POLITECNICO DI TORINO

Master of Science in Mechatronic Engineering

Master Degree Thesis

"Trajectory planning for a self-driving Electrical Vehicle: Design and development of a trajectory planning algorithm starting from the Occupancy Grid Map "

Tutor

Prof. Alessandro Rizzo

Candidate

Rocco Leo

ID: 278389

October 2021

ABSTRACT

During the 21st century, a consistent technological breakout has allowed a meaningful development of autonomous driving systems; this is due to state-of-the-art technologies which have contributed to the mentioned technological progress: the evolution of advanced sensors, definition of increasingly sophisticated technologies and an increase of computational power in generals influenced autonomous driving research field to reach advanced and challenging purposes. Self-driving systems could radically transform the current idea of transportation system, which inevitably would imply a significant change of our economy and society. Level 4 self-driving cars, which according to some automaker companies estimates, may be placed on the market in the next several years, would cause a complete social, economic, and technological revolution.

With this thesis project, we want to cover a specific problem of the huge world hiding behind the autonomous driving. By presenting a specific designed algorithm, the trajectory planning field for autonomous driving systems is addressed.

At high level, an autonomous driving system may be described by the so-called sense-act-plan procedure. The “sense” part is related to the sensors management, so that the vehicle surrounding environment can be detected the most effective way; the “act” part consists in the actuation system management; the “plan” part is where this work can be collocated. The project (VEGA: standing for “VEicoli a Guida Autonoma”) has been developed in Bylogix srl, a company which provides electrical and electronic engineering services and solutions for the Automotive industry, with a specific focus on autonomous driving.

The core of this work is the design of a trajectory planning algorithm, once data from the sensors are received and processed. In particular, by means of a LiDAR the outdoor environment, including obstacles, is sensed and an occupancy grid map

is given as input to the trajectory planner. Occupancy grid data, updated with a frequency of 10 HZ, are processed; thus, every 100 ms a trajectory is generated from the current vehicle position to a reference waypoint. From the generated trajectory, which guarantees the obstacles avoidance, the actuation variables are returned and sent by means of a CAN bus to the vehicle Electronic Control Unit.

After an overview related to the experimental setup (hardware devices and software technologies used in the implementation phase) and after a detailed description of the trajectory planning algorithm implementation, the final obtained result is presented. The prototype vehicle acquired the capability of driving, fully autonomously, in a predefined path generating a collision-free trajectory, avoiding the obstacles on the path. The reference output variables, that are steering angle and vehicle speed, are shown compared with the actual measured values: such comparison has proven to be more than satisfactory.

Contents

List of figures

List of tables

Chapter 1: Introduction

1.1 Paper purpose and layout

The main purpose of this thesis is to present the design and implementation of a trajectory planning algorithm for autonomous driving. The development has been performed in Bylogix s.r.l, which is a company providing electrical and electronic engineering services and solutions for the Automotive industry, with a specific focus on autonomous driving. The vehicle provided for testing operation is part of the VEGA project; the VEGA vehicle is presented in paragraph 1.4.

The core of this work is about the design of the trajectory planner, starting from an occupancy grid map provided as input, and the main requirement is to generate a safe trajectory which guarantees the obstacles avoidance. This paper is organized into 6 Chapters; in Chapter 1 an introduction about Autonomous Driving in general is provided, together with a presentation of the main technologies for AD and the most common trajectory planning algorithms in literature; plus, an overview on the specific VEGA project is also provided. In the second Chapter we start analyzing into specifics the assigned project, thus we describe the technologies used in the project both under hardware and software point of view; a focus on the occupancy grid map generation, which algorithm was already developed and tested, is reserved. Chapter 3 represents the core of this thesis, since the trajectory planning algorithm is described in details step by step. Chapter 4 is about the description of some tuning procedures, performed in order to find some parameters needed for the trajectory planner design. In Chapter 5 the final testing results are

presented and the paper ends with a Chapter about conclusions and future developments.

1.2 Autonomous vehicles introduction

Autonomous vehicles are also referred to as self-driving car, driverless car or robotic car, but independently from the used terminology they share the same technology. The idea of autonomous driving dates back to 1920, with experiments based on radio technology control system. The trail kept going from 1950 to nowadays, thanks to the development of automation and robotics technology to different fields of action, such as agriculture, medical, transportation, and manufacturing sectors. Specifically, for the last ten years the most relevant automobile industries have been starting to invest in researches about autonomous vehicles technology: we may mention Waymo Google, Uber, Tesla, Toyota, Bosch, etc.

Today, thanks to 5G and artificial intelligence, the whole world is waiting to see autonomous vehicles authorized to run on street. However, the disadvantages of autonomous cars are cost, mostly lost driving jobs and policy-making issues. On the other hand, the advantages would be safety, time saving avoiding traffic systems, vehicle parking space, pollution reduction, increasing of the vehicle life-time.

For the sake of completeness, we want to present the SAE international standard, which is described in the following figure [Fig. 1].

SAE level	SAE name	SAE narrative definition	Execution of steering and acceleration/ deceleration	Monitoring of driving environment	Fallback performance of *dynamic driving task*	System capability (*driving modes*)	BASt level	NHTSA level
Human driver **monitors the driving environment**								
0	**No Automation**	the full-time performance by the *human driver* of all aspects of the *dynamic driving task*, even when enhanced by warning or intervention systems	Human driver	Human driver	Human driver	n/a	Driver only	0
1	**Driver Assistance**	the *driving mode*-specific execution by a driver assistance system of either steering or acceleration/deceleration using information about the driving environment and with the expectation that the *human driver* perform all remaining aspects of the *dynamic driving task*	Human driver and system	Human driver	Human driver	Some driving modes	Assisted	1
2	**Partial Automation**	the *driving mode*-specific execution by one or more driver assistance systems of both steering and acceleration/deceleration using information about the driving environment and with the expectation that the *human driver* perform all remaining aspects of the *dynamic driving task*	**System**	Human driver	Human driver	Some driving modes	Partially automated	2
Automated driving system **("system") monitors the driving environment**								
3	**Conditional Automation**	the *driving mode*-specific performance by an *automated driving system* of all aspects of the *dynamic driving task* with the expectation that the *human driver* will respond appropriately to a *request to intervene*	System	**System**	Human driver	Some driving modes	Highly automated	3
4	**High Automation**	the *driving mode*-specific performance by an *automated driving system* of all aspects of the *dynamic driving task*, even if a *human driver* does not respond appropriately to a *request to intervene*	System	System	**System**	Some driving modes	Fully automated	3/4
5	**Full Automation**	the full-time performance by an *automated driving system* of all aspects of the *dynamic driving task* under all roadway and environmental conditions that can be managed by a *human driver*	System	System	System	**All driving modes**		

Figure 1: SAE levels for autonomous driving

The presented standard is a classification system for autonomous driving with six levels – ranging from fully manual to fully automated systems. This classification, published in 2014 by automotive standardization body SAE International, is based on the amount of driver intervention and attentiveness required, rather than the vehicle's capabilities, although these are loosely related.

In the following paragraph, after a brief introduction on the topic, we will focus on the main technologies involved in an autonomous driving system.

1.2.1 Autonomous driving system technology

During the 21st century the consistent technological breakthroughs have allowed the development of the Autonomous driving systems. The state-of-the-art technologies which have contributed to this technological progress, are:

- evolution of advanced sensors, to gather information about the world.
- definition of increasingly sophisticated algorithms.
- increasing of computational power in general.

Such technologies would allow the vehicles to better sense the surrounding environment, to process the input data in a more efficient way, i.e., in a fast time interval; plus, the use of advanced algorithms makes possible the definition of a suitable action in response to what the vehicle manage to sense through the advanced input acquirement system.

One of the prerequisites for autonomous vehicles (from now on abbreviated as AVs), is the ultra-reliability. Such reliability is reasonably difficult to achieve in a dynamic and complex environment in which many external factors could not be taken into account by the control system. Thus, it is evident that a brief discussion about current AV technology and its limitations is needed.

At high level, the used procedure can be described by the "sense-plan-act" design:

A set of sensors on the vehicle gathers information related to the environment in which the AV is inserted, then the implemented algorithm allows the

interpretation of the sensor data and, in last analysis, according to what the sensors manage to gather and according to the processing procedure some actions are taken by the vehicle, such as accelerating and directions changing. These plans are converted into commands to the vehicle's control system; the main commands in AVs systems are steering, throttle, brakes. Plus, it is important to underline that the "sense – plan – act" procedure cycles could run simultaneously on AVs. In particular, the frequency is directly proportionate to the required velocity in which an action should be taken; for example, a huge frequency is needed to execute an emergency braking, while for less critical situation a moderate frequency may be used. The "sense – plan – act" procedure is schematized in the following figure [Fig. 2]:

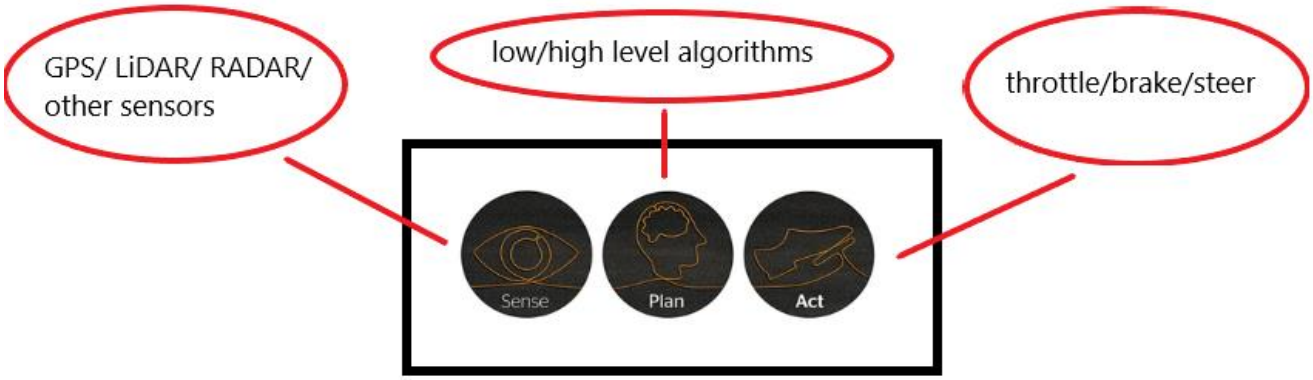


Figure 2: Sense - Plan - Act design

An overall description of the used technology is following.

1.2.2 “Sense”: the sensor suite

The sensor suite is often composed by the following devices:

- Lidars
- Radars
- Ultrasonic systems
- GPS

1.2.2.1 LIDARS

“Lidar” stands for Light detection and ranging system, that is a device widely used in the robotic field, including AVs. Lidars manage to establish distances with respect to an obstacle by means of laser range finders, computing the time-of-flight (ToF).

In order to understand the huge potentiality and the use of wide range of this technology, it is necessary to mention some facts. Nowadays, engineering start-ups are receiving consistent investments related to the development and application of the lidar technology, mainly from the automotive industry. An example is Aeva, a lidar company started by two former Apple engineers: It managed to raise $200 million from Hong Kong Sylebra Capital. The post deal market valuation reached a value of $2.1 billion. Plus, the innovation point of Aeva is the 4D lidar, which measures distances, velocity, preventing interferences from other sensors.

In general, at the moment, three most used Lidar implementations are:

1) Pulsed approach Lidar.
2) AMCW approach Lidar.
3) FMCW approach Lidar.

The Pulsed approach is based on the intensity measurement (it is referred to as incoherent measurement), its degree of accuracy is at centimeter level; plus, it is convenient for the simple setup procedure, the limitation is related to the low SNR at long ranges. AMCW approaches, on the other hand, are currently well developed (based on CMOS technology) and efficient in indoor environments. Finally, the FMCW approach would most likely be the baseline technology in AVs field, since its detection procedure allows a better resolution as regards the measurements between different orders of magnitude; plus, a simultaneous measurement of the target speed is guaranteed. The following table (Tab. 1) summarizes in a schematic way the pros, cons, and characteristics of each technology.

	PULSED	**AMCW**	**FMCW**
MEASURED PARAMETERS	Intensity of the pulse	Phase of modulated amplitude	Relative beat of modulated frequency
USE	Indoor/outdoor	Only indoor	Indoor/outdoor
PROS	Setup simplicity	Well established tech	Simultaneous speed measurement

CONS	Low SNR	Short ambiguity distance	Stability in operating conditions
RESOLUTION	1 cm	1 cm	0.1 cm

Table 1: LiDAR technologies

a quite detailed description of the technologies is explained in the following paragraphs.

Pulsed approach

The pulsed approach technique is mainly based on the modulation principle of the illumination beam: the main assumption is related to the fact that the speed of light in a given optical medium should be constant, thus the distance to the object is directly proportional to the time of flight. In fact, the distance is computed by multiplying the speed of light by the time needed by the light pulse to reach the target.

However, it should be considered that the measured time takes into account twice the distances to the target, since the light travels forth and back, thus a 0.5 factor should be considered in order to determine a correct value of the distance to the obstacle. In formula:

$$R = ToF\frac{c}{2}$$

Where R is the distance to the target, c is the speed of light in free space ($c = 3x10^8\ m/s$), Tof is the required time for the pulse of light to travel from the source to the target and back to the emitter.

In the following figure [Fig. 3] a simplified implementation diagram in shown:

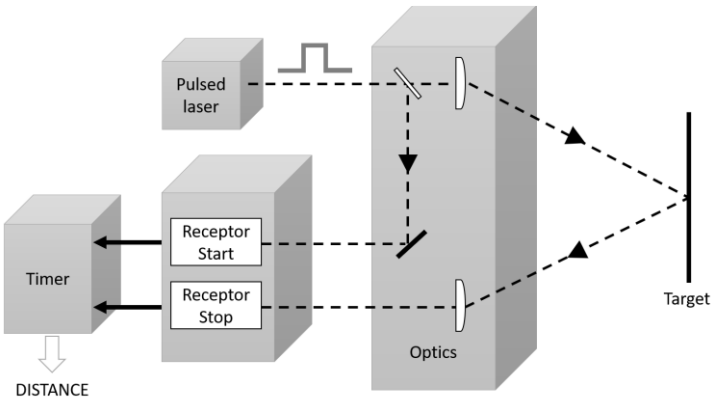


Figure 3: Lidars: Pulsed Approach

Continuous Wave Amplitude Modulated (AMCW) approach

The first difference with respect to the Pulsed approach is the use of a continuous light wave instead of laser pulses. Secondly, another important difference is about the used principle: AMCW is mainly based on a phase shift measurement, the distance to the target is defined by means of the phase shift of an intensity modulated periodic signal in its round-trip. Thus, the signal – that is a sinusoidal or a square wave of constant frequency (fM) – is emitted by the source; after the reflection with the obstacle/target the received signal is received by a collector. The phase shift of the received signal with respect to the emitted one is used to compute the distance R.

The computation of the phase shift $\Delta\Phi$ happens as follows:

$$\Delta\Phi = km * d = 2\Pi fM \frac{2R}{c}$$

Where km is the wavenumber related to the modulation frequency fM of the amplitude of the signal, d is the total travelled distance, R is the distance to the target and c is the speed of the light in the free space.

Finally, by inverting the formula above the distance can be computed as follows:

$$R = \frac{c}{2}\frac{\Delta\Phi}{2\Pi f M}$$

A scheme of the AMCW technology is shown in the following figure [Fig. 4]:

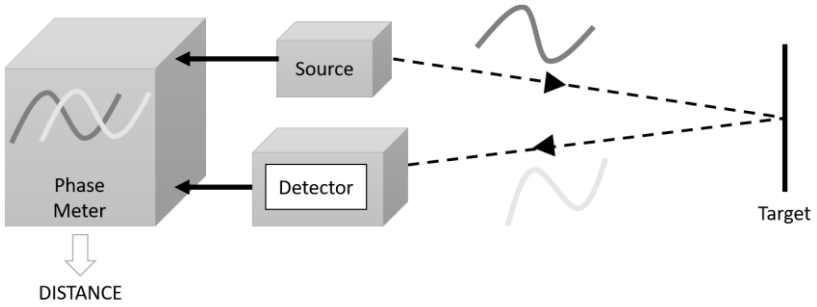


Figure 4: Lidars: AMCW approach

Continuous Wave Frequency Modulated (FMCW) approach

In the FMCW technology the quantity considered to determine the distance to the target is the frequency. The emitted optical frequency is periodically shifted, by varying the power applied to the source. What is exploited is the mixture of the emitted source with the reflected signal, this phenomenon creates a beat frequency. The beat frequency is due to the delay between the collected light and the reference signal. This frequency fr, supposing its variation according to a linear law, is directly

proportional to the time of flight, and consequently it is proportional to the distance to the target. The frequency computation is:

$$fr = slope\ \Delta\tau = \frac{B}{T} ToF = \frac{B}{T}\frac{2R}{c}$$

From which the distance to the target R cab ne derived:

$$R = fr\frac{cT}{2B}$$

Where B is referred to the bandwidth, T is the period of the ramp, $\Delta\tau$ is equal to the total travelled time, the other parameters have already been defined. The following figure [Fig. 5] shows all the mentioned parameters:

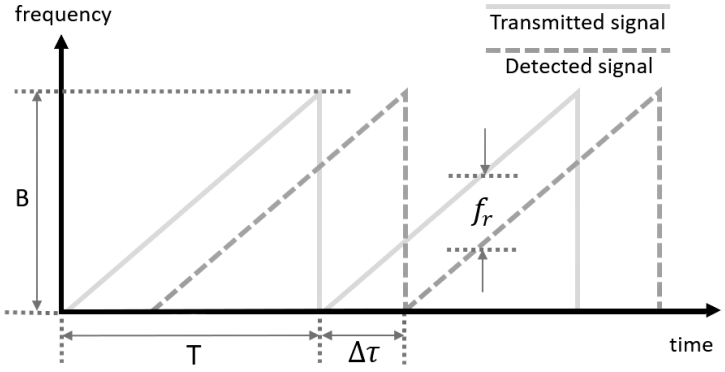


Figure 5: Lidars: FMCW approach

1.2.2.2 RADARS

Radars – that stands for Radio Detection And Ranging – are, as for lidars, fundamental components of the sensing system of an autonomous driving vehicle. Like lidar, radar technology is based on the ToF to calculate the distance to the target object. However, differently from lidars, radars, as

the name suggests, exploit radio waves: this characteristic leads to an important limitation; the technology operates properly on metallic objects, but nonmetallic objects, such as pedestrians, cannot be seen by radar sensor. As consequence, it is necessary to include with lidars and radars other sensors, so that the vehicle can have a more complete vision of the surrounding environment. In particular, radars work by means of the emission of electromagnetic waves, which are reflected when an obstacle is intercepted. The most relevant technology is the FMCW, similar technology used for lidars. FMCW radars emit continuous power and obstacles can be detected at very small distances, plus the velocity of the (moving) obstacle/target cab be easily computed thanks to the Doppler effect: considering the following image [Fig.6], the red wave represents the wave transmitted by the radar, the blue top one represents the reflected wave for an approaching vehicle, the blue bottom one represents the reflected wave for a moving away vehicle.

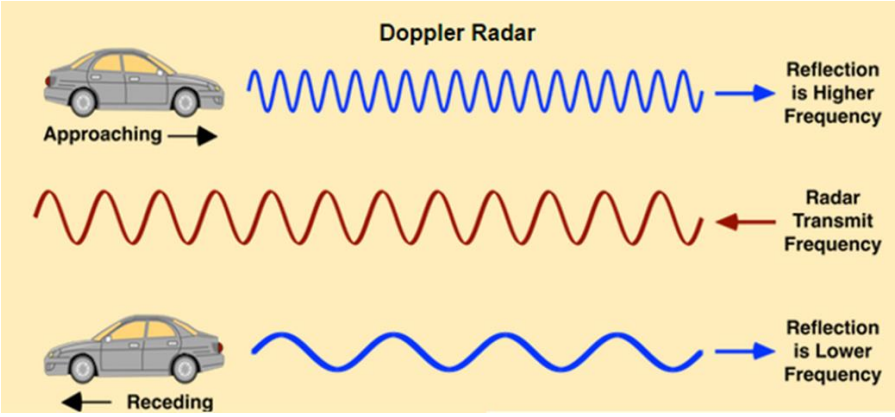


Figure 6: Radars Doppler effect

The Doppler effect equation is:

$$fD = \frac{2v}{\lambda}$$

Where *fD* is the measure frequency shift, λ represents the wavelength and *v* is the vehicle speed, which can be easily computed.

As regards the implemented hardware technology, FMCW radars are composed by:

- A frequency synthesizer: capable of setting the reference wave at the proper frequency.
- A power amplifier: capable of amplifying the emitted signal, so that the device can work in a long range (300m).
- An antenna: responsible for converting the electricity into electromagnetic waves, plus it sends and receives the reflected signal.
- A processor: responsible for processing the signals.

1.2.2.3 Ultrasonic systems

Ultrasonic sensors have an accuracy of short-range, around 1-10 meters, thus these kinds of sensors are mainly useful for parking assistance or as backup warning system.

The examined device works by emitting ultrasonic sound waves and convert the reflected sound into an electrical signal. The main components are the transmitter, which emits signals by means of piezoelectric crystals, and the receiver, which receives the sound reflected by the target. As for lidars, the distance from the target can be measured by considering the

time taken between the emission of the sound and its reception. The formula is:

$$R = c\frac{Tof}{2}$$

Where c is the speed of sound (343 meters/second), *ToF* is the time of flight.

