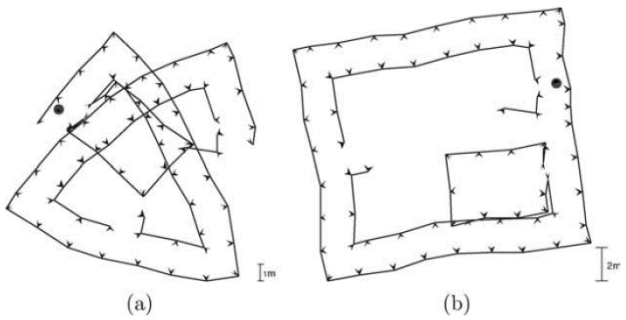


Figure 4: closed loop without (a) and with (b) integration [9]

When, instead, at the beginning of the loop a cone is re-identified the correspondent measurement is integrated into the map causing the loop to be closed. To obtain this, the SLAM algorithm must deform the whole loop to include the information of a connection between both ends of the loop. In this case the map is said "topologically consistent" or "globally consistent" (Lu and Milios, 1997; Duckett et al., 2002) [9].

1.7 The Computational Complexity: A Brief Overview About Strategies to Increase the Computationally Efficiency

As previously mentioned, another important issue that characterized the SLAM problem is the high dimensionality, hence, the computational cost. In fact, after each measurement, a SLAM algorithm has to estimate the vehicle pose which has three degrees of freedom and the whole map that has two times the number of cones degrees of freedom. So, the dimension of the SLAM problem is very large, and this is a problem for the computational point of view. Techniques to reduce the computational efforts can be of two types: optimal and conservative. Optimal algorithms aim to reduce required computation while still resulting in estimates and covariances that are equal to the full-form SLAM algorithm. Conservative algorithms result in estimates that have larger uncertainty or covariance than the optimal result. Usually, conservative algorithms, while less accurate, are computationally more efficient and, therefore, of value in real implementations. To increase the computational efficiency of the SLAM algorithms there are some map management strategies. These strategies can be split into three large categories: one based on the use of few landmarks using

active sensing, the second based on the possibility of applying sub-optimal update steps and other based on the way the information are added to the maps [10] [9]. Let us take a brief look at these strategies. The first method we spoke about is the one based on limiting the number of landmarks. One way to obtain this is using an active sensing system in order to look at only a limited part of the environment. To apply this strategy is important to identify within the environment the landmarks that are more probable of remaining visible during the vehicle motion. Other strategies are that based on the use of sub-optimal or near-optimal covariance updating like the covariance intersect and the partitioned update (N.B. these are both conservative techniques) [10]. The covariance intersect method was proposed by Uhlmann et al. (1997) and is an algorithm which combines state estimates and computes an update covariance matrix without an a priori knowledge of the correlations between states (cross-correlation between states). Using the covariance intersect algorithm each observation is conservatively fused into the estimated state matrix. This method is computationally efficient but as Uhlmann himself show us landmarks estimation have very large covariances. This large covariance can make data association difficult and does not allow the map to converge [10]. As far as partitioned update is concerned this is a method, proposed by Guivant and Nebot (2001), based on the observation that the update of a large proportion of the map is not convenient. Thus, they thought that is better to update only a select subset of map's landmarks during a particular observation reducing the computational complexity to O(N) instead of $O(N^2)$, where N is the number of states of the map (vehicles pose + landmarks pose). In these last years, the research has concentrated its effort in the development of alternatives map representations in order to improve the computational complexity keeping a good accuracy [11]. The two main method which fall in this category are the relative maps and the submaps techniques. Relative maps method was proposed by Csorba and Durrant-Whyte in 1997. The basic idea was to use the relative position between landmarks rather their absolute position as state variables. Since the relative position between landmarks are independent of any global coordinates frame and of vehicle's pose, relative locations are uncorrelated. Thus, the covariance is a diagonal matrix, and each observation can be fused into the SLAM filter easily updating the estimate of the observed relative positions [12]. In the 1999 Newmann developed this idea and demonstrated that although the relative maps

technique is very efficient for a computational point of view, it can give inconsistent estimates of relative states. To overcome this problem Newmann introduced the Geometric Projection Filter (GPF) which is a mechanism through he was able to enforce the consistency of relative estimated states. So, introducing correlation between the states Newmann was able to recover a consistent map. Newmann also highlighted some drawbacks which characterized this approach like the very difficult usage of data association [13]. For last, let us speak of submaps technique. This method consists in splitting a large map in a series of smaller submaps to reduce the computational effort and the covariance update process. One submaps approach was proposed in 1998 by Chong and Kleeman. While the vehicle moving in a certain environment a series of local submaps is generated and an estimate of relative position between these submaps were computed and kept. So that a search algorithm can be used to re-localize the vehicle when its estimate has a high probability of being in the area of a previous local submaps. This hierarchical approach allows to drastically reduce the storage effort and the processing time with respect a global approach but does not provide a globally consistent map of the environment [14]. In 2001 Williams demonstrate that a globally consistent map of environment can be reached using a submaps approach adding some opportune constraints. To conclude another technique is that introduced by Castellanos (Castellanos et al., 1999) using global and local frames of reference in the mapping process. Landmark frames were defined in the global reference frame and used to estimate sets of features and to generate a consistent global estimate of vehicle pose, the constraints between the various reference frames are used. A unique vehicle estimate is maintained with respect to each of the landmark frames. These distinct vehicle estimates are then fused together, using constraints, to recover a consistent estimate of the map states [10].

1.8 The Linearization Error

In the SLAM estimation problem, the equations are non-linear in the vehicle's orientation, thus due to this we have an estimation error that can increase unboundedly. This creates big problem when the map which must be linearized is very large. One of the effects of the linearization error is the distortion of distances between the landmarks although these

distances are well known from the measurements [9].

1.9 Requirements for An Ideal Slam Solution

In 2001 Frese and Hirzinger proposed for the first time three requirements that an ideal SLAM algorithm should fulfill. These conditions are based as explained Frese in his article "A discussion of simultaneous localization and mapping, 2006" on an intentionally inexpert view of the problem disregarding its apparent difficulty but asking how mapping should work when based on a common sense understanding of maps. These three requirements are the following:

1. **Bounded Uncertainty:** the uncertainty of any aspect of the map should not be much larger than the minimal uncertainty that could be theoretically derived from the measurements.
2. **Linear Storage Space:** the storage space of a map covering a large area should be linear in the number of landmarks O(n), where n is the number of landmarks.
3. **Linear Update Cost:** incorporating a measurement into a map covering a large area should have a computational cost at most linear in the number of landmarks.

As far as the first requirement is concerned, it states, simply, that any information which can be known from the measurements should be represented, even if in a roughly way, in the map. Thus, since a lot of relations can be known from the measurements, not representing one would violate this principle. The violation of bounded uncertainty principle causes the building of topologically not-consistent maps and the not closure of large loops. Of course, to create a more efficient SLAM algorithm it is possible to consistently approximate some relations. Instead, to explain the second requirements we can do an easy example. Let us image our vehicle moving along the racetrack delimited by the traffic cones. Since the map of the whole circuit is represented by the position of all the cones which delimit the race path, the size of map should be more or less proportional to the cones' number which represents the map's landmarks. But simply storing all measurements the map will not meet the linear storage space requirements. In fact, the storage space will be proportional to the number of measurements and not to the number of landmarks, moreover, the map's size would grow

during motion even when the vehicle repeatedly travels around the same area, as when the car travels more laps of the same racetrack. Finally, regarding third requirement, let us consider a building composed of two rooms A and B which have connected each other by few doorways. So, the map of the building consists of the map of both rooms A and B plus some information regarding the connections between the two environments. At first the measurement has to be incorporated into the map of A, taking into account the known effect of A on the connection between A and B. Then, the effect of these connections onto B must be computed. This is equivalent to incorporating several measurements concerning the connections into the map of B. However, computation can be deferred until the robot actually enters B, sharing the computational cost with all other measurements that have generated effects on the connections. As the number of landmarks in the connections is small, this should not take more time than incorporating the original measurement into the map of A. So, the total cost for integrating a measurement into a map of A and B should not be larger than the cost of integration into A plus the cost of integration into B, thus being linear in the number of landmarks. So, we can say that the first requirement links the map to the real environment since it states that almost all the environment's information must be represented. While the other two requirements are related to the efficiency requiring linear space and time consumption [9].

1.10 The State of the Art of the SLAM Process

The first approaches to the solution of the SLAM problem date back to the mid-eighties. The mathematical formulation of SLAM was still open. The first attempts are due to Moravec and Elfes (1985) [15]. They divided the map into a regular grid with square cells of fixed size. Each cell stores a real number among 0 and 1 representing the accumulated evidence of containing an obstacle. This approach is known as "evidence grids", it works well enough to integrate the noisy low-resolution information provided by ultrasonic sensors, but it cannot represent robot pose uncertainty. Thus, it is not suited to SLAM problem. During the same period, feature-based approach was also followed by several authors like Brooks (1985), Cathila and Laumond (1985), Cheeseman and Smith (1986), Durrant-Whyte (1988),

Faugeras (1989). These approaches can incorporate uncertainty in the robot pose and led to an estimation theoretic formulation of SLAM. The turning point was in 1988 with an important paper of Smith, Self and Cheeseman who first formulated a probabilistic approach for the solution of the SLAM estimation problem. They realized that landmark estimates are highly correlated because of the accumulated error of the robot pose, so they proposed to represent all landmarks and the robot pose in a unique state vector in combination with a full covariance matrix. The representation was called stochastic map and is based on an extended Kalman filter (EKF) [16]. This approach was followed, later, by many authors like Tardos (1992), Castellanos et al. (1999), Herbert et al. (1995), Newman (1999) that extended it, but the main problem of great computational effort remained. This because the original results of Gauss, obtained in the distant 1821, regarding the maximum likelihood estimation (ML) and later work in the field of surveying and photogrammetry (Triggs et al., 2000) are unknown in the SLAM field. In fact, the latter methods as opposed to those based on the extended Kalman filter are based on information matrix representation which has the advantage with respect to the covariance matrix to be a sparse matrix. So much so that the most time-consuming part of the SLAM algorithm is that related to the updating of covariance matrix, taking among $O(n^2)$ time for a map characterized of n landmarks. Hence, this limited the use of the SLAM algorithm to small map ($n < 100$) [9]. Then, Guivant and Nebot, the first in 2001 and the second in 2003, developed the Compressed Extended Kalman Filter approach (CEKF). This method allows the accumulation of measurements related to a local region of the environment that you want to map, this region will be characterized by a number k of landmarks. So, the measurement accumulation related to the local region will have a time cost of $O(k^2)$ independent from the overall map size n. When the vehicle leaves the region, the accumulated results must be propagated to the full EKF to perform the global update. This operation will have the cost of $O(kn^2)$. Therefore, landmarks are grouped in constellations and their coordinates expressed relative to some reference landmarks for each constellation. This makes that the correlations between landmarks belong to distant constellations can be neglected and an approximate global update can be performed with a time cost of $O(kn^{\frac{3}{2}})$ and with a storage space of $O(n^{\frac{3}{2}})$ [11] [17]. In 2002 Duckett et al. had a new idea based on Gaussian maximum likelihood estimation. The idea is to find the

optimum for a chosen robot pose or landmark keeping all other landmarks and robot poses fixed using an iterative equation solver called "relaxation" to the linear equation system that characterized the maximum likelihood estimation. In particular they apply one iteration updating each robot pose and landmark after each measurement with computation time O(kn) and O(kn) storage space. The big drawback of this method is that when there is big, closed loop, more iterations are necessary leading to $O(kn^2)$ computation time in the worst case [18]. This problem was solved by Frese and Duckett in 2004 by a method called Multilevel Relaxation (MLR). They employ a multilevel approach similar to the multigrid methods used in the numerical solution of partial differential equations. So, computation time could be reduced to O(kn) even when closing large loops [19]. Another method to face the SLAM problem is the so-called fast SLAM algorithm implemented in 2002 by Montemerlo et al. This algorithm is based on the fact that the landmark estimates are conditionally independent given the robot pose, thus a particle filter is implemented where every particle represents a sampled robot trajectory and associated Gaussian distribution of the different landmark positions. The computation time is O(Mlog(n)) while the storage space is O(Mn) where M is the number of particles and n is the number of landmarks. The main advantage of this algorithm is the ability to handle uncertain data association, instead, the major drawback is that the efficiency crucially depends on M being not too large. On the other hand, a large number of particles is necessary to close a loop with a large error [20]. In 2003 Eliazar and Parr, Hahnel et al. (2003) as well as Grisetti et al. (2005) extended the fast SLAM technique using plain evidence grids as particles. This method allows to build maps in difficult situations without any sensor processing such as landmarks extractions or scan matching [21]. Another important algorithm to face the SLAM problem is the Sparse Extended Information Filter (SEIF) developed in 2002 by Thrun et al. This method uses the information matrix instead of the covariance one to represent the uncertainty and take advantage of information matrix sparsity that requires a storage space of O(kn) and a computation time of $O(k^2)$. It is important to point out the fact that to compute the estimate a system of n linear equations must be solved. This can be done through relaxation but Thrun et al. proposed not to relax all n landmarks but only a subset of O(k) obtaining in this way a formally computation time of $O(k^2)$. This technique is called amortization, this because the

computation time needs to solve the equations is distributed over several update steps. At this point it is important to observe that the amortization method does not satisfy the "bounded uncertainty" requirement discussed in the previous paragraph. This because after closing the loop, the map's error will be much larger than the optimal estimate for a long time. So, to summarize, the main difference between relaxation and amortization method is that the first one needs O(n) iterations with $O(kn^2)$ time for reducing the equation error by a constant factor (Press et al., 1992) while the second one (the amortization) needs about $O(\frac{n^2}{k})$ update steps or $O(n/k)$ amount of time along the loop until the effect of closing the loop is almost incorporated into the estimate. Moreover, the SEIF algorithm can be modified in order to update all landmarks after each measurement and perform O(n) iterations after closing a loop [22]. In 2003 Paskin expresses the SLAM problem as a Gaussian graphical model implementing a new filter: The Thin Junction Tree Filter (TJTF). It creates and keeps a tree where the overall posterior distribution is represented as the product of a low-dimensional Gaussian at each node [23]. The following year (2004), Frese has implemented a similar algorithm based on so-called tree map that is organized hierarchically dividing the map into local regions and subregions. For each regions and subregions (so at each level of hierarchy), the algorithm stores a matrix representing the landmarks at the region's border. Thus, the large sparsely information matrix is divided into a sum of small matrices exploiting the special topology encountered in typical buildings. A measurement is integrated into a local subregion using $O(k^2)$ computation time. The global update necessary when moving to a different subregion requires only $O(k^3 \log n)$ computation time by virtue of the hierarchical decomposition. Computing an estimate for the whole map takes O(kn) time [9]. Until now we have spoken about SLAM algorithm based on probabilistic filters but there are also other methods that can substitute these or that can be complementary. For example, the Structure for Motion (SfM) techniques have an important role in the solution of the SLAM algorithm. These techniques allow to compute 3D structure of scene and camera position from a set of images (Pollefeys et al. 2004). These techniques are based on the extraction of salient features of incoming images in order to match them and perform a non-linear optimization named Bundle Adjustment (BA) to minimize the re-projection error (trigs et al. 1999; Engels et al.

2006). One method to face the SLAM problem through the SfM techniques is the Nister et al. (2004) visual odometry. This strategy consists in determine simultaneously the camera pose for each video frame and the position of features in 3D world, using only images in a casual way and in real time. Between 2006 and 2009 this method was improved by Mouragnon et al. adding a technique called Local Bundle Adjustment, reporting trajectories up to 500m. Then in 2007 Klein and Murray present a monocular method called Parallel Tracking and Mapping (PTaM). This approach is based on keyframes with two parallel processing threads. A keyframe is a video frame that is different enough from its predecessor in the sequence, to represent a new location. They are very useful to estimate efficiently the pose of the camera and reduce the redundancy information. Going back to the PTaM method, the first thread of execution performs the task of robustly tracking a lot of features while the second one produces a 3D point map thanks to the BA techniques. It would have to be underline that this method presents tracking failures when there are similar textures and moving objects. In (Konolige and Agrawal, 2008; Konolige et al., 2009) two techniques based on the SfM are presented, these techniques are the frameSLAM and View-Based Maps. The authors used a stereo camera mounted on a wheeled robot. The maps obtained are based on a structure consisting of a non-linear constraint graph between frames. The results of these two methods shows a good performance on long trajectories (about 10 km) under changing conditions like urban environments. Finally, other methods to solve the SLAM problem are the Bio-inspired models. An example of this methods is the RatSLAM. This algorithm proposed by Milford et al. (2004) is based on the models of rodents' hippocampus (the brain's part responsible of spatial memory) and can generate consistent and stable representations of complex environments using a single camera. The experiments carried out by the Milford himself and Wyeth in 2008 and Glover et al. in 2010 demonstrate good performances in real-time tasks in both indoor and outdoor environments. Moreover, it can close more than 51 loops of up to 5 km in length and at different hours of day. In 2010, Collet examines the behavior of ants in desert to understand how they are guided by visual landmarks. From this research it turns out that these techniques could be viable and easy to implement in a robot to perform a SLAM process [8].

Chapter 2 – Sensors Setup

2.1 The sensors available for the SLAM process

To perform the SLAM, process the PoliTo SC19 electric car is equipped with the following sensors: a LIDAR, an IMU, an RTK GNSS module and a stereo camera. In the following figure it is shown the position of these sensors mounted on the car.

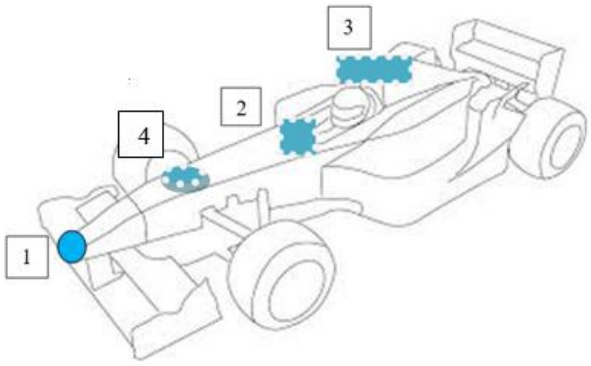


Figure 5: Sensors placement: 1) LIDAR, 2) IMU, 3) Stereo Camera, 4) RTK GNSS module

Thus, let us give a more detailed sight for each sensor:

The Velodyne LIDAR: The LIDAR (Light Detection and Ranging) is a technology which allows to measure distances by illuminating the target with laser light and measuring the reflection with a sensor. Differences in laser return times and wavelengths can then be used to make digital 3-D representations of the target. It is constituted by rotating mirrors that emits laser pulses [24].

Figure 5: the Velodyne LIDAR mounted on the car [36]

The results set of points that is obtained is called pointcloud. The main parameters that characterized it are the number of layers and the vertical plus the horizontal resolution represented by the angles between points. Its main drawbacks are that it is not so useful for recognizing objects (since it does not give color information) and in highly cluttered environments. Therefore, it is very expensive.

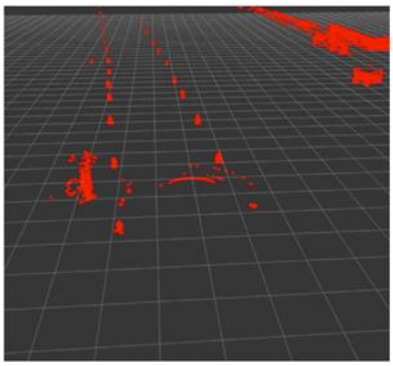


Figure 6: example of pointcloud obtained by a 40-layer single LIDAR [27]

The SBG Ellipse-N: it is a small-sized (46*45*24 mm) high-performance RTK (Real-Time Kinematic) Inertial Navigation System (INS) with an integrated Dual band and Quad Constellations GNSS (Global Navigation Satellite System) receiver. It provides Roll, Pitch, Heading, Heave, and centimetric GNSS position. It has been best suiting for dynamic environments and harsh GNSS conditions but can also operate in lower dynamic applications with a magnetic heading. The main drawback is the accelerometer noise [25].

Figure 7: the SBG Ellipse-N mounted on the car [25]

The ZED 2 stereo camera: it is a type of camera with two lenses positioning at distance similar to the human eyes characterized by a separate image sensor or film frame for each lens. In this way it is possible to simulate the human binocular vision giving the possibility to capture three-dimensional images through a process called stereo photography.

