Motorvehicle University of Emilia-Romagna

Master Degree in Advanced Automotive Electronic Engineering

MPC for Dynamic Obstacle Avoidance

Technical report on the group project activity

Course of Compliance Design of Automotive Systems

Group memebers:
Gianvincenzo Daddabbo
Gaetano Gallo
Alberto Ruggeri
Martina Tedesco
Alessandro Toschi

a.y. 2020-2021

Abstract

Every year 1.25 million people die and as many as 50 million are injured in road traffic accidents worldwide, according to United Nations statistics. Human error is involved in about 95% of all road traffic accidents in the EU, and in 2017 alone, 25300 people died on the Union's roads [1]. Autonomous cars can improve road safety and through new technologies it is also possible to reduce traffic congestion and CO2 emissions.

The aim of this report is to present the steps that we have followed as a group in order to implement a system for dynamic obstacle avoidance using an adaptive Model Predictive Control. In the simulated environment, the vehicle model is expected to encounter and safely avoid the dynamic obstacles along its way. On the other hand, when an obstacle is not present ahead of the autonomous vehicle, it is expected to follow a predetermined path.

Contents

List of Figures

List of Tables

1 Introduction

The task of avoiding obstacles is one of the key issues when it comes to the vehicular scenario and it is more difficult to perform it here than it is in static environments. A vehicle with an obstacle avoidance system is equipped with sensors that measure the distance between the car itself and the obstacle in the same lane. If an autonomous car encounters an obstacle, it is expected to move temporarily to another lane and move back to the original lane once it has driven past the obstacle. This type of control can be implemented using a Model Predictive Control (MPC). An MPC is an advanced control method that works in discrete time and uses a model of the system to make predictions about the system's future behavior. Figure 1 graphically shows the general concept behind a Model Predictive Control.

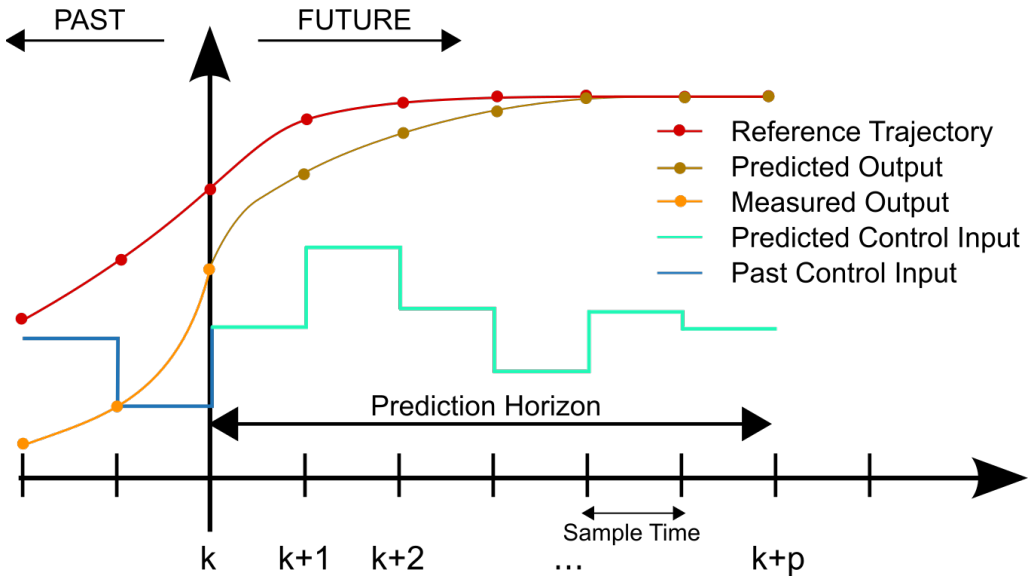


Figure 1: MPC algorithm scheme

The MPC solves an online optimization algorithm to find the optimal control action that drives the predicted output to the reference [2]. In other words, from a set of state values, and with respect to a model, it optimizes a problem around an objective and gives a sequence of control signals as outputs. The first set of control values are then used as inputs to the system plant, and after a short period, set as the *system time step*, the new state values are measured and the process is repeated. The overall objectives of the MPC are:

- prevent that input and output constraints are violated;
- optimize some input variables, while other outputs are kept in specified ranges,
- prevent the input variables from having excessive variations.

Figure 2 shows a block diagram for a Model Predictive Control.

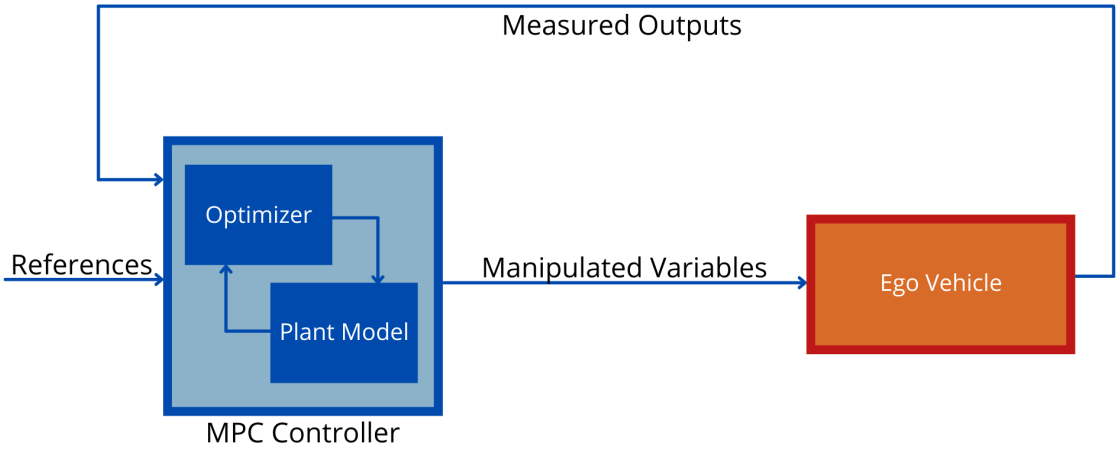


Figure 2: MPC Block Diagram

An MPC controller has two main functional blocks: the optimizer and the plant model. The dynamic optimizer allows to find the optimal input that gives the minimum value of the cost function taking into account all the constraints. Generally, a non linear model is used for the validation of the controller, while the plant model used for the MPC is a linearized version of the actual plant.
Our project is based on the use of an *adaptive* MPC which means that the plant state has to be measured again to be adopted as the linearization point for the next step of the predictive control. When the plant state is re-sampled, the whole process computes again the calculations starting from the new current state.

1.1 Model-Based Design

When developing a project, especially concerning embedded systems, it is crucial to follow a process model which illustrates the high-level activities and their phasing during development.
As stated in [3] *"Process models provide high-level perspective that helps team members understand what activities to do and what progress has been made on each of those development activities"*.
The development process of our system is based on the V-model shown in Figure 3. Starting from the general V-model we have tailored our own V-model in order to meet our needs; doing so we have managed to skip some unnecessary steps in the development process while still being compliant with the definition of the V-model itself.