1.2.2.4 GPS

A fundamental aspect of the AVs sensory system is GPS, standing for Global positioning System. This technology results to be essential for localization. The working principle of GPS operations is the following one:

- Computation of the distance of the receiver (autonomous vehicle) from visible satellites: using the speed of light equation $D = Tof\ c$, where c is the speed of light, *Tof* is the time of flight and D is the distances to the satellite.
- Trilateration of the receiver position: three satellites are used in order to locate univocally the receiver on the Earth surface, an additional one is used in order to reduce possible offset errors.
- Errors correction, induced by clock shifting or other causes: a possible solution is the use of differential GPS which works thanks to the cooperation of two receivers, with different clocks, one stationary whose location is known with high precision, one is a moving one (in our instance it could be the autonomous vehicle). The stationary receiver measures the timing error and provides the moving receiver with the

correct information. The following figure [Fig. 7] outlines this described mechanism:

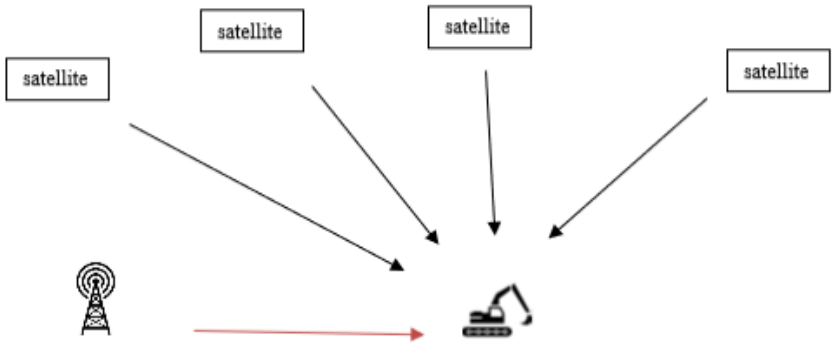


Figure 7: Differential GPS

Additionally, GPS is usually coupled with INS (Inertial Navigation System), which continuously calculates position, orientation, and velocity of the vehicle, by means of gyroscopes and accelerometers. INS is mainly used in the conditions in which GPS appears to be not available; anyway, it should be considered that, even if sophisticated system are used, systems relying on INS are subject to significant errors, of order of magnitude of meters.

1.2.3 "Plan – Act": Low/High level algorithms and actuators

This paragraph is aimed at presenting the most used algorithms, responsible for processing data coming from the sensors. It is appropriate to classify the AVs algorithms into high level algorithms and low-level algorithms.

The low-level algorithms deal with row data, in fact they deal with image processing, Lidar processing and radar processing. These algorithms gather data from the sensors and process this information giving back as output a key info that is then processed by high level algorithm, in order to take decisions.

Among the high-level algorithms, it is necessary to mention:

- Feature extraction algorithm – it would determine lane lines, or it identify a signpost.
- Object classification algorithm – it identifies and classifies different objects, i.e. fog, pedestrians, smog, traffic lights etc.
- Mapping and localization algorithm – it is necessary to continuously locate the vehicle, in order for it to go autonomously from a point A to a point B.
- Trajectory planning algorithm – this aspect is deepened in the next paragraphs.

The taken decisions, according to the processing strategy, are then translated in real actions thanks to the vehicle actuators. The main actuators are steering, throttle, brakes.

1.3 Trajectory Planning

1.3.1 Trajectory planning introduction

The main topic of this thesis is related to the design of trajectory planning algorithms.

This paragraph aims at introducing the topic and one of the current technologies of the motion planning is presented (A* algorithm).

By definition, a trajectory is represented by a sequence of states, each one visited by the vehicle. This set of points is not just a geometric entity, otherwise we would have referred to “path” instead of “trajectory”, but it is parametrized by time and, often, by velocity and acceleration. Trajectory planning is responsible for the real-time planning of the vehicle transition from a state A to a state B, satisfying all the vehicle constraints such as kinematic limits, navigation comfort, fuel consumption and so on. Plus, obstacles must be avoided, and the collision avoidance must be guaranteed with a high reliability. The trajectory planning procedure is cyclical: the path planner module generates a certain number of possible trajectories, so that the destination point B can be reached starting from A. The optimal trajectory is chosen according to the minimization of a specified cost function; the planning is scheduled at regular time interval, whose duration depends on the working frequency of the sensory suite.

One important prerequisite for the trajectory generation is the environment representation. In fact, the physical space should be transformed into the so-called state space. The state space contains all the possible vehicle configurations (position, orientation, linear or angular velocities); basically, the continuum environment must be transformed by means of a digital representation. This space discretization can be obtained by using different strategies, the technique used in this thesis is based on the occupancy grid. The occupancy grid discretizes the space into a grid: each

cell is associated with a probability of occupancy (1 if it is for sure occupied by an obstacle, 0 is the probability of occupancy is null). The advantage of this grid-based approach is related to the low computational power; on the other hand, the main disadvantage is linked to the difficulties in accounting robustly for non-linear dynamic.

The following figure [Fig. 8] shows all the steps involved in the trajectory generation:

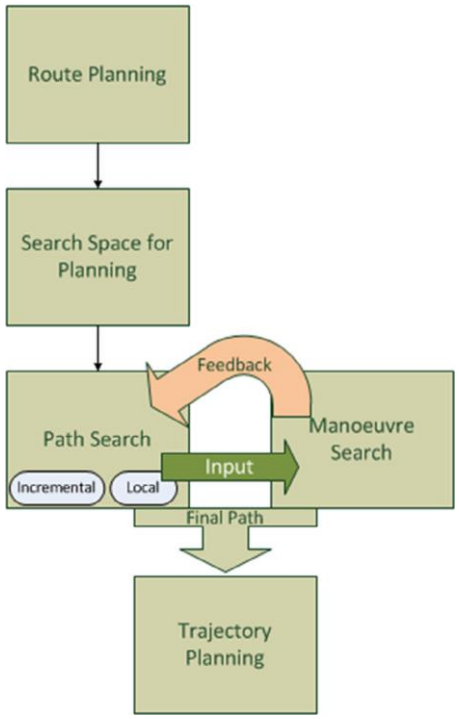


Figure 8: Trajectory planning steps

The *route planner* provides a route, it is the given input of the system. The second block in cascade "Search space for planning" represents the discretization of the surrounding environment, for example by using an occupancy grid. Then, planning can be subdivided into *incremental*

approaches, which find the best sequence of state transitions by also considering the previous ones, and *local approaches* related to the best single state transition. Both these approaches are considered as inputs for the *Manoeuvre Search*, which is concerned for determining the proper manoeuvre which places the vehicle in the most appropriate position. Because of the feedback between the Path Search and Manoeuvre Search blocks, the final path may change and, once it is created, the final trajectory planning is generated.

1.3.2 Trajectory planning algorithms

Usually, the most commonly used motion planning techniques fall within the field of graph search algorithms; the presented trajectory planning algorithms are the Dijkstra's algorithm and the A* algorithm.

Dijkstra Algorithm: The Dijkstra algorithm was introduced by Dutch computer scientist Edsger W. Dijkstra in 1959. The algorithm has the purpose of finding the shortest (the least costly) path from a starting point A to a destination point B. An example is shown in Figure 9.

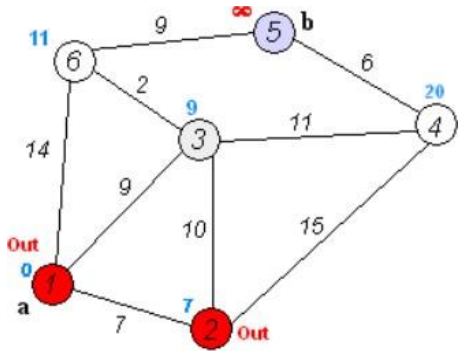


Figure 9:Dijkstra algorithm example

The nodes are classified into unvisited set and visited set. The edge weights between each node are known. At the beginning all the nodes are in the unvisited set.

First the starting node “a” is added to the visited set and removed from the unvisitedset. Since the distance of the starting node is 0, thenode value will be assigned as 0.

Next the values of the nodes close to node A are evaluated. If the cost is lower than the value of the node, then the node's value will be assigned as the total cost to reach that node. After that, the node with the lowest value will be selected as the next current node and it will be added into the visited set and removed from the unvisited set.

In general, the basic idea of Dijkstra is that: the optimum solution of the partial path is independent from the optimum solution of the whole path. For example, if the shortest path from node 1 to node 6 is 1->6 rather than 1->3->6, then in the final solution if the path starts from node 6, the previous path before 6 must be from 1->6. It tries to find the global optimum with the help of the local optimum.

However, the searching space of Dijkstra algorithm is large. Since the information of the destination is not used, the search process is quite inefficient. In order to overcome this issue, the A* algorithm is introduced.

A* Algorithm: The A* algorithm is the extension of Dijkstra's algorithm, thanks to the implementation of the heuristics, it reduces the calculation time compared with Dijkstra's algorithm. The main equation is shown as follows,

$$f(n)=g(n)+h(n)$$

In this equation n means node n, while g(n) is the actual cost from the original node to the node n, h(n) is the estimation of the optimum cost from node n to the destination node, f(n) is the estimation cost from the initial node to node n. The searching processis similar to Dijkstra's algorithm. The difference is that for Dijkstra's algorithm the next node explored node is the node with the minimum actual cost, while, for A*, the next explored node is the node with the minimum estimated cost, which is the sum of actual cost from initial node to node n and the estimation cost from node n to the destination. Thus, the searching process is more target orientated and the searching time is reduced.

1.4 The VEGA project: scenario

Bylogix has been among the Autonomous Driving pioneers in Italy, developing the first L4 autonomous driving vehicle based on **Nvidia Drive** platform in Italy.

Bylogix created a flexible system architecture that can host autonomous driving functionalities and ADAS features for testing, prototype and production vehicles.

A BEV Citroen E-Mehari was used as base vehicle; the electronic and electric architecture of the mass production car was in fact designed by Bylogix, allowing an easy integration of new features.

In the following picture [Fig. 10] a timeline of the VEGA ("VEicolo a Guida Autonoma") project is shown. The last step, reached by means of this thesis project, is the design of an obstacle-avoidance version of a trajectory planning algorithm.

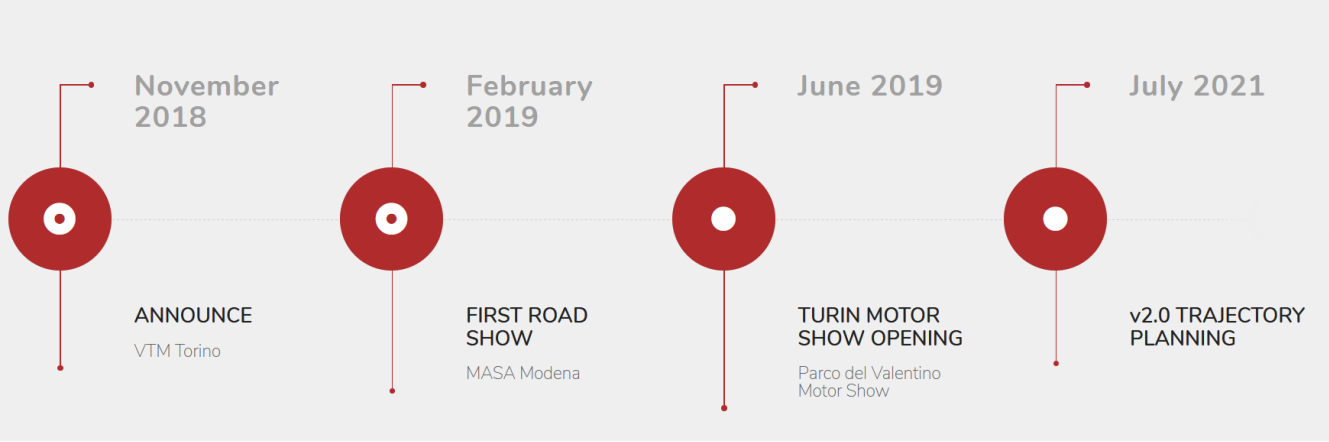


Figure 10: VEGA project timeline

The following figure [Fig. 11] is a representation of the base vehicle used to test the designed trajectory planner.

Figure 11: VEGA test vehicle

1	Surrounding perception, egomotion and geo-positioning sensors
2	Wired and wireless development network
3	Bylogix graphics IPC system
4	Donor car CAN-BUS network bridge
5	Dedicated CAN-BUS network

6 Bylogix By-Wire-Acceleration system

7 Bylogix Steer-By-Wire system

The technologies relevant for the project, among the ones mentioned above are: surrounding perception, ego motion and geo-positioning sensors (LiDAR, GPS sensor), the CAN bus network and the By-Wire acceleration and steer systems, which are used in the actuation phase.

Chapter 2: Experimental setup

In this chapter an overview of the different technologies, both hardware and software, which have been used for the realization of the autonomous driving project, are presented. This chapter represents the experimental setup used to develop the trajectory planning algorithm, which is analyzed in detail in the next chapter.

2.1 Hardware technologies

The hardware technologies presented in this paragraph consist in the devices used to "sense" the vehicle's surrounding environment (GPS and LiDAR sensors), the device used to elaborate and perform computational operations (NVIDIA TX2); plus, an overview on the CAN bus, used to send actuation variables to the ECU is provided.

2.1.1 Velodyne LiDAR: Puck-Hi Res

Velodyne LiDAR's Puck Hi-Res version has been used since it guarantees a high resolution in the 3D image capturing process. This kind of resolution is ensured by a FoV of 20° with a tight channel distribution of 1.33°; on the other hand, a standard device would have ensured a FoV of 30° but with an associated resolution considerably lower. The following figure [Fig. 12] underlines this comparison.

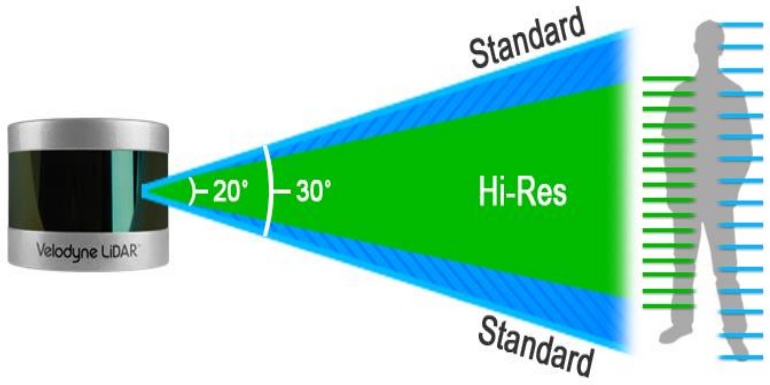


Figure 12: Velodyne LiDAR

Other specifications are the following ones: the horizontal FoV is of 360°, in this way the complete surrounding environment is detected, and real-time 3D LIDAR data is generated; The Puck Hi-Res has a range of 100 m and this guarantees a detailed detection at longer ranges at a low power consumption. Plus, the high resolution is provided by a close spacing between 16 supported channels; this generates 300,000 points/second from a 360° horizontal field of view and a 20° vertical field of view with ±10° from the horizon. Finally, the rotating parts are not visible since the device is encapsulated is a package allowing a wide range of operating temperature condition: for further details and specifications, please, refer to the table [Tab. 2] below.

SENSOR	LASER	MECHANICAL/ELECTRICAL SPECIFICATION	OUTPUT
16 Channels	Wavelength: 903 nm	Power consumption: 8W	Data Points generated: 300,000 points per second in single return mode, 600,000

			points per second in dual return mode
100 m range		Operating voltage: 9V-18V	100 Mbps Ethernet Connection
Accuracy ± 3 cm		Weight: 830g	
Vertical field of view: 20°		Operating temperature: -10°C to 60° C	
Horizontal field of view: 360°			
Rotation rate: 5 Hz -20 Hz			

Table 2: Velodyne specifications

2.1.1.1 The Point cloud

The velodyne LiDAR provides information of the 3D surrounding space in a row data format. However, the 3D space must be converted to a 3D image in order to be interpreted and processed: this can be done by means of point clouds.

Point clouds are a series of different "points", similar to pixels in a digital picture. Differently from pixels, a LiDAR point is made up of three coordinates — X, Y and Z — which refers to a specific position in a three-dimensional space. The union of these points makes up the point cloud, which can be easily interpreted and managed under a software point of view. The point cloud is used as starting point for the generation of the occupancy grid map of the environment surrounding the vehicle. Please, refer to the following paragraphs for further details. An example of point cloud is reported in the image below [Fig. 13]:

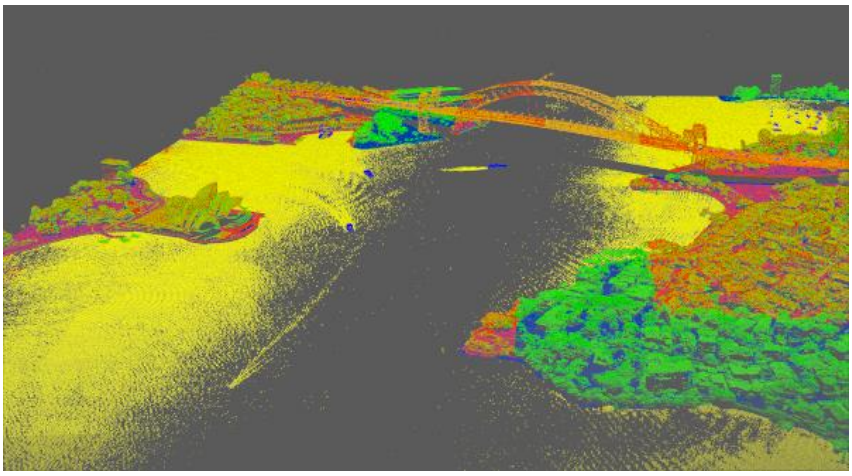


Figure 13: point cloud example

2.1.2 GPS sensor: Duro Inertial

The device used for the vehicle localization at each instant is the Duro Inertial. It combines the following technologies: GPS, IMU and RTK.

For an overview on GPS working principle, please, refer to the previous chapter; the inertial measurement unit (IMU) is used in our application to measure the orientation of the vehicle (Roll – Pitch – Yaw angles) using a combination of accelerometers and gyroscopes; it is also used to allow a satisfactory working of the GPS receiver also in critical condition when GPS-signals are unavailable. Plus, due to the GPS integration it gives the capability to gather as much accurate data as possible about the vehicle's current speed, turn rate, heading, inclination and acceleration.

The RTK (Real Time Kinematics) functionality, which has been activated in our application, is used to correct errors in the GNSS system: the working principle is based on the measurement of the phase of the signal's carrier wave, providing real-time corrections. In practice, RTK systems use a single base-station receiver and a number of mobile units. The base station re-broadcasts the phase of the observed carrier, and the mobile ones compare their own phase measurements with the one received from the base station.

In short, as shown in the following figure [Fig.14], the Duro Inertial device receives GNSS signals from satellites, the same signals are broadcasted to the RTK Base Station, and via internet the RTK corrections are delivered to the device by means of a GNSS data management software.

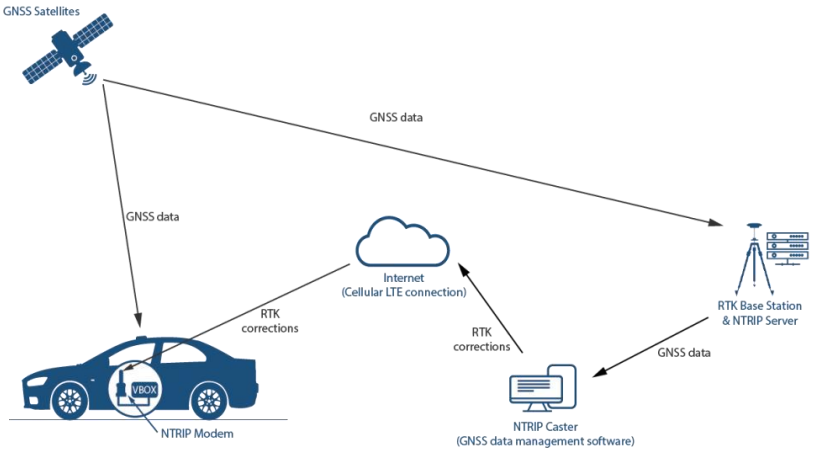


Figure 14: RTK system

The following table [Tab. 3] shows the high accuracy of position and velocity when the RTK is activated.

Performance During GNSS-RTK Outages		Position Accuracy 2-Sigma (m) RMS		Velocity Accuracy (m/s) RMS	
Outages	**Prior Position Mode**	**Horizontal**	**Vertical**	**Horizontal**	**Vertical**
1 second	RTK	0.02	0.06	0.035	0.020
5 seconds	RTK	0.05	0.09	0.040	0.030
10 seconds	RTK	0.17	0.16	0.055	0.045

Table 3: RTK accuracy

the Duro Inertial device is, essentially, a dual frequency GNSS receiver integrated with an inertial navigation system: this allows a centimeter-level positioning accuracy in outdoor environment. Plus, Duro Inertial combines Carnegie Robotics LLC (CRL's) SmoothPose™ sensor fusion algorithm, with Swift Navigation's Starling® Positioning Engine, to guarantee a robust positioning system in automotive application, even when there is not a sufficient GNSS availability. Some meaningful specification, taken from the datasheet are shown in the following [Tab. 4]:

Physical & Environmental	
Dimensions	130 mm x 130 mm x 65 mm
Weight	0.8 kg (Cast Al Housing)
Temperature	
Operating	-40° C to +75° C
Storage	-40° C to +85° C
Humidity	95% non-condensing
Sealing	IP67
Vibration[1]	
Operating and Survival (Random Vibe)	7.7 g
Operating and Survival (Sinusoidal Vibe)	5 g
Mechanical Shock[1]	
Operating	40 g
Survival	75 g

Electrical & I/O	
Power	
Input Voltage[2]	10 - 35 V DC
Typical Power Consumption[3]	5.0 W
Antenna LNA Power Specifications	
Output Voltage	4.85 V DC
Max Output Current	100 mA
External Connector Ports	
- 2 x RS232 Serial Ports with Optional Hardware Flow Control	
- Ethernet support up to 100 Mbps	
- PPS, PV, 3 x Event Inputs	
- CANBus with Selectable Termination Resistor	
- Configurable Digital Inputs and Outputs	
- 12 V at 1A and 5 V at 250 mA Power Outputs	

Performance Characteristics	
GNSS Signal Tracking	
GPS L1/L2, GLONASS G1/G2, BeiDou B1/B2, Galileo E1/E5b SBAS (WAAS, EGNOS, GAGAN, MSAS)	
Position Update Rate (GNSS+INS)	Up to 10 Hz
GNSS Data Rates[4]	
Measurements (Raw Data)	Up to 10 Hz
Standard Position Outputs	Up to 10 Hz
RTK Position Outputs	Up to 10 Hz
Swift Binary Protocol (SBP) and NMEA-0183	
Maximum Operating Limits[5]	
Velocity	515 m/s

Table 4: Duro Inertial specifications

2.1.3 NVIDIA TX2

In our project all the computational efforts are entrusted to the NVIDIA Jetson TX2. This kind of device has been chosen mainly because of its high-performance capability. The whole autonomous driving systems works with a frequency of 10 Hz (every 0.1 seconds fresh data are ready to be processed) and the NVIDIA TX2 guarantees the execution of all the needed computations in the time range of 0.1 s.