Figure 8: the ZED 2 stereo camera mounted on the car [37]

The dimensions of the camera are described by the following technical drawing from taken from the camera datasheet:

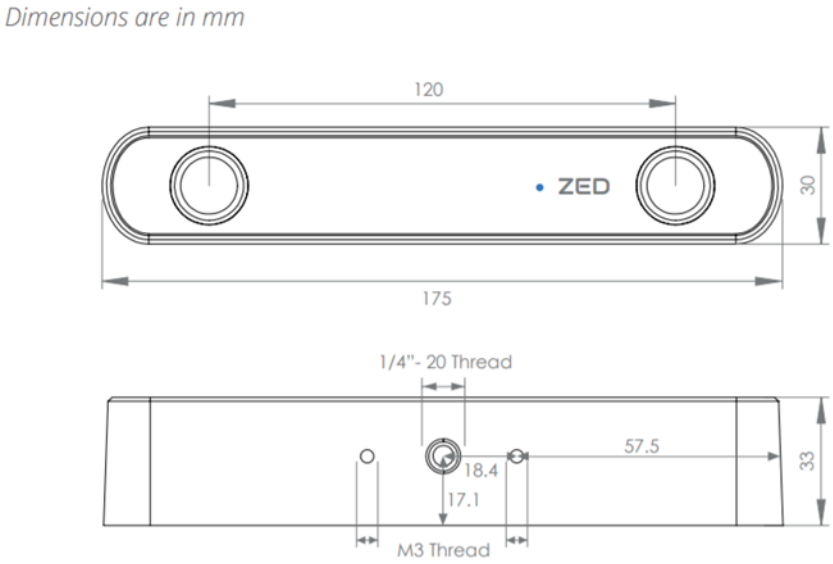


Figure 9: ZED 2 stereo camera technical sketch [37]

Moreover, the zed camera is equipped also with the following motion/environments sensors: an IMU-Accelerometer Gyroscope, two temperature sensors, a barometer and a magnetometer as shown in the sensors diagram below:

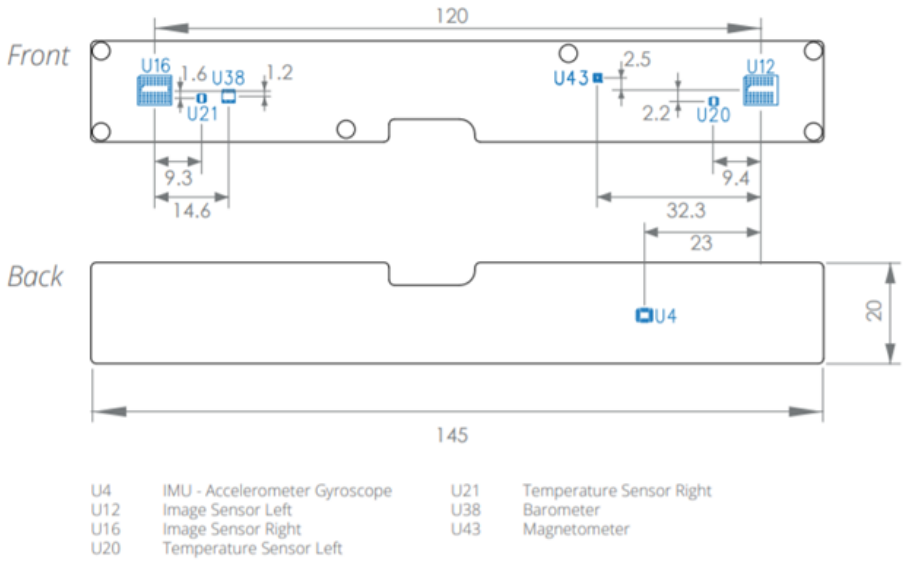


Figure 10: sensors placement on the ZED 2 stereo camera [37]

Finally, the raw data coming from the zed camera are processing through the zed SDK software as shown in the following picture:

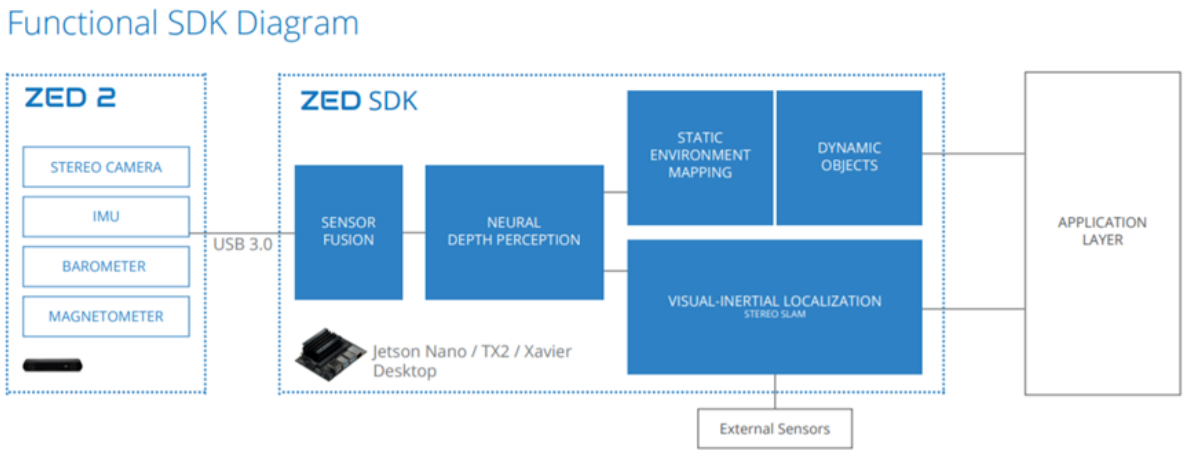


Figure 11: Functional SDK Diagram [37]

The EMLID Reach M+: it is a 35g of weight RTK GNSS module. This module uses the Real-time kinematic (RTK) positioning technique. This satellite navigation technique is used to improve the precision of position data derived from satellite-based positioning systems

(global navigation satellite systems, GNSS), hence, the GPS. It uses measurements of the phase of the signal's carrier wave in addition to the information content of the signal and relies on a single reference station or interpolated virtual station to provide real-time corrections, providing up to centimeter-level accuracy. The dimensions in mm are shown in the following drawing [26]:

Figure 12: the EMLID reach M+ module mounted on the car [26]

Finally, in the following figure are represented the connectors pinout:

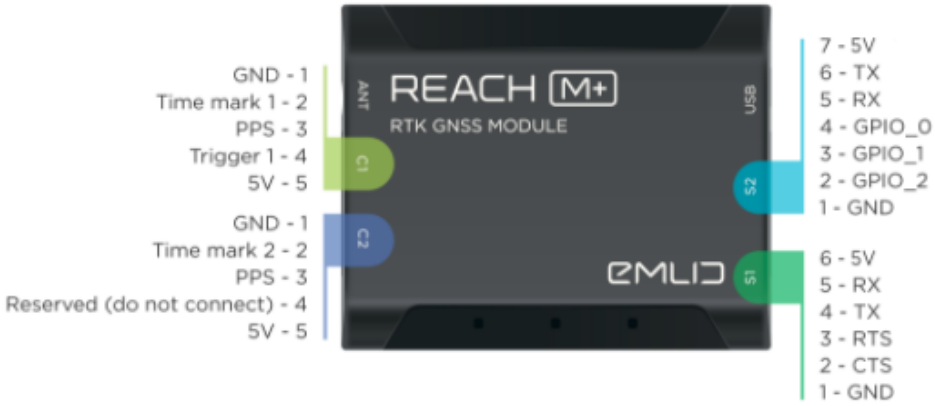


Figure 13: connectors pinout scheme [26]

2.2 Sensors Configurations

With these sensors we can have three sensor setup configurations for the perception:

- *Stereo camera or depth camera*: the 3D cone measurement is based on stereo matching. This solution benefits from the availability of several commercial solutions. Its main drawback is that the precision decreases a lot with the distance.
- *LIDAR only*: the 3D cone measurement is based on clustering techniques. Its main drawback is that we have color information only for close cones.
- *Camera + LIDAR*: two global maps are obtained, one coming out by stereo camera data and the other one coming from the LIDAR ones. These two maps will be merged in order to obtain a more accurate global map. The drawback of these configuration is the complex fusion and synchronization and the extrinsic calibration necessary for example when a sensor is moved or reattached [27]. The mapping algorithm which this thesis speaks about uses the data coming only by SBG Ellipse-N and by the stereo camera but in a near future will be used also the LIDAR sensor. Thus, we can represent the actual SLAM process as follow:

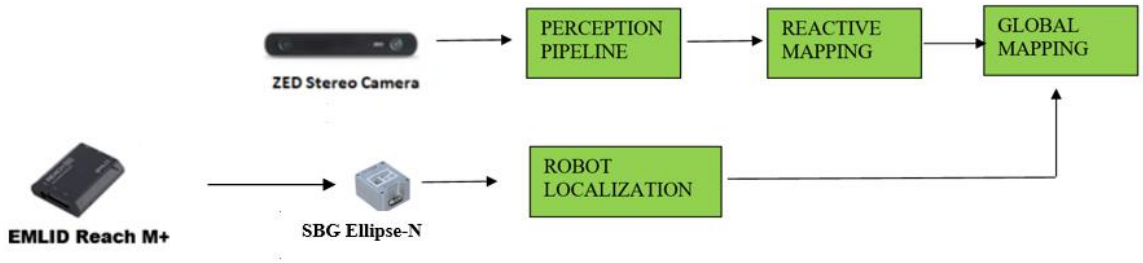


Figure 14: Overall scheme of the PoliTo SC19 electric car SLAM process

2.3 Sensors Synchronization

Since sensors data can be affected by unknown latency, the signals coming from the sensors need to be synchronized. In particular, in our case, the synchronization is between the perception sensor (the zed stereo camera) and the perceive sensor (the SBG ellipse N).
The reference time that we can consider can be three: the time stamps from ROS messages, the ACU (Autonomous Computing Unit) system time and the GPS. As far as the time stamps from ROS messages is concerned, this is not the time the measurement took place, but the time at which the message was processed by the hardware driver. So, the synchronization consists in estimating a delay for each sensor through datasheet, measurement and trial and error procedures and matching the message timestamps with fixed offsets. If the synchronization is based on ACU as time master relative time synchronization inside the system needed. This method requires sensors able to sync via PTP (Precision Time Protocol). Thus, the synchronization consists in matching messages based on internal timestamp. Finally, we can use GPS. The GPS antenna is used to acquire a reference time that is included in all messages. Following this method, also the ACU must be synced to such a GPS time as well [27]. In our case we have not set yet a temporal synchronization thus the time stamps from ROS messages are used.

Chapter 3 – The Localization Algorithm

As previously explained, the SLAM process is composed of two simultaneous actions, the localization, and the mapping. The localization is the ability of a robot to establish its own position and orientation within a fixed reference frame (the map reference frame). To reach the localization target it is necessary to solve an estimation problem that consists of fusing the raw data coming from the proprioceptive sensors to obtain a robust odometry. Since the race car is a non-linear system, this estimation problem is solved through an extended Kalman filter (EKF) which allow to estimate the maximum likelihood of random variables over time like the car velocity, to estimate hidden state like the slip angle and to perform the sensor fusion. In the following sections of this chapter, we are going to explain how it is faced the localization problem.

3.1 The Kalman Filter

The Kalman filter named after Rudolf E. Kalman, one of the primary developers of its theory, is an efficient recursive filter that estimates the internal state of a linear dynamic system from a series of noisy measurements. It works following two step-process. The first one is the prediction-step, in this phase, the Kalman filter computes the estimation of the current state variables, along with their uncertainties. The second step, instead, consists in the updating of the current state estimation observed through a weighted average. The task of the weights is that the values with smaller estimated uncertainty are trusted more. The weights are computed from the covariance, a measure of the estimated uncertainty of the prediction of the system's state. This process is repeated at every time step, with the new estimate and its covariance informing the prediction used in the following iteration. Thus, the Kalman filter is a recursive estimator. This means that to compute the estimate for the current state it is necessary to have only the estimated state from the previous time step and the current measurement. The Kalman filter model assumes the true state at time k is evolved from the state at (k − 1) according to the following equation:

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

Where:

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

At time k an observation (or measurement) z_k of the true state x_k is made according to the following equation:

$$z_k = H_k x_k + v_k;$$

Where:

- H_k is the observation model, which maps the true state space into the observed one;
- v_k is the observation noise, which is assumed to be zero mean Gaussian white noise with covariance R_k: $v_k \sim N(0, R_k)$.

The initial state, and the noise vectors at each step $\{x_0, w_1, \ldots, w_k, v_1, \ldots, v_k\}$ are all assumed to be mutually independent. In what follows, the notation $\hat{x}_{n|m}$ represents the estimate of at time n given observations up to and including at time m ≤ n. The state of the filter is represented by two variables:

- $\hat{x}_{n|m}$ that is the a posteriori state estimate at time k given observations up to and including k;
- $P_{k|k}$ the a posteriori estimate covariance matrix (a measure of the estimate accuracy of the state estimate).

The term a posteriori is used to indicate the fact that a value includes the observation information from the current state. The term a priori when a value doesn't include the observation information coming from the current state. The prediction phase can be described by the following equations:

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

The update phase, instead, can be described by these equations:

- Innovation or measurement pre-fit residual: $\tilde{y}_k = z_k - H_k \hat{x}_{k|k-1}$;

- Innovation (or pre-fit residual) covariance: $S_k = H_k P_{k|k-1} H_k^T + R_k$;
- Optimal Kalman gain: $K_k = P_{k|k-1} H_k^T S_k^{-1}$;
- Updated (a posteriori) state estimate: $\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k \tilde{y}_k$;
- Updated (a posteriori) estimate covariance: $P_{k|k} = (I - K_k H_k) P_{k|k-1}$;
- Measurement post-fit residual: $\tilde{y}_{k|k} = z_k - H_k \hat{x}_{k|k}$;

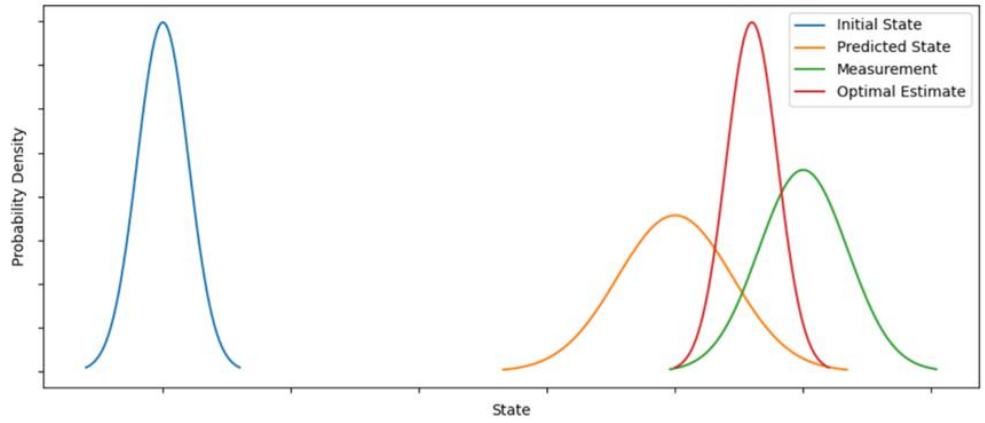


Figure 15: Kalman Filter example [27]

Since the vehicle dynamical model is not a linear one, the Kalman filter cannot be used. Thus, the Extended Kalman Filter or the Unscented Kalman Filter must be used [28].

3.2 The Extended Kalman Filter

The Extended Kalman Filter (EKF) is the nonlinear version of the Kalman Filter. In fact, in the EKF, the state transition and observation models do not need to be linear functions of the state but must be differentiable functions. Let us consider the following equations representing the first the process equation and the second the observation one:

$$x_k = f(x_{k-1}, u_k) + w_k;$$
$$z_k = h(x_k) + v_k;$$

The w_k and v_k represent the zero mean multivariate Gaussian noises with covariance Q_k and R_k respectively. u_k is the input (or control) vector. The function f is a nonlinear state transition function and can be used to compute the predicted state from the previous estimate and similarly the function h is a nonlinear sensor model which maps the state into measurement space and can be used to compute the predicted measurement from the predicted state. However, f and h cannot be applied to the covariance directly. Instead, a matrix of partial derivatives (the Jacobian) is computed. At each time step, the Jacobian is evaluated with current predicted states. These matrices can be used in the Kalman filter equations. This process essentially linearizes the non-linear function around the current estimate. Unlike its linear counterpart, the extended Kalman filter in general is not an optimal estimator. In addition, if the initial estimate of the state is wrong, or if the process is modeled incorrectly, the filter may quickly diverge, owing to its linearization. Another problem with the extended Kalman filter is that the estimated covariance matrix tends to underestimate the true covariance matrix and therefore risks becoming inconsistent in the statistical sense without the addition of "stabilizing noise". Since our physical system is represented as a continuous-time model and discrete-time measurement are frequently taken for state estimation through a digital processor we can describe the extended Kalman filter with the following equations. The system model and the measurement one is given by:

$$\dot{x}\,(t) = f(x(t), u(t)) + w(t);$$
$$z_k = h(x_k) + v_k;$$

Where: $w(t) \sim N(0, Q(t))$, $v_k \sim N(O, R_k)$ and $x_k = x(t_k)$.

The initialize equation will be:

$\hat{x}_{0|0} = E[x(t_0)]$, $P_{0|0} = E[\big(x(t_0) - \hat{x}(t_0)\big)\big(x(t_0) - \hat{x}(t_0)\big)^T]$;

The prediction-step is characterized by the following system:

$$\begin{cases} \dot{\hat{x}}(t) = f(\hat{x}(t), u(t)) \\ \dot{P}(t) = F(t)P(t) + P(t)F(t)^T + Q(t) \end{cases}$$

Solved with:

$$\begin{cases} \hat{x}(t_{k-1}) = \hat{x}_{k-1|k-1} \\ P(t_{k-1}) = P_{k-1|k-1} \end{cases} \rightarrow \begin{cases} \hat{x}_{k|k-1} = \hat{x}(t_k) \\ P_{k|k-1} = P(t_k) \end{cases}$$

Where:

$$F(t) = \frac{\partial f}{\partial x}|_{\hat{x}(t),u(t)};$$

The update-step is characterized by the following equations:

$$K_k = P_{k|k-1}H_k^T\big(H_k P_{k|k-1}H_k^T + R_k\big)^{-1};$$

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k\Big(z_k - h\big(\hat{x}_{k|k-1}\big)\Big);$$

$$P_{k|k} = (I - K_k H_k)P_{k|k-1};$$

Where:

$$H_k = \frac{\partial h}{\partial x}|_{\hat{x}_{k|k-1}};$$

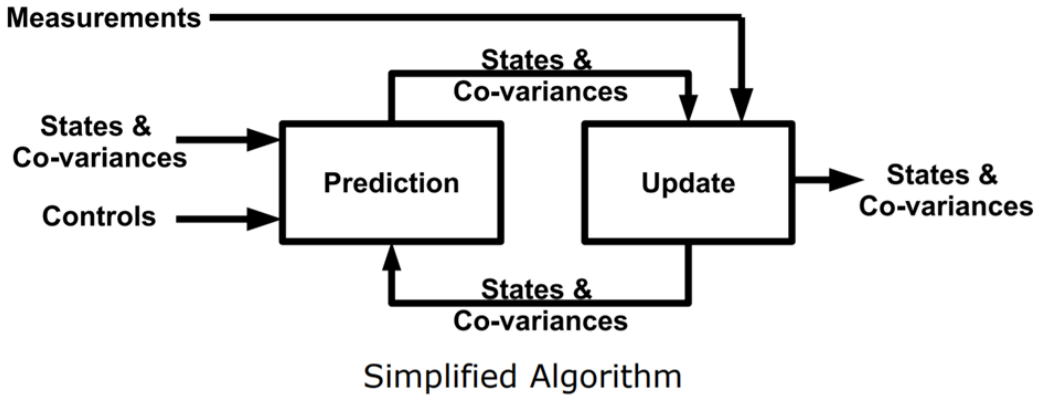


Figure 16: The Extended Kalman Filter simplified algorithm scheme [27]

For highly nonlinear model the Unscented Kalman Filter (UKF) is used. In this filter the probability density is approximated by a deterministic sampling of points which represent the underlying distribution as a Gaussian. The nonlinear transformation of these points is intended to be an estimation of the posterior distribution, the moments of which can then be derived from the transformed samples. The transformation is known as the unscented transform. The UKF tends to be more robust and more accurate than the EKF in its estimation of error in all the directions [28].

3.3 Localization Algorithm description

In Stefano Favelli's thesis "Robust positioning for autonomous racing vehicles" it is faced the localization problem that with the mapping algorithm dealing in this thesis will constitute the SLAM algorithm of the race car PoliTO SC19. The localization algorithm consists of three ROS nodes: the odometry publisher, the EKF node and the navsat_transform node.

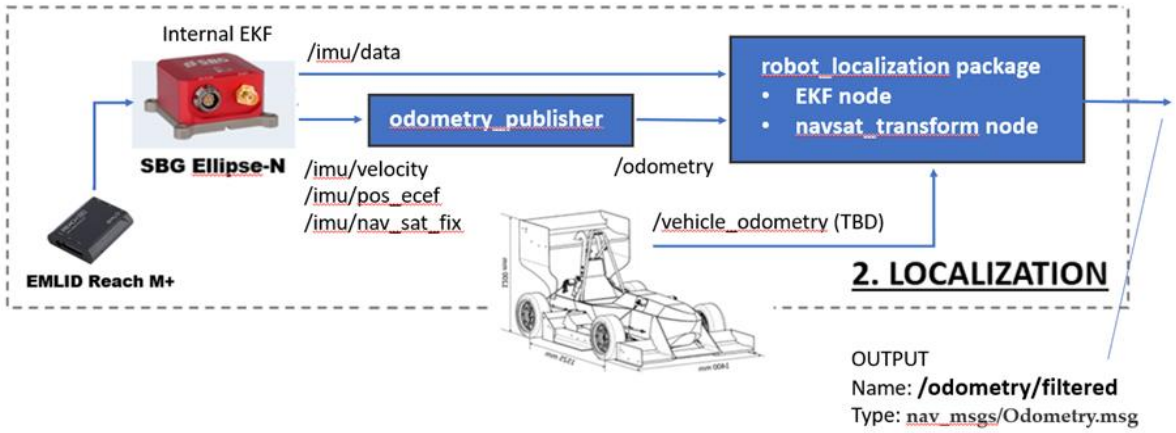


Figure 17: Localization software architecture

As far as the odometry publisher node is concerned this is a custom designed Python script to combine the output of SBG Ellipse-N into a dedicated odometry topic to be used by ROS Robot Localization Package. Instead, as far as the localization package is concerned this is a collection of state estimation nodes, each of which is an implementation of a nonlinear state estimator for robots moving in 3D space. It contains two state estimation nodes,

ekf_localization_node and ukf_localization_node. In addition, robot_localization provides navsat_transform_node, which aids in the integration of GPS data. The EKF localization node implements an Extended Kalman Filter which uses an omnidirectional motion model to project the state forward in time, and projected estimate using perceived sensor data. Regarding the navsat_transform node, this receives as inputs the nav_msgs/Odometry message coming from the EKF localization node, a sensor_msgs/Imu message containing an accurate estimate of robot's heading and a sensor_msgs/NavSatFix message containing GPS data. Finally, it produces an odometry message in coordinates that is consistent with the robot's world frame. So, the working principle of the localization package used is an EKF fuses the odometry message coming from the odometry_publisher node and the Imu/data message and it establishes the transform between odom and base_link frames. The navsat_transform node integrates the GPS data and produces an odometry topic /odometry/gps. Finally, an EKF fuses the odometry_publisher output topic, the Imu/data one and the navsat_transform node output topic establishing transforms between map, odom, and base_link reference frames and it publishes an odometry/filtered topic that will be used in the mapping algorithm to complete the SLAM process.