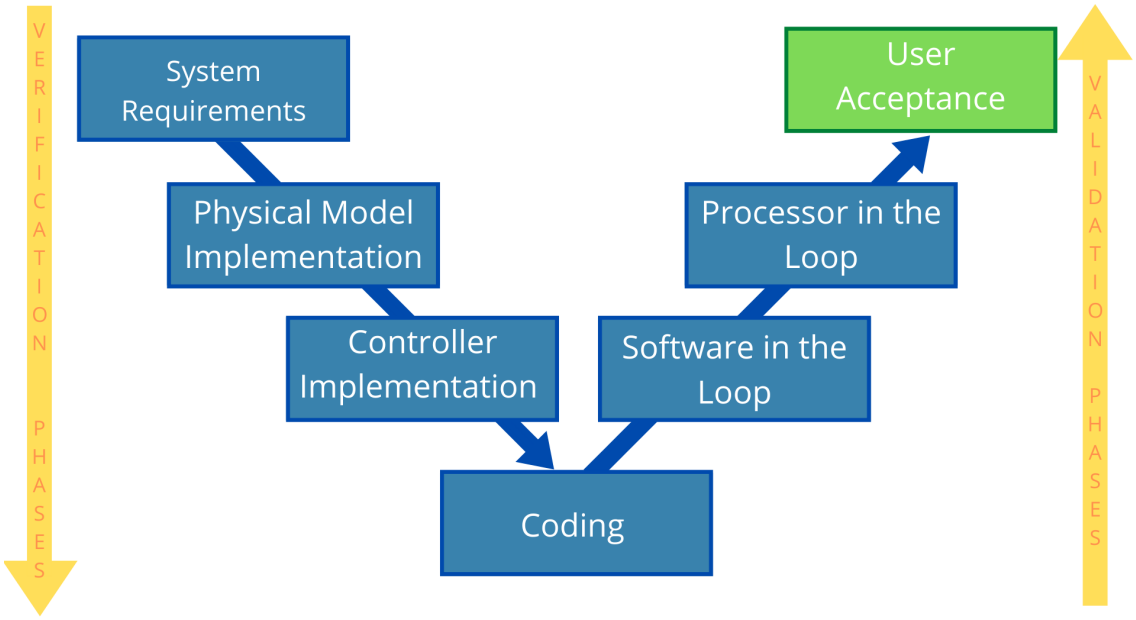


Figure 3: System Development V-model

As shown above the V-model is a representation of system development process that highlights verification steps on one side and validation steps on the other of the 'V'. The left side of the 'V' identifies the verification phase, containing the steps that lead to code generation while the right side identifies the validation phase which ends with the user acceptance. Below a brief description of the steps which constitute our V-model:

- **System requirements**: this is the first step of each project, it consists in the requirements gathering from the customer;
- **Physical model implementation**: this phase contains the system design of our vehicle in terms of sensors required and dynamic and kinematic model;
- **Controller implementation**: this phase is the focus of our project, designing an adaptive MPC.
- **Coding**: exploiting some specific tools, we can automatically generate code for embedded deployment and create test benches for system verification, saving time and avoiding the introduction of manually coded errors;
- **Software-In-the-Loop**: once generated, the code is tested in a simulation environment;
- **Processor-In-the-Loop**: during this phase the code is uploaded on a demo board and than it is tested again;
- **User Acceptance**: in this phase the final product is tested in order to verify the compliance with the initial costumer requirements.

1.2 Required tools

The project is based on MATLAB, a proprietary multi-paradigm programming language and numeric computing environment developed by *MathWorks*, and Simulink, a MATLAB-based block diagram environment for multi-domain simulation and Model-Based Design.

The following list represents the software requirements for the project, including all the applications needed for the proper execution and test:

- **MATLAB R2019a or newer**;
    1. **Curve Fitting Toolbox**: needed to make the reference map gentler;
    2. **Automated Driving Toolbox**: needed to get a reference map from real world data;
    3. **Model Predictive Control Toolbox**: needed to have all the useful resources to develop the MPC.
- **Simulink 9.3 or higher**;
    1. **Simulink Test**: used to perform automated tests and requirements verification;
    2. **Embedded Coder**: used for automatic code generation.

2 System Requirements

Defining the project requirements is an essential and crucial step to accomplish in the starting phase of the project. Indeed, from both the perspective of the customer and of the project developers, detailed requirements are necessary to deliver exactly what is needed.
Firstly, the requirements are defined by the customer at a high level, then "translated" by the developers to a lower and technical level and later on validated to ensure that the project requirements are correct, free of defects/bugs, and meets the needs of the users.
For the sake of this project we have assumed to receive the requirements from a hypothetical customer. Namely, there are six high level requirements given for this obstacle avoidance implementation that are mainly related to the accuracy of the system and to the driver comfort:

1. Maximum lateral error from reference of $0.75m$;
2. Detection of obstacles within $100m$ ahead of vehicle;
3. Move on left lane within a predetermined safe zone from the obstacle[1];
4. Once passed the obstacle, come back on right lane with no less than $10m$ but no more than $50m$ ahead of it;
5. Maximum lateral acceleration of $2m/s^2$;
6. All previous requirements satisfied in speed range from $10km/h$ to $100km/h$.

[1] Third and fourth requirement are requested when an obstacle is detected.

3 System Implementation

As briefly discussed in the *Introduction* section, nowadays one of the smartest techniques deployed for autonomous vehicles control is the Model Predictive Control, hence our decision to develop a controller in order to accomplish the path following and obstacle avoidance task. The design of such a controller needs a deep analysis of the project requirements and a smart system partitioning to identify different tasks that can be developed autonomously and independently from others to be then merged in order to satisfy the control problem.

3.1 Analysis of requirements

Starting from the detailed system requirements described in Section 2, we can obtain low level requirements suitable for the controller implementation; each of these is associated with a draft of the validation method needed to assert each of them. Table 1 shows the system requirements, low level requirements and the corresponding validation methods.