The NVIDIA® Jetson TX2 System is a combination of performance, power efficiency, integrated deep learning capabilities and rich I/O. The Jetson TX2 is ideal for many applications including: Intelligent Video Analytics, Drones, Robotics, Gaming Devices, Virtual Reality, Augmented Reality, Portable Medical Devices and Autonomous Driving.

The NVIDIA Jetson TX2 main technical features are reported in the following:

- **256 core NVIDIA Pascal GPU**. Fully supports all modern graphics APIs, and is GPU compute capable. The Pascal GPU architecture offers major performance improvements and power optimizations. TX2's CPU Complex includes a dual-core 7-way superscalar NVIDIA Denver 2 for high single-thread performance with dynamic code optimization, and a quad-core Arm Cortex-A57 geared for multithreading. This is the main characteristics that made us opt for this device in our automotive application. The huge computational power both in single-threading and multithreading, for example, allows the execution of for/while loops in a time interval more than acceptable.

- **Advanced HD Video Encoder**. Recording of 4K ultra-high-definition video at 60fps.
- **Advanced HD Video Decoder**. Playback of 4K ultra-high-definition video at 60fps with up to 12-bit pixels.
- **128-bit Memory Controller**.128-bit DRAM interface providing high bandwidth LPDDR4 support.
- **1.4Gpix/s Advanced image signal processing**: Hardware accelerated still-image and video capture path.
- **Audio Processing Engine**. Audio subsystem enables full hardware support over multiple interfaces.

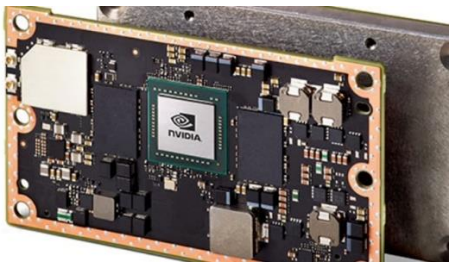


Figure 15: NVIDIA TX2

2.1.4 CAN network

A CAN network has been used to actuate the outputs of the trajectory planner software. The actuation variables are the target speed, converted as throttle position and the steering angle, intended as the angle of the front wheels with respect to the vehicle longitudinal axis.

The Controller Area Network (CAN), invented by Robert Bosch GmbH in 1980 for automotive applications, is an asynchronous serial bus composed by 2-wire differential bus; the network is characterized by the absence of node addressing; thus, the network is based on the broadcasting concept.

To allow a better cost control, CAN communication is normally based on two separate hardware components:

1) The CAN protocol controller, responsible for the ISO/OSI data-link layer
2) The CAN transceiver, responsible for the ISO/OSI physical layer

The CAN protocol controller is typically embedded in an MCU, it handles all the data-link layer aspects of the protocol, it is responsible for frames transfer/reception, error handling, communication with the MCU through a register-based interface. Plus, it supports polling/interrupt, it can use dedicated RAM to buffer incoming/outgoing messages and it support intelligent incoming message reception.

The CAN transceiver, on the other hand, is typically a dedicated component outside the MCU and handles all the physical layer aspects of the protocol.

From a high-level point of view, the type of communications provided by the communication services which implement the functionalities to send/receive CAN frames are:

- Message-based communication: CAN frames are sent once given the ID, data length, and payload bytes

- Signal-based communication: higher level objects, representing application data, are sent. An example of signals set, which has been used in our application, is the following:

```
BO_ 660 SCU_ACTUATION: 8 Vector__XXX

SG_ SCU_ACTUATION_TargetSpeed : 31|0@0+ (1,0) [0|255] "∞" Vector__XXX

SG_ SCU_ACTUATION_TargetAngle : 15|32@0- (0.1,0) [-3276.8|3276.7] "∞" Vector__XXX
```

The two signals above, contained in the actuation signal set, ensure the actuation of the steering speed (angular speed at which the steer is rotated), and the steering angle.

Each signal is defined by the following attributes:

- **Signal name**: i.e., TargetSpeed
- **Length in bit**: i.e., 15 bit long (the specific message start from the 32° bit)
- **Byte order** (the @0 notation states a big-endian order, with most significant bit first)
- **Scaling Factor**: i.e., 0.1 is a multiplicative factor
- **Offset:** i.e., 0
- **Minimum value:** i.e., -3276.8
- **Maximum value:** i.e., 3276.7

2.2 Software considerations

As regards software considerations, there is need to underline that our job was not to work on the occupancy grid map generation, but this latter is given as input through a program already developed, whose main steps are described in the following. In order to understand properly the way in which the map is generated, it is essential an overview on what an occupancy grid map is. Plus, some considerations on the used reference frames will be provided, to understand the main steps executed by the occupancy grid map generator.

2.2.1 Used Reference frames

The local reference frames are vehicle-based. The vehicle motion involves a roto-translation the local frames which are:

- *Velodyne frame*
- *Base_link frame*
- *GPS frame*

All these three reference systems move at the same way according to the vehicle motion, the only difference stays in the point in which the origin of the systems is located. As the name suggests, the *velodyne frame* origin is

located exactly in correspondence of the Velodyne LiDAR, the *base_link frame* origin is the same of the velodyne one bust translated on the ground, the *GPS frame* origin is located on the ground and in correspondence of the GPS sensor, which is located on the top of the vehicle.

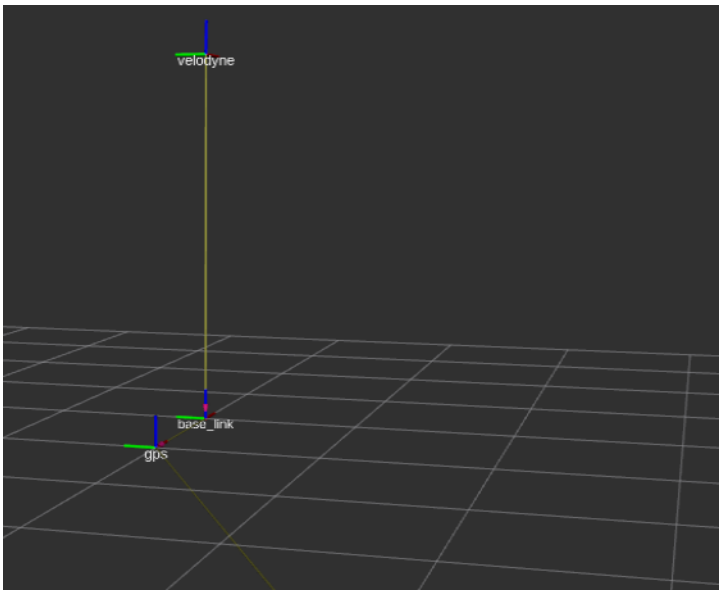


Figure 16: Velodyne, Base link and GPS frames

The global reference frame is named *map*: it stays fixed even when the vehicle is moving.

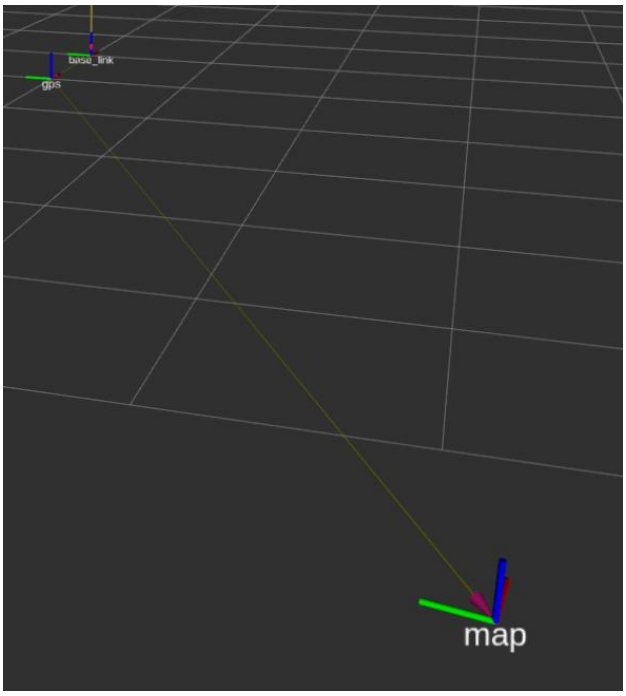


Figure 17: map global frame

The most frequent transformations performed in the occupancy grid generator (but also in the trajectory planner) are the ones from *GPS frame* to *map frame* and the other way around. For instance, in order to have the coordinates of a point in the GPS frame, given its coordinates in the map frame the procedure to perform is the following one:

1) The first step is to express the coordinates of the point in a temporary reference frame, which has the same origin of the local frame (GPS frame), but with the axis aligned with the global frame. This can be easily done through an algebraic sum:

$$x\_local\_temp = x\_global - gps\_x\_pos;$$

$$y\_local\_temp = y\_global - gps\_y\_pos;$$

where gps_x_pos and gps_y_pos denote the coordinates of the local frame origin in the global one. Please, note that in this case the z-axis has not been taken into account because the two reference frames are supposed to be on the ground (z is in both cases equal to zero).

2) The second transformation consists in the rotation of the temporary reference frame, the rotation between the reference systems is performed as follows:

$$x\_local = x\_local\_temp\ cos(yaw) - y\_local\_temp\ sin(yaw)$$

$$y\_local = x\_local\_temp\ sin(yaw) + y\_local\_temp\ cos(yaw)$$

The same result can be obtained considering a rotation matrix around the z-axis.

2.2.2 The Occupancy grid map: an overview

An occupancy grid map is a discretization of the surrounding environment in cells; each cell indicates the probability of occupancy. The map is used to differentiate the free space from the portion of environment occupied by obstacles. The standard approach consists in dividing the map in cells, and

each cell is represented in grayscale with a tonality depending on the probability of occupancy of that cell: the black color is used to represent a cell 100% occupied, white color is used for cells 100% free.

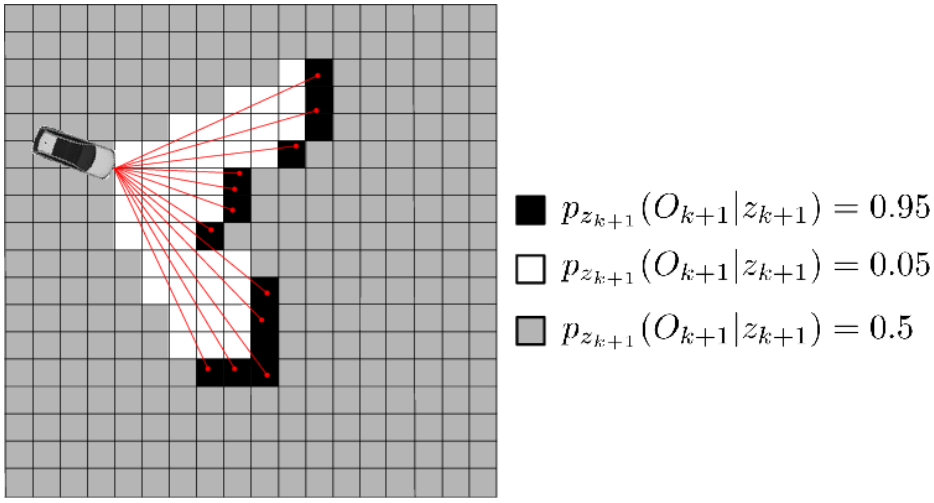


Figure 18: standard occupancy grid map representation

In the following we present a theoretical model which is the one used most frequently in literature for the occupancy grid map generation. In practice, a pragmatic approach, which is presented in the next paragraph, has been used.

This is a mainly statistical analysis whose final objective is to determine the occupancy probability of each single cell. The information assumed to be known is the position of the vehicle and therefore of the sensors (determinable via GPS/INS) and the data coming from the sensors (the position in OGCS - Occupancy Grid Coordinate System - of the detected obstacle detection). The assumptions made are:

1) Each cell is assumed either free or occupied

2) The cells are independent (independence between binary random variables)

Referring to the above assumptions and through the use of the Bayes filter, the model that establishes the probability of occupancy of each cell and then of the entire grid can be determined.

The description of the mapping task is formalized as:

$$m* = \ argmax_m \ P(m|x1, z1, \dots, xt, zt)$$

Where $m*$ represents the map of the environment, given sensor data; x1… xt is the dataset used to indicate the vehicle pose provided by the GPS; z1… zt represents the sensor data.

Please, note that each cell is a binary random variable and:

$P(mi) = 1$ indicates and occupied cell, $P(mi) = 0$ indicates a free cell, $P(mi) = 0.5$ states the fact that no information related to that cell is provided.

The statistical model makes use of the joint probability distribution:

$$P(m) = P(m1, m2, .., mn)$$

Thus, the probability distribution of the whole occupancy grid map is given by the probability that cell 1 (m1) is occupied AND cell 2 is occupied, … , and cell n is occupied.

Using the assumption of the independence between the cells one can write:

$$P(m) = \Pi \ P(mi)$$

and the estimation of the occupancy grid map from data can be written as:

$$P(m|z1:t\ , x1:t) = \ \Pi \ P(mi|z1:t\ , x1:t)$$

Where $z1:t$ represents the sensor data, $x1:t$ represents the poses of the sensor, mi is the binary random variable related to a single cell.

An additional step is to use a Bayes filter in order to estimate a binary random variable. By means of a static state binary Bayes filter one can write:

$$P(mi|z1:t\,,x1:t) = \frac{P(zt|mi\,,z1:t-1\,,x1:t)P(mi|z1:t-1,x1:t)}{P(zt|z1:t-1,x1:t)}$$

By means of the Markov theorem and the Bayesian rule one can obtain the following form:

$$P(mi|z1:t\,,x1:t) = \frac{P(mi\,|zt,xt)\;P(zt|xt)P(mi|z1:t-1\,,x1:t-1)}{P(mi|xt)P(zt|z1:t-1\,,x1:t)}$$

Where $P(mi\,|zt,xt)$ is the probability of a cell being occupied given the current observation and the current pose; $P(zt|xt)$ is the probability of an observation given the current pose, this term is kind of tricky to be estimated; $P(mi|z1:t-1\,,x1:t-1)$ is the probability that a cell is occupied given all the past information; $P(mi|xt)$ is the probability of occupancy of a cell knowing the current pose, this term is also tricky to be estimated; $P(zt|z1:t-1\,,x1:t)$ is the probability of an observation given past positions and observations.

Then, computing the ration between the probability of occupancy and the probability that a cell is not occupied:

$$\frac{P(mi|z1:t,x1:t)}{P(-mi|z1:t,x1:t)} = a*b*c$$

Where $a = \frac{P(mi|zt,xt)}{1-p(mi|zt,xt)}$; $\quad b = \frac{P(mi|z1:t-1,x1:t-1)}{1-p(mi|z1:t-1,x1:t-1)}$; $\quad c = \frac{1-P(mi)}{p(mi|)}$;

a represents the term which uses the current observation (zt) and the current pose of the sensor ($|zt,xt$).

b represents the recursive term, which contains all the data up to $t-1$.

c represents the prior information, that is what one can say about the map without any observation.

At this point, the log odds notation is introduced for efficiency reasons, in this way the occupancy grid update happens faster;

The notation is the following one:

$$l(x) = log\ \frac{P(x)}{1-P(x)}$$

the product between a, b, c terms is transformed into a sum and the final result is:

$$l(mi|z1:t, x1:t) = l(mi|zt, xt) + l(mi|z1:t-1\ , x1:t-1) - l(mi)$$

$l(mi|z1:t, x1:t)$ is the current estimation of the cell, information which is stored. $l(mi|zt, xt)$ is the log odds of the inverse sensor model, which is the information related to the current observation. $l(mi|z1:t-1\ , x1:t-1)$ is the recursive term and $l(mi)$ represents the prior information.

In short, one obtains:

$$l_{t,i} = inv_{sensor_{model}(mi,xt,zt)} + l_{t-1,i} - l_0$$

$l_{t,i}$ is the new state of cell i at time instant t, $inv_{sensor_{model}(mi,xt,zt)}$ is the term depending on the current observation, $l_{t-1,i}$ is related to the past history and l_0 is the prior information.

The model, expressed as above, results highly efficient since only algebraic sums must be computed, in this way operations can be easily parallelized.

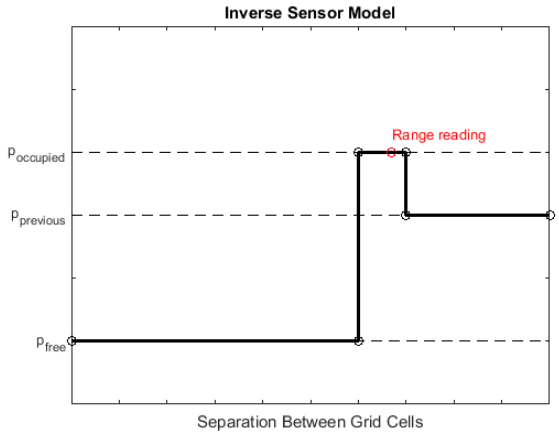


Figure 19: inverse sensor model

2.2.3 Occupancy grid node

According to the current implementation, the occupancy grid is represented by a certain number of cylinders placed in correspondence of the detected objects.

Circles (in 2D representation) have been used as data structure for the obstacle detecting because of an efficiency reason. In principle, only two kinds of information are needed for the representation: the coordinates of the center and the radius. Thus, the program of trajectory planning receives in input a data structure easily manageable and easy to elaborate and process. The theoretical canonic procedure, described in the previous paragraph, has not been considered in practice in our work because the data to be transmitted would have been of a consistent dimension (there would have been the need of transmitting for each cell discretizing the space information related to the occupancy probability).

- *occupancy_grid_node*: this node is responsible for creating the occupancy grid. The main steps are the following:

1) The LiDAR row data, expressed in the velodyne frame, is transformed in Point cloud data which is still expressed in the local reference frame.

2) The Point cloud data is transformed to map frame but not considering the translation vector from gps to map, i.e. the point cloud is translated in the gps frame and rotated according to the vehicle orientation with respect to the fixed frame map (not considering roll angle). This step is necessary in order to eliminate the apparent rotation of the point cloud; by means of this transformation the point cloud remains fixed even though the vehicle is moving.

3) The point cloud is converted into an image, in this way the 3D point cloud representation is associated to a bi-dimensional representation, which is denoted as a bird-eye view. This step is executed by applying first a convolution matrix, then the Hoshen Kopelman filter. In this way, by means of the convolution matrix, it is possible to highlight all the different blobs present at each instant of time; a blob is a section of point cloud in which there is associated a cluster of data. On the other hand, the Hoshen Kopelman filter is responsible for the extraction of the blobs from the image.

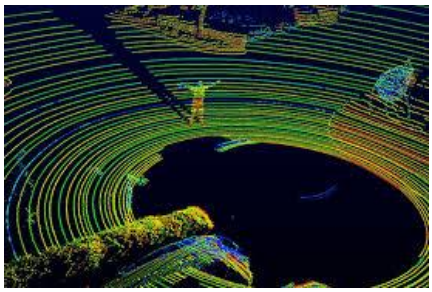


Figure 20: Blobs example

4) To each blob a rectangular bounding box is associated

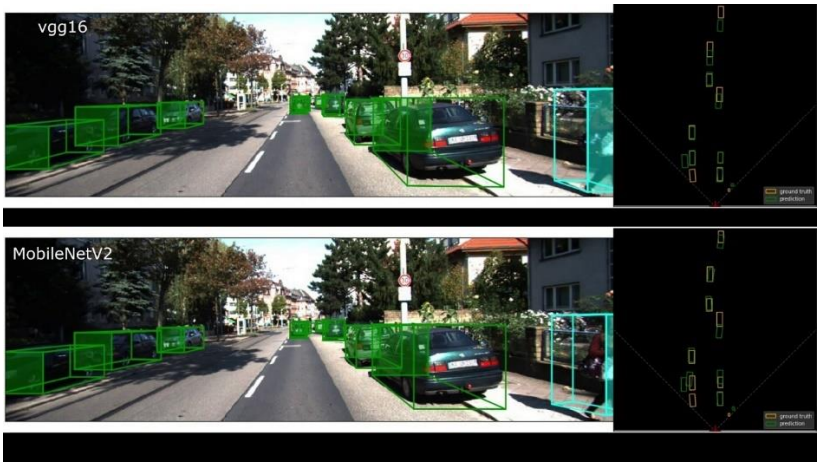


Figure 21: Bounding box representation

5) Each blob is represented as one or more circles, depending on its size.

The algorithm determining the circles position works as follows: the bounding box is divided in half, if the central point of the dividing line is inside the blob a circle is centered in that point. The initial bounding box is thus divided into two parts, for each part the same algorithm is applied. So, iteratively, a series of circles are positioned on each blob. The center of each circle is then translated by considering the position of the GPS with respect to the fixed reference system (map frame). As a result, the apparent motion is eliminated.

6) The actual occupancy grid is created: an array of confirmed elements, each

with the following parameters:

- id

- center

- radius

- volume

- time_stamp

In order to compute the volume of the cylinder that has as base the found circle, the high is extracted from the point cloud.

7) Every time new data comes in from Lidar, the current occupancy grid (an array of elements) is compared to the array of confirmed elements to determine whether or not each element has already been identified. The update/erase logic is as follows:

a) the element is found inside the array of confirmed elements:

the element is again confirmed and its time_stamp is updated to the current time instant.

b) the element is NOT found:

The new element is added to the array of confirmed elements and the id is incremented by 1.

c) a confirmed element has not been detected for at least 1 second:

The confirmed element is erased

The discriminant for (a) or (b) is:

$$\Delta c = (elem.xc - confirmed_{el}.xc)^2 + (elem.yc - confirmed_{el}.yc)^2$$

Currently, for an element to be found Δc must be less than 0.7 m.

Δc indicates if the new detected circle is from fresh data or if It is still related to a section of the object already detected.

Please, consider that fresh data are acquired each 0.1 seconds, since the whole system is working at a frequency of 10 HZ. This is due to the fact that 0.1 s is the needed time required by the processor to do all the computation needed for the generation of a valid trajectory from the current position to the first waypoint.