Chapter 4 – The Mapping and The Perception Algorithm

Nowadays, mapping is one of the most active research fields in robotics. The environment, within a certain robot is moving, is represented on maps by means of geometric representations which distinguish and show the free space from that occupied by obstacles. There are different types of maps reported in the literature, broadly divided in metric and topological maps. The first ones can represent the geometric properties of the mapped environment, whereas the second ones describe the connectivity between different location. Examples of metric maps are occupancy grid maps (Gutmann et al. 2008) and landmark-based maps (Klein and Murray 2007), while examples of topological maps are the graph-based maps of Fraundorfer et al. 2007. In particular, the landmark-based maps identify and keep the 3D location of certain salient environment features. The advantages of these kind of techniques are that since only isolated landmarks are captured from the structure of the environment, the memory resources and the costs are relatively slow. For these reasons, these types of maps are not ideal for obstacle avoidance or path planning, since the lack of a landmark in a place does not imply that the space is free. Instead with regard the topological maps, these represents the environment as a list of significant places that are connected by arcs (like graph). This kind of world's representation simplified the problem of mapping large extension. The main drawback of these kind of solution is the high local error, thus, a global optimization it is necessary. Therefore, another limitation of topological maps is the lack of metric information, thus it is impossible to use these maps for guiding purposes [8]. In our case it is adapted to our system the mapping algorithm published in the GitHub PerceptionAndSlam_KTHFSD1718 package [29]. This algorithm is a landmark-based one. Although these kinds of algorithms, generally, are not used for obstacle avoidance and path planning, it is a convenient solution as the environment, within the car moves, can be considered as free space where the only obstacles are the traffic cones that delimits the racetrack. Of course, to build the map of the racetrack it is necessary to detect the traffic cones that delimits it. This process is called visual perception and it is performed by a turnkey

algorithm, called SSD mobilenet v1 from tensor flow models directory, based on artificial neural network that elaborates the zed stereo-camera images and allows to recognize the traffic cones along the path. The visual perception process is performed thanks to a Python ROS node which subscribes to zed camera image, zed camera point cloud and zed odometry topics. It takes a zed camera image, passes it through SSD mobilenet object detection to detects cones and then it creates a local map. Thus, this node publishes two topics: the processed images with bounding boxes and depth on cones and the local maps, which is a NumPy array containing the local coordinates of the cones in the 3D space and their color code.

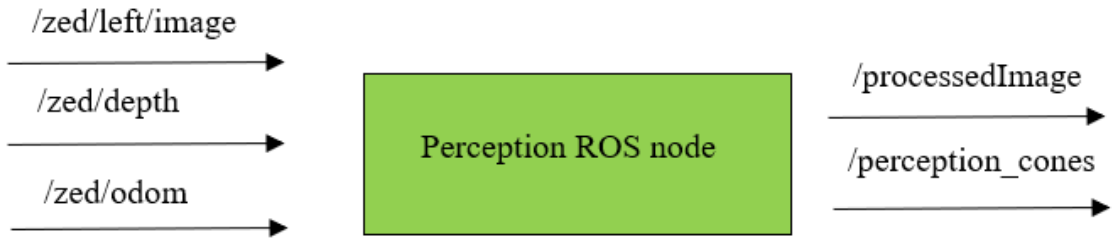


Figure 18: Perception ROS node scheme

4.1 Artificial Neural Network

Artificial Neural Networks (ANNs) are a programming tool inspired by the biological neural networks which constitute the animal brain. The beauty and the power of this tool is in the fact that the programmer does not tell the computer how to solve problems. But as if the machine as its own intelligence, it learns itself from a dataset the relationship between inputs and outputs. Thus, let us look at how a neural network is made and how it works. As first thing, we must define the basic concepts that constitute the ANN architecture. These concepts are the perceptron, the neuron, the activation function, the weights, and the bias. A perceptron is a binary classifier which takes several binary inputs and generates a single binary output.

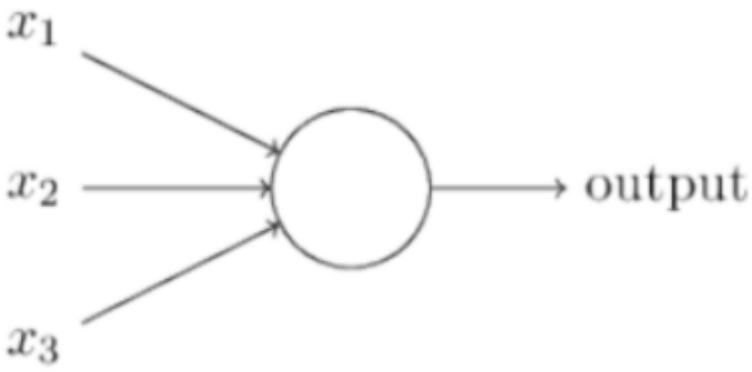


Figure 19: Perceptron [38]

Afterwards, the scientist Frank Rosenblatt introduces the concept of weight. The weight (θ) is nothing but a real number that, multiplies to an input value, does nothing but quantify the influence which that given input will have on the output. So, the output will depend on the value of the sum $\sum_j \theta_j x_j$ where θ_j represents the weight related to the input x_j. Formally:

$$output = \begin{cases} 0 \quad if \quad \sum_j \theta_j x_j \leq threshold \\ 1 \quad if \quad \sum_j \theta_j x_j > threshold \end{cases}$$

If b $\equiv -\ threshold$, we can rewrite the previous relationship as follows:

$$output = \begin{cases} 0 \quad if \quad \sum_j \theta_j x_j + b \leq 0 \\ 1 \quad if \quad \sum_j \theta_j x_j + b > 0 \end{cases}$$

The b value represents the perceptron bias and can be interpreted as the measure of how easy it is to look at perceptron to output 1. Another kind of neuron is represented by sigmoidal neuron. This, unlike perceptron, produces an output that is not either zero or one but is a value within a certain range. In fact, it is represented by the sigmoid function, which is similar to the step function but smoother:

$$\sigma = \frac{1}{1 + e^{-z}}$$

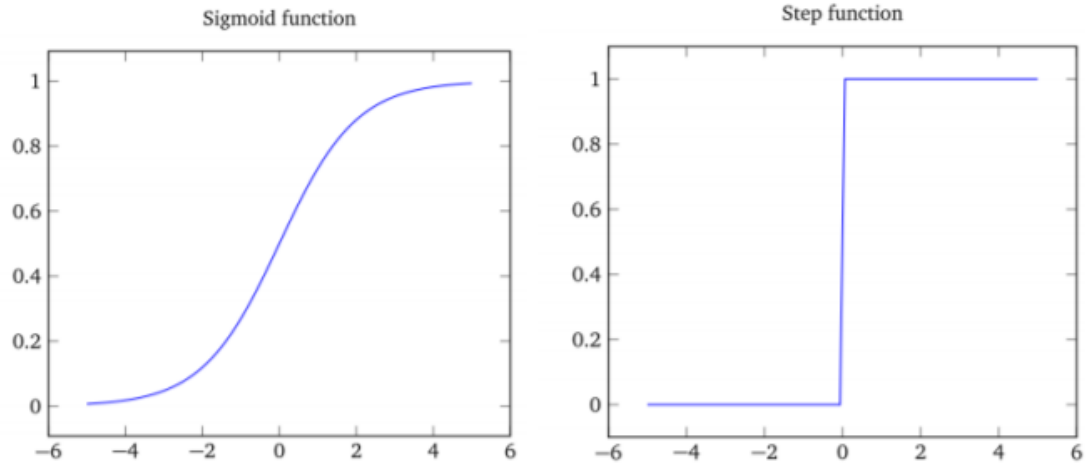


Figure 20: Sigmoid function and Step function comparison [38]

At this point, using these elements we can describe the generic architecture of an ANN. The structure of an ANN consists of three layers: the input layer, which is the leftmost one, the output layer, which is the rightmost one and the hidden layer, which is the middle one. Each layer is composed of a column of neurons and each layer's neuron relates to the neurons of the successive layer through arrows which represents the weights.

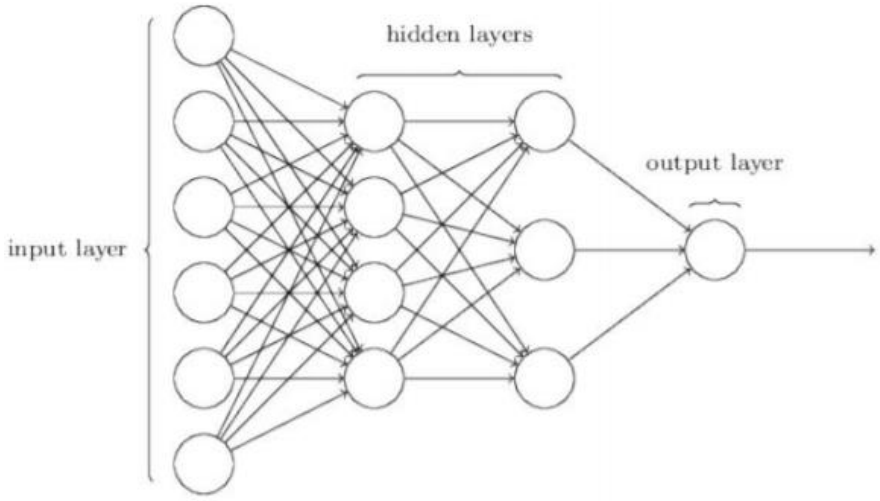


Figure 222: the architecture of an ANN [38]

As we can observe from the figure 21, the neural network, generally, holds more than one only hidden layer. When this happens, the ANN is called *deep* neural network.
As previously mentioned, inside the neuron, each input is multiplied for its own weight and then added together with the bias value. The value thus obtained is called activation input $z = f(x, \theta, b)$. Then the activation input pass through the so-called activation function whose result is propagated to the successive neuron.

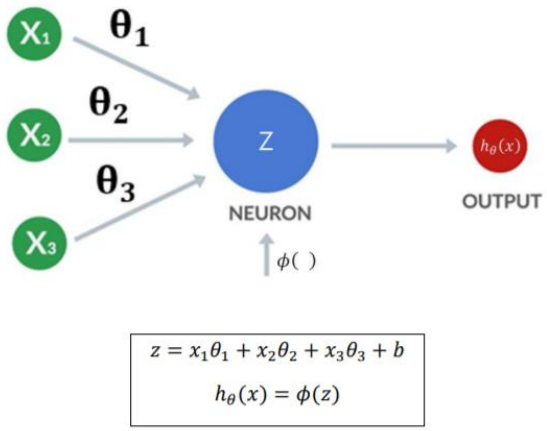
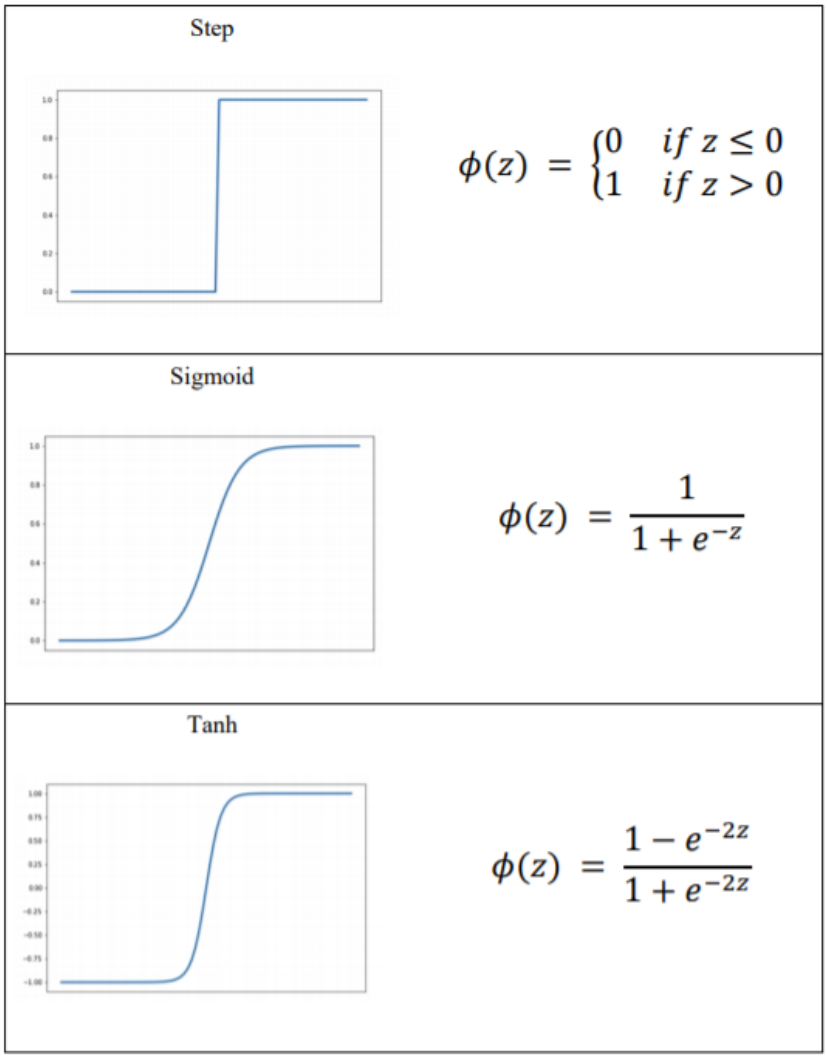


Figure 233: the activation function [38]

In the following figure are reported some examples of activation function:



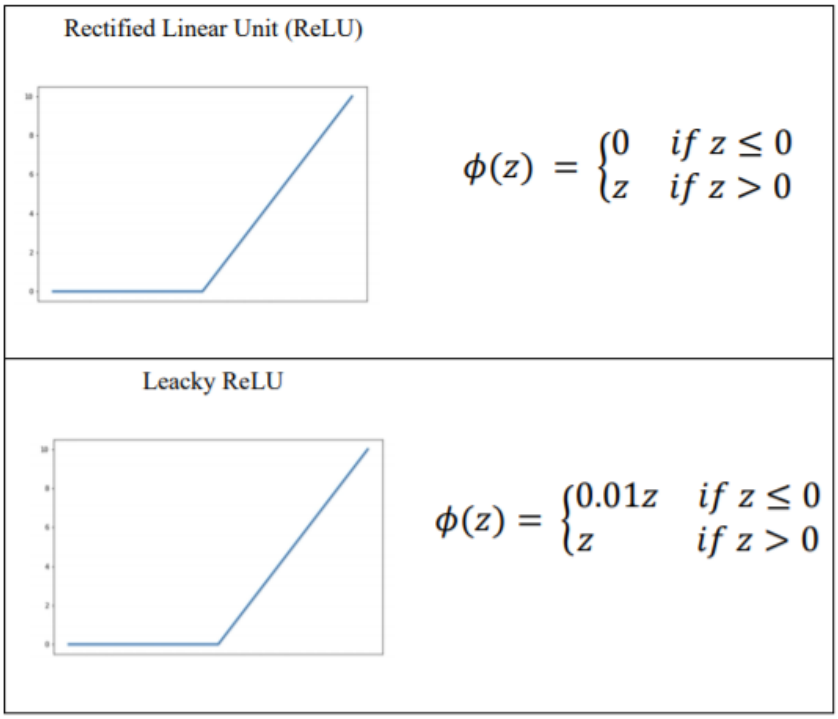


Figure 24: examples of activation function [38]

It is important to note that using the step function as opposed to the others represented in the figure 23, the quantitative information related to the activation input is lost. In fact, with a step function an activation input of 1 or an activation input of 10 000 will produce the same result.

To work properly a neural network must be trained. The training consists in giving to the ANN the inputs from a dataset and compare its hypothesis outputs $h_\theta(x)$ with the outputs y from the dataset. From this we can understand the importance of having a large dataset to obtain accurate hypothesis. The hypothesis function can be considered as it were a multivariate linear regression (a linear regression with multiple variable) which associate each feature of an example to the output. Thus, if we consider the number of features equal to n and the number of training examples equal to m, we can write the input (feature) of i^{th} training example as $x^{(i)}$ and the value of feature j of i^{th} training example as $x_j^{(i)}$. So, the hypothesis function $h_\theta(x)$ can be written as follow:

$$h_\theta(x) = [\theta_0 \quad \theta_1 \quad \dots \quad \theta_n] \begin{bmatrix} x_0 \\ x_1 \\ \dots \\ x_n \end{bmatrix} = \theta^T x$$

where:

$$x = \begin{bmatrix} x_0 \\ x_1 \\ \dots \\ x_n \end{bmatrix} \epsilon \mathbb{R}^{n+1} \qquad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \dots \\ \theta_n \end{bmatrix} \epsilon \mathbb{R}^{n+1}$$

are respectively the input vector and the weight vector.

The accuracy of the hypothesis function referring to the actual value of the output is given by the cost function or mean square error:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2$$

Since the value of $h_\theta(x_i)$ is a function of the weights θ and of the biases b we need to find those values that best approximate the predicted function to the real function, i.e., we want the cost function to be as small as possible. To do this it can be used the gradient descendent technique, whose aim is that to update the weights and bias values in order to minimize the cost function. The gradient descent algorithm repeats the following rule until convergence:

$$\theta_j := \theta_j - \alpha \frac{\partial J(\theta)}{\partial \theta_j}$$

where θ_j represents the $j\text{-}th$ weight; the same equation is used for bias value.

The parameter α is an hyperparameter, a value that is used to control the learning process. Substituting the expression of the h_θ in the case of multivariate linear regression we obtain that the gradient descendent algorithm takes the following form:

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^i) - y^i\right) x_j^i \quad for\ j := 0, \dots, n$$

The choice of the learning rate α is very important because:

- A very small value brings to slow convergence.
- A very large value can lead to divergence (bounce from one side of the curve to the other without ever reaching the minimum).

Since two features can have very different ranges of values, the speed of the gradient descent can vary and oscillate inefficiently when the variables are irregular. For this reason, it is necessary to apply the feature scaling technique using the following formula:

$$x_i := \frac{x_i - \min(x_i)}{\max(x_i) - \min(x_i)}$$

An alternative to this method is the mean normalization technique expressed by the following formula:

$$x_i := \frac{x_i - average(x_i)}{\max(x_i) - \min(x_i)}$$

[30]

But in an ANN, the task performed by the gradient descendent technique in linear regression it is performed by backpropagation algorithm. This process develops through the determination of the derivative of the cost function. To understand backpropagation let us consider we have a dataset and a training example (x, y). Forward propagation subsists in the calculation of:

$$\begin{aligned}
a^{(1)} &= x \\
z^{(2)} &= \Theta^{(1)} a^{(1)} \\
a^{(2)} &= g(z^{(2)}) \quad (\text{add } a_0^{(2)}) \\
z^{(3)} &= \Theta^{(2)} a^{(2)} \\
a^{(3)} &= g(z^{(3)}) \quad (\text{add } a_0^{(3)}) \\
z^{(4)} &= \Theta^{(3)} a^{(3)} \\
a^{(4)} &= h_\Theta(x) = g(z^{(4)})
\end{aligned}$$

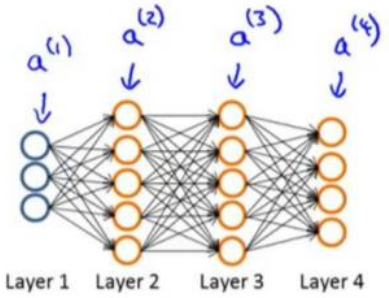


Figure 25: Forward propagation [38]

where $a^{(i)}$ is the activation values of $i\text{-}th$ layer, thus, the value of the activation function

applied to z, i.e., $g(z^{(i)})$. Finally, $a^{(4)}$ is the last activation value and corresponds with the hypothesis value $h_\Theta(x)$. Considering that $a_j^{(l)}$ is the activation value of node j in layer l, the intuition of backpropagation algorithm consists in the computation of error values of node j in layer l called $\delta_j^{(l)}$. Thus, if L is the last layer of the ANN, the error in the node j will be:

$$\delta_j^{(L)} = a_j^{(L)} - y_j$$

In vector form the equation will be:

$$\delta^{(L)} = a^{(L)} - y = h_\theta(x) - y$$

Now let us proceed with the calculation of $\delta^{(L-1)}$ back to $\delta^{(2)}$, hence the name backpropagation ($\delta^{(1)} = 0$, since $a^{(1)}$ corresponds to the features x).

The delta error in a generic layer l will be:

$$\delta^{(l)} = \left((\theta^{(l)})^T \delta^{(l+1)}\right) .* g'(z^{(l)})$$

where $g'\left(z^{(l)}\right) \cong a^{(l)} .* (1 - a^{(l)})$ is the derivative of the activation function g evaluated with the activation inputs $z^{(l)}$, $\theta^{(l)}$ is the weights matrix of l layer and the operator .* represents the element-wise multiplication. At this point we can determine the Δ matrix values at each iteration as follow:

$$\Delta_{ij}^{(l)} = \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$$

Finally, we define the matrix D as the derivative of the cost function $\frac{\partial J(\theta)}{\partial \theta_{ij}^{(l)}} = D_{ij}^{(l)}$:

$$D_{ij}^{(l)} := \begin{cases} \frac{1}{m}\left(\Delta_{ij}^{(l)} + \lambda \theta_{ij}^{(l)}\right) & if\ j \neq 0 \\ \frac{1}{m}\Delta_{ij}^{(l)} & if\ j = 0 \end{cases}$$

When j = 0 the value $D_{ij}^{(l)}$ coincide with the case of bias.