System requirements	Low level requirements	Validation methods
Maximum lateral error from reference of $0.75m$	Inequality constraint: $e_y = abs(Y_{reference} - Y_{vehicle}) <= 0.75m$	Check in each point that the difference between the actual vehicle position and the reference path is not greater than $0.75m$
Detection of obstacles within 100 m ahead of vehicle	Sensor fusion able to provide useful data in the range $0 - 100m$ from the vehicle	Drive toward a static obstacle and confirm that is detected when is $100m$ far
Move on left lane within a predetermined safe zone from the obstacle	Inequality constraint when obstacle detected: $dist = position_{vehicle} - position_{obstacle} \in SafeZone$	Confirm that when changing lane the distance from the obstacle measured by the sensors is greater than the predetermined safe zone
Once passed the obstacle, come back on right lane with no less than $10m$ but no more than $50m$ ahead of it	Inequality constraint when obstacle passed: $10m <= dist <= 50m$	Confirm that when changing lane the distance from the obstacle measured by the sensors is greater than the predetermined safe zone
Maximum lateral acceleration of $2.0m/s^2$	Constraint on Model Predictive Control input: $a.max = 2.0$	Check that the maximum value registered by the IMU−accelerometer is not greater than $2.0m/s^2$
All previous requirements satisfied in speed range from $10km/h$ to $100km/h$	Verify all previous constraint when simulating at speed range $10 - 100km/h$	Verify all the above methods with vehicle travelling from $10km/h$ to $100km/h$

Table 1: System requirements and possible validation methods

3.2 Partitioning

Regarding the system partitioning, which as already said in Section 3 is a crucial part of the implementation of a system, we have decided to base the implementation of our

project on the tasks shown in Figure 4.

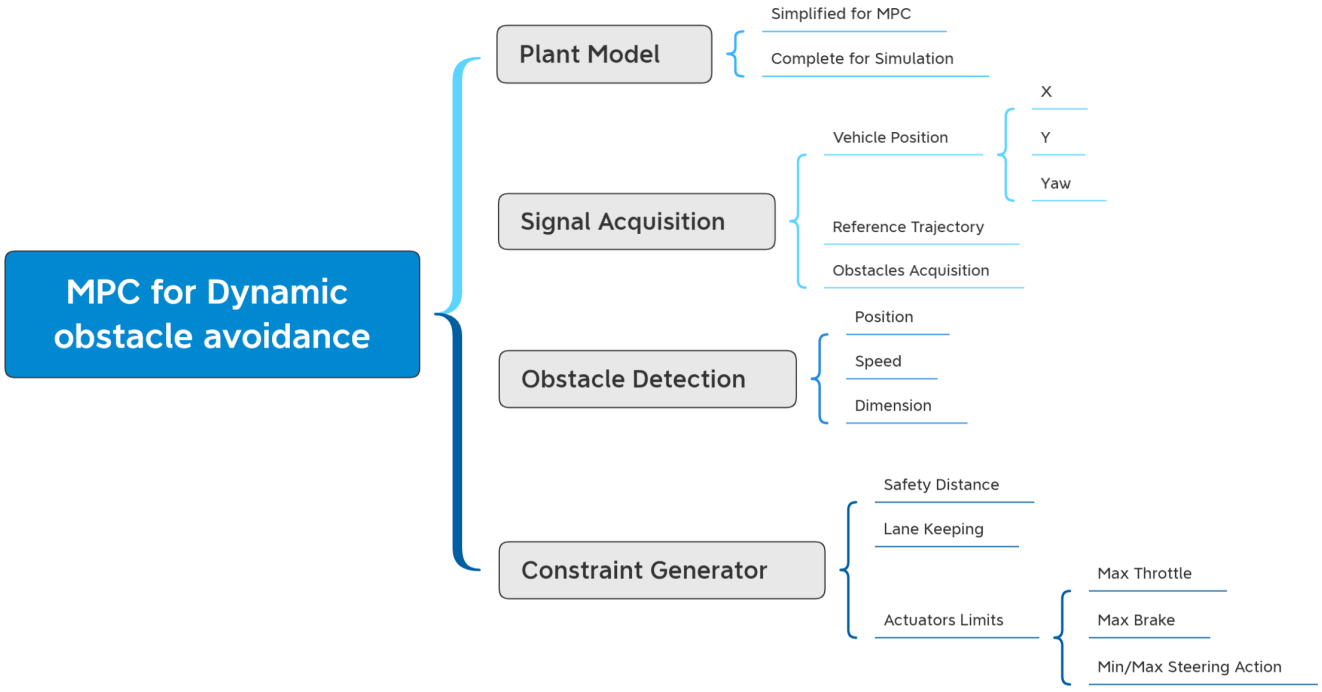


Figure 4: System Partitioning

As shown in the figure above, our project can be divided into four macro-areas:

- Plant Model: is the mathematical model of the vehicle which represent the development environment of our project. We have decided to adopt two models, a simplified one to be used in the MPC and a more accurate one to be used in the simulation;
- Signal Acquisition: refers to the gathering of data related to the vehicle position, trajectory, and obstacle acquisition, all coming from the sensors mounted on our vehicle;
- Obstacle Detection: here the properties of the obstacle such as position, speed, and dimension are evaluated;
- Constraint Generator: in this phase we set the constraints which the MPC needs to comply with. These constraints are necessary in order to make our model as realistic and safe as possible.

These sections will be described in depth in the following chapters.

4 Plant Model

The starting point for Model Based design is to develop and implement a plant model, so a model representing the Physics of the system considered.
Different models are available in literature to model a vehicle. Since the target of the project is to develop a lateral controller, one of the most suitable model is the so called *Bicycle Model*.
The kinematic bicycle model is described by the following non linear system:

$$\dot{X} = V cos(\psi + \beta) \tag{1}$$

$$\dot{Y} = V sin(\psi + \beta) \tag{2}$$

$$\dot{\psi} = \frac{V cos(\beta)}{l_f + l_r} \left(tan(\delta_f) - tan(\delta_r)\right) \tag{3}$$

$$\beta = tan^{-1}\left(\frac{l_f tan(\delta_r) + l_r tan(\delta_f)}{l_f + l_r}\right) \tag{4}$$

Where:

- X and Y are the coordinates of the body of the vehicle in the global reference frame,
- ψ is the yaw (orientation angle),
- β is the slip angle of the body,
- $\beta + \psi$ is the body speed direction known as *Course Angle*,
- δ_f is the front steering angle,
- δ_r is the rear steering angle,
- V is the magnitude of the body speed,
- l_f is a geometric parameter which indicates the distance of the CoG from the front wheel,
- l_r is a geometric parameter which indicates the distance of the CoG from the rear wheel.