An image of an occupancy grid elaborated by the *occupancy_grid_node* is shown:

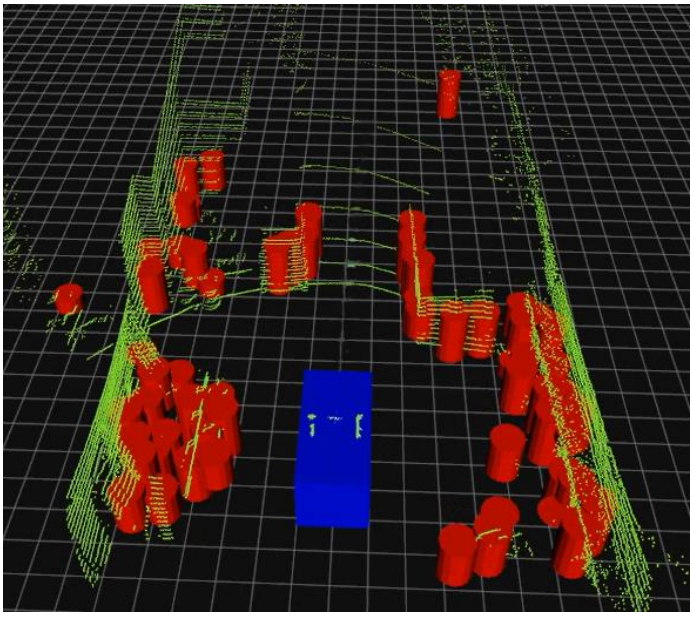


Figure 22: occupancy grid returned by the occupancy_grid_node

2.2.4 ROS: an overview

The implementation of the designed algorithms of detection and planning has been done in ROS (Robotic Operating Systems).

ROS is an open-source, meta-operating system for robotic applications, but is also used in the autonomous driving field. It provides the services of

an operating system, including hardware abstraction, transferring of messages between processes, and package management.

ROS can be described by means of three different levels: the Filesystem level, the Computation Graph level, and the Community level: we will provide a brief description of the first two.

ROS Filesystem Level

The filesystem level covers resources which are on disk, such as:

- Packages: Packages are the main unit for organizing software in ROS. A package contains ROS runtime processes (*nodes*), a ROS-dependent library, datasets, configuration files.
- Metapackages: Metapackages are specialized Packages containing a group of related packages.
- Package Manifests: Manifests (package.xml) provide metadata about a package, including its name, version, description, license information, dependencies.
- Message (msg) types: defining the data structures for messages sent in ROS.
- Service (srv) types: defining the request and response data structures for services in ROS.

ROS Computation Graph Level

The *Computation Graph* level is the peer-to-peer network of processes which are processing data. it includes:

- Nodes: Nodes are processes performing computations. ROS is designed to be modular, so a ROS system may include many nodes; some of the nodes present in our application are:

a) *duro_node*: this node reads raw data from the socket and processes them using the

   Swift binary protocol library in order to obtain the current position, orientation and Velocity of the vehicle.

b) *static_tf_node1*: this node broadcasts the transform between velodyne and base_link.

c) *static_tf_node2*: this node broadcasts the transform between base_link and gps.

d) *occupancy_grid_node*: this node is responsible for creating the occupancy grid

e) *trajectory_planning_node*: this node is responsible for creating the trajectory as output: this is the node in which the designed algorithm of trajectory planning has been translated

- Messages: Nodes communicate with each other by passing messages. A message is simply a data structure, comprising typed fields, similar to a data structure used in C (ROS supports both primitive types and customized ones).
- Topics: the logic used in ROS to transport messages is a publish / subscribe semantic. When a node sends out a message it publishes it to an appropriate topic. On the other hand, a node that is interested in a certain kind of data subscribes to the appropriate topic.
- Services: Request / reply, differently from the publish/subscribe logic, is done by means of services, which are defined by a pair of message structures: one for the request and one for the reply. A providing node

offers a service under a name and a client uses the service by sending the request message and awaiting the reply.

- Bags: Bags are used for saving and playing back ROS message data. Bags are used to test algorithms; in our application they have been used to analyze the results of the *trajectory_planning_node*, in practice they have been used mainly for debugging operations.

Chapter 3: The trajectory planning algorithm

3.1 Trajectory planning algorithm design and implementation

In this chapter we present the implementation of the trajectory planning algorithm.

The developed program receives as input the data from the occupancy grid map and returns the signals which are actuated through the CAN bus, which are the steering angle and the target speed. Every 100 ms the *trajectory_plannig_node* receives fresh input data from the sensors (GPS vehicle position and orientation) and an updated occupancy grid; thus, with a frequency of 10 Hz the actuation variables are sent to the vehicle's ECU.

As anticipated in the previous chapter, the node receives information of the surrounding obstacles through an array of objects, each object represents a circle whose most relevant data is its center and radius. An additional input is a list of waypoints, representing the ideal path. In short, the core of the algorithm is to determine an obstacle – avoidance trajectory from the vehicle current position to a certain waypoint.

In the following figure [Fig. 23], a schematic representation of the inputs/outputs is given.

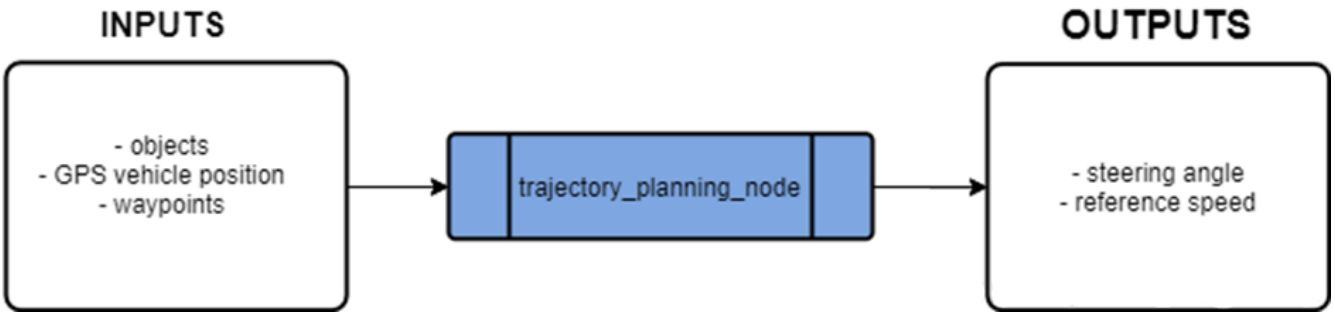


Figure 23: trajectory_planning_node I/O

The main performed steps for the output generation are listed in the following: each one of them is carefully analyzed in the next paragraphs; plus, for clarity reasons, the most critical operations will be presented through flow charts, whose legend is:

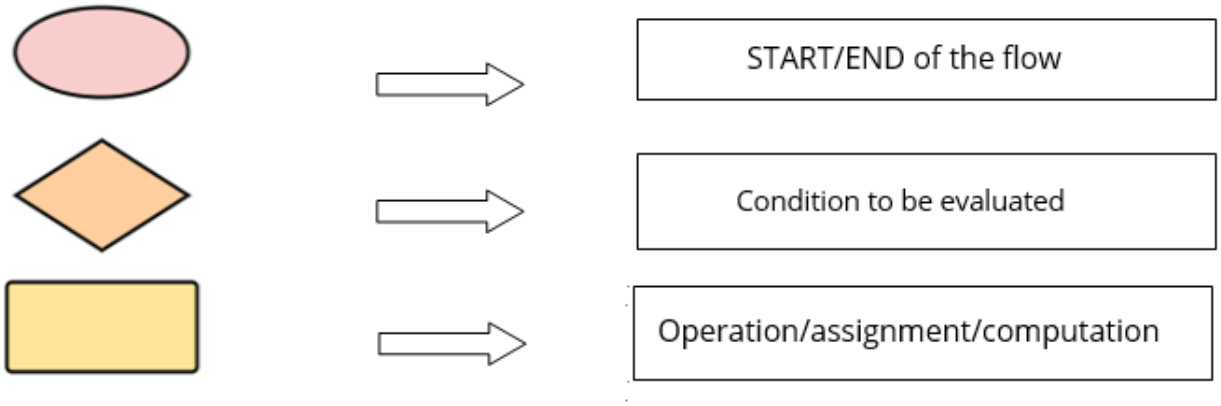


Figure 24: flow charts legend

The trajectory planning fundamental steps are:

1) Waypoints and objects loading
2) Get reference waypoint
3) Get length to goal
4) Transform waypoint's reference frame
5) Get optimal path and speed profile
6) Get look ahead index
7) Find steering angle to actuate
8) Find reference speed to actuate
9) Check emergency braking condition
10) Set steering angle PI configuration
11) Set actuation variables to the CAN bus

3.1.1 Waypoints and objects loading

The very first step consists in the waypoints list loading: this operation is performed only once when the program is executed and the vehicle is started. Waypoints are pre-recorded by driving the vehicle in a non-autonomous way around the established path; then, a resolution variable (*wp_res*) is set – in our application it is set to 1 m. in short, during the non-autonomous drive every meter the position given by the GPS sensor is saved in the map reference frame: these X,Y coordinates (the Z coordinate is neglected) represent the waypoint list given as input to the

trajectory_planning_node. In a real-case scenario, waypoints may be provided, for example, by specific Google Map API. By way of explanation, in the following figure [Fig. 25] two waypoints of a generic path are represented.

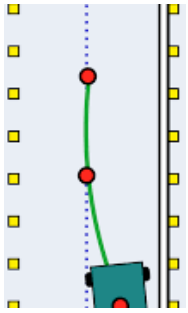


Figure 25: example of waypoints

Then, each time the program is executed that is every 100 ms, an array of circles (*objects*) is read as input. All the circles, in principle, cover the whole space that results occupied by an obstacle. The structure "*objects",* used by the trajectory planner, may be graphically represented as follows:

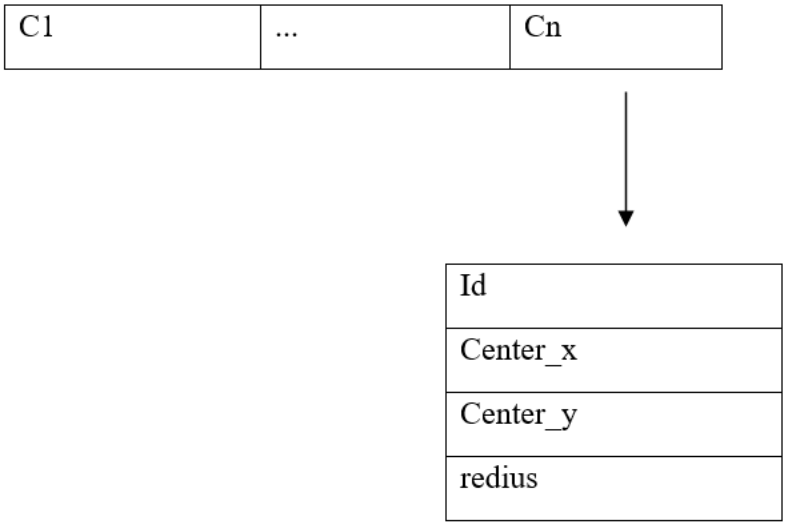


Figure 26: "objects" structure to represent obstacles on the path

3.1.2 Get reference waypoint

The block diagram of the function *get_reference_wp* is provided in the following figure [Fig. 27] :

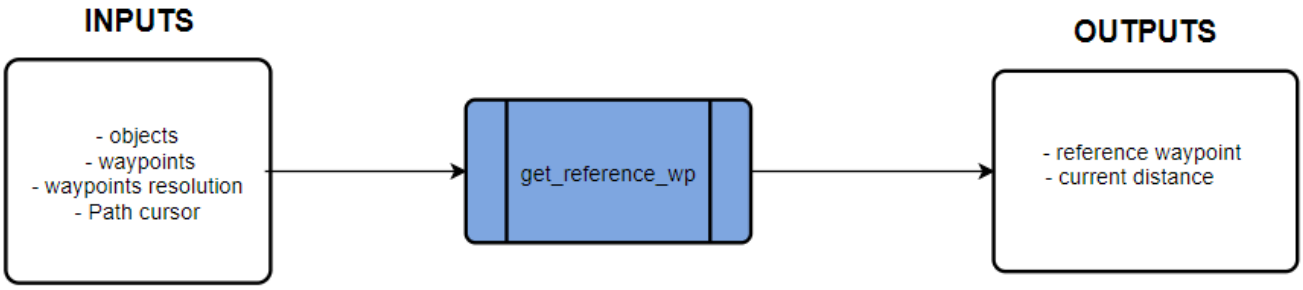


Figure 27: get_reference_wp I/O

The purpose of this function is to return, given the waypoint list, a reference waypoint: then the collision-free trajectory is generated from the vehicle current position to the reference waypoint. A further output is the linear distance between the vehicle and the chosen waypoint (*current_dist*): this variable is used to compute the length to the final waypoint (*len_to_goal*), needed to set a correct speed actuation value; this aspect is explained in the related paragraph.

The inputs are the list of objects (obstacles), the waypoints list and their resolution and the Path cursor; this latter represents the position of the car within the waypoints list.

It is basically an array index which indicates the waypoint closest to the current car position; The accuracy of this position depends on the resolution of the array of waypoints. To find the path cursor we evaluate the distance from the GPS car position and the considered waypoint (i.e., waypoints[pathCursor]) and when this distance is lower than 2.5 m, the path cursor is considered to be found. The value of 2.5 is chosen because we need to consider that the GPS is mounted approximatively 2 meters behind the car front axis, and considering a waypoint on the vehicle body does not make any sense.

The goal of this strategy is to select the reference waypoint at the furthest point where an object is identified. This guarantee, for instance, that in a sharp bend the trajectory planner has vision of the entire bend and not of only a portion: if the waypoint was located, for example, in the middle of the bend the car would have a limited vision and the second half of the bend may be driven by means of unnecessary dangerous maneuvers. In the following figure [Fig. 28] it is shown an example of reference

waypoint (the white dot represents the chosen waypoint and it is located a little behind the last object on the path detected by the LiDAR in that moment):

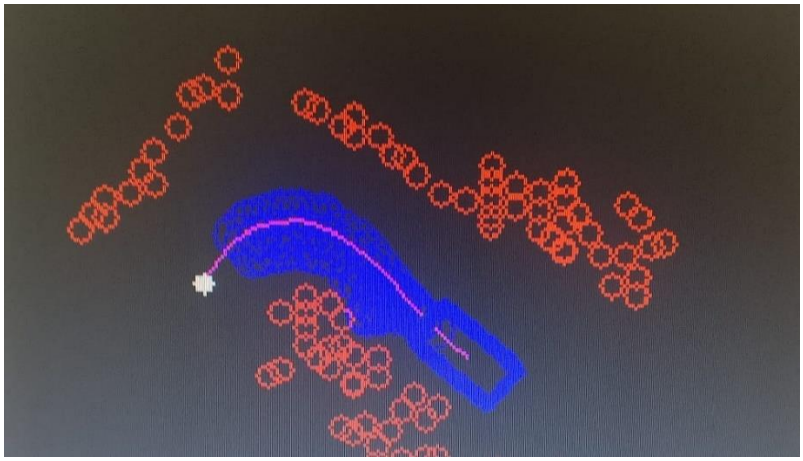


Figure 28: reference waypoint

To explain the function a flow chart is provided in Figure 29.

In order to extract the appropriate waypoint, we initialize the following variables:

- *min_dist* = 100, it represents the linear distance between the considered waypoint and the nearest obstacle (circle).
- *wp_dist* = 100, it represents the linear distance between the car and the considered waypoint.

the variables mentioned above are initialized to a high value, so that they will be re-computed at least once. Then, the assignment *i* = *pathCursor+30.0/wp_res* is performed: We start from a point 30 meters in front of the car and then we have first to check whether the considered point is beyond the end of the path or not. If this is the case, then we need

to choose the last point of the array (i.e. waypoints.size() -1), we compute the *current_dist* and terminate. If this is not the case, we start going backward from the initial point at 30 meters until either *min_dist* < 9 or *wp_dist* < 18: the exit conditions are if the distance with respect to an obstacle is less than 9 meters (we have reached a waypoint close to the last identified object) or if the distance between the vehicle and the waypoint is less than 18 meters (we do not have to keep analyzing the waypoint list backward since it does make any sense to consider a waypoint too much close to the car).

Plus, as the reference waypoint is identified backwards it is necessary to consider that this would end up selecting the waypoint slightly beyond the last identified object. For this reason, another 6 meters is subtracted from the identified point, taking care to perform this operation only if the exit condition from the previous cycle was the *min_dist* one.

Finally, the current_dist is computed as a linear distance:

$$current\ dist = \sqrt{(wp_x - gpsPos_x)^2 + \left(wp_y - gpsPos_y\right)^2}$$

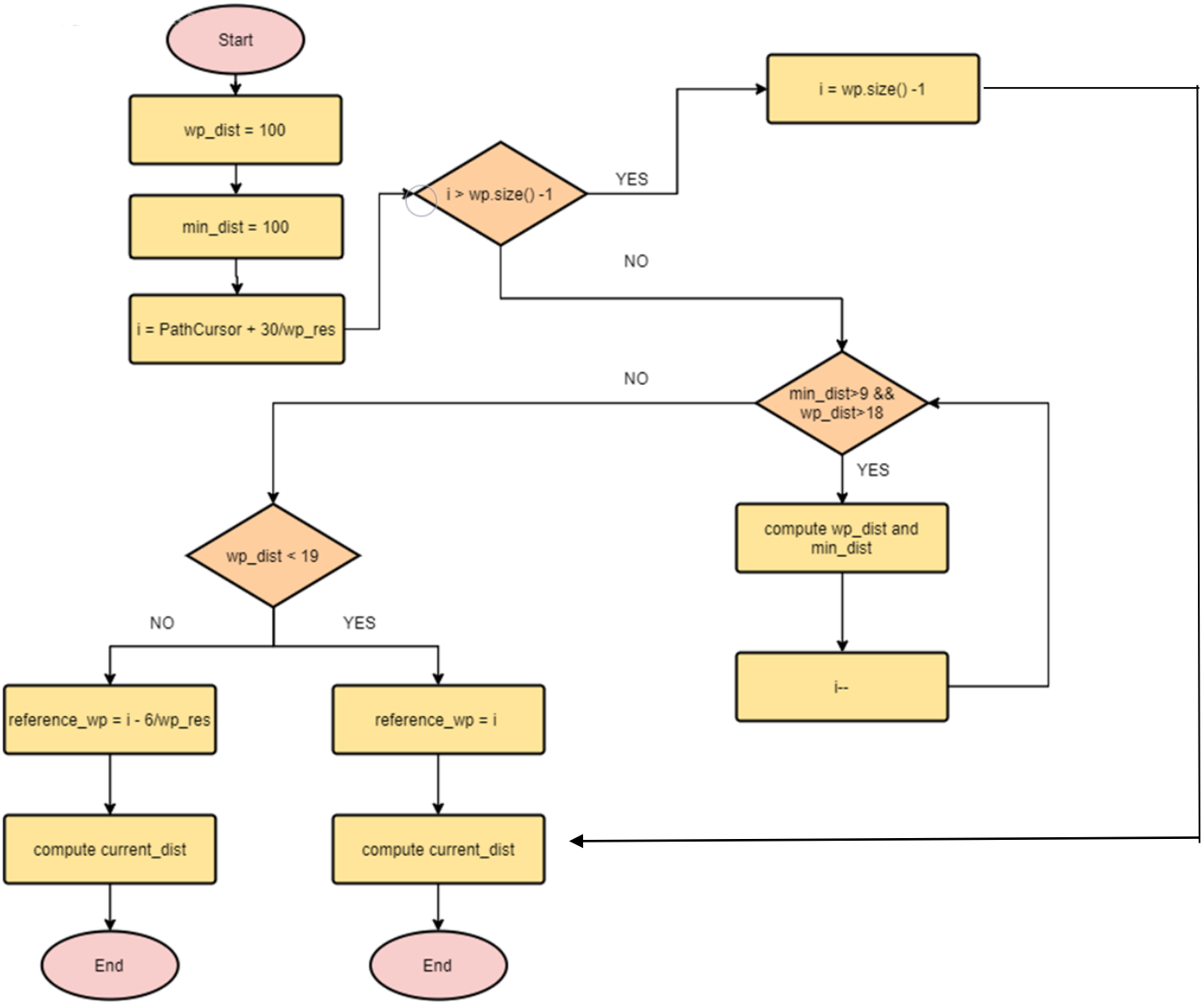


Figure 29: get_reference_wp flow chart

3.1.3 Get length to goal

This function computes the length to goal, that is the final waypoint of the list, starting from the *current_dist* and adding all the rest of the path (calculated as sum of linear distances between consecutive waypoints).

```
len_to_goal = current_dist;
for (size_t i=wp_idx; i<waypoints.size()-1; i++) {
        len_to_goal = len_to_goal + sqrt(pow(waypoints[i+1][0] - waypoints[i][0],2)
                    + pow(waypoints[i+1][1] - waypoints[i][1],2));
}
```

Figure 30: len_to_goal computation

The *len_to_goal* variable measures approximatively, since the distance between waypoints is computed linearly, how many meters the vehicle has to cover in order to reach the final waypoint of the path. This variable is used when the reference speed is sent in actuation: if the *len_to_goal* is sufficiently low (the vehicle has almost reached out its destination), the reference speed will be set to zero, so that the vehicle would reach the final point of the path with a null velocity.

3.1.4 Waypoint reference frame transformation

The reference waypoint is transformed in the GPS reference frame (local frame) since at the next step the function generating the geometric path, which is a spiral, from the current vehicle position to the reference waypoint, happens to work in local frame: the function only receives as input the waypoint in the local frame and the starting vehicle position in the global frame is not provided.

3.1.5 Get optimal path and speed profile

At this stage the geometric path to reach the reference waypoint is generated; as shown in the following input/output representation, given the reference waypoint previously determined, the *get_optimal_path* function returns the optimal trajectory, a Boolean variable indicating whether the emergency braking procedure has to be activated, and a speed profile the vehicle should follow in order to execute the determined trajectory. The I/O representation is shown in the following figure [Fig. 31]:

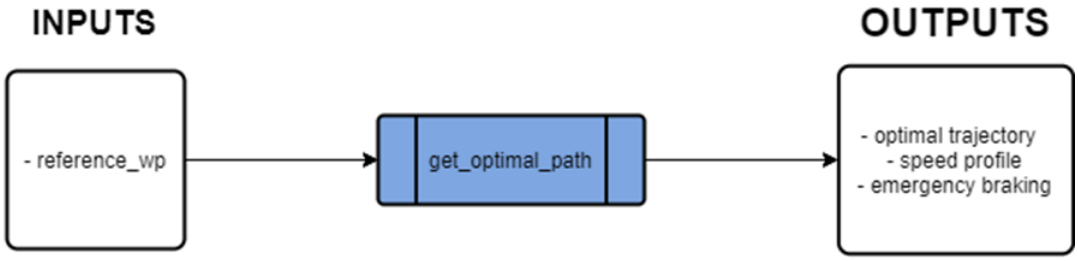


Figure 31: get_optimal_path I/O

Please, refer to the flow chart shown in Figure 32 in order to fully understand the operations performed to obtain the outputs listed above.