Lastly, let us define the problem of underfitting and overfitting. The terms overfitting and underfitting describe how much the model learns and generalizes the new data. A good machine learning model must be capable of generalizing new data well. With the term underfitting we refer to a model that has a high-cost function, i.e., with poor performance. With the term overfitting we refer to when the model of a network is too tied to the training

data and does not generalize about new data, so the resulting test cost function is higher [31].

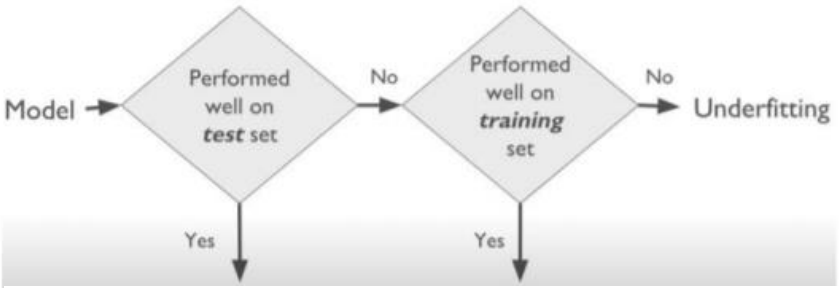


Figure 26: logical scheme which describes the underfitting and overfitting scenario [38]

4.2 Visual Perception: The Single Shot MultiBox Detector

The SSD (Single Shot MultiBox detector) is a deep network based object detector that does not resample pixels or features for bounding box hypothesis, but it is as accurate as approaches that do that. This results in a significant improvement in speed for high-accuracy detection. The fundamental improvement in speed comes from eliminating bounding box proposals and the subsequent pixel or feature resampling stage. Thus, using SSD, we only need to take one single shot to detect multiple objects within the image, while regional proposal network (RPN) based approaches such as R-CNN series that need two shots, one for generating region proposals, one for detecting the object of each proposal. Thus, SSD is much faster compared with two-shot RPN-based approaches. The SSD approach is based on a feed-forward convolutional network that produces a fixed-size collection of bounding boxes and scores for the presence of object class instances in those boxes, followed by a non-maximum suppression step to produce the final detections. The early network layers are based on a standard architecture used for high quality image classification. To these layers are added an auxiliary structure to the network to produce detections with the following key features:

- *Multi-scale feature maps for detections*: at the end of the truncated base network there are some convolutional feature layers. These layers decrease in size progressively and

allow predictions of detections at multiple scales. The convolutional model for predicting detections is different for each feature layer.

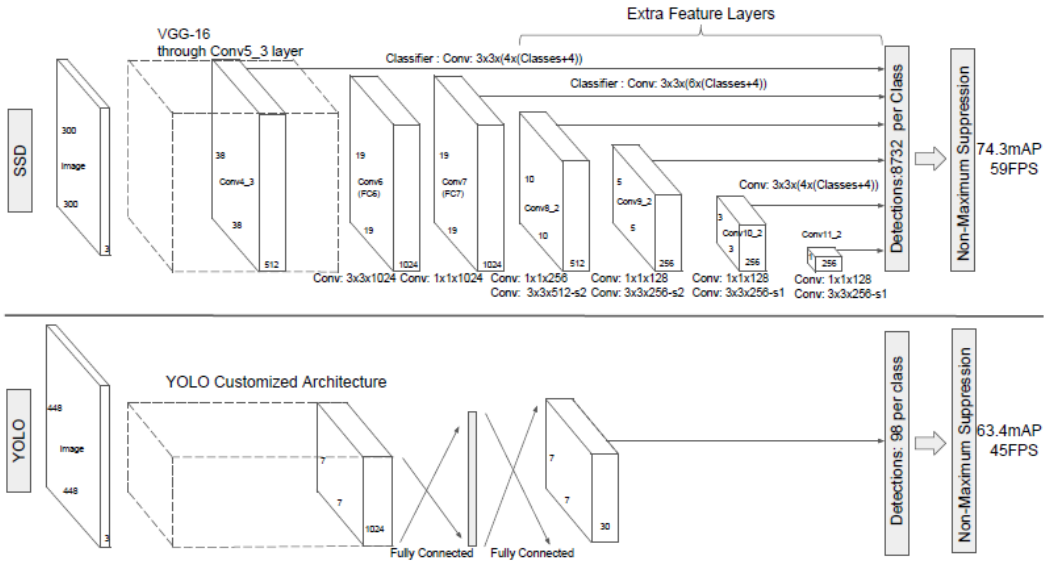


Figure 27: A comparison between two single shot detection models: SSD and YOLO [32]

- *Convolutional predictors for detection*: each added feature layer can produce a fixed set of detections predictions using a set of convolutional filters. For instance, if we have a feature layer of size m*n with p channels, the basic element to predict parameters of a potential detection is a 3*3*p small kernel that produces either a score for a category, or a shape offset relative to the default box coordinates.

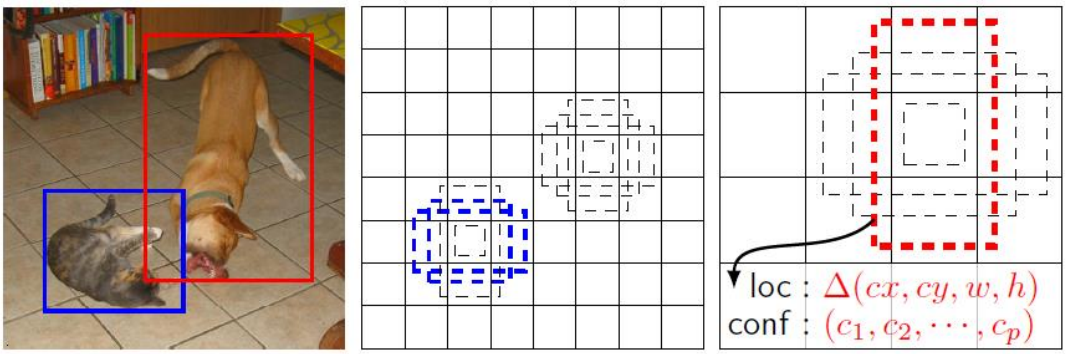


Figure 28: SSD framework [32]

- *Default Boxes and aspect ratios*: A set of default bounding boxes are associated with each feature map cell, for multiple feature maps at the top of the network. The default boxes tile the feature map in a convolutional manner, so that the position of each box relative to its corresponding cell is fixed. At each feature map cell, it is predicted the offsets relative to the default box shapes in the cell, as well as the per-class scores that indicate the presence of a class instance in each of those boxes.

The cost function consists of two terms: L_{conf} and L_{loc} where N is the matched default boxes:

$$L(x, c, l, g) = \frac{1}{N}\left(L_{config}(x, c) + \alpha L_{loc}(x, l, g)\right)$$

$$L_{loc}(x, l, g) = \sum_{i \in Pos}^{N} \sum_{m \in \{cx, cy, w, h\}} x_{ij}^{k} \text{smooth}_{\text{L1}}(l_i^m - \hat{g}_j^m)$$

$$\hat{g}_j^{cx} = (g_j^{cx} - d_i^{cx})/d_i^w \qquad \hat{g}_j^{cy} = (g_j^{cy} - d_i^{cy})/d_i^h$$

$$\hat{g}_j^w = \log\left(\frac{g_j^w}{d_i^w}\right) \qquad \hat{g}_j^h = \log\left(\frac{g_j^h}{d_i^h}\right)$$

L_{loc} is the localization loss or rather the $smooth_{L1}$ cost between the predicted box (l) and the ground-truth box (g) parameters. These parameters include the offsets for the center point (cx, cy), width (w) and height (h) of the bounding box. The confidence loss is the normalized exponential cost function over multiple classes confidences (c) and the weight term α is set to 1 by cross validation.

$$L_{conf}(x,c) = -\sum_{i\in Pos}^{N} x_{ij}^{p} log(\hat{c}_i^p) - \sum_{i\in Neg} log(\hat{c}_i^0) \quad \text{where} \quad \hat{c}_i^p = \frac{\exp(c_i^p)}{\sum_p \exp(c_i^p)}$$

Let us consider having m feature maps for prediction, we can calculate S_k for the k-th feature map. S_{min} is 0.2, S_{max} is 0.9. That means the scale at the lowest layer is 0.2 and the scale at the highest layer is 0.9. All layers in between are regularly spaced. Thus, the equation to compute s_k is:

$$s_k = s_{\min} + \frac{s_{\max} - s_{\min}}{m-1}(k-1), \quad k \in [1,m]$$

[32]

4.3 The Mapping Algorithm Description

As explained previously, the mapping is a characteristic process of the SLAM, where the cones detected by the zed stereo camera are plotted on a map which represents the whole racetrack. This process is fundamental for the car navigation, thus, for the path-planning and actuation. In particular, during the first lap of the race, the racetrack map is unknown. In this phase, the trajectory planning needs to be short-term or reactive to the cones detected immediately. Once the first lap is traveled, the racetrack map is built. Thus, the trajectory planning become long term or pro-active. The mapping process has been implemented through two ROS nodes written in Python 3.7: the reactive mapping node and the global mapping node, adapting the ones presented in the GitHub PerceptionAndSlam_KTHFSDV1718 package.

4.3.1 The Reactive Mapping Node

The reactive mapping node is a Python ROS node which take as input (subscribes) the /perception_cones topic and produces as output (publishes) the /reactive_cones topic. The

/perception_cones topic is an extended one-dimensional array produced by the perception ROS node (see the figure 19) with a size of 4N, where N is the number of the cones detected in the processed image. Every cone is characterized by a data packet of 4 values whose format is [x, y, z, color] where x, y, z are the three-dimensional coordinates of detected cone in the camera reference frame and color is the numeric code that identifies the detected cone's color. Instead, regarding the /reactive_cones topic, this is an extended one-dimensional array with a size of multiples of 4 too. But in this case, for every cone detected there is a 4 values data packet with the following format [centroid_x, centroid_y, variance, color], where the first two values are centroid of cone position, the third value is the position variance and finally the fourth one is the cone color code.

Figure 29: The Reactive Mapping Node scheme

The reactive mapping node consists of two functions or methods: the callback_perception() and the cluster_frames() one. Let us consider that the camera is detecting N cones every frame. The callback_perception() method integrates data from three successive frames and creates a 3N×4 matrix that will be the input of the cluster_frames() function which is called within the callback_perception() one. The cluster_frames() method assigns a cluster identification number to every cone. If the number of cones in a cluster is major than the 65% of the maximum number of times that a cone can be detected (each cone can be detected maximum 3 times because we are analyzing 3 frames at a time, so each cone should be seen at least 2 times), they are legitimated, and their positions are used to compute centroid (the mean value) and the variance in cone positions. At the end, the /reactive_cones topic is published [29]. To describe more in detail the reactive mapping algorithm the following two flowchart are represented. The first one represents the callback_perception() method while the second one, that has been divided into two parts, represents the cluster_frames() function.

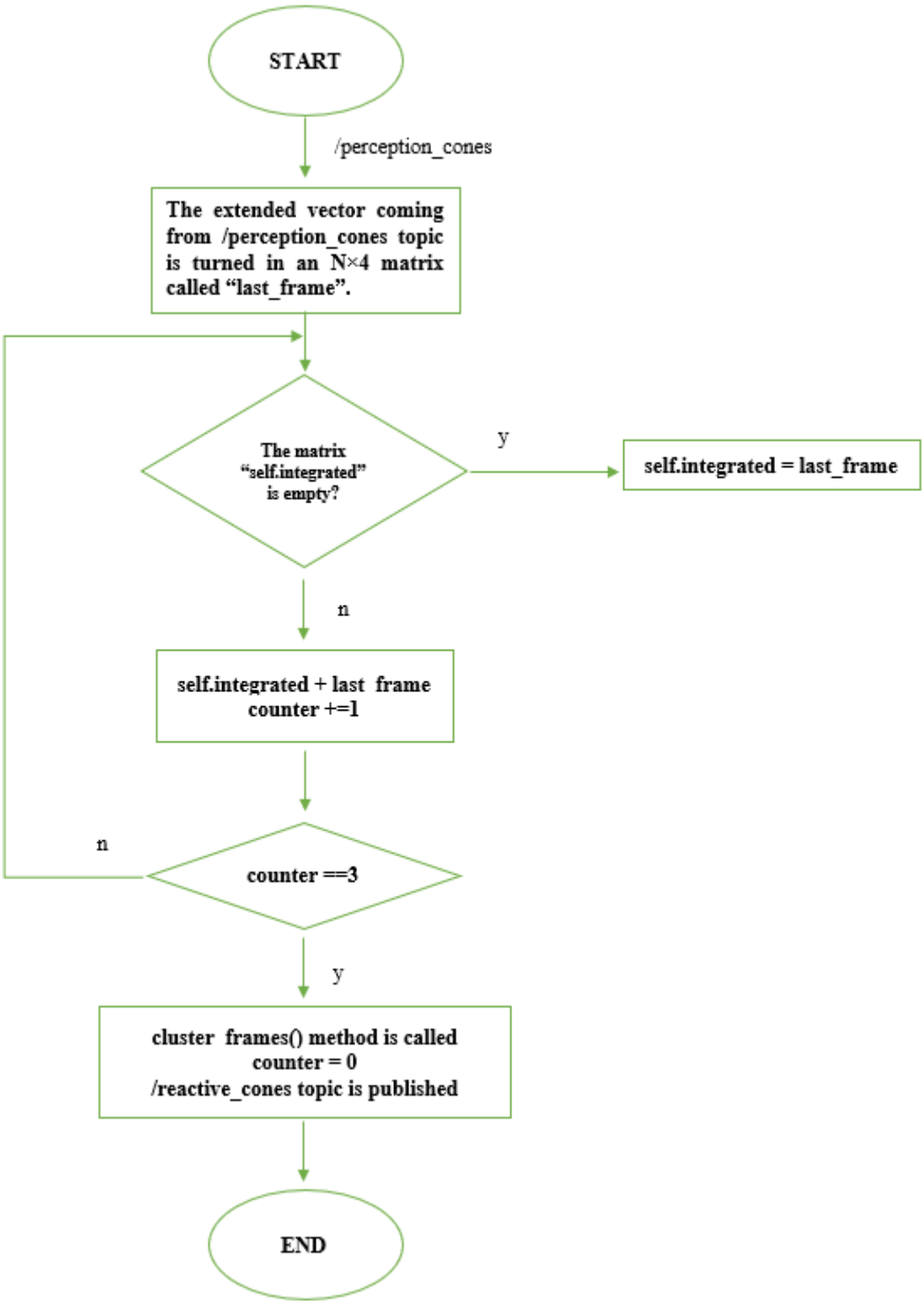


Figure 30: callback_perception() method flowchart, N.B. self.integrated + last_frame means that if the first is a 2x4 matrix and the second one is 1x4 the results will be the matrix 3x4 with the element of the previous ones.

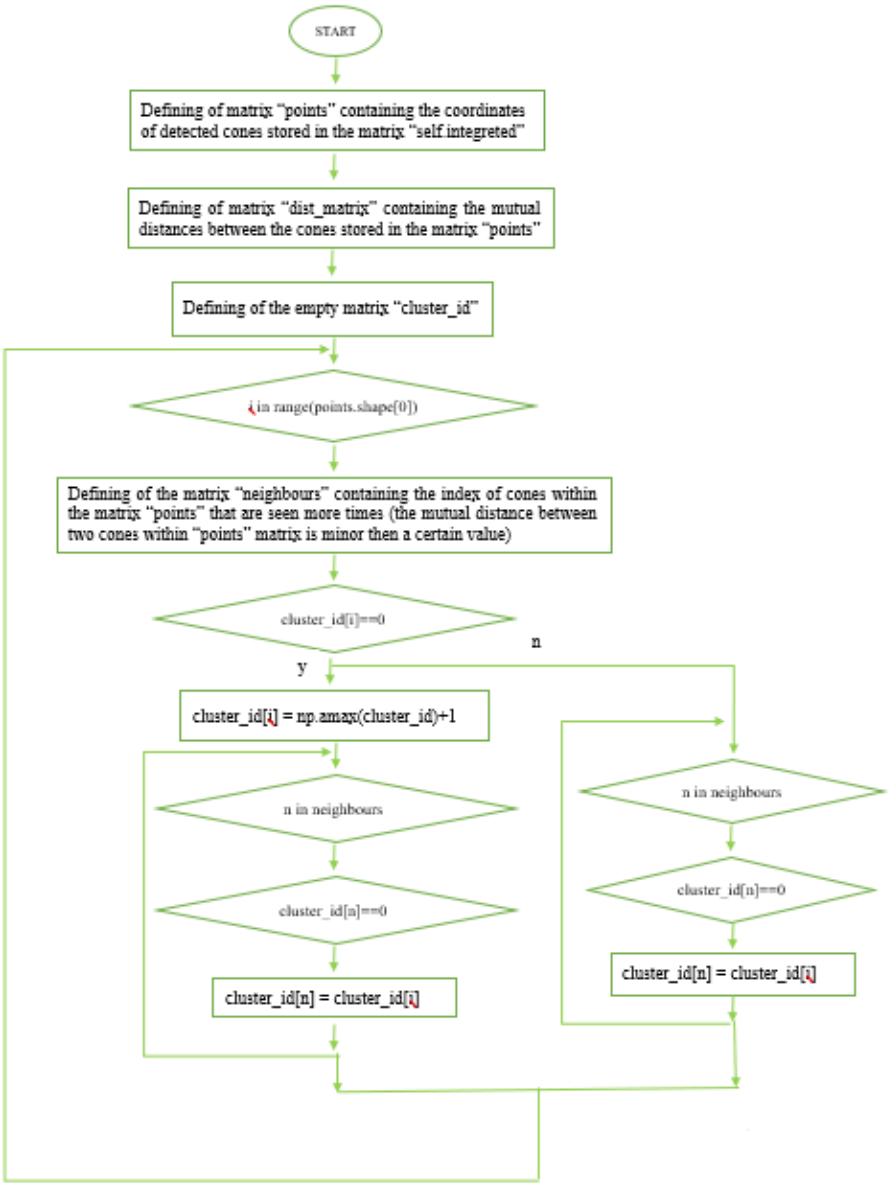


Figure 32: the cluster_frames() method flowchart part 1

Therefore, at the end of this for loop (the outermost one) we will have assigned a certain cluster_id to each cone and the cones whose mutual distance is less than certain value will have the same cluster_id. At this point we create a dictionary called cluster_dict where each unique value of the cluster_id is associated to the number of times it is present in the cluster_id vector. After the dictionary definition the cluster_frames() method develops as follow:

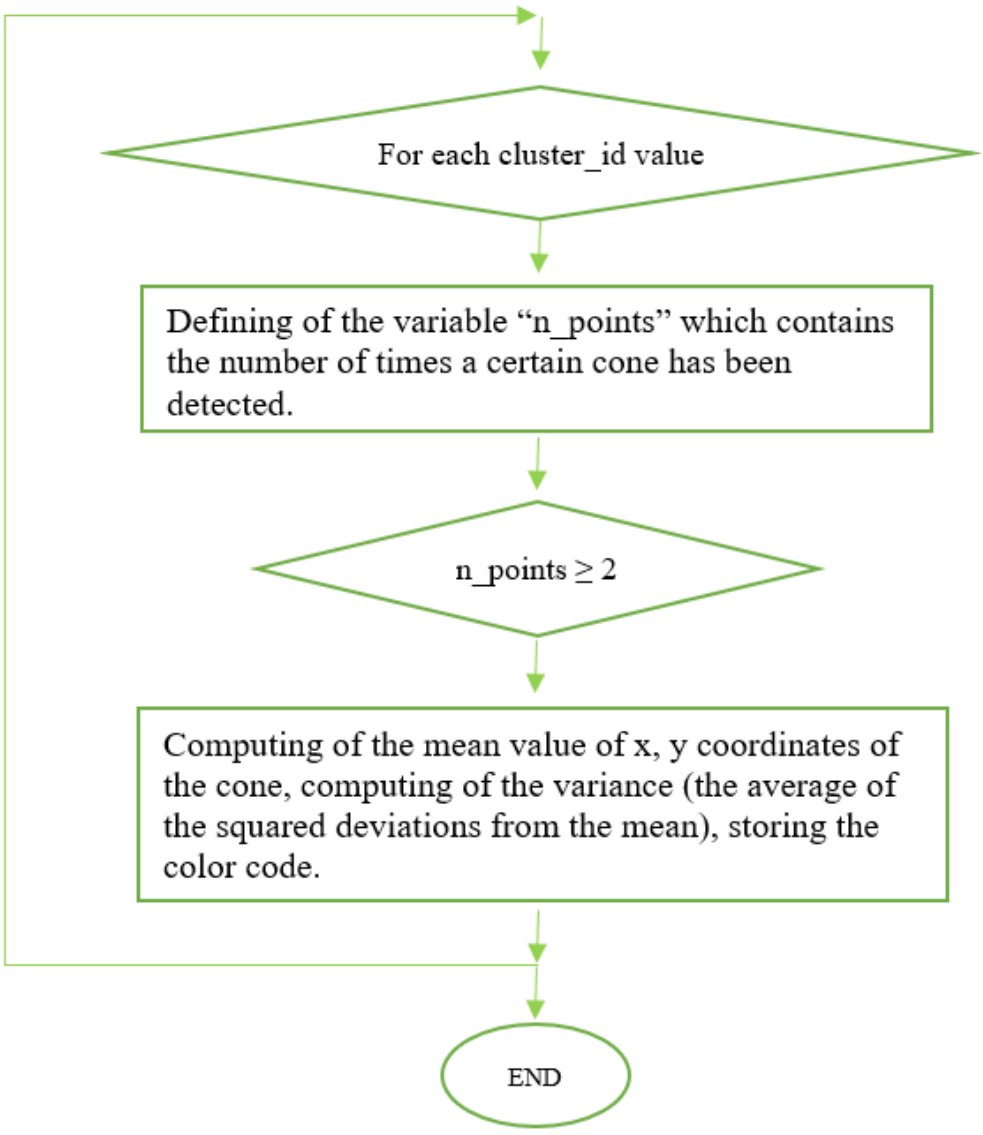


Figure 31: the cluster_frames() method flowchart part 2

4.3.2 The Global Mapping Node

The global mapping node is a Python ROS node which has the task to create a global map of the racetrack as accurate as possible. This because all the long-term decisions of the car are based on this map. Hence, if in the global map there are cones that should not be there or some cones are not stored in the map, the car could stop prematurely, it could divert toward a not real cone or in worst of hypothesis it could crash against a cone which is not marked in the map. The origin of the global map created by this ROS node overlaps with the initial position of the car. This ROS node, in fact, receives as inputs two topic: the /odometry/filtered topic coming from the localization package containing the position and the orientation of the vehicle with respect its initial state and the /reactive_cones which is the topic publishes by the reactive mapping node. Instead the global mapping output is constituted by the /global_map_markers topic which is an extended one-dimensional vector with the following format: [x, y, color, covariance, hits, inFOV, id] where x and y are the position coordinates in the global reference frame, color is the cone's color code, covariance is the covariance related to the position, hits is the number of time that a cone has been detected, inFOV is a flag that indicates if a cone is in the camera field of view and finally id is an identification number that characterized each cone.