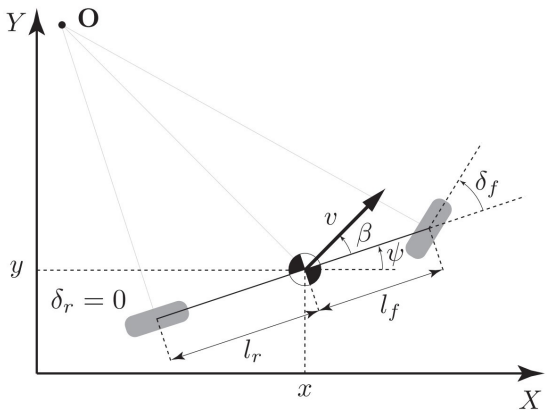


Figure 5: Bicycle kinematic vehicle model

This model can be further simplified assuming that, as in the most of commercial vehicles, only the front wheels are able to steer, so the rear steering angle δ_r is constant and equal to 0.
Standard MPC controller works on linear systems, so we decided to implement a linearized version of the Bicycle Model to feed the controller. This linearized model is described by the following State Space equations:

$$\dot{x} = Ax + Bu \tag{5}$$

$$y = Cx + Du \tag{6}$$

where:

$$x = \begin{bmatrix} X \\ Y \\ \psi \\ v \end{bmatrix} \quad A = \begin{bmatrix} 0 & 0 & -Vsin(\psi) & cos(\psi) \\ 0 & 0 & Vcos(\psi) & sin(\psi) \\ 0 & 0 & 0 & \frac{tan(\delta)}{l_r+l_f} \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & \frac{Vtan(\delta)^2+1}{l_r+l_f} \\ 1 & 0 \end{bmatrix} \tag{7}$$

$$u = \begin{bmatrix} Throttle \\ \delta \end{bmatrix} \qquad C = I^{4\times4} \qquad D = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

These matrices are obtained evaluating the Taylor expansion of the non linear bicycle model, as explained in the MathWorks example [4].
A more complex vehicle model can be useful to test the performance of the controller to be developed. The *Dynamic Bicycle Model* [5] can be built starting from the previous non linear model, deriving the following second order derivative equations:

$$\ddot{x} = \dot{\psi}\dot{y} + Throttle \tag{8}$$

$$\ddot{y} = -\dot{\psi}\dot{x} + \frac{2}{m}\left(F_f cos(\delta) + F_r\right) \tag{9}$$

$$\ddot{\psi} = \frac{2}{I_z}\left(l_f F_f - l_r F_r\right) \tag{10}$$

$$\dot{X} = \dot{x}cos(\psi) - \dot{y}sin(\psi) \tag{11}$$

$$\dot{Y} = \dot{x}sin(\psi) + \dot{y}cos(\psi) \tag{12}$$

where m is the mass of the vehicle, I_z is the inertia of the vehicle with respect to the vertical axle passing for the CoG of it, while F_f and F_r are respectively the side slip forces acting on the front and rear wheels, and they can be evaluated as:

$$F_f = 2C_f\left(\delta - \theta_f\right) \tag{13}$$

$$F_r = 2C_r\left(-\theta_r\right) \tag{14}$$

with C_i side slip friction coefficient of the i-th wheel couple and θ_i side slip angle of the wheels.
Tyre side slip angles can be approximated by the following equations:

$$\theta_f = tan^{-1}\left(\frac{\dot{y} + l_f\dot{\psi}}{\dot{x}}\right) \tag{15}$$

$$\theta_r = tan^{-1}\left(\frac{\dot{y} - l_r\dot{\psi}}{\dot{x}}\right) \tag{16}$$

The models introduced are, more or less, independent of the longitudinal dynamics of the vehicle, since they link this dynamic with a simple coefficient, that is the $Throttle$. In our model, this coefficient should be considered as an acceleration of the vehicle, and it can be both positive (driving) or negative (braking). Range of values for this parameter is dependent on lots of variables, of course, such as the engine power, the vehicle mass and inertia, the ground type, the rubber of the wheels and so on. Our assumption is to give a fixed interval for this parameter, to simulate a small commercial vehicle on dry asphalt, which can have a maximum acceleration of $4m/s^2$ and a maximum braking deceleration of $-0.8g$. To summarize:

- $Throttle \in [-7.85, 4.00]m/s^2$

Parameters considered for the bicycle model are taken from real vehicle data [6] and are reported in the following table:

ID	**Vehicle name**	**Wheel base** $[m]$	**l_r** $[m]$	**l_f** $[m]$	**Mass** $[kg]$	**Inertia** $[kg \cdot m^2]$
1	*Hyundai Azera*	2.843	1.738	1.105	1200	1000
2	*BMW 325i*	2.570	1.369	1.201	1251	2027
3	*Ford E150*	3.505	1.634	1.871	2995	6536
4	*Suzuki Samurai*	2.032	0.870	1.162	1229	1341
5	*Volkswagen Beetle*	2.408	0.996	1.412	857	1289

Table 2: Vehicle data considered for development and validation

Values reported in the table 2 are stored in a MATLAB file and can be accessed through the *loadParameters* function, which take as input the vehicle ID and returns the parameters associated with that vehicle. To avoid to call improperly this function, a default parameters set as been provided with the following values:

Vehicle name	**Wheel base** $[m]$	**l_r** $[m]$	**l_f** $[m]$	**Mass** $[kg]$	**Inertia** $[kg \cdot m^2]$
Default	2	1	1	1000	1000

We used data of vehicle 1, *Hyundai Azera*, for the development phase, where mass and inertia are not the real ones of the vehicle, but are given with realistic values, while other data of the previous table 2 are meant to be used in the validation phase, to test the controller with different vehicles. For what concern the side slip friction values, we considered two fixed values for all the vehicles that are:

- $C_f = 1.0745 \times 10^5$ N/rad
- $C_r = 1.9032 \times 10^5$ N/rad

while in the *"default"* condition they are both 10^5 N/rad.

4.1 Model comparison

As specified in the subsection 3.2 we have decided to use two models, a simplified and linearized model for the MPC and a more complex one for the simulation of the model. The former is defined as *Kinematic Bicycle Model* and the latter as *Dynamic Bicycle Model*. Hence we have decided to test both of these systems to show how they behave with different throttle and steering inputs using the *Simulink Test* tool. Thus, the tests performed are:

- **Free evolution test**: this test has been performed considering a constant steering of 0° and a constant throttle of 0 m/s^2;
- **Only throttle test**: this test has been performed keeping the steering angle constant and equal to 0° and varying the throttle as shown in Figure 6;
- **Constant steering test**: this test has been performed keeping the throttle constant and equal to 0 m/s^2 and the steering angle constant and equal to 2°;
- **Ramp steering test**: this test has been performed keeping the throttle equal to 0 m/s^2 and giving a ramp steering angle signal (varying linearly from 0° to 36°);
- **Small sinusoidal steering test**: this test has been performed keeping the throttle constant and equal to 0 m/s^2 and giving a sinusoidal steering angle signal with frequency 0.2 Hz and amplitude 5°;
- **Big sinusoidal steering test**: here we have performed the same test as before (sinusoidal steering input) but with a larger amplitude of the sine wave (15°);
- **Combined test 1**: this test has been performed keeping the steering angle constant and equal 2° and varying the throttle as shown in Figure 6;
- **Combined test 2**: this test has been performed keeping the throttle equal to 0.2 m/s^2 and giving a ramp steering angle signal (varying linearly from 0° to 36°).