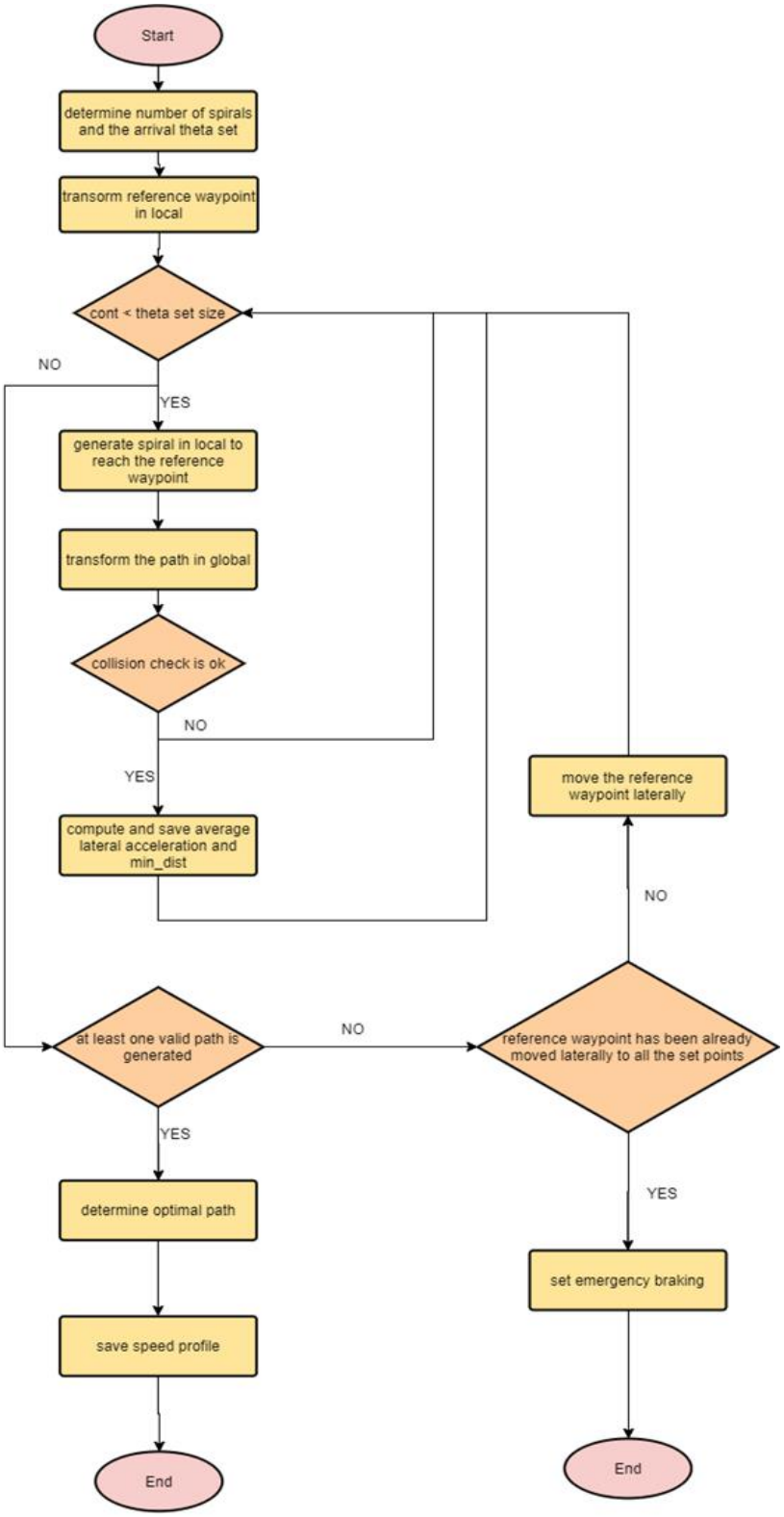


Figure 32: get_optimal_path flow chart

The first step consists in determining the number of possible geometric paths which must be computed to reach the reference waypoint; we decided to generate 11 trajectories with a delta angle between them of 15 degrees (reaching in this way a maximum final angle of +-75 degrees). Thus, at first analysis, a spiral set is generated: this initial set may or may not contain the optimal trajectory which guarantees the obstacle avoidance; an example of spiral set from the current position to the waypoint is shown in the following figure [Fig. 33]:

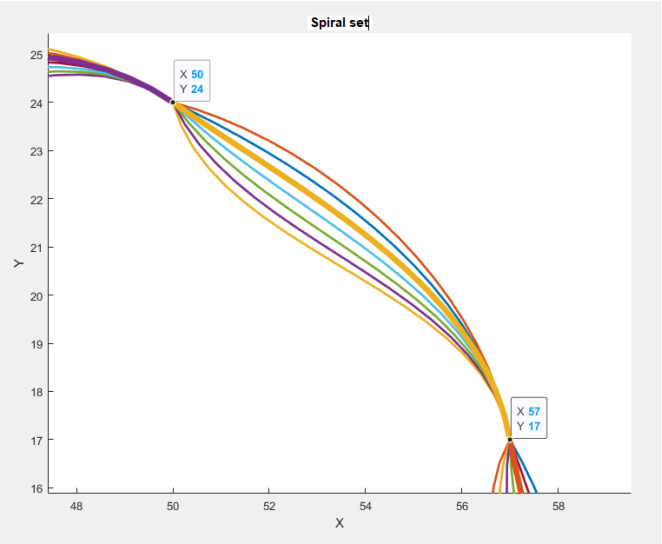
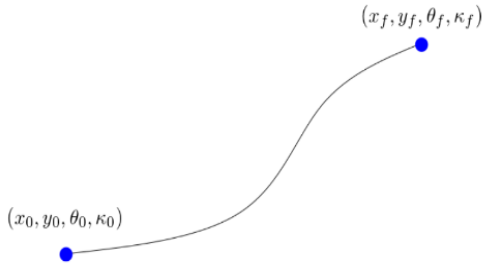


Figure 33: spiral set example

Once determined the spiral set cardinality, once transformed the reference waypoint in the local frame, the first spiral of the set (associated with the first final angle *theta*) is generated. Thus, by means of the *theta* variation of 15° the whole spiral set is computed in a loop. Let us, now, consider a single trajectory generation:

The main objective is to generate a geometric path that must fulfill some constraints on the curvature, this is necessary to guarantee a comfortable

autonomous driving. One mathematic entity which allows simple curvature constraints checking is the spiral. Spirals are defined by their curvature *k* as a function of arc length *s*. In the following, as function of the arc length, an algebraic representation of the spiral, of the x and y location is provided:



$$\kappa(s) = a_3 s^3 + a_2 s^2 + a_1 s + a_0$$

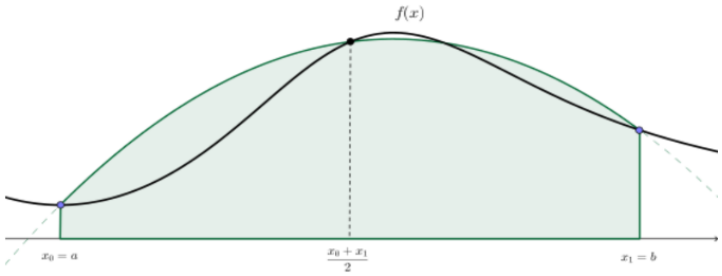
$$x(s) = x_0 + \int_0^s \cos\big(\theta(s')\big) ds'$$

$$y(s) = y_0 + \int_0^s \sin\big(\theta(s')\big) ds'$$

Figure 34: spiral algebraic representation

Please, notice that boundaries on the initial and ending state are related to the point coordinates, the curvature and the theta angle.

Since spiral position does not have a closed form solution, Simpson's rule may be used to numerically evaluate the Fresnel integrals: the rule consists in dividing the integration interval into *n* regions and in evaluating the function at each region boundary.



$$\int_0^s f(s')ds' \approx \frac{s}{3n}\left(f(0) + 4f\left(\frac{s}{n}\right) + 2f\left(\frac{2s}{n}\right) + \cdots + f(s)\right)$$

Figure 35: Simpson's rule

Applying Simpson's rule with n=8, θ(s) has a closed form solution expressed as follows:

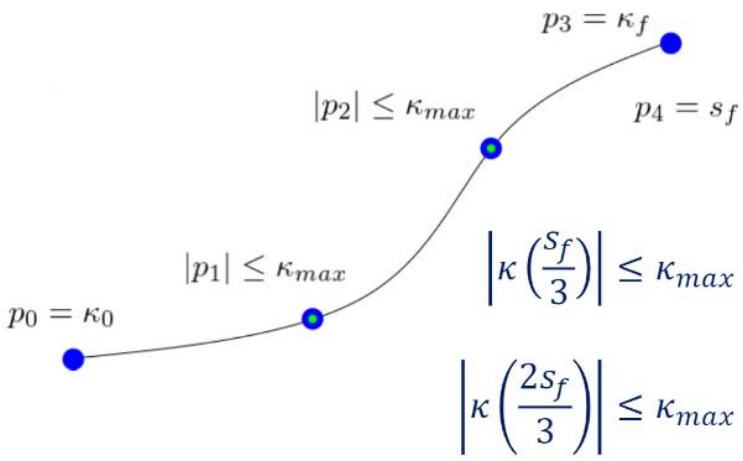
$$\theta(s) = \theta_0 + \int_0^s a_3 s'^3 + a_2 s'^2 + a_1 s' + a_0 ds'$$
$$= \theta_0 + a_3 \frac{s^4}{4} + a_2 \frac{s^3}{3} + a_1 \frac{s^2}{2} + a_0 s$$

Thus, substituting the integral for the spiral position one can write:

$$x_s(s) = x_0 + \frac{s}{24}\left[\cos(\theta(0)) + 4\cos\left(\theta\left(\frac{s}{8}\right)\right) + 2\cos\left(\theta\left(\frac{2s}{8}\right)\right) + 4\cos\left(\theta\left(\frac{3s}{8}\right)\right) + 2\cos\left(\theta\left(\frac{4s}{8}\right)\right) + 4\cos\left(\theta\left(\frac{5s}{8}\right)\right) + 2\cos\left(\theta\left(\frac{6s}{8}\right)\right) + 4\cos\left(\theta\left(\frac{7s}{8}\right)\right) + \cos(\theta(s))\right]$$

$$y_s(s) = y_0 + \frac{s}{24}\left[\sin(\theta(0)) + 4\sin\left(\theta\left(\frac{s}{8}\right)\right) + 2\sin\left(\theta\left(\frac{2s}{8}\right)\right) + 4\sin\left(\theta\left(\frac{3s}{8}\right)\right) + 2\sin\left(\theta\left(\frac{4s}{8}\right)\right) + 4\sin\left(\theta\left(\frac{5s}{8}\right)\right) + 2\sin\left(\theta\left(\frac{6s}{8}\right)\right) + 4\sin\left(\theta\left(\frac{7s}{8}\right)\right) + \sin(\theta(s))\right]$$

In order to determine the spiral parameters, we formulate an optimization problem, subject to the curvature constraints; the curvature constraints are imposed at 1/3rd and 2/3rd of the way along the path as shown below:



An additional constraint is referred to the bending energy: its minimization guarantees an even curvature distribution along the spiral, promoting comfort.

Thus, the final optimization problem may be written as:

$$\min f_{be}(a_0, a_1, a_2, a_3, s_f) \text{ s.t.} \begin{cases} \left|\kappa\left(\frac{s_f}{3}\right)\right| \leq \kappa_{max}, & \left|\kappa\left(\frac{2s_f}{3}\right)\right| \leq \kappa_{max} \\ x_S(0) = x_0, & x_s(s_f) = x_f \\ y_S(0) = y_0, & y_S(s_f) = y_f \\ \theta(0) = \theta_0, & \theta(s_f) = \theta_f \\ \kappa(0) = \kappa_0, & \kappa(s_f) = \kappa_f \end{cases}$$

Optimization, however, must be performed in the local frame attached to the vehicle, as mentioned before; this is done to simplify the optimization by setting the starting boundary conditions to zero.

Once solved the optimization problem, the spiral parameters can be finally determined and the spiral positions are obtained. The parameters are:

$$a_0 = p_0$$

$$a_1 = -\frac{11p_0/2 - 9p_1 + 9p_2/2 - p_3}{p_4}$$

$$a_2 = \frac{9p_0 - 45p_1/2 + 18p_2 - 9p_3/2}{p_4^2}$$

$$a_3 = -\frac{9p_0/2 - 27p_1/2 + 27p_2/2 - 9p_3/2}{p_4^3}$$

At this point the program receives a discrete set of points on the obtained spiral, which reach the chosen waypoint.

Once the single spiral is generated, it must be validated by means of a collision check. Before doing that, the spiral is expressed in the map reference frame because the collision checker receives as input, beside the trajectory, also the *objects* (circles representing the obstacles), which are provided with respect to the global reference frame.

3.1.5.1 Collision check

An input/output representation of the collision checker is shown below:

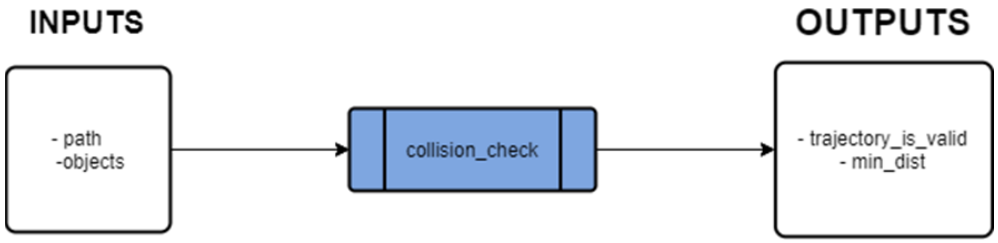


Figure 36: collision_check I/O representation

The provided outputs are: a Boolean variable (*trajectory_is_valid*) which indicates whether the trajectory received as input is valid (collision-free) and the variable *min_dist* which contains the minimum distance along the spiral to the closest object, this latter information is used to choose the optimal path.

Again, a supporting flow chart is provided:

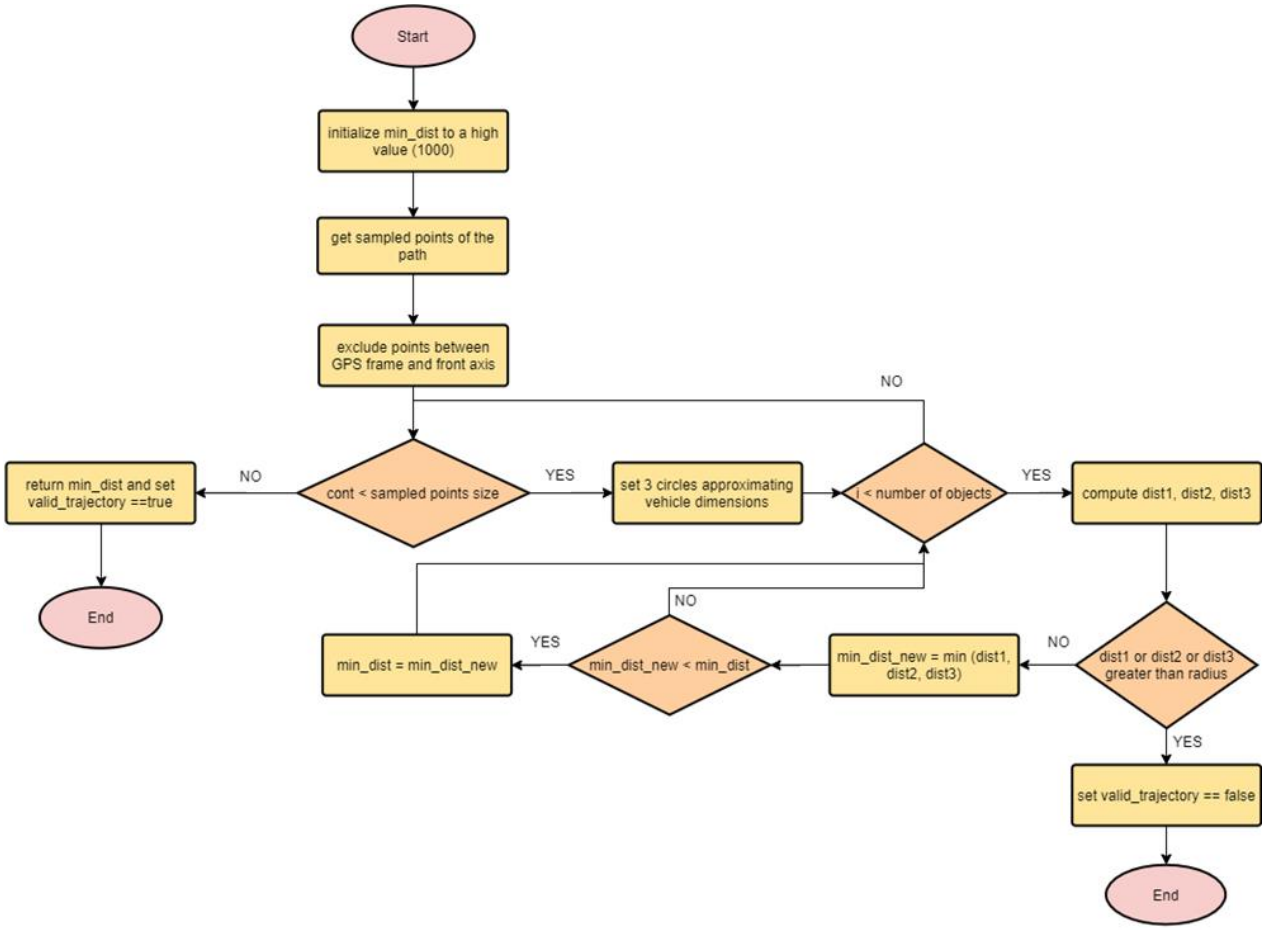


Figure 37: collision_check flow chart

This function checks that the entire generated trajectory is at a safe distance greater than or equal to a certain radius (set to 1.1 m) from each obstacle.

Remembering that the path starts from the GPS frame, i.e., 2 meters behind the front axle, points that are less than this distance from the start of the path are excluded. Then, for every other sampled point of the path we check whether the car will collide with something or not. This is done approximating the car with 3 circles with radius 1.1 m. In particular, we consider 1 circle at the point of the trajectory and 2 circles with centers at

+1.7 m and -1.7 m respectively from that point. The direction of the line that connects the centers of the 3 circles is the tangent of the theta angle of the considered point on the trajectory:

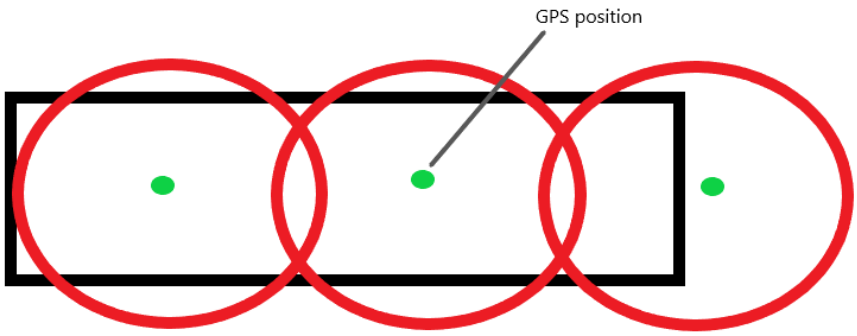


Figure 38: vehicle circles representation

To determine the coordinates of the centers of the circles located at the front and rear axis of the vehicle, knowing the coordinates in global reference frame of the GPS sensor (which is the starting point of the spiral) we consider:

$$y - y_{gps} = \tan\theta\ (x - x_{gps})$$

Which is the equation for the lines passing through the GPS position and with direction along the theta angle of the considered point on the spiral; then, in order to find the coordinates of the required points we impose a distance between the GPS position and a generic point equal to d (1.7 m). Thus, the second equation to be considered is:

$$(y - y_{gps})^2 + (x - x_{gps})^2 = d^2$$

Considering both the equations in a linear system one obtains the x-position of the two centers (and consequently the y-position):

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

with the parameters, computed offline, equal to:

$$a = 1 + \tan\theta^2$$

$$b = -2x_{gps}(1 + \tan\theta^2)$$

$$c = x_{gps}^2(1 + \tan\theta^2) - d^2$$

Then, for each sampled point of the spiral, the distances between the centers of the three circles with the centers of the circle representing the obstacles are iteratively computed (*dist1, dist2, dist3*): as soon as one of these distances is greater than the safe distance (1.1 m) the program terminates and the *valid_trjectory* variable is set to false. If the three computed distances are all less than the safe distance, the *min_dist* variable is updated and the loop keep going until all the sampled points of the spiral are analyzed. At the end the *min_dist* variable contains the most critical distance between the vehicle on the path and an obstacle.

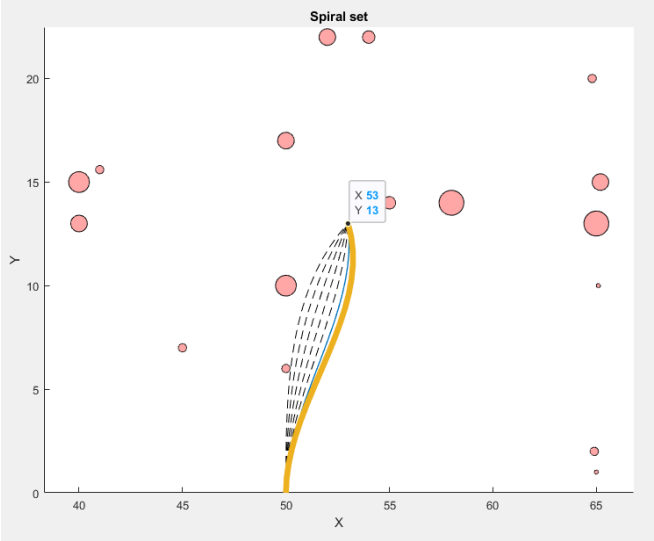


Figure 39: valid/non-valid trajectories of a spiral set

In the figure above, considering as reference waypoint the point with coordinates (53; 13), the first 5 spirals starting from the left side are defined not valid by the collision checker because of their closeness to the obstacles; the remaining two spiral are valid, the last one on the right side (plotted in orange) is chosen as the optimal one according to the criterion described in paragraph 3.1.5.3.