Figure 32: The global mapping node scheme

This node is characterized by three functions: the callback_odom(), the callback_reactive() and the update_cone_db(). The first method needs to compute from the /odometry/filtered topic the translation and the rotation matrices that needs to pass from the reactive reference frame to the global one. The second method needs to take from the /reactive_cones topic the

pose and the color of each cone detected by the camera expressed in the reactive reference frame. And lastly, the third method is the update_cone_db(). This function is called with a fixed frequency the so-called update frequency that is a design parameter. The logic behind this method is that if a certain cone is seen more times during the traveling along the circuit, the covariance related to its position decreases, on the other hand, if a car, performing more laps of the same racetrack, does not see a certain cone frequently, the cone's position covariance increase till the covariance reaches a minimum value and the cone is removed from the map [29]. Thus, we can represent the global mapping algorithm with the following flowchart.

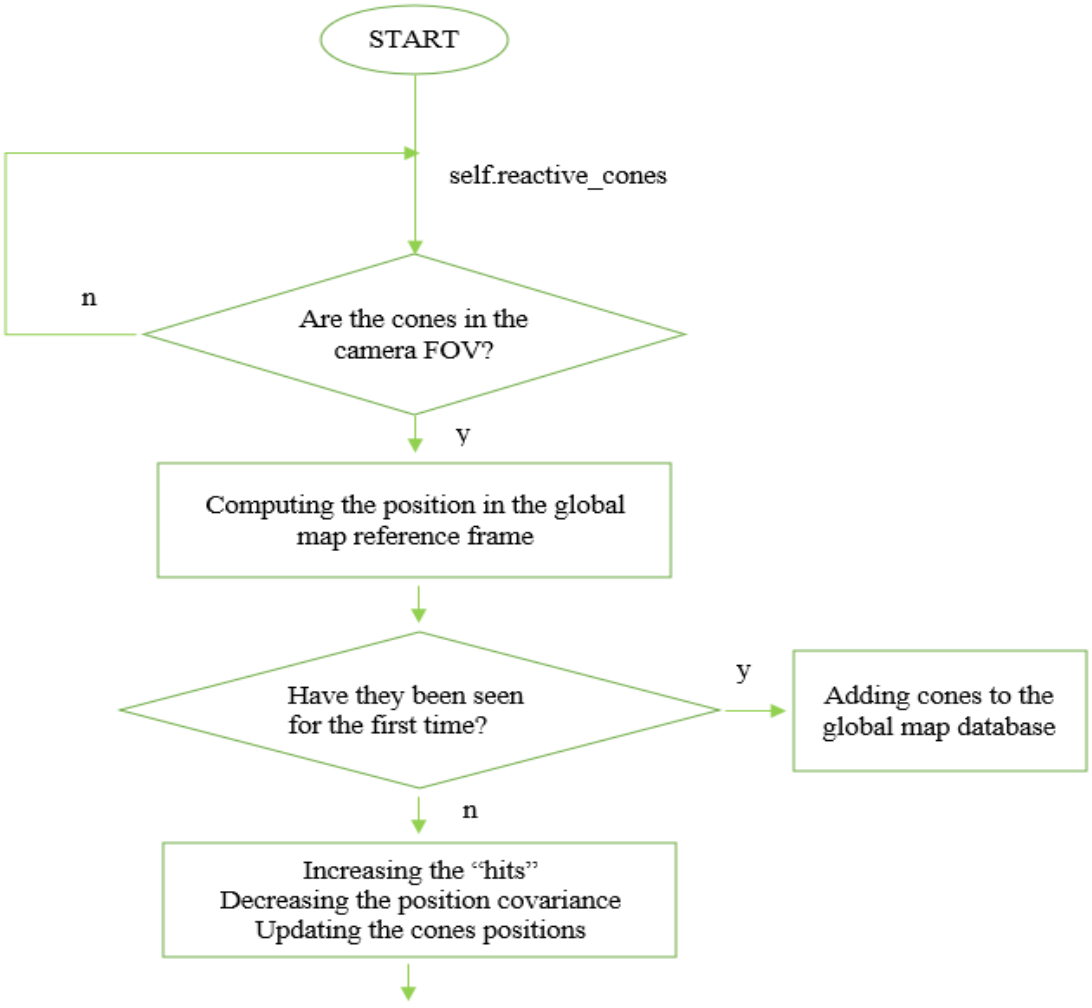


Figure 33: the global mapping algorithm flowchart (part_1)

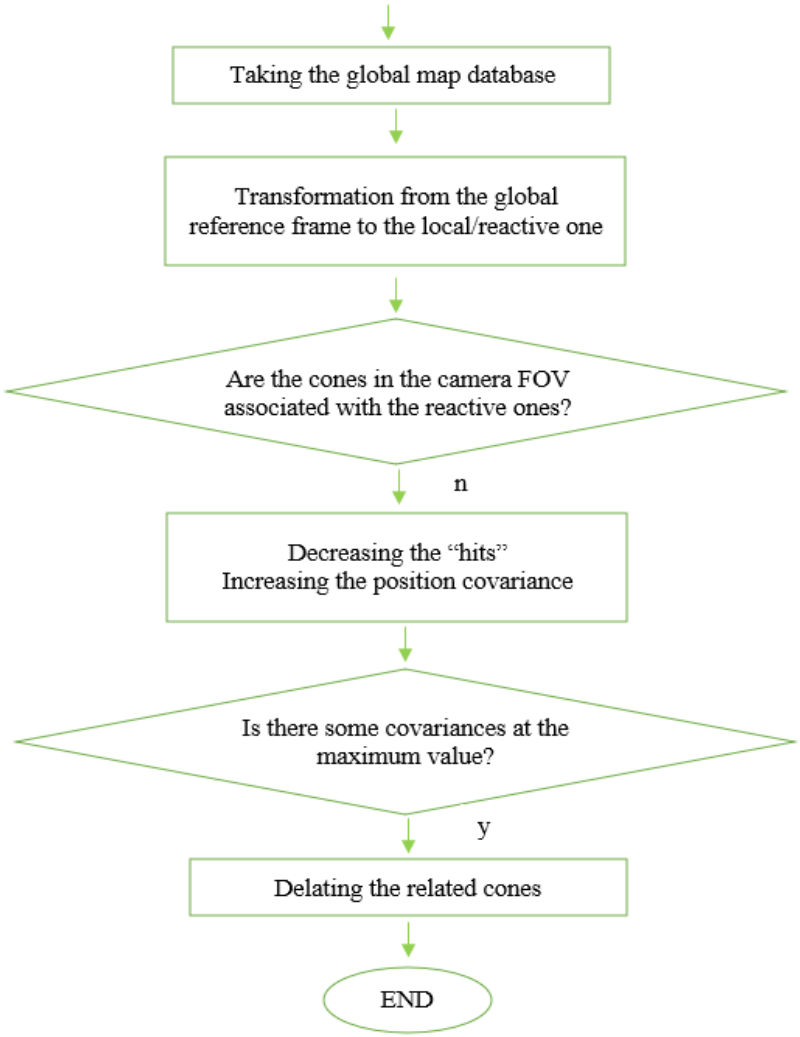


Figure 34: the global mapping algorithm flowchart (part_2)

Thus, let us describe the flowchart above. First of all, the /reactive_cones topic values are converted by the callback_reactive() function in the so-called matrix self.reactive_cones. At this point only the cones which are in the camera field of view are processed by the mapping algorithm. In fact, the stereo camera has a 120 degrees field of view with a depth range of 0.3-25m, but we can reduce this one delating the cones that are out of the FOV chosen by us. This is done for optimize the mapping algorithm, in fact this way you are doing nothing but reduce the number of features in the map (see the chapter 1, the section 1.7 about the map computational complexity). All this is possible as we know the width of the path and the

distance that exists between two successive cones. Moreover, setting a maximum distance of field of view of 4 m instead of 25 m and minimum one of 1.7 m instead of 0.3 m, the algorithm it can process a pair of cones a time reducing the time complexity of the algorithm. After that, thanks to the transformation matrix computed from the callback_odom() method, the position of detected cones it transformed from the camera reference system to the global one. At this point, if the cones detected are seen for the first time, they are added to the global map database, instead the "hits" value is increased, the covariance is decreased, and the position is updated. The cone position is updated using the exponential moving average. This kind of average is a first-order infinite impulse response filter that applies weighting factors which decrease exponentially. An infinite impulse response is a property applying to a lot of linear time-invariant system which are characterized by having an impulse response h(t) which never becomes zero, but it continues to draw near indefinitely. This means that the cone position is updated giving a major weight to the older positions than the new ones. This because the mapping process is characterized by the error accumulation problem (see the chapter 1, the section 1.6 about the error accumulation problem). The EMA for a series Y could be calculated recursively as follow:

$$S_t = \begin{cases} Y_1, & t = 1 \\ \alpha Y_t + (1-\alpha) S_{t-1}, & t > 1 \end{cases}$$

where:

- The coefficient α represents the degree of weighting decrease, a constant smoothing factor between 0 and 1. A higher α discounts older observations faster.
- Y_t is the value at a time t.
- S_t is the value of the EMA at any time t.

Finally, it is important to speak about S_1 initialization, S1 initialization effect on the resultant moving average depends on α; smaller α values make the choice of S1 relatively more important than larger α values, since a higher α discounts older observation faster. In the previously equation S_1 has been initialized as $S_1 = Y_1$ [33]. More in detail, in our case, α is equal to 0.9. Thus, the updated position is computed as follow:

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

At this point, the part of code concerning the ability of forgetting the cones begins. All the cones stored in the global map database are transformed in the local reference frame and if they are not in the camera FOV the correspondent hit decrease and consequently the covariance increase. Finally, the covariance value is checked and if it is lower than the threshold value the cone is delated. In particular, the covariance decreases linearly with the number of hits as shown in the following equation:

$$cov = cov_{Max} - \Delta \times hit$$

where cov_{Max} is the covariance maximum value, hit is the number of times a cone has been detected and $\Delta = cov_{Max} \frac{hit_{Max}-1}{hit_{Max}}$ where hit_{Max} is the maximum number of hits.

Chapter 5 – The Mapping Algorithm Testing

After the choice and the implementation of the algorithm it was necessary to test it. The mapping algorithm testing was carried out through simulation, in particular with EUFS simulator and through on-track testing, even if this last one is still incomplete.

5.1 The Computer Simulation

A computer simulation is a tool to virtually investigate the behavior of the system under study. To do this, a real-life or hypothetical situation is modeled on a computer so that it can be studied to see how the system works. Thus, by changing variables in the simulation, predictions may be made about the behavior of the system [34]. More in detail, the simulation in robotics should satisfy the following requirements:

- The simulation should be real-time, this means that it interfaces with the autonomous software stack.
- The sensors representation should be as realistic as possible.
- The simulated system should be physically truthful as good as possible.

The main benefits of a computer simulation are first the fact that the test is performed at home, thus, there is a no time-consuming preparation, and no ready-to-race vehicle is needed. Another benefit is represented to the fact that a computer simulation reduces the critical issues on real vehicle. And finally, the simulated scenarios are always available while in real life scenarios we are extremely conditioned by whether effects and surroundings [27].

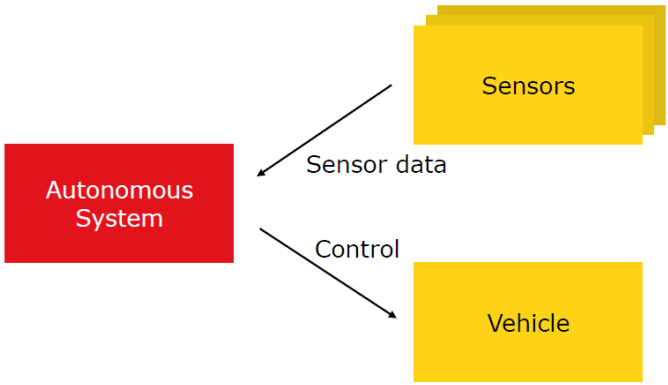


Figure 35: real-life architecture [27]

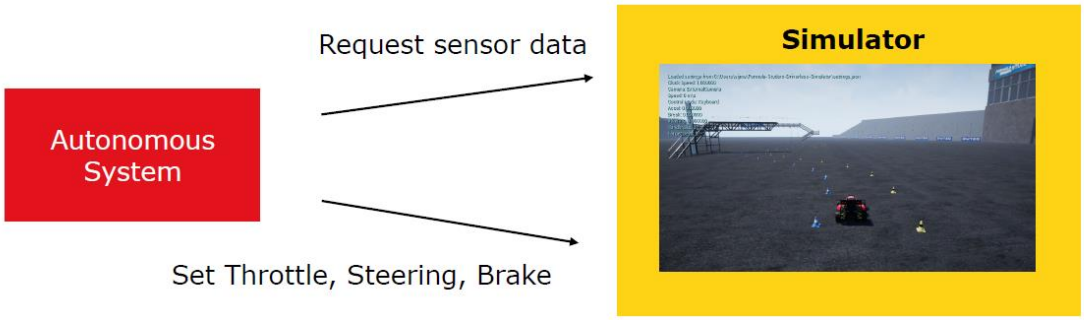


Figure 36: simulation architecture [27]

The simulator used to test the mapping algorithm is the EUFS one. Which is a simulation tool implemented in Gazebo by Edinburgh University Formula Student team. It allows to select five different track scenarios. A small track and a big track which are fixed track used for rapid prototyping and consistency. The ski pad which is a preconfigured track as per the formula student rules. The acceleration track which is a straight circuit preconfigured as per the formula student rules. And finally, a randomly generated track. The vehicle model is implemented using the gazebo_race_car model which allow us to make a custom vehicle

model. This is very important when we want to simulate the dynamical behavior of the car because this is strongly influenced by the vehicle model. The sensors with which the car is equipped by default are the following:

- Velodyne VLP16 LIDAR.
- ZED stereo camera.
- IMU.
- GPS.
- Wheel odometry.

If additional sensors are needed it is possible to use the ones available through the ros-melodic-robotnik-sensor package [35].

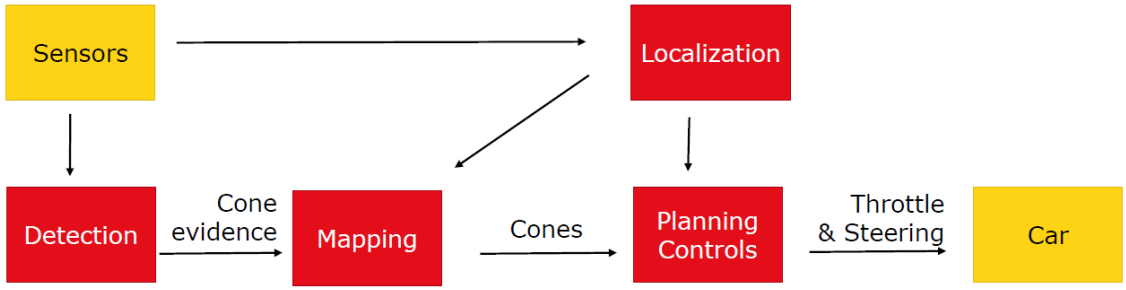


Figure 37: The EUFS simulator architecture [27]

As we can observe from the figure 36, the simulated parts when we use EUFS simulator are the sensors, the car, and the environment (the yellow boxes) while all the autonomous software stuff is the same one that is used during a real experience (the red boxes) [27]. But in our case, since the IMU sensor model is not representative of which we equipped the car (it produces different raw data with respect the IMU used), we cannot simulate the Localization node. Thus, we can test only the mapping without the localization, and to do this the ground truth odometry provided by the simulation it is used. The mapping algorithm has been tested within three kind of circuits provided by the simulation: the acceleration racetrack, the small racetrack and the big one.

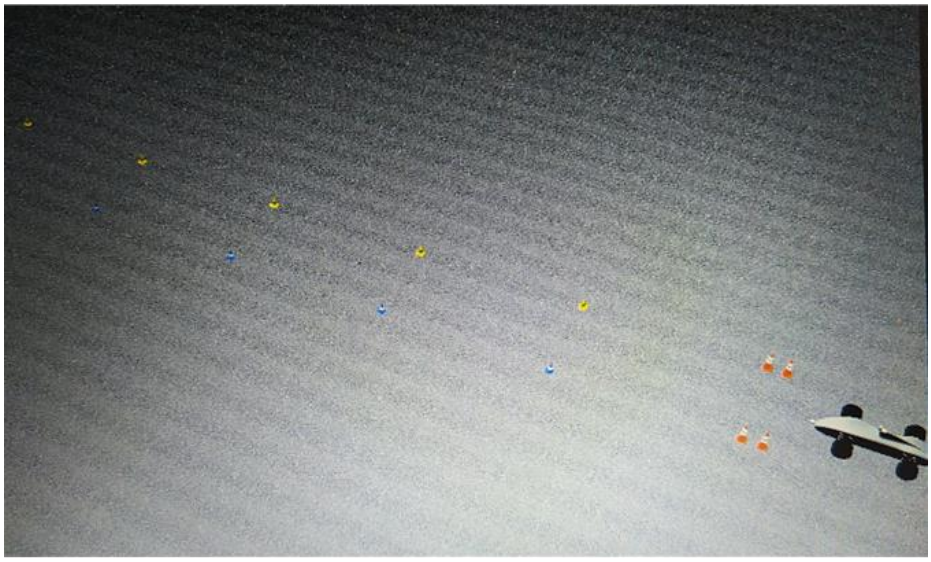


Figure 38: the acceleration racetrack

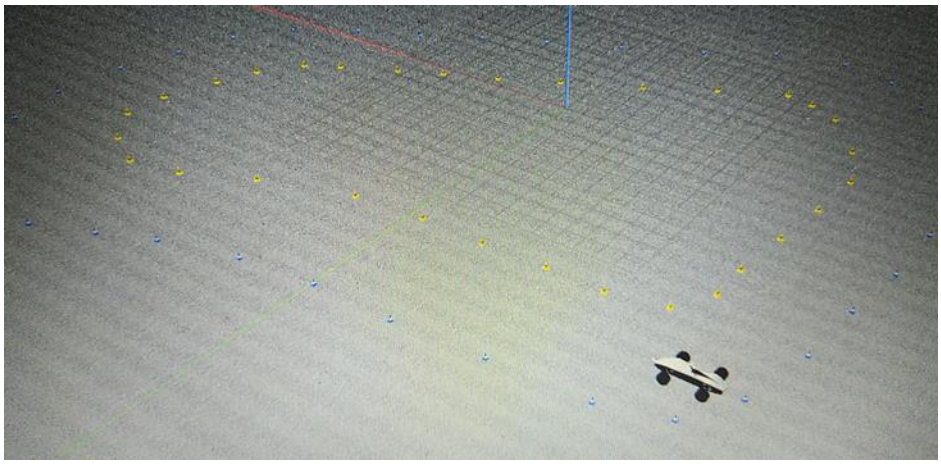


Figure 39: the small racetrack

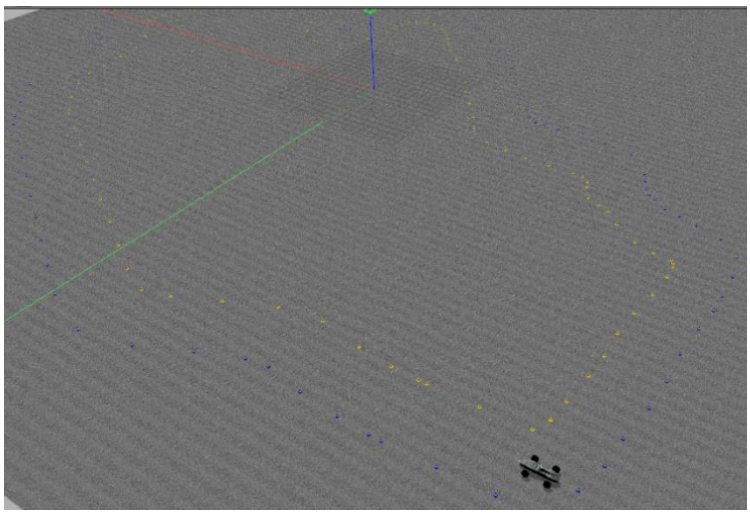


Figure 40: the big racetrack

With the acceleration circuit we can test the behavior of the algorithm in a velocity range from 2 m/s to 10 m/s and we can also set the update frequency in relation with the car speed. With the small and big tracks simulation, instead, we can test the behavior of the mapping algorithm in a closed circuit and hence we can make some consideration on how the car navigation should be to obtain a good map of the environment. In the future if we will reach to implement in the simulation also the localization node thanks to a suitable Gazebo IMU model, we will test the complete SLAM process and in particular the ability of the slam algorithm to map a big, closed racetrack where the error accumulation is one of the biggest issues. The first result of the simulation is the following one:

Figure 41: acceleration racetrack, car speed 2 m/s, update frequency 10Hz (part 1)

The picture above shows us the map obtained when the car travelled along the acceleration racetrack with a maximum speed of 2 m/s and the mapping algorithm worked with an update frequency of 10 Hz. The number of traffic cones that constitutes the acceleration circuit are 46 cones, while in the map are represented 176 cones. This problem remains increasing the

car speed or modifying the update frequency. So, what is the cause of this error? It depends on the mapping algorithm or it depends on the simulation?
If we observe the simulation results, we can see that the cones detected are not distributed homogeneously along the path, but they form small homogeneous groups which are repeated throughout the path as highlighted in the following picture:

Figure 42: acceleration racetrack, car speed 2 m/s, update frequency 10Hz (part 2)

So, we can understand that this is a tickrate error. This means that the odometry messages and the perception_cones are not synchronized because of the big computational effort during the processing of the camera frames. To solve this problem, we have created a new ROS node called synchronizer that use the ROS filter called the AproximateTimesynchronizer which uses an adaptative algorithm to match messages based on their timestamp. Thus, we have implemented a Python script that tell us if the perception_cones messages and odometry ones are synchronized and if not, it matches them. The Python script is the following one:

```python
import numpy as np
from numpy import array
from std_msgs.msg import Float64MultiArray
from rospy_tutorials.msg import Floats
from rospy.numpy_msg import numpy_msg
from sensor_msgs.msg import Image, CameraInfo
from nav_msgs.msg import Odometry

rospy.init_node('sync_test')

def gotimage(perc_sub, odom_sub):
    #assert image.header.stamp == camerainfo.header.stamp
    print ("got an Image and Odometry")
    last_odometry = odom_sub
    last_perc = perc_sub
    pub_perc = rospy.Publisher('/perc', numpy_msg(Floats), queue_size=10)
    pub_perc.publish(last_perc)
    pub_odom = rospy.Publisher('/odom_filt', Odometry, queue_size=10)
    pub_odom.publish(last_odometry)

perc_sub = message_filters.Subscriber('/perception_cones', numpy_msg(Floats))
odom_sub = message_filters.Subscriber("/ground_truth/state_raw", Odometry)
#odom_sub = message_filters.Subscriber("/robot_control/odom", Odometry)


ats = message_filters.ApproximateTimeSynchronizer([perc_sub, odom_sub], queue_size=10, slop=15, allow_headerless=True)
ats.registerCallback(gotimage)
rospy.spin()
```

Figure 43: the ApproximateTimesynchronizer pyhton script

So, this node publishes two topic the /perc and the /odom_filt one. These two topics are the filtered topic, hence the synchronized topic, that correspond to the /perception_cones and /ground_truth/state_raw topics.

In the following are shown the results obtained using the AproximateTimesynchronizer filter when the racecar travel along the acceleration circuit with different speeds and different update frequencies.

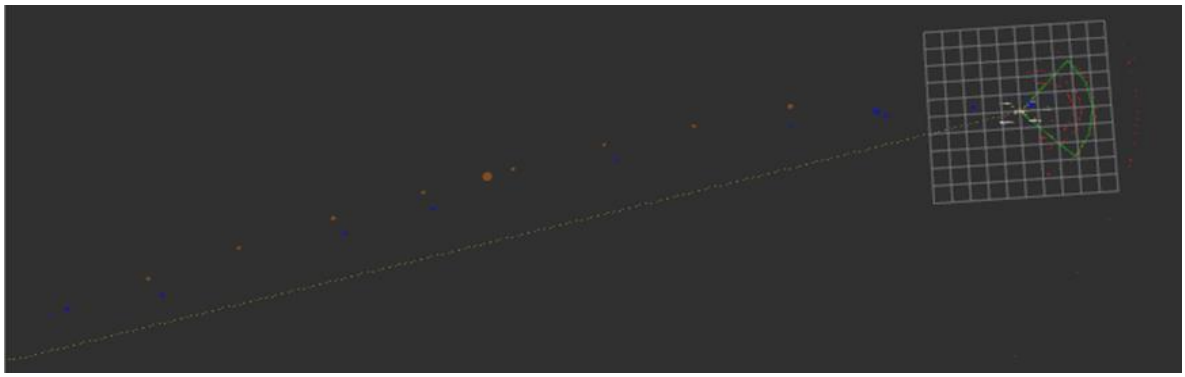


Figure 44: acceleration circuit simulation, with car speed of 2m/s and the update frequency of 10Hz

The figure 44 shows us the acceleration circuit mapped when the car traversed it with a maximum speed of 2 m/s and with a mapping update frequency of 10Hz. The number of cones detected are 23/46.

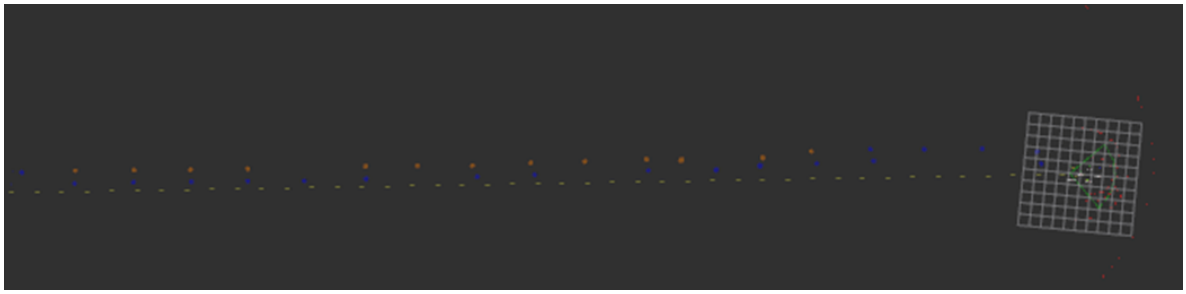


Figure 45: acceleration circuit simulation, with car speed of 5m/s and the update frequency of 10Hz

The figure 45 shows us the acceleration circuit mapped when the car traversed it with a maximum speed of 5 m/s and with a mapping update frequency of 10Hz. The number of cones detected are 42/46.

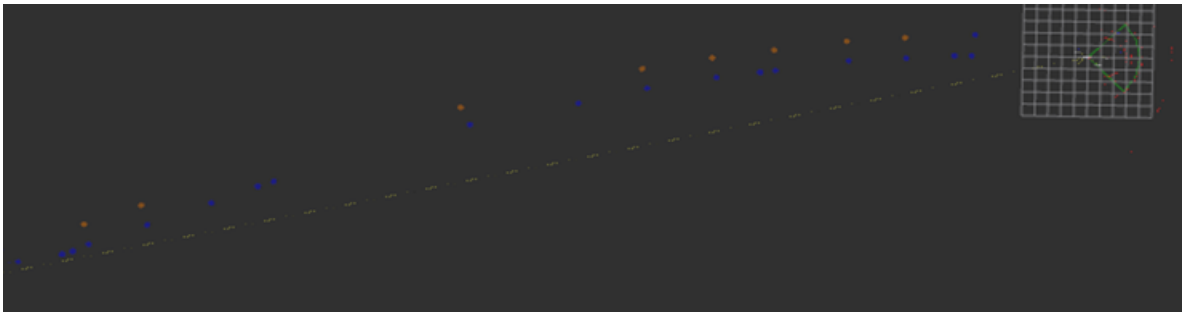


Figure 46: acceleration circuit simulation, with car speed of 10m/s and the update frequency of 10Hz

The figure 46 shows us the acceleration circuit mapped when the car traversed it with a maximum speed of 10 m/s and with a mapping update frequency of 10Hz. The number of cones detected are 32/46.

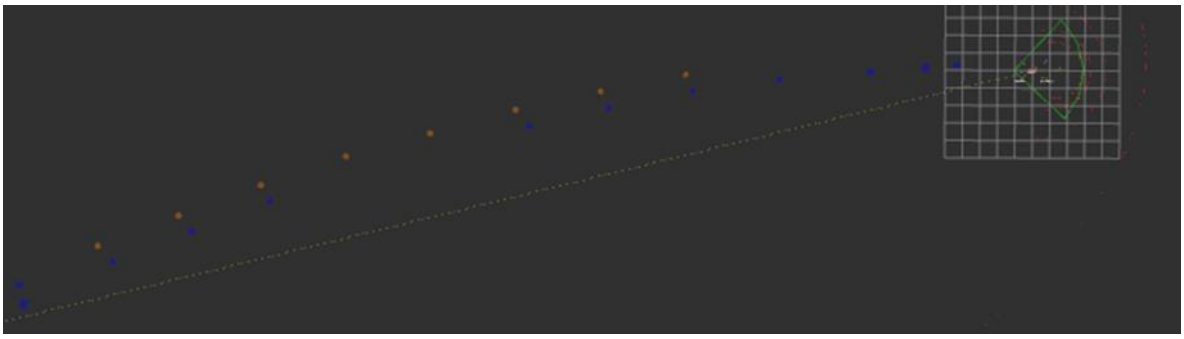


Figure 47: acceleration circuit simulation, with car speed of 2m/s and the update frequency of 5Hz

The figure 47 shows us the acceleration circuit mapped when the car traversed it with a maximum speed of 2 m/s and with a mapping update frequency of 5Hz. The number of cones detected are 25/46.

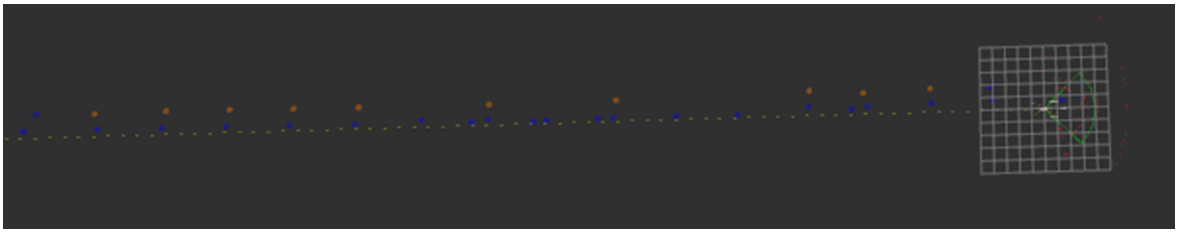


Figure 48: acceleration circuit simulation, with car speed of 5m/s and the update frequency of 5Hz

The figure 48 shows us the acceleration circuit mapped when the car traversed it with a maximum speed of 5 m/s and with a mapping update frequency of 5Hz. The number of cones detected are 38/46.

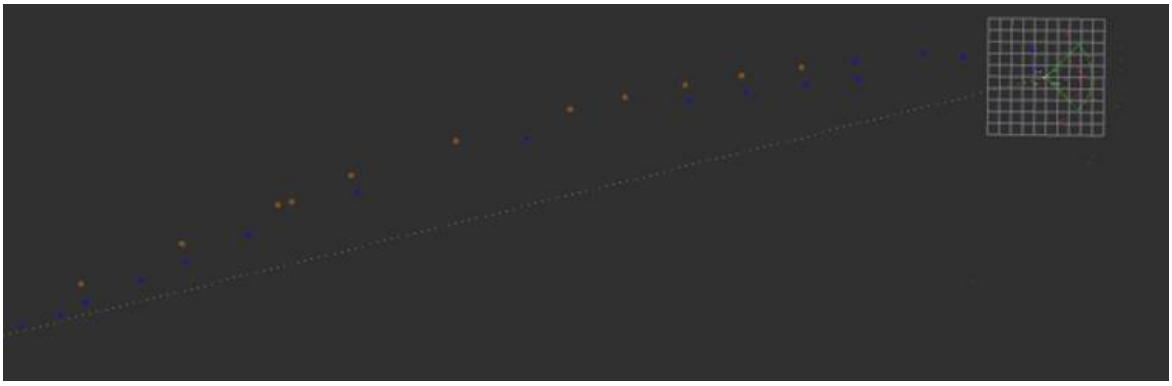


Figure 49: acceleration circuit simulation, with car speed of 10m/s and the update frequency of 5Hz

The figure 49 shows us the acceleration circuit mapped when the car traversed it with a maximum speed of 10 m/s and with a mapping update frequency of 5Hz. The number of cones detected are 33/46.

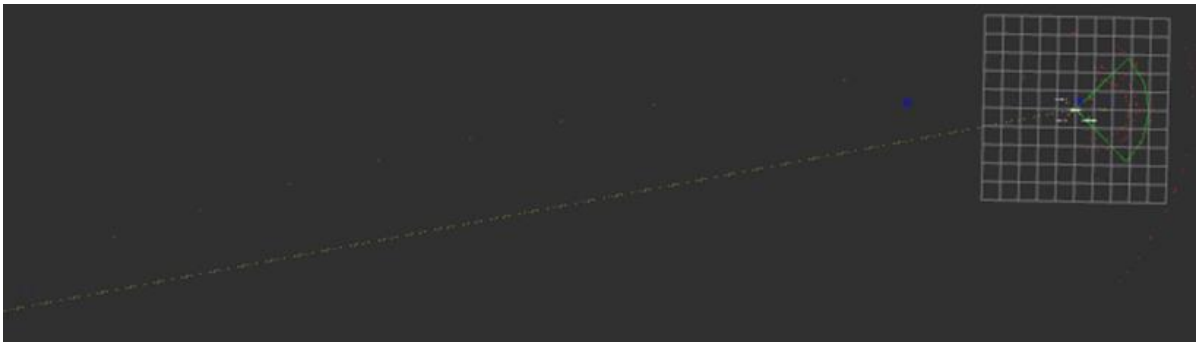


Figure 50: acceleration circuit simulation, with car speed of 2m/s and the update frequency of 60Hz

The figure 50 shows us the acceleration circuit mapped when the car traversed it with a maximum speed of 2 m/s and with a mapping update frequency of 60Hz. The number of cones detected are 24/46.

Figure 51: acceleration circuit simulation, with car speed of 5m/s and the update frequency of 60Hz

The figure 51 shows us the acceleration circuit mapped when the car traversed it with a maximum speed of 5 m/s and with a mapping update frequency of 60Hz. The number of cones detected are 34/46.

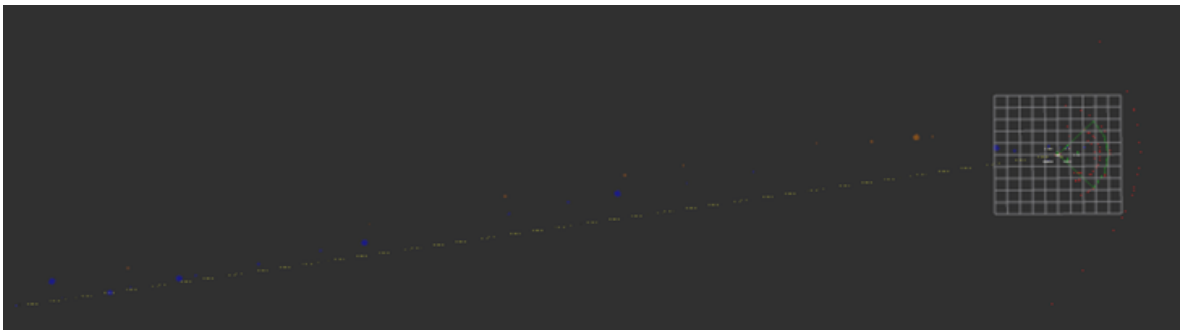


Figure 52: acceleration circuit simulation, with car speed of 10m/s and the update frequency of 60Hz

The figure 52 shows us the acceleration circuit mapped when the car traversed it with a maximum speed of 10 m/s and with a mapping update frequency of 60Hz. The number of cones detected are 27/46.

From these results we can do two important considerations. The first one is that the best results have been obtained with a maximum car speed of 5 m/s. The second one is that the update frequency with which the algorithm works well should be greater or equal to 10Hz. With a frequency of 10 Hz and a car speed of 5 m/s we obtain one of the best results. Moreover, it is important to note that every cone on a global map starts as a huge blob and with time its radius reduces. This indicates the car is more confident of cones being present there. Thus, if we use a high frequency (see the case where the car speed is 5 m/s and frequency is 60 Hz) we still have a satisfactory number of detected cones, but their radius is very small this because the car sees the cones more times at high frequencies then at low frequencies, so the cones confidence radius decreases. To understand the update frequency, we can image that the car has two eyes (the stereo camera) and these eyes are opened and closed with a certain frequency that is the update frequency. Thus, to tune this parameter, we have to make two consideration. The first one is that while the car is travelling between two successive cones, we want that the car sees more times the cones that have in front of it. The second consideration is that the frequency must be lower than the frequency with ROS

publishes the messages that is about 100Hz.

In the following pictures are shown the results obtained with the car traveling in a small-closed loop at different speeds and different update frequencies.

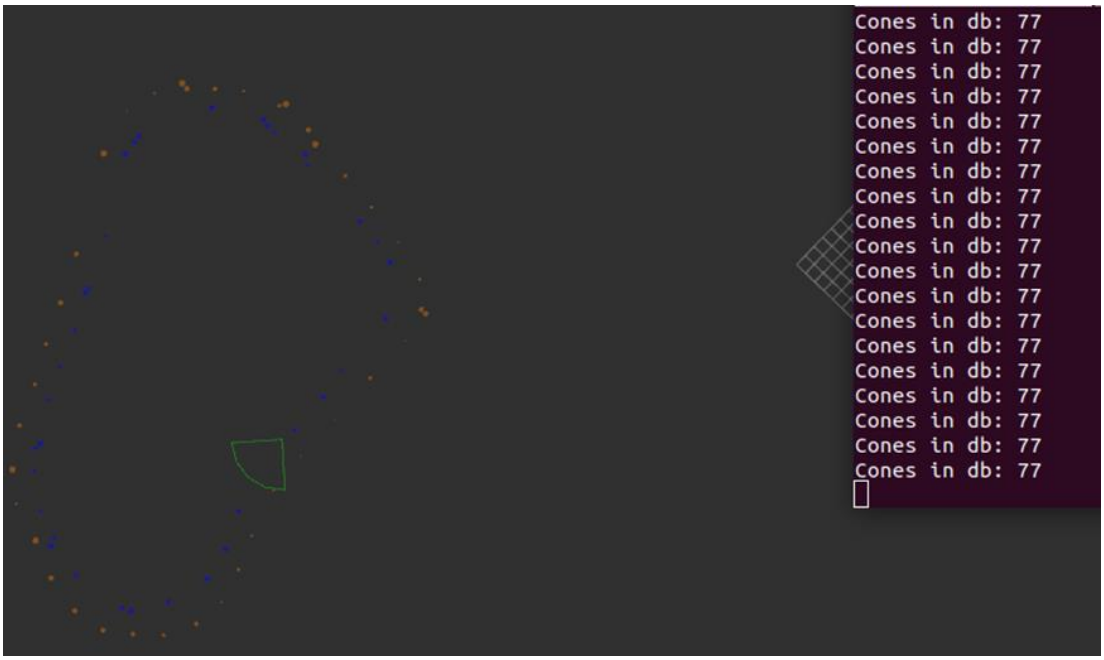


Figure 53: small-track simulation, with car speed of 1.5m/s and the update frequency of 5Hz

The figure 53 shows us the small-closed racetrack mapped when the car traversed it with a maximum speed of 1.5 m/s and with a mapping update frequency of 5Hz. The number of cones detected are 77/67.

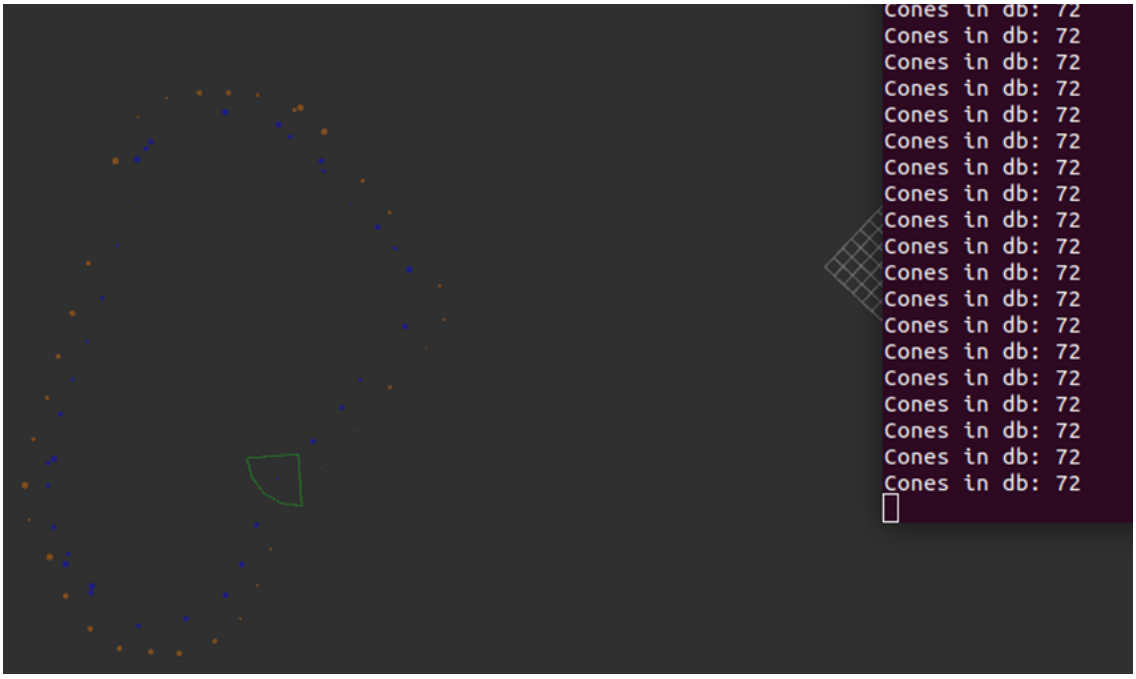


Figure 54: small-track simulation, with car speed of 1.5m/s and the update frequency of 10Hz

The figure 54 shows us the small-closed racetrack mapped when the car traversed it with a maximum speed of 1.5 m/s and with a mapping update frequency of 10Hz. The number of cones detected are 72/67.