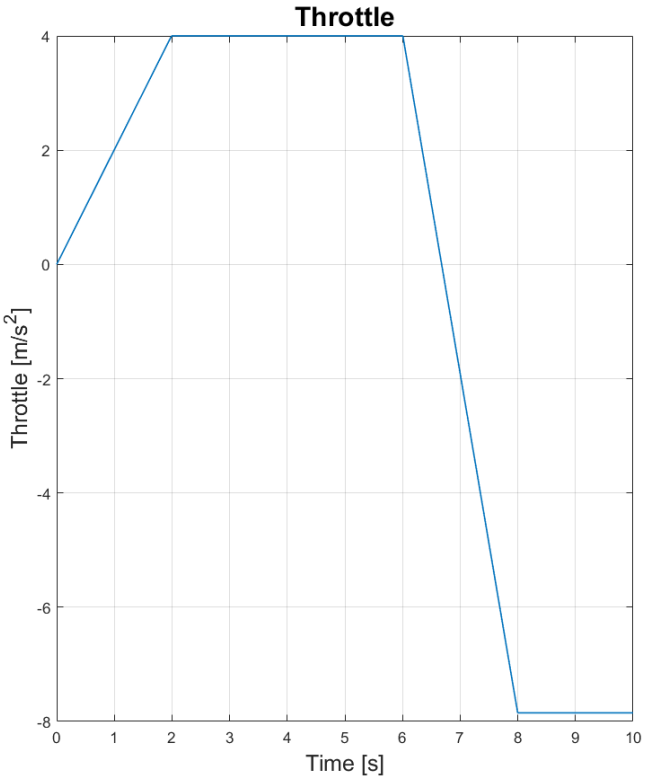


Figure 6: Throttle Input used in *Only throttle test* and *Combined test 1*. Maximum and minimum values are coherent with the range described in Section 4 in order to explore the whole set of throttle values.

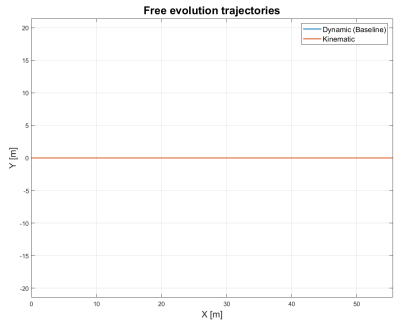
4.1.1 Simulink Test

The above-mentioned tests have been performed using the *Simulink Test* tool implemented by MathWorks. Simulink Test provides tools for authoring, managing, and executing systematic, simulation-based tests of models, generated code, and simulated or physical hardware. It includes simulation, baseline, and equivalence test templates that let you perform functional, unit, regression, and back-to-back testing using software-in-the-loop (SIL), processor-in-the-loop (PIL), and real-time hardware-in-the-loop (HIL) modes. With Simulink Test you can create non-intrusive test harnesses to isolate the component under test. You can define requirements-based assessments using a text-based language, and specify test input, expected outputs, and tolerances in a variety of formats. [7]

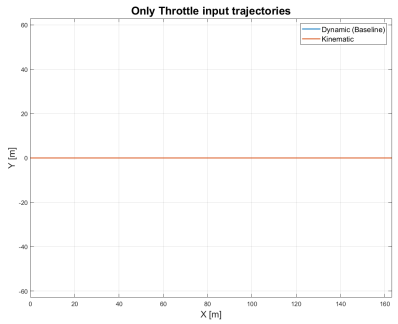
4.1.2 Results analysis

Exploiting Simulink Test, we have created two test harnesses : *Dynamic* and *Kinematic*. Those are referred to the models described at the beginning of this section. Then, we have executed an Equivalence Test which allowed us to make a comparison between two simulations. In particular, the Dynamic Model has been considered as the *baseline* which our Kinematic Model has been compared to in the test. The baseline represents the expected output being more accurate than the Kinematic model, which in turn is called *Compare to* model inside the Simulink Test automatic report generator[2]. Since these tests aim to give a qualitative analysis of the two models and compare them, we have not included equivalence criteria. The only parameter we have set up is the relative tolerance assigning to it a value of 1% in order to ignore the negligible offsets between the dynamic model and the kinematic model.
Figure 7 shows the results of the simulations we have carried out. As shown in Figures 7a and 7b respectively, as expected the two models behave in the same way when no inputs are present or when only the longitudinal input (throttle) is changed.
Another relevant result is shown in Figures 7e and 7f: as long as the amplitude of the sinusoidal input is fairly small, the two models behave in a similar fashion but the greater the amplitude of the sinusoidal input, the greater the deviation of the kinematic model from the dynamic one.
The other images shown below, underline as well the offset between the two vehicle models, which is due to the fact that when using the dynamic model we take into account lateral slips which are not considered in the kinematic model.

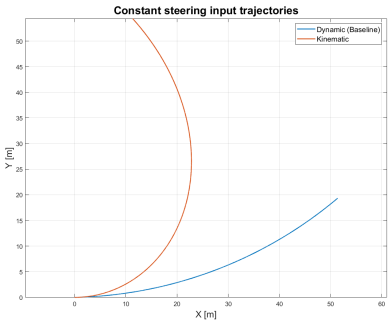
[2]The "Model Comparison - Test Report" file generated by Simulink Test is included in the /Documentation/Test Reports/ file path.



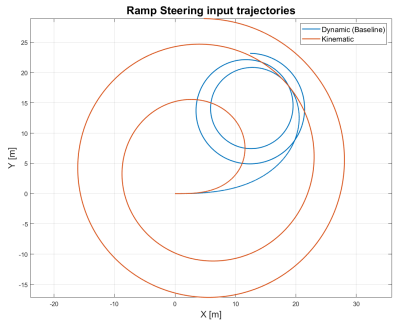
(a) Free evolution test - trajectories



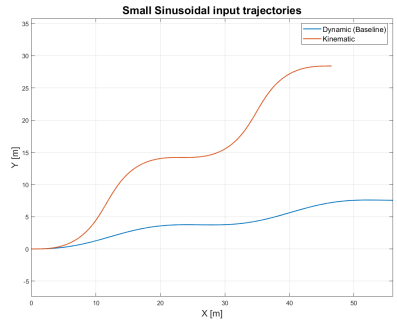
(b) Only throttle test - trajectories



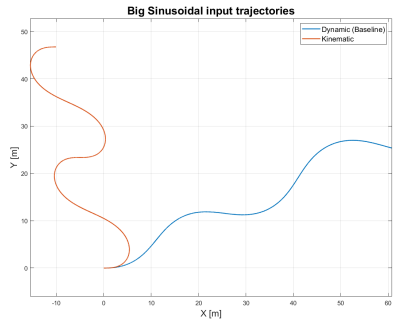
(c) Constant steering test - trajectories