Once terminated the collision check, if the trajectory results valid it is saved locally, together with the *min_dist* and its average lateral acceleration profile, which is determined by the *speed_profile_generator* function: basically, the spiral is given to another function, called locally, which returns for each point of the spiral the velocity, lateral acceleration and longitudinal acceleration the vehicle should assume in order to execute that trajectory. The method used for

the speed profile generation is presented below.

3.1.5.2 Speed profile

Given a path a speed profiler can be generated thus, for each sampled point of the trajectory, velocity, longitudinal and lateral accelerations can be computed: a kinematic value is associated to each of path points.

In order to compute the most relevant kinematic quantities some parameters have been chosen, so that a certain driving comfort is guaranteed. The imposed parameters are listed in the following table [Tab. 5]:

PARAMETER	DESCRIPTION	ASSIGNED VALUE
V_{max}	Maximum admissible speed, chosen empirically	15 m/s
$a\ lat_{max}$	Maximum admissible lateral acceleration	1 m/s^2
$a\ long_{max\_acc}$	Maximum admissible longitudinal acceleration	1 m/s^2
$a\ long_{max\_dec}$	Maximum admissible longitudinal deceleration, in the following it is referred to also as $a\ long_{min}$	-2 m/s^2
V_{start}	Starting velocity	0 m/s at the starting time instant; in general, it is set at the current velocity detected by the GPS sensor

Table 5: parameters for speed profile generation

It should be noted that the maximum admissible speed has been determined not in the design/simulation phase of the trajectory planning algorithm, but during the phase of testing. We noticed that the speed limit of 15 m/s ensures a completely autonomous safe driving, all the turns are driven comfortably and at a relatively low speed all the non-linearity phenomena, such as road-tire contact or the vehicle dynamics, can be neglected, since empirically they do not affect the correct execution of the algorithm.

The speed profile is generated through different steps:

Step 1

For each sampled point of the path a speed limit $Vlim_n$ is determined as the minimum between the maximum admissible speed and the maximum speed that allows to drive the turn comfortably. The kinematic quantity which takes into account the driving comfort is the lateral acceleration: the lowest the lateral acceleration is the smoothest the turn is driven. This is the reason why the maximum admissible lateral acceleration has been set to a relatively low value (1 m/s^2).

In formula we compute for each of the *n* points of the path:

$$Vlim_n = \min\left\{V_{max}\ , \sqrt{\frac{a\ lat_{max}}{|Kn|}}\right\}$$

Where $|Kn|$ is the absolute value of the inverse of the radius of curvature R. In the following figure [Fig. 40] a point P of a generic path is represented and red arrow represents its radius of curvature.

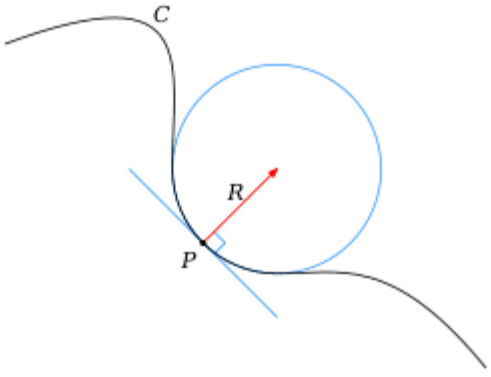


Figure 40: curvature radius of a point on a path

Thus, at the end of the first step a speed limit profile is generated.

STEP 2

At this stage the assumption we have made is to consider uniform acceleration between two consecutive sampled points of the trajectory; thus, we consider the following formula:

$$V_n = \sqrt{{V_{n-1}}^2 + 2\, along_n\, d}$$

Where V_n is the speed associated to the point *n* to be computed; ${V_{n-1}}^2$ is initialized to 0 for the fist point of the trajectory; $along_n$ is the longitudinal acceleration at the considered point; d is the curvature distance between two consecutive points.

Now, V_n, $along_n$, $alat_n$ can be computed (the obtained values may be subject to a correction, explained a STEP 3):

- For the considered point of the trajectory, one should determine the admissible interval of $along_n$. Considering the existing condition of the root above:

$$along_n > -\frac{{V_{n-1}}^2}{2d} = a^*$$

Another trivial requirement is: $a\ long_{min} < along_n < a\ long_{max}$. Since a^* is clearly always a negative number, from the study of sign the admissible interval is:

$a\ long_{min} < along_n < a\ long_{max}$, if $a^* \leq a\ long_{min}$

$a^* < along_n < a\ long_{max}$, if $a^* > a\ long_{min}$

For reasons of clarity, let us denote the extremis of the range by means of the following notation:

$$x1 < along_n < x2$$

- Let us compute the velocities evaluated in $x1$ and $x2$:

$$V_1 = \sqrt{{V_{n-1}}^2 + 2\ x1\ d}$$

$$V_2 = \sqrt{{V_{n-1}}^2 + 2\ x2\ d}$$

- Let us consider the limit determined at STEP 1: $Vlim_n = L$, we need to impose that the computed velocity must be always less then L. Now we may distinguish three cases:

1) $L > V_2 > V_1$

We choose as speed value the maximum admissible one; the final value is $V_n = V_2$

2) $V_1 < L < V_2$

In this case: $V_n = L$

3) $L < V_1$

This is the most critical case, since all the values less then the speed limit are not admissible because of the longitudinal acceleration constraints. We impose as velocity $V_n = L$, but this would violate the acceleration limit constraints, thus some corrections are performed at STEP 3

- Let us compute the remaining kinematics quantities:

$alat_n = \frac{{V_n}^2}{R}$, where R is the curvature radius;

$along_n = \frac{{V_n}^2 - {V_{n-1}}^2}{2d}$;

STEP 3

This step is performed only if the longitudinal acceleration lower bound is not fulfilled.

At this stage the trajectory is analyzed backwards, in order to perform corrections.

The longitudinal acceleration value is imposed to: $along_n = a\ long_{min}$.

The value of V_{n-1} is modified considering the imposed value of longitudinal acceleration:

$$V_{n-1} = \sqrt{{V_n}^2 - 2\, along_n\, d}$$

The value of the lateral acceleration is updated as well, using the same formula described at the previous step.

Please, note that with this approach the end of the path that the vehicle is supposed to drive is not detected, thus at the final waypoint the vehicle would not arrive with null velocity. In practice, this aspect is taken into account right before the reference speed is sent to the CAN bus; the reader may refer to the paragraph related to the block *find_reference_speed* for further details.

By means of this approach, considering a generic path shown in Figure 41, the results shown in figures 42, 43, 44 can be obtained:

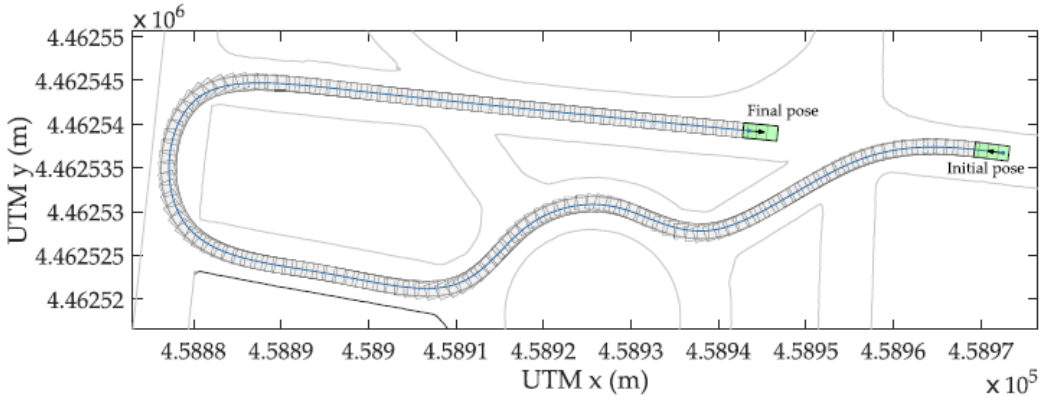


Figure 41: a generic path

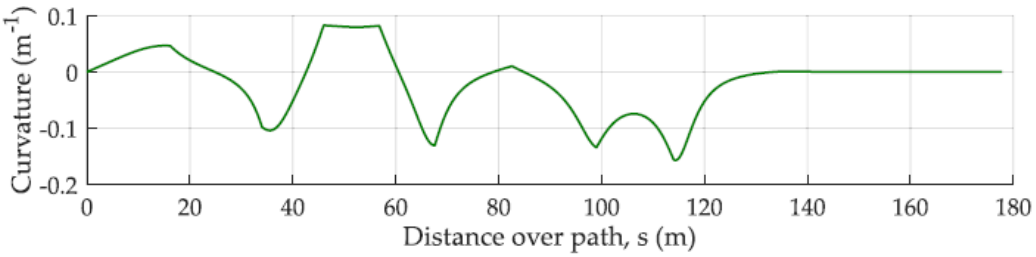


Figure 42: curvature profile example

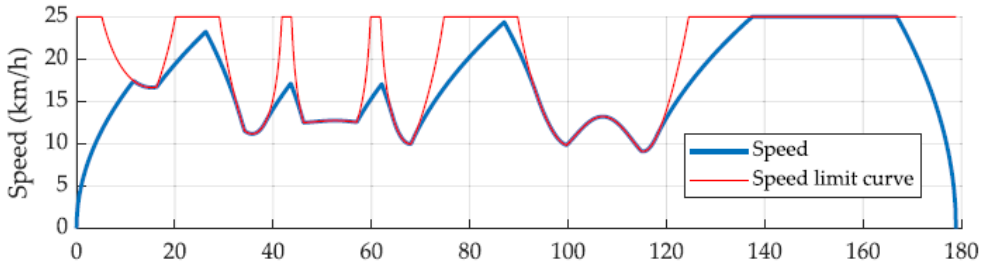


Figure 43: speed profile example

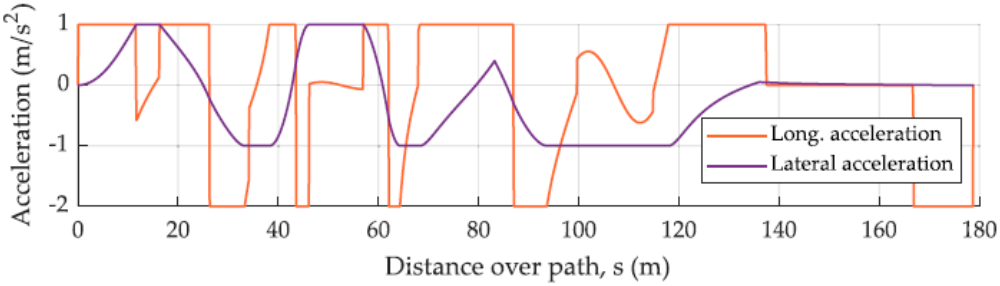


Figure 44: acceleration profile example

3.1.5.3 Optimal Path choice

So, the speed profile is analyzed when the reference speed is sent in actuation, while on the lateral acceleration profile an arithmetic average is computed, and the result is locally saved if the trajectory is determined to

be valid. The average lateral acceleration and the *min_dist* are the two parameters used to establish the optimal path.

The operations of spiral generation, collision check and eventual saving of these relevant parameters (average lateral acceleration and minimum distance on the path to the closest object) are repeated for the entire spiral set, varying at each iteration the arrival *theta* in proximity of the reference waypoint.

At this point we distinguish two cases:

a) in the spiral set there is at least one valid trajectory

in this case the valid candidate trajectories of the spiral set are analyzed in order to return the optimal one. Each valid trajectory is associated with a value calculated as a weighted sum of the average lateral acceleration and the minimum distance to obstacles. The optimal trajectory will be the one with the lowest value. Note that in this way a low average lateral acceleration and a large minimum distance are preferred: a low lateral acceleration maximize, by definition, the driving comfort; with this approach trajectories with a low curvature are preferred. The *min_dist* parameter is used to prefer the trajectories which are sufficiently far away from the obstacles, this for safety reasons. For each trajectory *n* among all the valid ones, a parameter $value_n$ is associated and it is determined as follows:

$$value_n = w_{acc} * avg_{lat_{acc_n}} + \frac{w_{dist}}{\min\_dist_n}$$

Please, notice that since min_dist_n is located at the denominator a low value of this distance contributes to the increasing of $value_n$; on the other hand, because of the direct proportionality, a high average lateral acceleration contributes to the increasing of $value_n$. The optimal

trajectory returned by the *get_optimal_trajectory* function is the one associated with the minimum $value_n$.

The coefficients w_{acc} and w_{dist} are determined offline and experimentally; the tuning operations of the above weighting coefficients is presented in Chapter 4.

Once determined the optimal spiral, it is returned by the function together with its speed profile.

b) in the spiral set none of the trajectory is valid

if this is the case, other spiral sets are considered by laterally moving the waypoint. A new spiral set is considered only if the previous spiral set was not containing a valid trajectory; the number of different spiral sets that may be analyzed, including the one with arrival point coincident with the reference waypoint, in the worst-case scenario is equal to 7 since the waypoint is moved laterally up to +- 1.5 meters with a resolution of 0.5 m. As shown in the following figure [Fig. 45], the reference waypoint is moved laterally along the line perpendicular to the angle determined by the line passing through the GPS position and the reference waypoint.

If all the possible spiral sets are analyzed (the reference waypoint has been moved to all the possible established locations), and a collision-free trajectory is still not found, the *emergency braking* variable is set to true, and the function terminates.

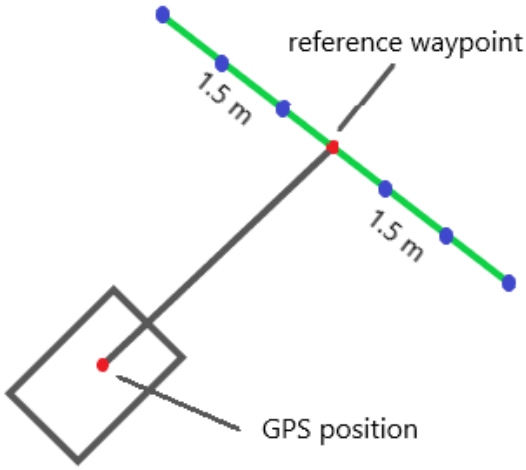


Figure 45: reference waypoint is eventually moved laterally

3.1.1.1 Recap

In the following, for clarity reasons, a recap referred to the *get_optimal_path* function is provided; the main steps are:

1) generate a valid trajectory to reach the reference waypoint.

2) transform the trajectory from the local reference frame to the global one.

3) check if the trajectory is collision free or not.

4) save the average lateral acceleration and the minimum distance to obstacles and add this path to the vector of valid trajectories.

We repeat steps 1-3 for all 11 trajectories but only entering step 4 for valid and collision-free trajectories. If none of the 11 paths is valid and collision-free the waypoint is moved laterally starting from the right side and then steps 1-4 are repeated again for all new 11 trajectories.

5) determine the optimal trajectory. Each valid trajectory is associated with a value calculated as a weighted sum of the average lateral acceleration and the minimum distance to obstacles. The optimal trajectory will be the one with the lowest value. We enter step 5 if at least one of the 77 trajectories is valid and collision-free. (77 because 7 possible arrival points are tested, with 11 trajectories for each of them). If none of those trajectories is practicable the vehicle will go in emergency braking.

3.1.6 Get look ahead index

At this point of the program the trajectory up to the reference waypoint and the related speed profile are determined. Remembering that in the next time slot (after 100 ms) all the operations are repeated with the new vehicle position and the updated occupancy grid map, the matter is to understand how to compute in the current time slot the steering angle to send to the CAN bus, and which point of the speed profile needs to be sampled. The steering angle, but also the reference speed in some conditions, are referred to the so-called look-ahead point.

The look-ahead is a spiral point determined by using the following rule:

- if the spiral does not contain an inflection point:

  in this case the look-ahead point is located in the middle of the spire (at 50% of its length)

- if the spiral contains an inflection point in the section of the spire between the 30% and 70% of its length

the look-ahead coincides with the inflection point, which is identified when a changeover from negative to positive is detected on the curvatures on the spiral.

With this rule application it is guaranteed that when there is an inflection point – there is the necessity to steer, for instance, first on the right side and after the inflection on the left side – the output steering angle will be computed so that the initial maneuver is actually on the right side: differently the vehicle, erroneously, may not turn on the right side and the part of the trajectory before the inflection point would be ignored.

From the following schematic representation [Fig. 46] it is evident that, in the shown case, if the look ahead is chosen after the inflection point, the steering angle, which is related to the inclination of the blue line, would cause a wrong maneuver and for sure the vehicle will not pass through the inflection point. Remembering that the spiral, at this stage, is a collision-free trajectory, the vehicle has to follow its path in order to satisfy the collision-avoidance requirement: in this proposed scenario the steering angle to return as output must be related to the yellow line inclination (the detailed steering angle computation will be presented in the related paragraph).

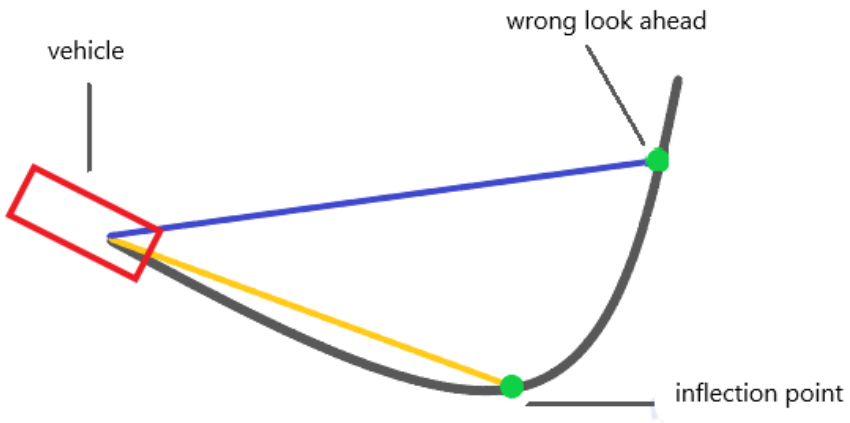


Figure 46: look ahead with an inflection point

3.1.7 Find steering angle to actuate

In order to determine the steering angle to actuate, the pure pursuit approach is used [Fig. 47]:

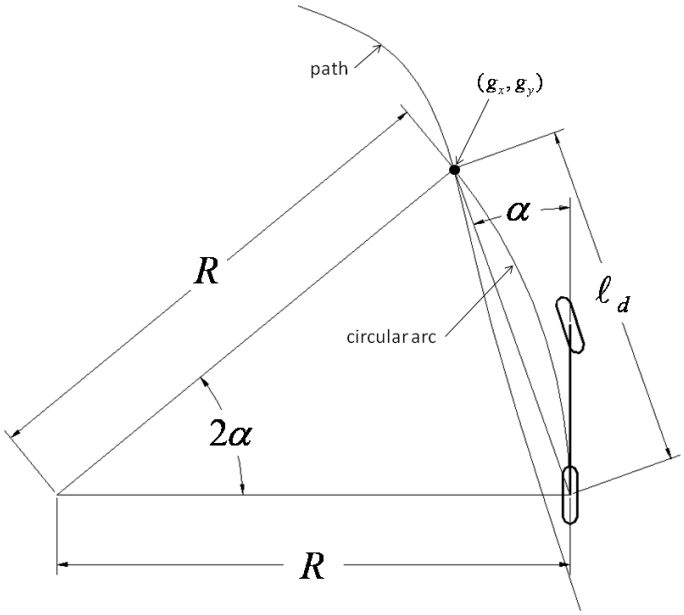


Figure 47: pure pursuit

The pure pursuit method is one of the most common approaches for path tracking problems in autonomous driving applications. The pure pursuit method consists in calculating the curvature of a circular
arc connecting the rear axle location to the look-ahead point. ld is the distance between the look: ahead point and the current rear axis position and L is the distance between the front axis and rear axis (wheelbase). The vehicle's steering angle δ can be determined using only the look ahead point location and the angle α between the vehicle's heading vector and the look-ahead vector. Applying the law of sines and using a simple bicycle-model, the steering angle is computed as:

$$\delta = \tan^{-1}\left(\frac{2\,L\sin(\alpha)}{ld}\right)$$

At this stage some variables need to be computed; referring to the following figure [Fig.48], the available data are

- ld^* since the look-ahead point coordinates have been previously determined and the GPS position is known;
- α^*, which can be easily computed as $\tan^{-1}\left(\frac{look\_ahead\_y}{look\_ahead\_x}\right)$, once expressed the look-ahead coordinates in the GPS reference frame;
- the linear distance between the GPS and the rear axis (RA) is measured and it is equal to about 1m;
- L is also measured and it is set to 2.4 m;

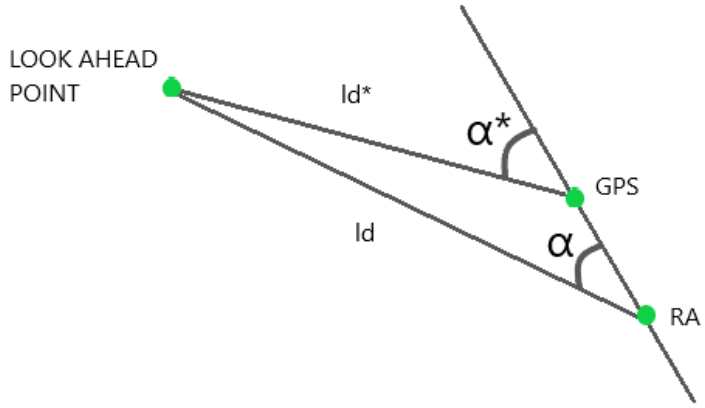


Figure 48: parameters needed for steering angle computation

The missing data are ld and α. ld can be computed using the low of cosines as:

$$ld = \sqrt{L + (ld^*)^2 - 2 * ld^* \cos(180 - \alpha^*)}$$

α can be computed using the low of sines:

$$\alpha = \sin^{-1} \frac{ld^* \sin(180 - \alpha^*)}{ld}$$

Once computed the steering angle, at this level, a further simple control is applied; the lines of code implementing this aspect are:

```
double delta_steering = pure_purs - pure_purs_prev;
pure_purs = pure_purs_prev + w_steering * delta_steering;
pure_purs_prev = pure_purs;

if (len_to_goal < 9) {
        pure_purs = pure_purs_prev;
}

pure_purs_prev = pure_purs;
```

In short, the steering command returned at the previous time instant is considered and a delta between the current steering angle, obtain through the pure pursuit, and the previous one is computed; the actual command sent as output is equal to the previous one, summed to the computed delta which is weighted through a coefficient $w\_steering$ set to 0.1. By means of this procedure abrupt steering angle variations are avoided: what has been implemented is basically a smoothing control.