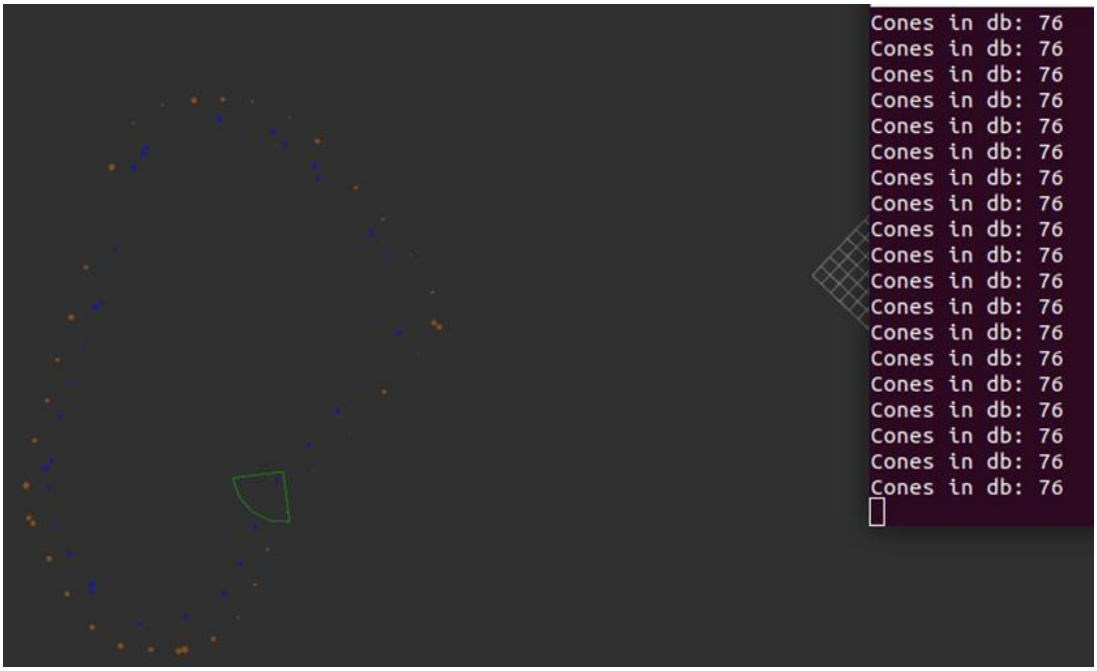


Figure 55: small-track simulation, with car speed of 1.5m/s and the update frequency of 20Hz

The figure 55 shows us the small-closed racetrack mapped when the car traversed it with a maximum speed of 1.5 m/s and with a mapping update frequency of 20Hz. The number of cones detected are 76/67.

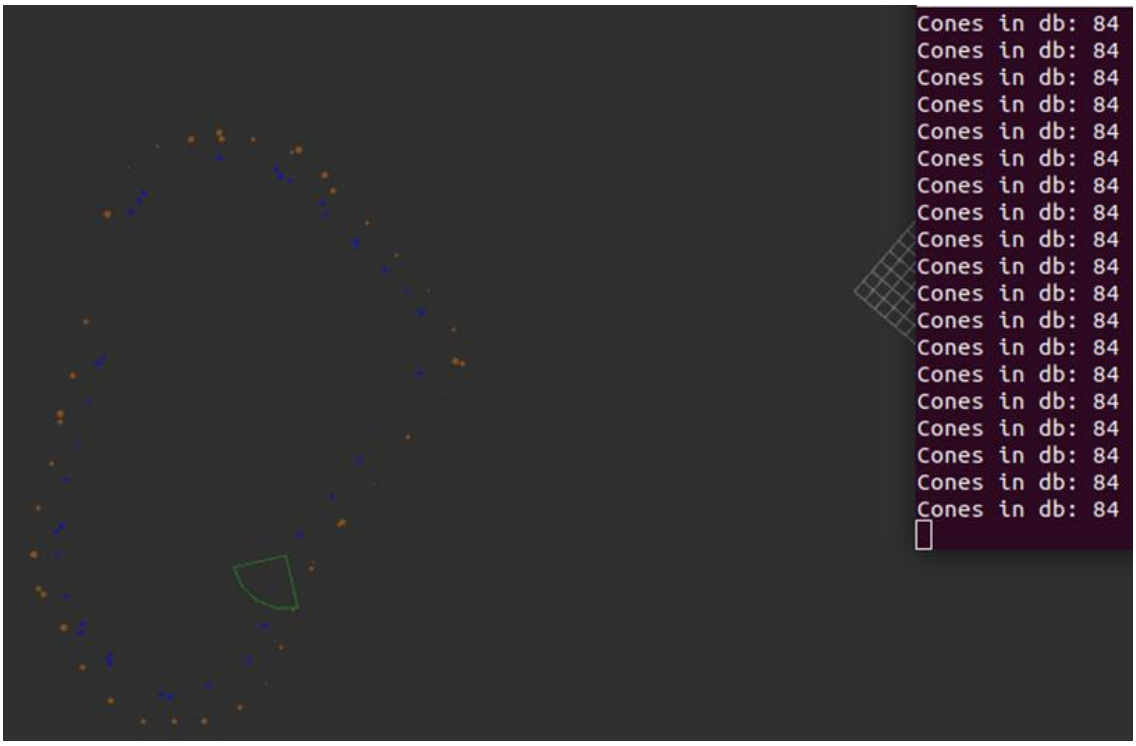


Figure 56: small-track simulation, with car speed of 1.5m/s and the update frequency of 30Hz

The figure 56 shows us the small-closed racetrack mapped when the car traversed it with a maximum speed of 1.5 m/s and with a mapping update frequency of 30Hz. The number of cones detected are 84/67.

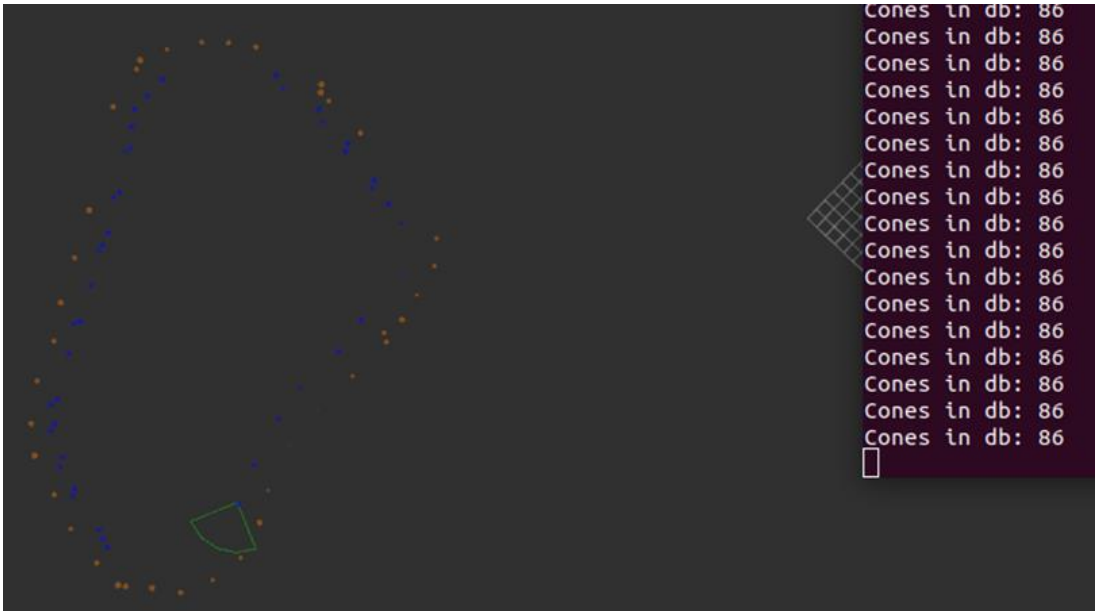


Figure 57: small-track simulation, with car speed of 2.5m/s and the update frequency of 5Hz

The figure 57 shows us the small-closed racetrack mapped when the car traversed it with a maximum speed of 2.5 m/s and with a mapping update frequency of 5Hz. The number of cones detected are 86/67.

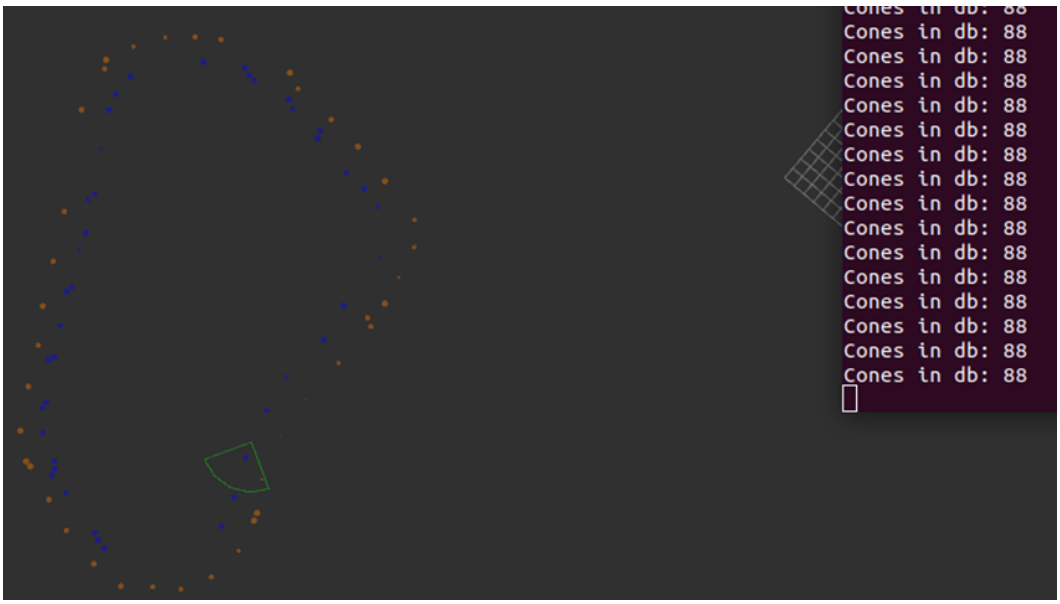


Figure 58: small-track simulation, with car speed of 2.5m/s and the update frequency of 10Hz

The figure 58 shows us the small-closed racetrack mapped when the car traversed it with a maximum speed of 2.5 m/s and with a mapping update frequency of 10Hz. The number of cones detected are 88/67.

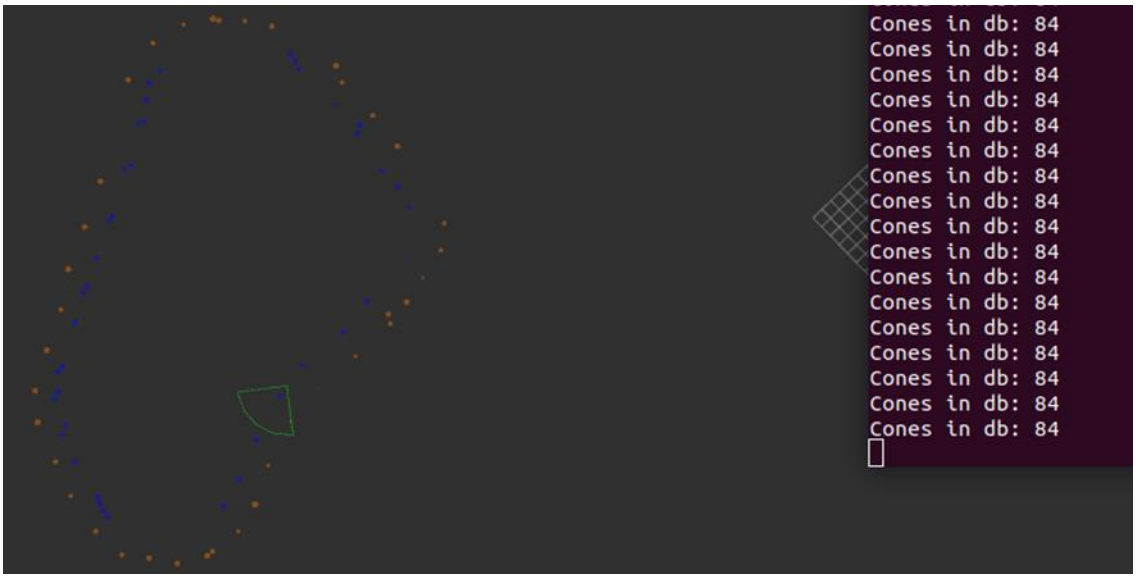


Figure 59: small-track simulation, with car speed of 2.5m/s and the update frequency of 20Hz

The figure 59 shows us the small-closed racetrack mapped when the car traversed it with a maximum speed of 2.5 m/s and with a mapping update frequency of 20Hz. The number of cones detected are 84/67.

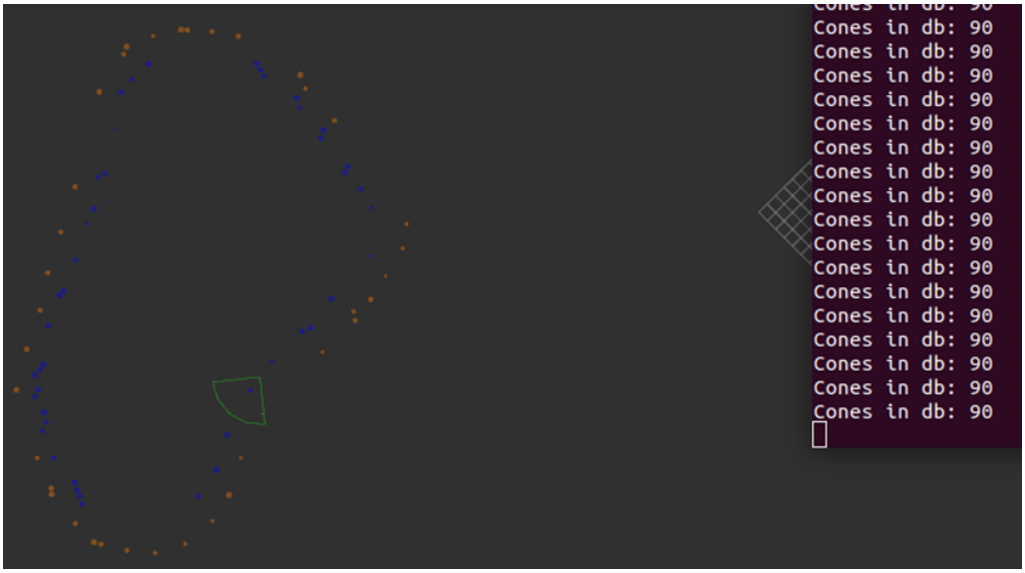


Figure 60: small-track simulation, with car speed of 2.5m/s and the update frequency of 30Hz

The figure 60 shows us the small-closed racetrack mapped when the car traversed it with a maximum speed of 2.5 m/s and with a mapping update frequency of 30Hz. The number of cones detected are 90/67.

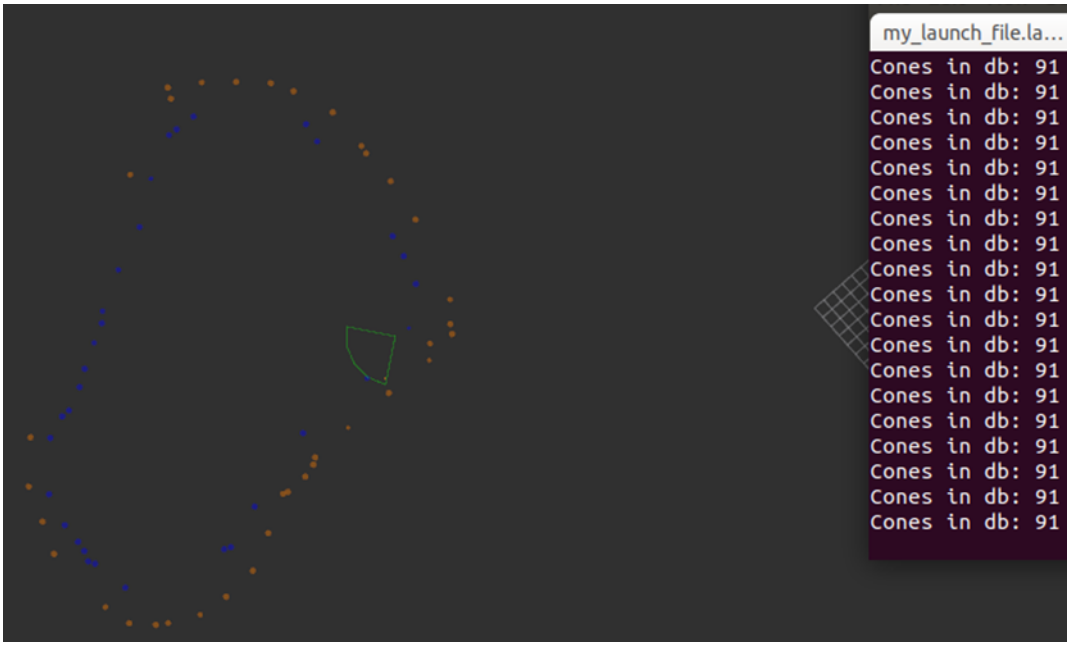


Figure 61: small-track simulation, with car speed of 7m/s and the update frequency of 5Hz

The figure 61 shows us the small-closed racetrack mapped when the car traversed it with a maximum speed of 7 m/s and with a mapping update frequency of 5Hz. The number of cones detected are 91/67.

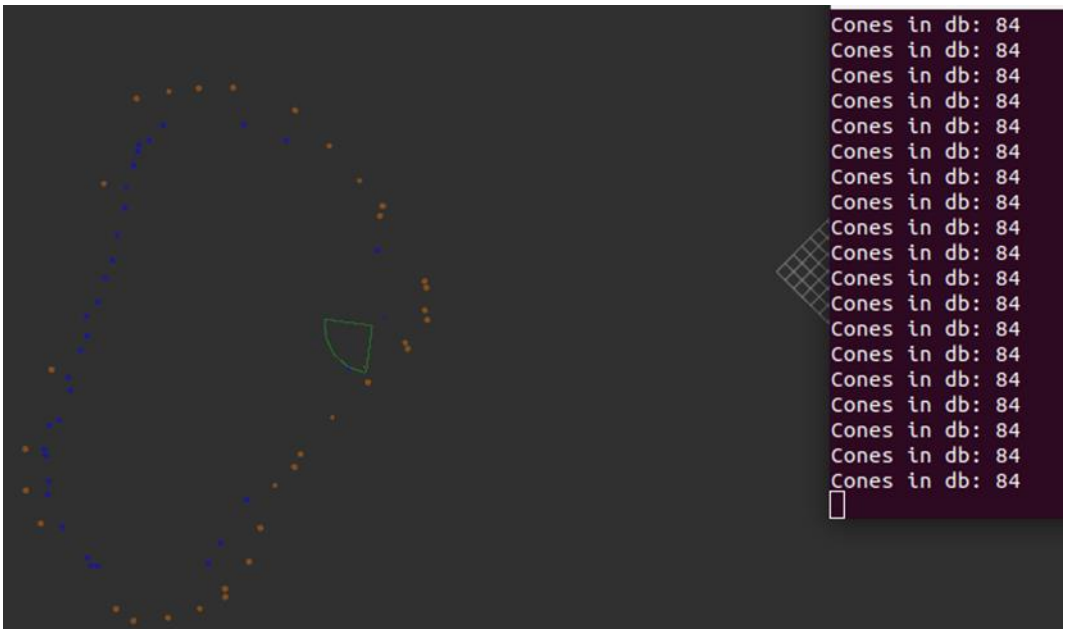


Figure 62: small-track simulation, with car speed of 7m/s and the update frequency of 10Hz

The figure 62 shows us the small-closed racetrack mapped when the car traversed it with a maximum speed of 7 m/s and with a mapping update frequency of 10Hz. The number of cones detected are 84/67.