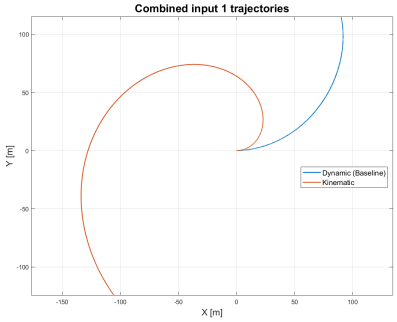
(d) Ramp steering test - trajectories



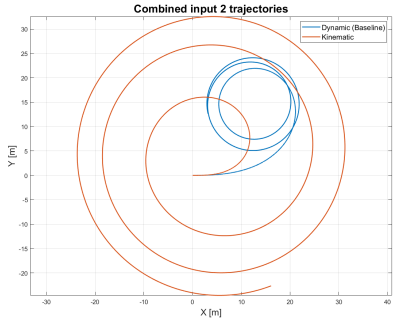
(e) Small sinusoidal test - trajectories



(f) Big sinusoidal test - trajectories



(g) Combined 1 test - trajectories



(h) Combined 2 test - trajectories

Figure 7: Output trajectories of the test set

After the analysis of the results, we decided to provide to the MPC the linearized bicycle model for the vehicle plant, because the controller is usually updated with the current state in some fractions of a second, and for this reason the differences between the model and the *"Real World"* system, that in our case is represented by the dynamic model, shouldn't affect too much the behaviour of the controller itself. As a matter of fact, by looking at the images above it is clear that the difference between the two models output is negligible at the first time steps of the simulation.
Moreover, as already discussed in this section, the performances of the two models are very similar if compared with small inputs, and this condition is fulfilled if we consider to use our controller in normal driving scenarios, such as, for instance, an highway.
Keeping in mind these assumptions, we go on with the project, considering to apply a change in the vehicle model for the MPC if we are not able to achieve good results with this one.

5 Signal Acquisition

All autonomous vehicles must have a sufficient knowledge of the surrounding environment and obtain accurate location information. For this purpose different sensors need to be adopted on a single vehicle, as a matter of fact different sensor have different limitations which can be overcome through sensor fusion. This approach is able to compensate the limitations of one sensor with the strengths of another.

5.1 Sensors used in obstacle avoidance

When dealing with autonomous driving, three are the core functions of interest:

- perception: visual perception (e.g., camera) and radar perception (e.g., LIDAR);
- planning: GPS, Inertial Navigation System (INS), and HD Maps;
- control: image identification, deep learning and artificial neural networks.

5.2 Perception: what a self-driving car sees

The three primary autonomous vehicle sensors for perception are camera, radar and lidar. Working together, they provide the car visuals of its surroundings and help it to detect the speed and distance of nearby objects, as well as their three-dimensional shape. [8]

5.2.1 Camera

Cameras are the main type of sensors for creating a visual representation of the surrounding world, as a matter of fact autonomous vehicles rely on cameras placed on every side to stitch together a 360 degrees view of their environment. Though providing very accurate images up to 80 meters, cameras have limitations:

- the distance from target objects needs to be calculated in order to know exactly where they are;
- mechanical issues: mounting multiple cameras as well as keeping them clean;
- issues related to low visibility conditions (e.g., rain, fog, nighttime);
- heavy graphic processing needed.

5.2.2 Radar

Radars can overcome cameras limitations in low visibility scenarios and improve object detection. The working principle of radars is based on radio pulses emitted by a source. Once these pulses hit an object they travel back to the sensor providing information about its speed and location. The maximum distance range for radar-based sensors is 250 meters for Long-Range Radars, 100 meters for Medium-Range Radars and 30 meters for Short-Range Radar.

5.2.3 Lidar

The name LiDAR stands for Light Detection and Ranging and its working principle is analogous to the radar's (Sub-section 5.2.2) but using a different part of the electromagnetic spectrum. Radars use radio waves or microwaves while lidars use light near the visible spectrum. This type of sensors make it possible for an autonomous car to create a 3D point cloud map and they provide shape and depth of the surroundings and its actors.

5.3 Planning: know where a self-driving car is going

"As the core of the planning layer, localization and navigation technologies including GPS, INS, and HD maps aim to assist self-driving cars in planning routes and navigating in real time." [9]

The above-mentioned technologies which have to be always active, allow the vehicle to go from point A to point B following the optimal route, and must re-compute in real-time the path if the optimal one has any unexpected diversions using a well tuned controller.

5.3.1 GPS

GPS stands for Global Positioning System. A GPS receiver is able to compute the current position of the vehicle together with the current time, acquiring and analysing signals received from at least four of the constellation of over 60 low-orbit satellites. The accuracy on the position information is 1 meter.

5.3.2 INS

INS stands for Inertial Navigation System and allows the accurate sensing of high-precision 3D position, velocity, and attitude information of the car via the inertial measurement unit (IMU). It can make up for the deficiency of GPS localization, which is not real-time enough.

5.3.3 HD Maps

HD Maps are crucial when it comes to Self-Driving Vehicles, these high-definition 3D maps are highly accurate and contain details not normally present on traditional maps. Such maps can be precise at a centimetre level. HD Maps consist of two map layers: static and dynamic HD Map. The former is usually collected in advance and contains lane models, road information, and road attributes while the latter, stored on a cloud platform, contains real-time traffic information. Moreover, dynamic HD Maps can be updated in real-time through information collected from vehicles and infrastructures on the cloud.

5.4 Our Assumptions

Our model is based on several assumptions regarding data acquisition from the sensors and how these data are translated into useful information. Below the list of assumptions we have made:

- we suppose no error in the information collected from the sensors;
- our car knows all the information related to the road model, such as lane width and number of lanes by acquiring them from cameras installed on the vehicle and HD Maps;

- the vehicle knows its position in the XY reference frame, through the use of GPS and INS;
- the route planning done by our controller exploits the latitude and longitude information to plan out a complete route based on the HD Maps, GPS, and INS. Moreover, while driving, the car also compares the HD maps with the perception information from sensors to get changing environment information for dynamically planning routes and making decisions;
- we suppose to have both a Long-Range Radar and Lidar mounted on the vehicle, hence to be able to detect obstacles in the range of 200 meters together with their velocities.

Though we include all the above information on the MATLAB code, we suppose to collect them using sensors, as previously described.

5.5 Reference trajectory

After implementing the vehicle model and keeping in mind the assumptions made above, we have generated and imposed a reference trajectory to the vehicle in order to have a path to follow during the development and the testing phases. To do so, we have taken the desired scenario from a real map using the open source website *OpenStreetMap*[10] (Figure 8). The website allows the user to export any route as a *.osm* file containing the global coordinates of the area together with some useful metadata such as street name, number of lanes, speed limits, etc.