Please, notice that if the *len_to_goal* variable is less than 9 m the steering angle command is not updated because the vehicle is in proximity of the path end.

3.1.8 Find reference speed actuation and check emergency braking

the optimal trajectory chosen, for each sampled point, has associated a speed value: so, the matter is to choose which speed value needs to be

actuated at the current time interval. The lines of code used to determine the speed actuation value are the following:

```
if (gps_speed > 1.5) {
        desired_speed = min_speed;
}
else {
        desired_speed = confirmed_path.v_points[lookahead_idx];
}

if (em_braking || len_to_goal < 10) {
        std::cout << "emergency braking\n";
        desired_speed = 0;

}
double delta_speed = desired_speed - desired_speed_prev;
desired_speed = desired_speed_prev + w_speed * delta_speed;
desired_speed_prev = desired_speed;
```

In general the reference speed to actuate corresponds, for safety reasons, to the minimum speed on the generated velocity profile: in this way for sure the constraint on the maximum lateral acceleration (driving comfort) is guaranteed. However, if the current detected velocity of the vehicle (*gps_speed*) is less than 1.5 m/s, the reference speed is the one associated to the look-ahead point on the trajectory; in this way, when the vehicle is driving at a very low speed, by considering the velocity on the look ahead point, most likely, if the curvatures on the generated spiral are not significantly high, the new reference speed would cause an acceleration.

The speed actuation value is different if the emergency braking Boolean variable is set to true; if this is the case a valid trajectory cannot be generated, most likely because the collision checking is not successful: in this scenario an alert message of emergency braking is generated, and the

reference speed is set to zero. It is important to underline that, for technical hardware reasons, the brake by wire was not implemented on the prototyped vehicle: thus, the *trajectory_planning_node* has not been set to send as output variables acting on the brake. The only action which can be performed in case of emergency is to set the reference speed to zero, causing a releasing pressure on the throttle pedal; obviously, the time interval in which the vehicle arrests is influenced by the current velocity of the vehicle. By the way, when the vehicle goes in emergency braking a deceleration is immediately sensed, and in the majority of the cases after this deceleration in the next time intervals a valid trajectory is found and the vehicle keep moving on the predefined path. However, we highly recommend, since we could not act with the designed algorithm on the brake pedal, to assist the autonomous driver and to brake in cases of real emergency.

Another case in which the reference speed is set to zero is in proximity of the end of the path. When the *len_to_goal* variable is less than 10 m, the output is set to zero.

Please, notice that the same smoothing control used for the steering angle actuation has been implemented: by means of this kind of control, when the vehicle is starting, the first returned reference speed is reached gradually, the same reasoning is valid for the vehicle stopping at the end of the path (the zero velocity is reached with a smooth profile).

3.1.9 Send actuation variables to CAN bus

At this final stage, the determined reference speed and reference steering angle are sent via the CAN bus to the ECU. Anyway, both the signals are not actuated as they are, but they are handled by a controller already implemented in the ECU. In particular we had the chance to act on the steering PI controller, by tuning its parameters. This tuning operation is presented in the next chapter. On the other hand, the ECU speed controller parameters are not manageable, thus an error between the reference speed and the actual one was observed: the obtained results are shown in chapter 5.

It is important to underline that an 18:1 steering ratio has been used: the pure pursuit control returns a steering angle intended as the angle variation of the front wheel of the car. In our case 1° on the car's wheel angle corresponds to a 18° steering angle. Thus, the steering angle sent to the CAN bus is the output returned by the pure pursuit, considering a multiplicative factor of 18.

Chapter 4: The tuning procedure

in this chapter we present two tuning procedures which have been performed to determine some of the used parameters. The first tuning procedures is needed to establish the values of w_{acc} and w_{dist} , which are some coefficients used to choose the optimal path from the current vehicle position to the chosen reference waypoint: further details are provided in the previous chapter.

The second tuning procedure is related to the PI control system already developed in the ECU, responsible for the steering angle control.

4.1 Optimal path weighting coefficients tuning

the used tuning procedure consists in the following analysis:
Firstly, we considered the steering angles recorded during a non-autonomous driving test; let us refer to this plot as “Ground truth”, which is represented in the following figure [Fig. 49]:

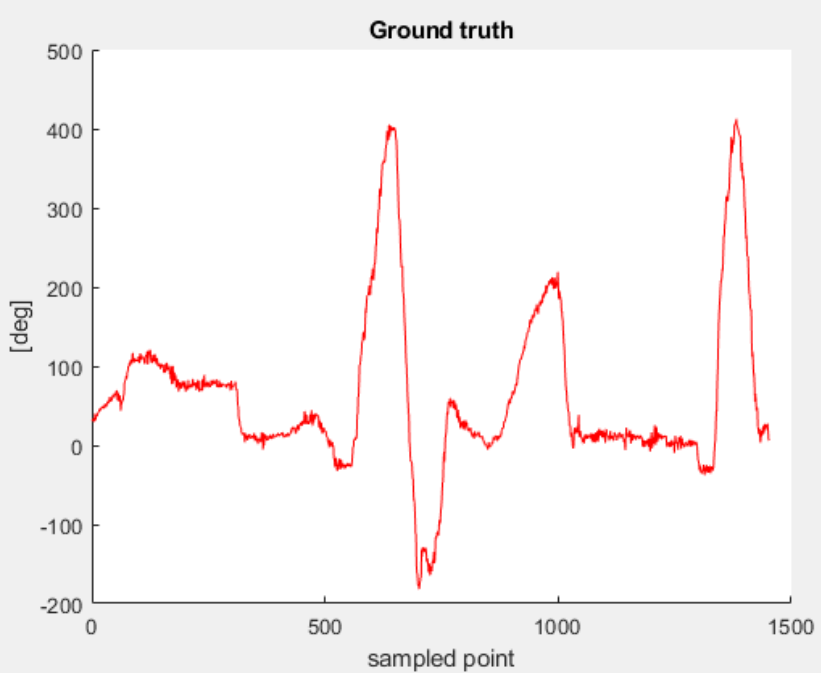


Figure 49: steering angle Ground truth

Then, we considered the following values for each weighting coefficient to be tuned: 0.25, 0.5, 0.75, 1, 1.25. First we fixed w_{acc} to a reference value and we considered a variation of the w_{dist} coefficient. For each value assumed by w_{dist}, for each point of the path the difference between the Ground truth and the reference steering angle given as output is computed; then average and standard deviations are calculated on this delta vector.

The following plot [Fig. 50] shows the average (and standard deviation) of the difference between the steering angle ground truth and the reference one for a portion of the path, considering the distance coefficient variation.

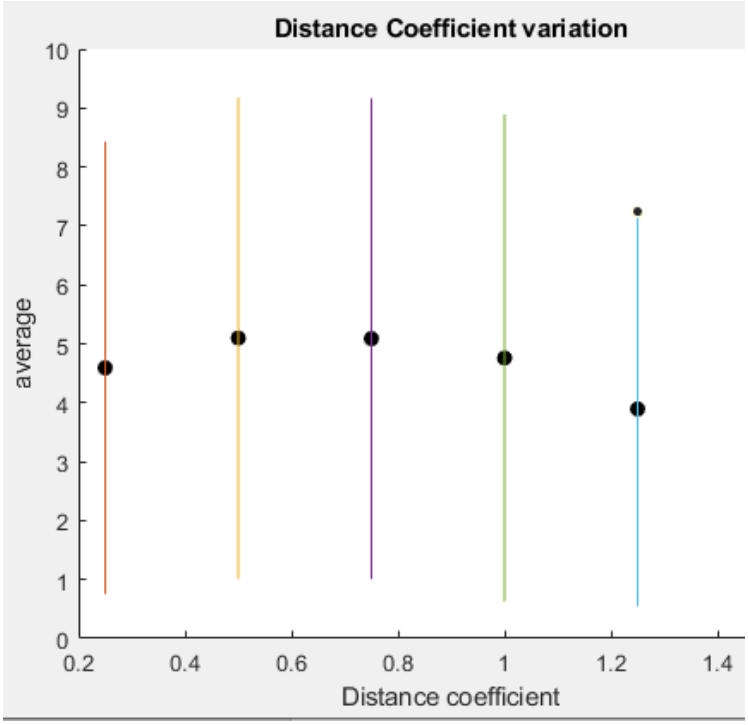


Figure 50: average and standard deviation of the difference between the ground truth and the measured steering angle, considering a variation of the distance coefficient

The w_{dist} coefficient is chosen as the one to which is associated the less average on the delta vector defined above. However, in this case, even though a minimum is identifiable at 1.25, an extra test has been performed with the value of 1.5 since the minimum identified above is not a relative minimum. From the following plot [Fig. 51] it is evident that the performances with $w_{dist} = 1.5$ are worse, thus $w_{dist} = 1.25$, being now a relative minimum, is chosen.

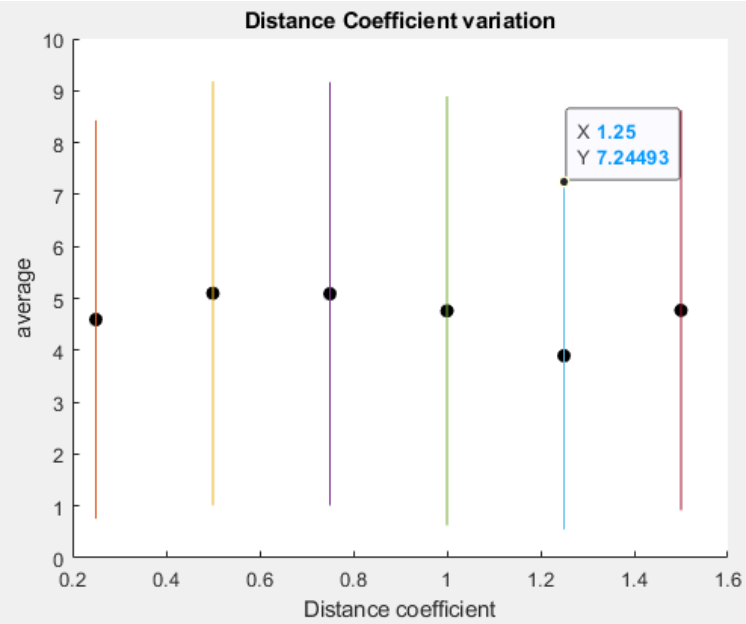


Figure 51: average and standard deviation of the difference between the ground truth and the measured steering angle, considering a variation of the distance coefficient. an extra test has been performed (w_dist = 1.6)

At this point, fixed w_{dist}, w_{acc} is tuned applying the same reasoning using for the previous tuning. The plot related to w_{acc} is the following one [Fig. 52]:

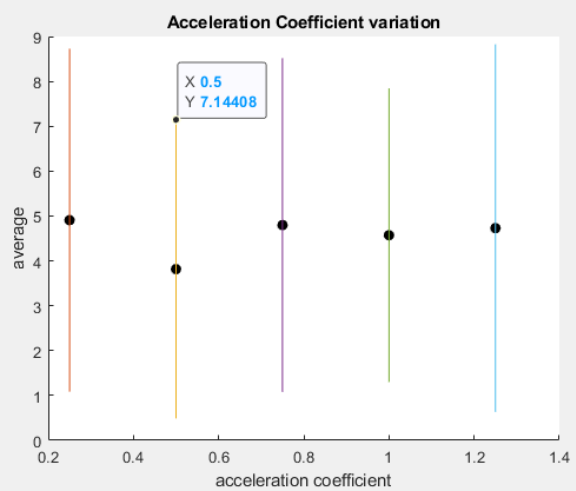


Figure 52: average and standard deviation of the difference between the ground truth and the measured steering angle, considering a variation of the acceleration coefficient

Finally, the chosen weighting coefficient used to determine the optimal spiral are:

$$w_{acc} = 0.5$$

$$w_{dist} = 1.25$$

4.2 PI controller tuning for the steering angle

as mentioned in the previous chapter, the reference speed angle sent to the ECU is modified by a controller already implemented and embedded in the ECU. From the documentation we were provided, we realized that a PI controller tuning was necessary. The block scheme of a generic PI control system is presented in the following figure [Fig. 53]:

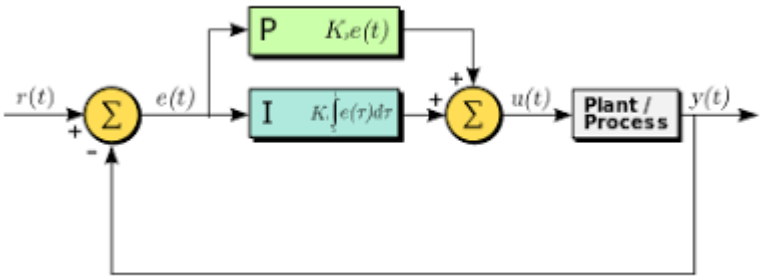


Figure 53: PI control system

In our scenario, the reference is the steering angle returned by the trajectory planning node, Kp and Ki are the coefficients to be tuned, $u(t)$ is the control variable, that is the controlled steering angle which is actually actuated through a rotation of the steer, and $y(t)$ is the measured steering angle. In particular, the proportional action consists in multiplied

Kp by the error: if the error is large and positive, the control output will be proportionately large and positive. If there is no error the control action is not performed. On the other hand, the integral action takes into account the errors over time: the I block seeks to eliminate residual errors by adding a specific control effect; the integral term will stop growing as soon as the error is eliminated. Kp and Ki tuning has been performed experimentally: we simply compared for a small part of the path the reference steering angle with the real one, which is detected by analyzing proper CAN messages. Basically, a trial and error procedure has been applied: we performed different tests with different values of Kp and Ki. The most relevant ones are shown in the following.

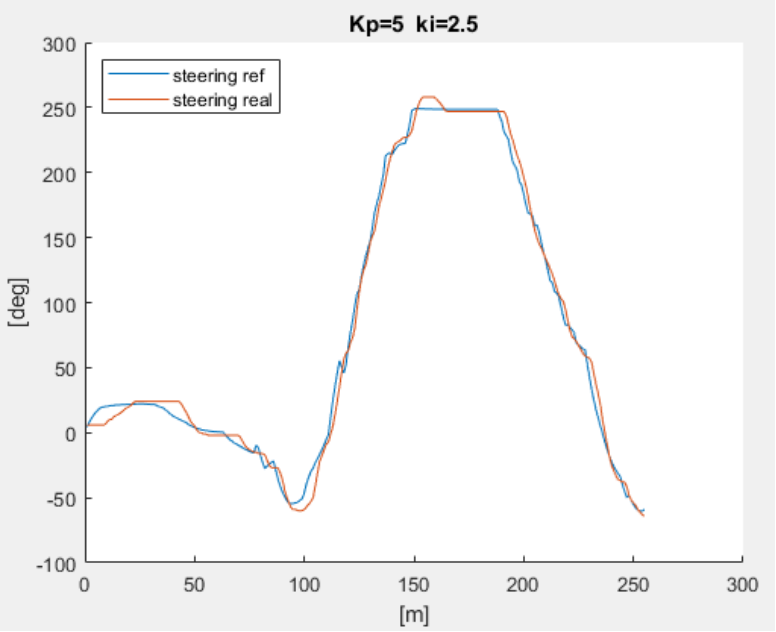


Figure 54: steering angle profiles comparison with the indicated parameters

In the plot represented above, the actual signal tends to follow the reference, but some overshoots are observed. The most visible one is at 150 m. even though the error between the two signals is not that

consistent, we discarded the values above since better performances, with different tuning values, have been found.

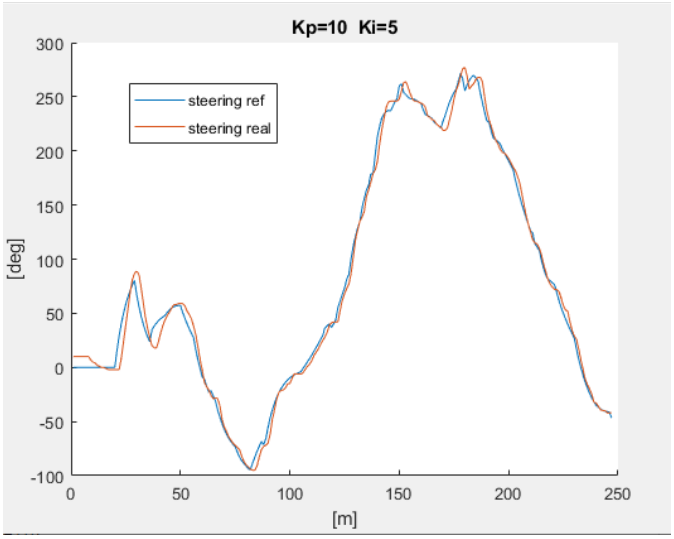


Figure 55: steering angle profiles comparison with the indicated parameters

In the plot above the overshoot is almost null, however, a slight delay may be observed in the whole profile, especially at 50m. a delay of the steering angle was preferred to be avoided, since a delay in the steering maneuver may cause emergency situations. The following two figures are the ones corresponding to the best performances

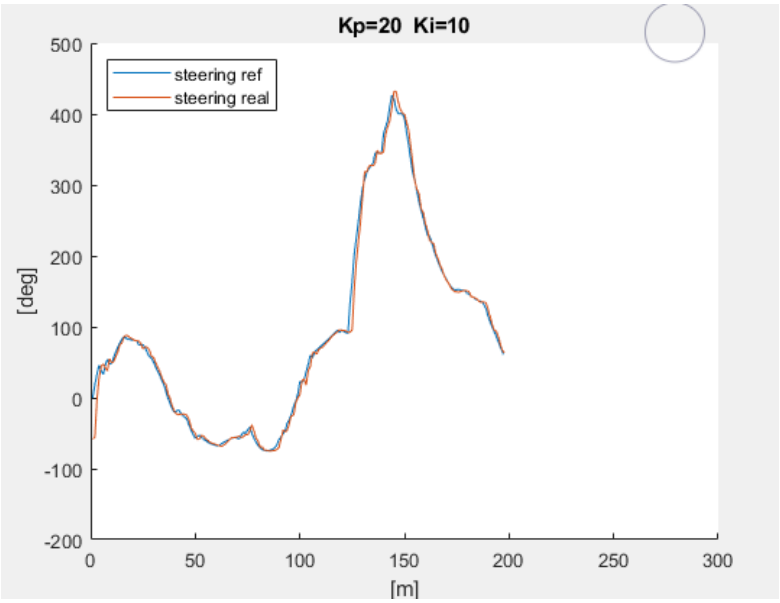


Figure 56: steering angle profiles comparison with the indicated parameters

.

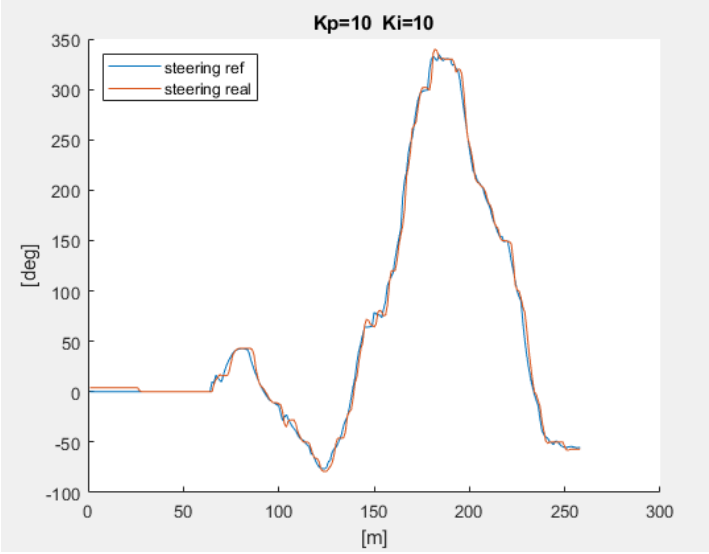


Figure 57: steering angle profiles comparison with the indicated parameters

The values of Kp=10 and Ki=10 have been chosen because, even if a really slight overshoot is detected at 170m, the reference profile is almost completely overlapping to the measured steering angle.

Chapter 5: Final results

In this chapter the obtained results are discussed. We will present first the setup and then the experimental results are shown.

5.1 Test setup

The autonomous driving test has been performed in a pre-established path around the building where Bylogix s.r.l is located. With reference to the following figure [Fig. 58], the considered path is around the yellow building.

Figure 58: test path from Google Maps

The trajectory planning node, together with the occupancy grid node, is launched on the NVIDIA TX2 which is connected to the ECU. By means of a CAN bus, the steering angle and vehicle speed are actuated every 100 ms. The VEGA car, provided by Bylogix, has been used in this testing phase of the project; the driving path has been established a priori, thus the waypoints have been pre-recorded and given as input to the trajectory

planning node. The recorded waypoints, in the global reference frame, are shown in the following figure [Fig. 59]:

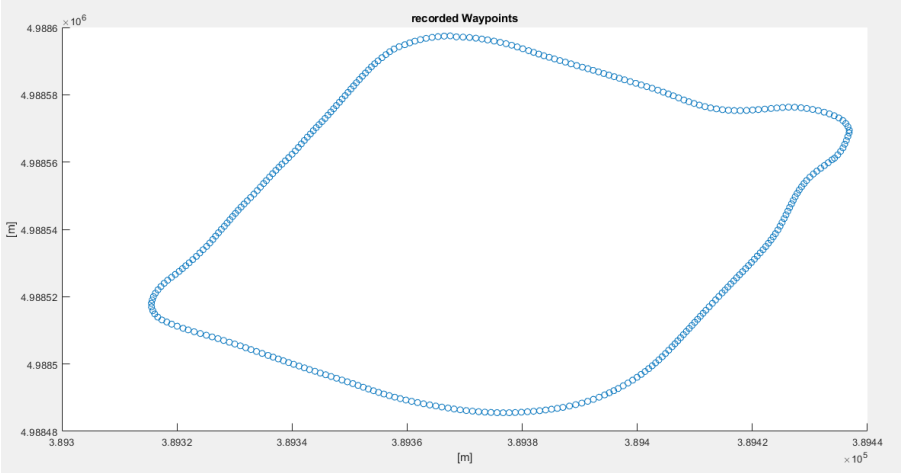


Figure 59: recorded waypoints

Please, notice that in the figure above all the recorded waypoints are represented, but the trajectory planner, according to the rule explained in paragraph 3.1.2, will choose at each time interval the waypoint to use as reference. The waypoints are recorded, during a setup non-autonomous driving test, every 1 m. It can be also noted that in the west and east area there is a denser waypoints distribution: this apparent anomaly is justified since in those mentioned areas the detected GPS signal was not as strong as in the rest of the path. It is necessary to underline that this kind of anomaly does not affect the correct working of the algorithm, since the reference waypoint choice is not influenced by the density of waypoints in the area close to the current vehicle position. This phenomenon is evident in the following zoomed figure [Fig. 60].