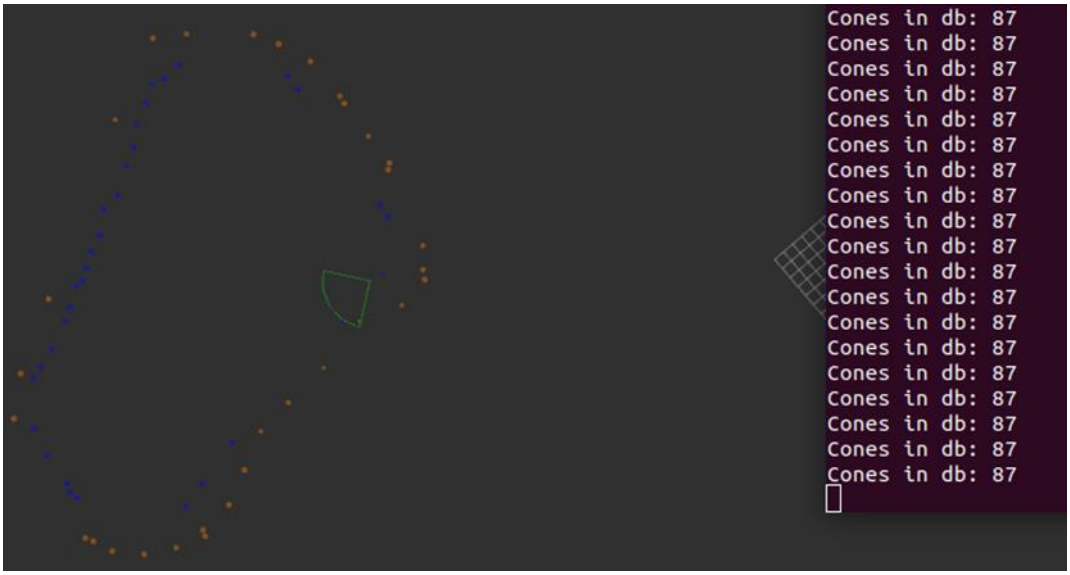


Figure 63: small-track simulation, with car speed of 7m/s and the update frequency of 20Hz

The figure 63 shows us the small-closed racetrack mapped when the car traversed it with a maximum speed of 7 m/s and with a mapping update frequency of 20Hz. The number of cones detected are 87/67.

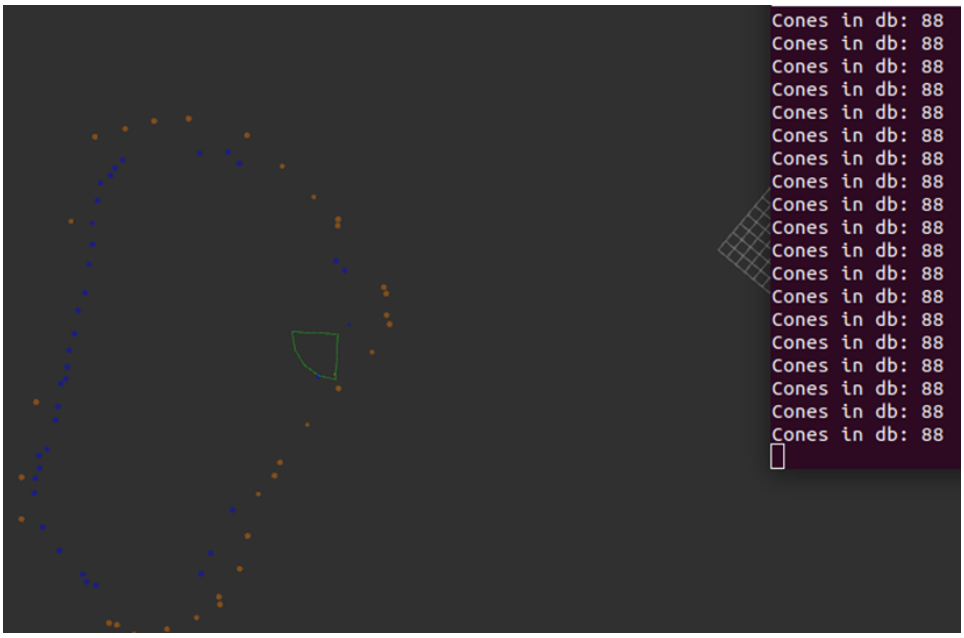


Figure 64: small-track simulation, with car speed of 7m/s and the update frequency of 30Hz

The figure 64 shows us the small-closed racetrack mapped when the car traversed it with a maximum speed of 7 m/s and with a mapping update frequency of 30Hz. The number of cones detected are 88/67.

The total number of cones of the circuit used in this simulation is 67 and in all the simulations the algorithm has stored a quantity of cones more than 15 % of the total number. This is because between the simulated odometry and the simulated camera frames there is a variable delay that produces in some cases a ‘‘trail’’ of cones.

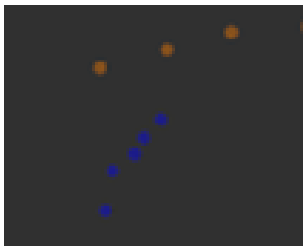


Figure 65: example of "trail" of cones

From this simulation results we can see that the algorithm robustness drops drastically with a car velocity of 7 m/s thus it suggested to use this algorithm with lower car speeds. Moreover, the error related to the relative position of the cones with respect to the camera plus the error related to the bad synchronization between messages accumulate after a certain number of laps and some cones disappear because they are not anymore in the same position, so they have a too big uncertainty. From this simulation we can observe the best results are obtained with two laps.
Here we shown the result obtained in the big circuit and small circuit with a car speed of 5 m/s and an update frequency of 10 Hz. These two results are equivalent from the conclusions point of view. In fact, the big-track test become important when we test together localization and mapping algorithm because it can evaluate the ability of the SLAM algorithm to close large loops.

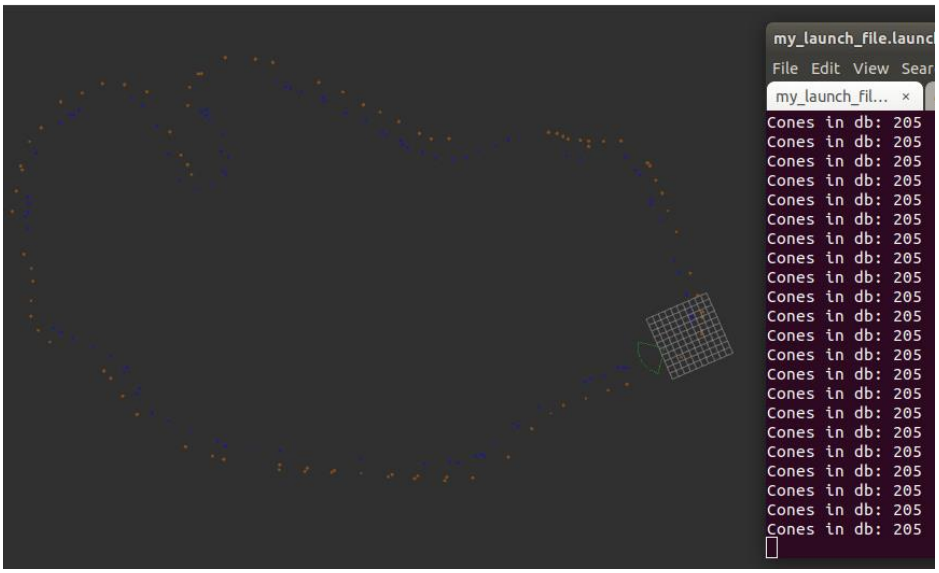


Figure 66: the mapping result in the big racetrack traveled two times with a car speed of 5 m/s and an update frequency of 10Hz.

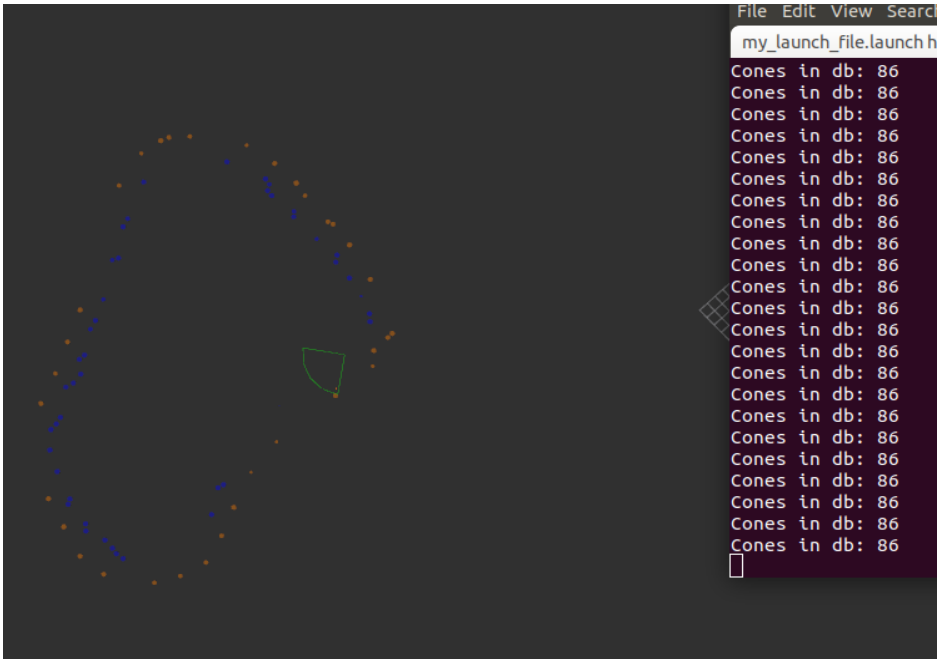


Figure 67: the mapping result in the small racetrack traveled two times with a car speed of 5 m/s and an update frequency of 10Hz.

5.2 The On-Track Testing

The on-track testing consists in testing the algorithm in the real system. This kind of test is extremely important because it is the only way to test the hardware and software together. In addition, in our case it is also the only way we currently have to test localization and mapping together. Its main drawback is the cost in fact this kind of tests are extremely expensive especially when something goes wrong. For this reason, it is very important to planning accurately the experience and it is good practice to run computer simulations before this stage. In this way we can significantly reduce the possible accidents. Unfortunately, we had only one experimental session to test on-track the mapping algorithm. During this test we have used the zed camera odometry and not the localization one so we have a great accumulation error (so we cannot correctly close a loop) but we have no problem with message synchronization. Moreover, we have a problem with the battery which powers the sensors and Xavier Jetson computer. This situation has forced us to simplify the racetrack and to use an extension cord for the alimentation. The circuit was a straight racetrack composed of a pair of orange cones at the beginning, four blue cones at one side and four yellow cones the other one and four cones per side at the end. The obtained results are the following:

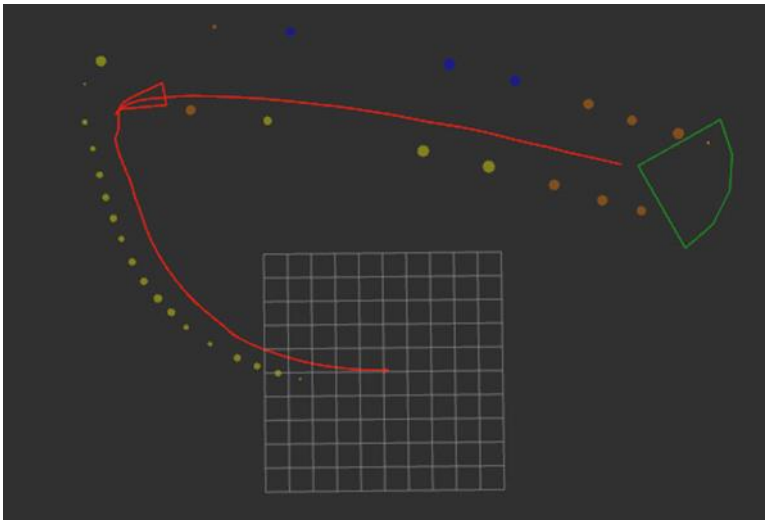


Figure 68: global map with an update frequency of 25Hz

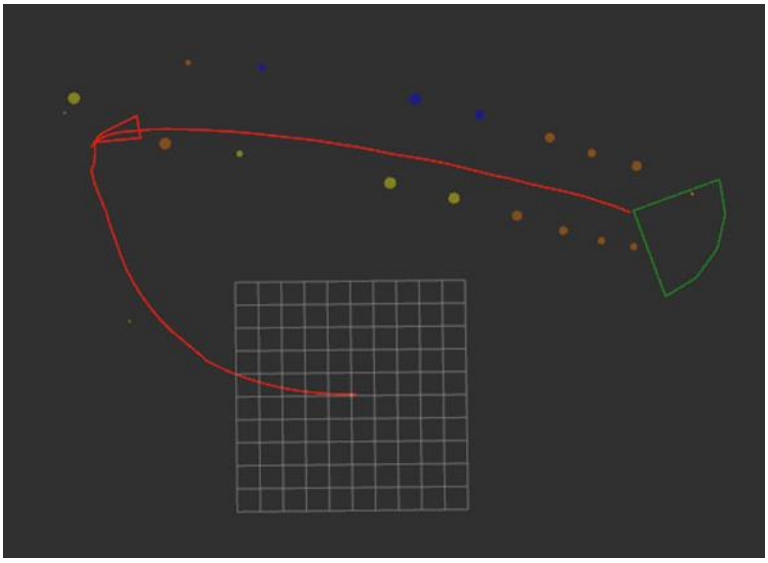


Figure 69: global map with an update frequency of 40Hz

In the figure 66 we have used an update frequency of 25 Hz, so the global map is updated more slowly in fact we can see the trail left by an object integral with the car. A different situation we have in the second image with an update frequency of 40 Hz, in fact we can see as the yellow trail disappears. This because at every update of map the car always see the same cone but moved slightly so we can see only a yellow point that moves together with the car. To better understand the difference between the first case and the last one we can say that in the image with 40 Hz frequency the car eyes were always open without batting an eye while in the image with 25 Hz frequency the car has opened and closed intermittently its eyes. The red line is the odometry path.

Chapter 6 – Conclusion and Future Works

In this experimental thesis work, it is faced the mapping problem which is a characteristic issue of the SLAM process in an autonomous racecar. In the first phase of the work, a research about the SLAM process state of the art it has been done. In particular it has been seen what are the critical issues that characterize the mapping problem, what methods has been used by the scientists to solve them and what requirements should be satisfy by an ideal mapping algorithm. After that, it was decided to adapt the KTH formula student driverless mapping algorithm available on GitHub website to our racecar. This mapping algorithm uses the zed stereo camera to detect the cones and to build the global map of the racetrack. More specifically this algorithm assigns at each detected cone an identification number, the position coordinates with respect the car starting point and the position covariance radius which is function of the number of times that a certain cone it has been seen. The cone position is updated using the exponential moving average. Moreover, the algorithm checks if a cone belonged to the global map it is not seen frequently. In this case the covariance radius increases until it reaches a threshold value, and the cone is removed from the map. After this phase, the algorithm it has been tested. The algorithm testing took place in two phases: the computer simulation and the on-track testing. For the computer simulation, the EUFS simulator it is been used. The main problems that have been addressed during this phase are those related to the libraries and software compatibility and the tickrate error between the stereo camera image topic and the odometry one. In fact, it has been used TensorFlow library which works only with Python versions major than 3, ROS Melodic which works with Python 2.7 and the EUFS simulator which works with ROS Kinetic. Therefore, it was made sure that all these libraries and programs could coexist despite their version incompatibility. And to solve the tickrate problem it is created a ROS node which uses a ROS filter called the AproximateTimesynchronizer based on an adaptative algorithm to match messages considering their timestamp. The results obtained from this computer simulation are that the algorithm robustness drops drastically with a car velocity of 7 m/s thus it suggested to use this algorithm with lower car speeds (the best result obtained it is with 5 m/s). Then we can observe that the update frequency with which the algorithm works

well should be greater or equal to 10Hz. And finally, we can observe the best results are obtained with two laps of the racetrack. As far as the on-track testing is concerned this is very important to test the algorithm together with the hardware and the localization algorithm together with the mapping one. Unfortunately, we have done only one significant experimental session which allows us to confirm some results obtained with the computer simulation, but we have not tested the localization and mapping algorithm together. This work is only the first step in the implementation of a robust mapping algorithm, and I believe that the next steps can be the following. First, it is necessary to test the localization algorithm together with the mapping one and we have to see if between the zed camera image topic and the odometry localization one there is a tickrate error. If there will be this error, I suggest modifying the localization node including in the sensor fusion the zed odometry. In fact, from the experimental session we have observed that the zed odometry is synchronized with the zed image topic. Then, we should implement a mapping algorithm using the LIDAR for example using the gmapping ROS package. Thus, we can fuse the two maps, the one obtained using the zed camera and the one obtained using the LIDAR to have a more robust result.

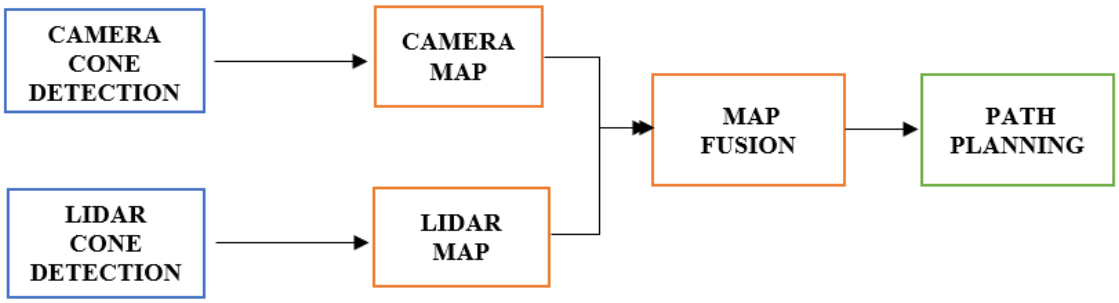


Figure 70: Camera and LIDAR map fusion

List Of Figures

Bibliography

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

[2] N. K. K. D. S. P. S. S. a. O. A. O. James M. Anderson, «James M. Anderson, Nidhi Kalra, Karlyn D. Stanley, Paul Sorensen,Constantine Brief History and Current State of Autonomous Vehicles,» 2014.

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

[4] «Wikipedia (2021), Society of Automotive Engineers,» [Online]. Available: https://it.wikipedia.org/wiki/Society_of_Automotive_Engineers;.

[5] «SAE Interantional, Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles J3016_201806,» [Online]. Available: https://www.sae.org/standards/content/j3016_201806/;.

[6] «Society of Automotive Engineers (2018), SAE International Releases Updated Visual Chart for Its "Levels of Driving Automation" Standard for Self-Driving Vehicles,» [Online]. Available: https://www.sae.org/news/press-room/2018/12/sae92 international-releases-updated-visual-chart-for-its-%E2%80%9Clevels-of-drivingautomation%E2%80%9D-standard-for-self-driving-vehicles, verified in 11/10/2020.

[7] «Official formula student website,» [Online]. Available: https://www.formulastudent.de/fsg/rules/.

[8] J. R.-A. J. M. R.-M. Jorge Fuentes-Pacheco, «Visual simultaneous localization and mapping: a survey,» 2012.

[9] U. Frese, «A Discussion of Simultaneous Localization and Mapping,» 2006.

[10] S. B. W. H. D.-W. a. T. B. Gamini Dissayaky, «Map Management for Efficient Simultaneous Localization and Mapping (SLAM)».

[11] G. J. a. N. E., «Optimization of simulataneous localization and map-building algorithm for real-time implementation,» *IEEE Transactions on Robotics and Automation,* p. 17(3):242–257., 2001.

[12] C. M. a. D.-W. H., «New approach to map building using relative position estimates,» in *Proc. of SPIE: Navigation and Control Technologies for Unmanned Systems II, Vol. 3087*, The International Society for Optical Engineering,, 1997.

[13] Newmann, «On the structure and solution of the simultaneous and localization and map building problem,» in *Ph.D. Thesis*, University of Sidney, Australian Centre for Field Robotics , 1999.

[14] K. L. Chong K., «Large scale sonarray mapping using multiple connected local maps,» in *Field and Service Robotics*, A. Zelinsky, A cura di, Springer-Verlag: Berlin, 1998.

[15] 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.

[16] R. a. S. P. Cheeseman, «On the representation and estimation of spatial uncertainty,» *International Journal of Robotics,* pp. 5:56-68, 1986.

[17] G. J. a. N. E., «Solving computational and memory requirements of feature-based simultaneous localization and mapping algorithms,» *IEEE Transactions on Robotics and Automation,* p. 19(4):749–755, 2003.

[18] M. S. S. J. Duckett T., «Fast on-line learning of globally consistent maps,» *Autonomous Robots,* p. 12(3):287–300., 2002.

[19] L. P. D. T. Frese U., «A multigrid algorithm for simulataneous localization and mapping,» *IEEE Transactions on Robotics,* p. 21(2):1–12., 2004.

[20] T. S. K. D. W. B. Montemerlo M., «Fast-SLAM: A factored solution to the simultaneous localization and mapping problem,» in *Proc. of the AAAI National Conference on Artificial Intelligence* , Edmonton, 2002.

[21] P. R. Eliazar A., «DP-SLAM: Fast. Robust Simultaneous Localization and Mapping without Predetermined Landmarks,» in *Proc. of the 18th International Joint Conference on Artificial Intelligence*, Acapulco, 2003.

[22] T. S., «Robotic mapping: A survey,» in *Exploring Artificial Intelligence in the New Millenium*, G. L. a. B. Nebel, A cura di, Morgan Kauffman.

[23] P. M., «Thin junction tree filters for simultaneous localization and mapping,» in *In Proceedings of the 18th International Joint Conference on Artificial Intelligence* , San Francisco, 2003.

[24] «Wikipedia (2021), Lidar,» [Online]. Available: https://en.wikipedia.org/wiki/Lidar; .

[25] «SBG system website,» [Online]. Available: https://www.sbg-systems.com/products/ellipse-series/.

[26] «Emlid website,» [Online]. Available: https://emlid.com/reach/.

[27] «FSG_Driverless_Accademy_20200926 slides seminar».

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

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

[30] M. Nielsen, «Neural Network and Deep Learning,» 2019.

[31] A. Ng, «<<Machine Learning>> Stanford University with Coursera,» 2020.

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

[33] «Wikipedia (2021), Moving Average,» [Online]. Available: https://en.wikipedia.org/wiki/Moving_average#Exponential_moving_average.

[34] «Wikipedia (2021), Simulation,» [Online]. Available: https://en.wikipedia.org/wiki/Simulation; .

[35] «Gitlab, EUFS simulator,» [Online]. Available: https://gitlab.com/eufs/eufs_sim.

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

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

[38] G. Petti, «Master of Science Thesis in Aerospace Engineering, "Optimization Methodologies Study for development of Prognostic Artificial Neural Networks.,» 2020.