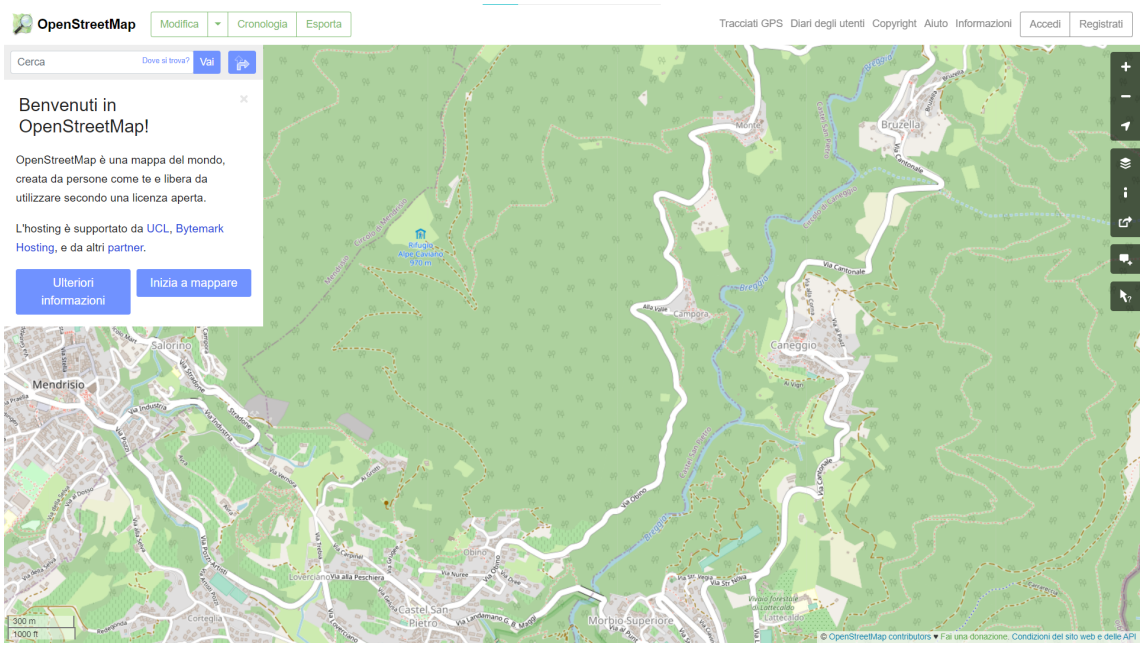


Figure 8: Selecting the area of interest in *OpenStreetMap*

Next, we have imported the *.osm* file in MATLAB exploiting the *Driving Scenario Designer* tool (Figure 9) provided as part of the Automated Driving Toolbox. The Driving Scenario Designer tool splits the whole selected route in as many pieces as the number of different streets names present. Then, the user can select only the pieces of road he/she is interested in and export its waypoints in a *.mat* file.

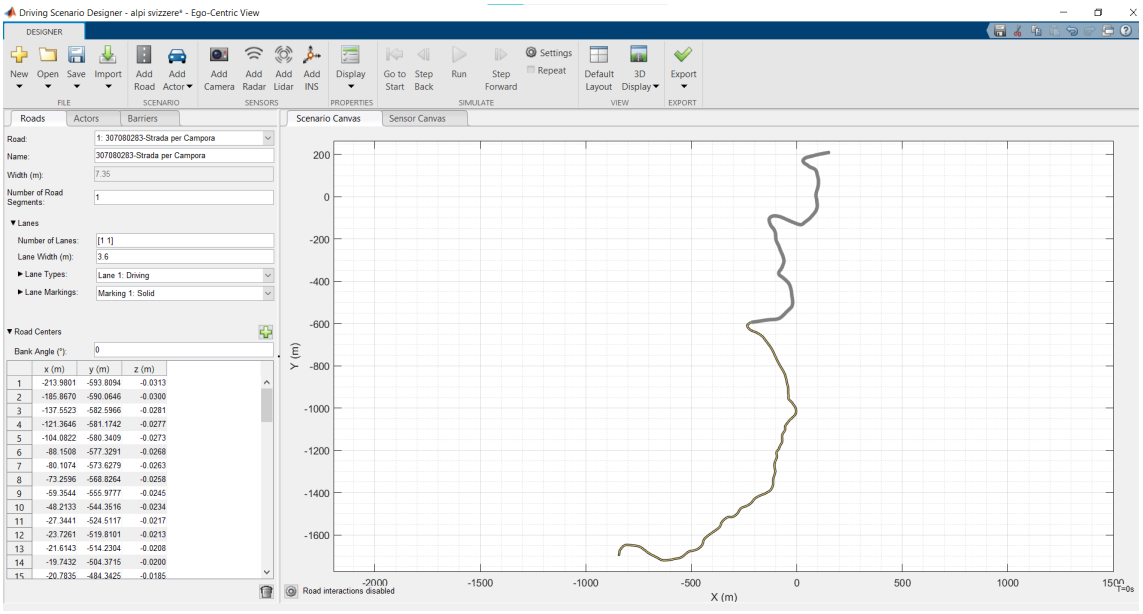


Figure 9: Importing maps on MATLAB using *Driver Scenario Designer*

Afterwards, the *.mat* file is used to generate the corresponding map in the XY reference frame. Since the thus generated map is composed by an insufficient number of waypoints for our applications, we have decided to create a function *ScenarioLoading* able to do an up-sample of the map because a higher density of points on it optimizes the path following algorithm. The up-sampling function interpolates linearly the original map with a fixed step depending on the reference speed and the sampling time selected for the controller. It is worth mentioning, as shown in Figure 10, that the up-sampled path (in orange) and the original waypoints (in blue) slightly deviate in some sections of the route due to the smoothing applied in order to obtain a more gentle trajectory.

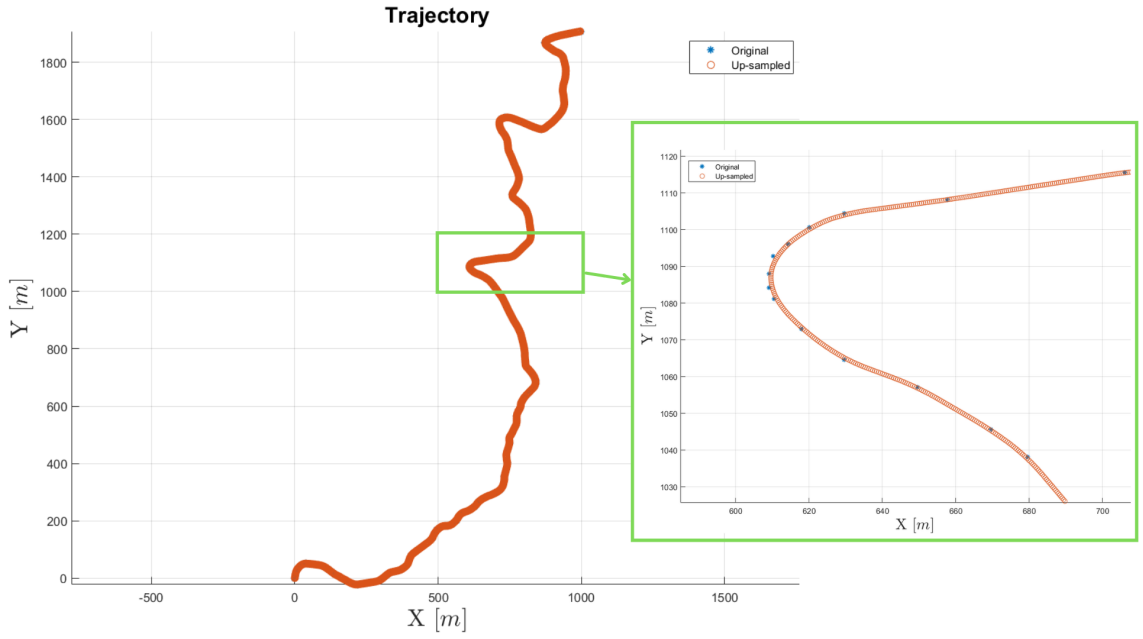


Figure 10: Comparison between original and up-sampled path

6 Constraints

In this section we are going to introduce the constraints we have used for the project. A plausible set of constraints is fundamental in order to perform a realistic simulation, which is required in order to have reasonable simulation outcomes.