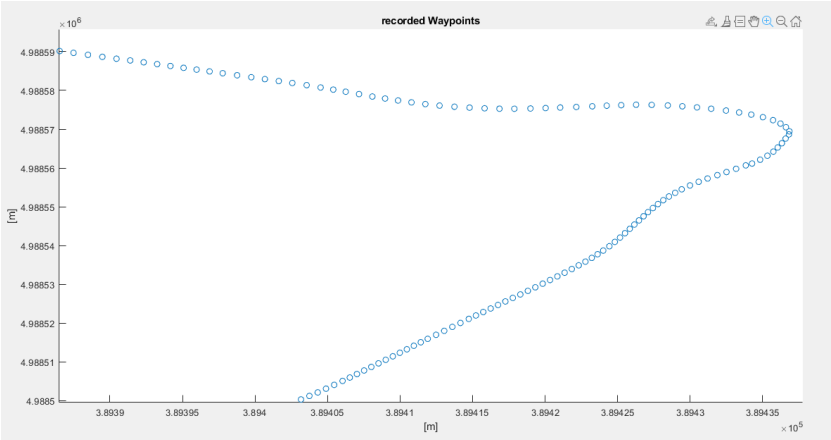


Figure 60: zoomed recorded waypoints

The obstacles detected by the Velodyne LiDAR on the complete path are shown in the following figure. Please, notice that both the waypoints representation and the obstacles one are obtained only after the full path is driven by the vehicle, thus when a new obstacle is detected the related circles representation is saved. The following figure [Fig.61] shows the detected obstacles, represented by circles.

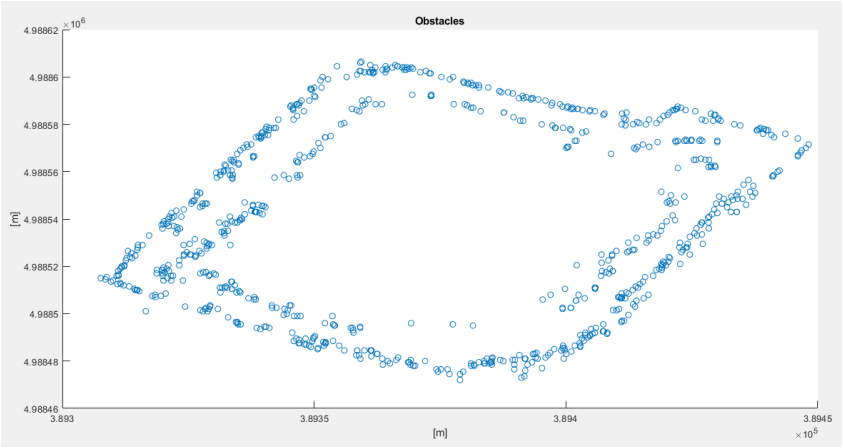


Figure 61: detected obstacles on the full path

5.2 Focus on the results

In this paragraph we focus on the analysis of the most relevant characteristics of the final result. The complete vehicle trajectory is displayed in the following image [Fig. 62]. Please, notice that the line representing the full path trajectory is obtained by adding up the single spirals, which are generated every 100 ms and each one of them guarantees the most relevant requirement: obstacle avoidance. In fact, the trajectory in no point is intersecting the red circles, which represent both the obstacles positioned on the path and the road boundaries.

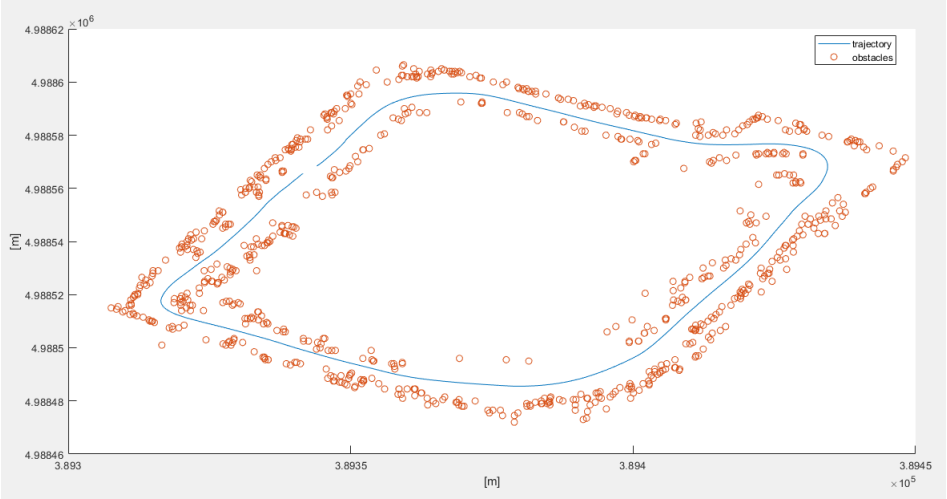


Figure 62: full trajectory

The plots below are a schematization of the actuation variables values recorded during the autonomous driving test. Each variable is compared with its reference, that is the value sent through the CAN bus to the ECU.

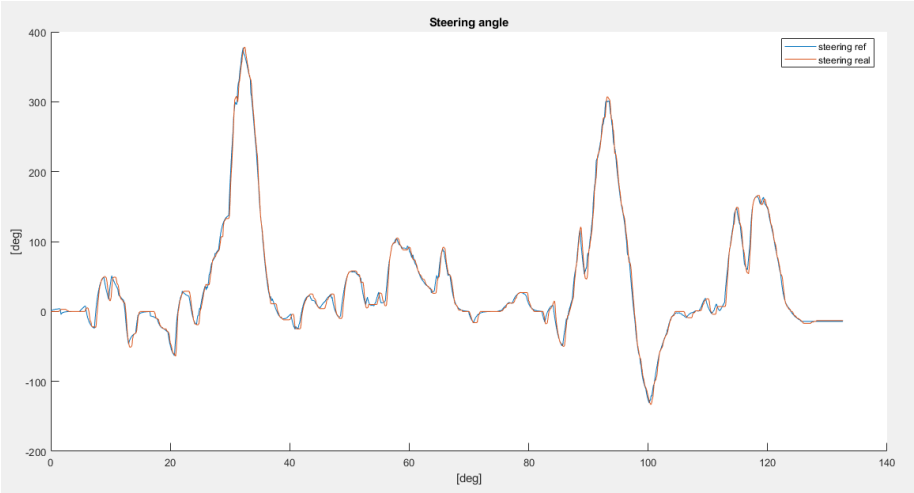


Figure 63: real steering angle and reference steering angle on the full path

In the figure above [Fig. 63] the reference steering angle (red line) and the real one (blue line) are displayed. The values, which are expressed in degrees, indicate the angle variation of the steer and are not referred to the front wheels angle variations. The obtained result is more than satisfactory, since the reference and the actual steering angle are almost identical; plus, the result is also coherent with the path, in fact the three peaks clearly observable in the plot identify the three sharp turns on the path.

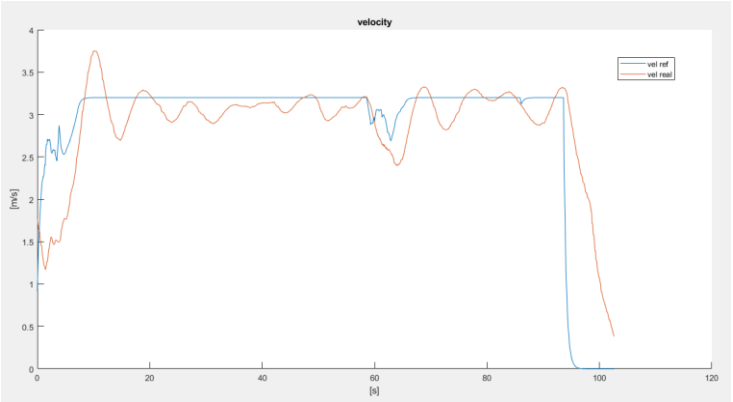


Figure 64: real velocity and reference velocity on the path

The plot above [Fig.64] is a representation of the velocity actuation variable: the reference is the blue line and the real value is represented by the red line. Both the values are expressed in m/s. It is important to underline that the two trends are comparable, however, some oscillations of the actual velocity can be noticed. This behavior is related to the design of the speed controller in the ECU that was already implemented, to which we could not get access during our project. Nevertheless, the oscillations observed in the plot are not practically perceptible since their average amplitude is of about 0.5 m/s.

In the following, some relevant results, which are also useful to clarify how the trajectory planner is actually working, are shown.

As first result we want to underline an example of obstacle avoidance, which has been correctly executed by the vehicle with the autonomous driving mode enabled. The following figure [Fig. 65] shows the generated optimal valid trajectory which has been chosen since it guarantees obstacle avoidance, it maximizes the distance from the obstacles and minimizes the average lateral acceleration, ensuring driving comfort. Specifically, the yellow point on the trajectory indicates the chosen look ahead point, and the white one is the reference waypoint, chosen in that particular time slot.

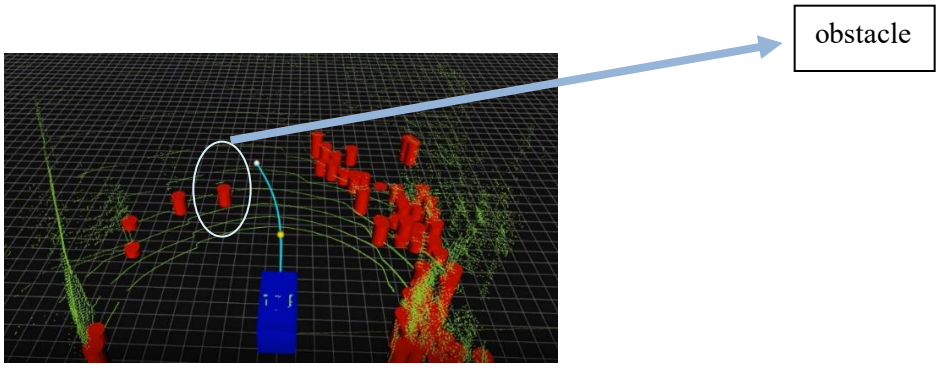


Figure 65: example of trajectory guaranteeing obstacle avoidance

In the following figure a picture is presented as proof of obstacle avoidance. It is evident a change of trajectory if figure 66 and figure 67 are compared. The used obstacle has been built to emulate a static pedestrian.

Figure 66: obstacle avoidance, the obstacle is detected

As soon as the obstacle is detected, in the next time slot the trajectory generated guarantees the obstacle avoidance, thus a steering angle variation can be observed, so that the vehicle would not crash and keep going on the established path.

Figure 67: obstacle avoidance, the obstacle is overcome

The next result we would like to underline is about the way in which the look - ahead point (red point in fig 68, yellow point in fig 69) is chosen.

In the following figure [Fig. 68], the chosen look ahead point coincides with the inflection point of the spiral; in this specific case this choice happens to be really relevant since it allows the vehicle to drive the sharp turn correctly. the way in which the look ahead point is chosen affects the steering angle, which in this case should be relatively high in order for the vehicle to drive the turn without any crash.

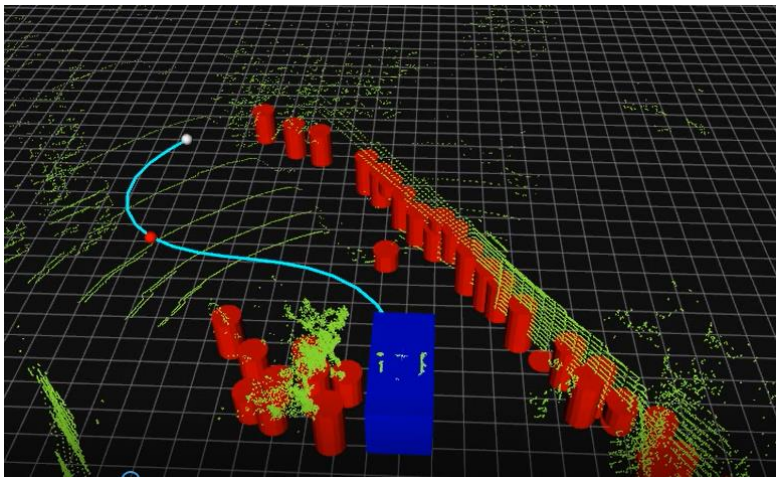


Figure 68: practical example of an inflection point chosen as look ahead

In the following figure [Fig.69], since there are not inflection points on the optimal spiral, the look ahead point is chosen at half of the spiral; please, consider that the trajectory starts from the point where the GPS sensor is located, that is a couple of meters behind the front axis. When there is no inflection point the look ahead point position is not as critical as in the previous case, however, a too much close point may not guarantee sufficient steering angle variations and a too much faraway look ahead point may cause unexpected behaviors, because in the next time

slots a spiral with a completely different shape may be generated in order to avoid some obstacles, not currently visible. Empirically, we have established to set in normal conditions the look ahead point on the middle of the spiral: the obtained observed results are satisfactory.

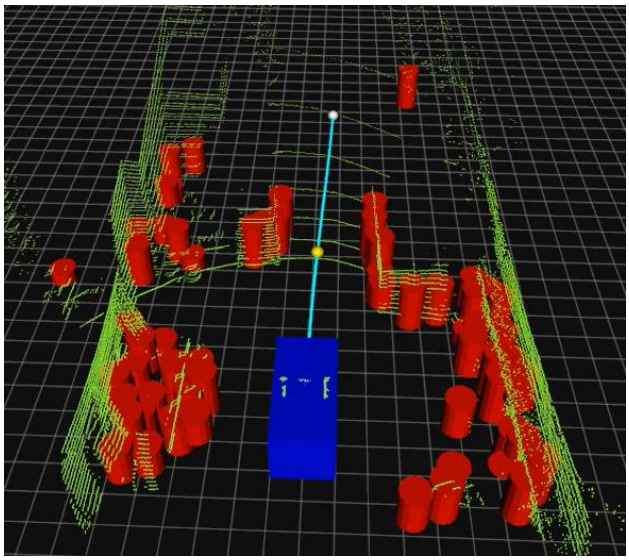


Figure 69: look ahead choice when there is no inflection point

The last result we want to focus on is related to the emergency braking condition. In the following figure [Fig. 70] the trajectory displayed in violet is the last one tested by the trajectory planner node and it is not valid because at the ending part the safety distance constraint from the obstacles is not fulfilled. In case of emergency braking, as explained in Chapter 3, the reference speed is set to a null value and the steering angle is the same one provided at the previous time step. As a matter of fact, in the scenario shown below, the emergency condition is not highly critical, in fact, after an initial deceleration, in the next time steps the emergency braking condition disappears because valid trajectories are found and the

vehicle can keep driving on its path. in fact, we observed the violet trajectory only for 200/300 ms.

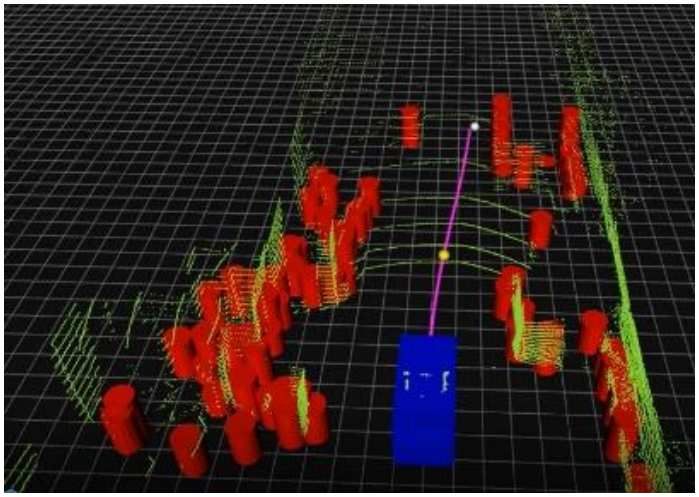


Figure 70: practical example of emergency braking

Chapter 6: Conclusion and future development

In conclusion, what has been realized is a software program capable of processing row data coming from the detection sensors, in order to generate an obstacles avoidance trajectory to reach the final waypoint of the path. The trajectory planner receives in input the occupancy grid map and with a frequency of 10 Hz a spiral trajectory, which connects the current vehicle position to a reference waypoint, is returned; on the spiral, with a criterion experimentally validated, a specific point that gives indication of the actuation variables (car speed and steering angle) is properly chosen.

It is important to underline that the final performances are satisfactory if the surrounding environment is static; at the current stage, the program is not equipped with any artificial intelligence: the line following procedure, the street signals recognition are not implemented yet. Plus, the algorithm only detects obstacles, without differentiating pedestrians from cars or other static objects: the current behavior is based only on the obstacle avoidance; thus, the vehicle will only try to avoid obstacles on the path, calculating every 100 ms an optimal trajectory; if a valid trajectory cannot be found the velocity is set to zero. Plus, in case of emergency braking, the only action we could take has been to send in output a null reference speed, so the vehicle will keep moving for a few meters, according to its inertial mass: this limitation is due to the fact that a braking By-wire system was not implemented on the VEGA vehicle.

The future suggested developments are linked to the critical issues mentioned above; plus, it is imperative to work on a proper sensor fusion,

integrating the Velodyne LiDAR and the GPS sensor with cameras: in this way different objects, by means of computer vision, can be classified and, consequently, the vehicle may be able to take proper decisions according to the kind of object met on its path.

Bibliography

[1] Medium. 2021. *How RADARs work*. [online] Available at: <https://medium.com/think-autonomous/how-radars-work-1eb523893d62> [Accessed 12 October 2021].

[2] Techcrunch.com. 2021. *TechCrunch is now a part of Verizon Media*. [online] Available at: <https://techcrunch.com/2021/01/04/lidar-startup-aeva-raises-another-200m-ahead-of-its-debut-as-a-public-company/?guccounter=1&guce_referrer=aHR0cHM6Ly93d3cuZ29vZ2xlLmNvbS8&guce_referrer_sig=AQAAACRIPrEJUCEXlKqKrszG5Ng-m6p02OaZ9whXXY1EmHh5jPFgR9unL4iQUGeIdErmhWv3CCrBfKUsn2e_Duxv2d4aYJtvH0CAl-rumF5yOr1gh0mCT4d61aYoJ_5RBkT0gMCyMT030GU4SR2K6bbzjU7J3y8Tm0cIyudwd70ZvLXl> [Accessed 12 October 2021].

[3] FierceElectronics. 2021. *What is an Ultrasonic Sensor?*. [online] Available at: <https://www.fierceelectronics.com/sensors/what-ultrasonic-sensor#:~:text=An%20ultrasonic%20sensor%20is%20an,sound%20that%20humans%20can%20hear).> [Accessed 12 October 2021].

[4] Anderson, J., 2021. *Brief History and Current State of Autonomous Vehicles*. [online] Available at: <https://www.jstor.org/stable/10.7249/j.ctt5hhwgz.11> [Accessed 12 October 2021].

[5] Katrakazas, C., Quddus, M., Chen, W. and Deka, L., 2015. Real-time motion planning methods for autonomous on-road driving: State-of-the-art and future research directions. *Transportation Research Part C: Emerging Technologies*, 60, pp.416-442.

[6] Talamino, J. and Sanfeliu, A., 2019. Anticipatory kinodynamic motion planner for computing the best path and velocity trajectory in autonomous driving. *Robotics and Autonomous Systems*, 114, pp.93-105.

[7] Artunedo, A., Villagra, J. and Godoy, J., 2019. Real-Time Motion Planning Approach for Automated Driving in Urban Environments. *IEEE Access*, 7, pp.180039-180053.

[8] Samuel, M., Hussein, M. and Binti, M., 2016. A Review of some Pure-Pursuit based Path Tracking Techniques for Control of Autonomous Vehicle. *International Journal of Computer Applications*, 135(1), pp.35-38.

[9] Mouhagir, H., Cherfaoui, V., Talj, R., Aioun, F. and Guillemard, F., 2017. TRAJECTORY PLANNING FOR AUTONOMOUS VEHICLE IN UNCERTAIN ENVIRONMENT USING EVIDENTIAL GRID. *IFAC-PapersOnLine*, 50(1), pp.12545-12550.

[10] Pipe, J. and Zwart, N., 2013. Spiral trajectory design: A flexible numerical algorithm and base analytical equations. *Magnetic Resonance in Medicine*, 71(1), pp.278-285.

[11] Li, Y. and Ruichek, Y., 2014. Occupancy Grid Mapping in Urban Environments from a Moving On-Board Stereo-Vision System. *Sensors*, 14(6), pp.10454-10478.

[12] Positioninguniversal.com. 2021. *CAN bus*. [online] Available at: <https://www.positioninguniversal.com/post/what-is-the-can-bus> [Accessed 12 October 2021].

[13] Rui, Y., 2014. Research on Identification of Vehicle Steering Angle. *Applied Mechanics and Materials*, 608-609, pp.787-793.

[14] Ghita, N. and Kloetzer, M., 2012. Trajectory planning for a car-like robot by environment abstraction. *Robotics and Autonomous Systems*, 60(4), pp.609-619.

[15] NVIDIA. 2021. *NVIDIA Jetson TX2: High Performance AI at the Edge*. [online] Available at: <https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-tx2/> [Accessed 12 October 2021].

[16] Velodyne Lidar. 2021. *Smart Powerful Lidar Solutions | Velodyne Lidar*. [online] Available at: <https://velodynelidar.com/> [Accessed 12 October 2021].

[17] Swiftnav.com. 2021. *Duro Inertial Ruggedized GNSS Receiver Ideal for Outdoor Deployments*. [online] Available at: <https://www.swiftnav.com/duro-inertial> [Accessed 12 October 2021].