6.1 Input constraints

The vehicle model presented in the Section 4 assumes that the inputs δ_f (front steering angle) and Throttle (driving and braking) can be controlled directly. In practice, however, low-level controllers are used to transform the aforementioned commands into physical control signals.

6.1.1 Steering angle constraints

Regarding the steering angle control, we suppose that our MPC directly affects the Electronic Power Steering (EPS) system by sending to it the target steering wheel position. Then, the EPS control unit calculates the optimal steering output based on the target steering wheel position received and sends the information to an electric motor to provide the necessary action on the wheels. For the project, we have assumed this link to be ideal and so our MPC directly control the angle of our front wheels. The values used are reported in the Table 3 and they have been chosen according to real data[11].

	min	**max**
Steering angle	-36 deg	36 deg
Steering rate	-60 deg/s	60 deg/s

Table 3: Steering constraints

6.1.2 Throttle constraints

As stated in Section 4 the vehicle model used in our simulation is, more or less, independent of the longitudinal dynamics of the vehicle. Moreover, the set of simulation we carry out are to be performed at constant speed, as close as possible to the target speed value. Physical limits on the actuators and comfort requirements impose bounds on the throttle and its rate of change according to the Table 4.

	min	**max**
Throttle	-7.85 m/s^2	4 m/s^2
Throttle rate	-20 m/s^3	8 m/s^3

Table 4: Throttle constraints

6.2 Output/State constraints

MPC algorithms allow to define a set of constraints on the States/Output variable of the model. For our project, this feature allows to describe a forbidden zone in the nearby of the obstacle when it is detected. In order to properly define this area, we refer to the art.148 and art.149 of the Italian reference legislation "*Codice della strada*" which state

that while traveling and overtaking, the vehicles must keep a safety distance from the vehicle in front of them.

6.2.1 Safety distance

In order to estimate the optimal safety distance that each vehicle must keep from the other, several factors must be taken into account:

- the alertness of the driver's reflexes;
- the type and response of the vehicle;
- the vehicle speed;
- the visibility and weather conditions;
- the slope of the road and the characteristics and conditions of the road surface;
- the traffic.

Under the previous assumptions it's clear that it is impossible to exactly evaluate the safety distance because there are a multitude of factors that continuously change over time. Among these, the most important one is the speed at which the vehicle is travelling. A common formula relates the safety distance to the square of the speed:

$$SafetyDistance[m] = \left(\frac{v[km/h]}{10}\right)^2 \tag{17}$$

Moreover the safety distance is strictly correlated to the stopping space that is the sum of the reaction space and braking space. The former is the distance traveled by the vehicle in the reaction time (usually 1 second), the latter is proportional to the square of the speed and inversely proportional to the deceleration and the friction coefficient.

$$BrakingSpace = v^2/(2a \times \mu) \tag{18}$$

The following table presents the value of the braking space associated to different speed and friction coefficient values:

Vehicle speed	Braking space on dry asphalt ($\mu = 0,8$)	Braking space on wet asphalt($\mu = 0,4$)
20 km/h	2.0 m	3,9 m
40 km/h	7,9 m	15,7 m
60 km/h	17,7 m	35,4 m
80 km/h	31,5 m	62,9 m
100 km/h	49,2 m	98,3 m
120 km/h	70,9 m	141,7 m

Table 5: Braking distance with respect to vehicle speed and road condition

In our work we have decided to consider the *SafetyDistance* evaluated as in the Eq. 17 to define the aforementioned *"forbidden zone"*.

7 Controller Setup

Matlab's Model Predictive Control Toolbox provides functions, an app, and Simulink blocks for designing and simulating controllers using linear and nonlinear model predictive control (MPC) [12]. Thanks to the toolbox we specified the plant model, horizons, constraints, and weights. By running closed-loop simulations, we evaluated controller performance and adjusted its behavior by varying weights and constraints at run time.

7.1 MPC Initialisation

Initialisation phase requires the definition of all the parameters needed for the execution of the controller. First of all we have created a MPC object passing as input the discretized state-space model of the plant, the prediction horizon and the control horizon.
Assuming that the car is already traveling at a speed V, the initial conditions for the other states is set taking the first point in the reference path, and for the two inputs it is set equal to zero. Exploiting the function *odometer* we computed the total distance covered by the car following the scenario in order to let the simulation terminates exactly when the road ends.
Subsequently we defined the constraints on inputs (Table 3 and 4), constraints on the state and the weights for the cost function, which correctly tuned, guarantee that the vehicle follows the reference without going off the road. The cost function of the MPC takes as input the weights for all the states of the model, the input manipulated variables and the manipulated variables rate. In our case, the states of the plant model are four, and to tune the controller, we have to give a weight to each one of them. Our choice is to give the same weight to states x and y, which represent the position of the vehicle in the reference frame, and then to play with the weights on angle θ and v to tune the MPC.
Moreover we decided to give 0 weight to the manipulated variables.
Constant constraints, as the ones defined for the inputs, can be given to the controller directly in the initialisation phase, while constraints on the states, which are time varying, can be passed at runtime my means of the matrices E, F and G.
MPC algorithm try to solve the equation:

$$E \cdot u(k+j) + F \cdot y(k+j) \leq G \tag{19}$$

for all the steps in the prediction horizon, scanned by index j, where y is the vector of measured and unmeasured outputs and u is the vector of manipulated variables at instant k.

7.2 Simulink Model

Once the controller has been initialised, all is set to begin the simulation. In order to run closed-loop simulations a Simulink model has been designed because of the availability of many useful tools to test and validate the model. For the sake of clarity in reading and debugging the whole system, some of the Simulink blocks have been grouped in subsystems and a mask has been applied to some of them. In Figure 11 the main system is depicted: the process simulated by these blocks is coherent with the MPC algorithm described in the previous subsection and the numbers in green enumerate the four major subsystems which will be analyzed below.

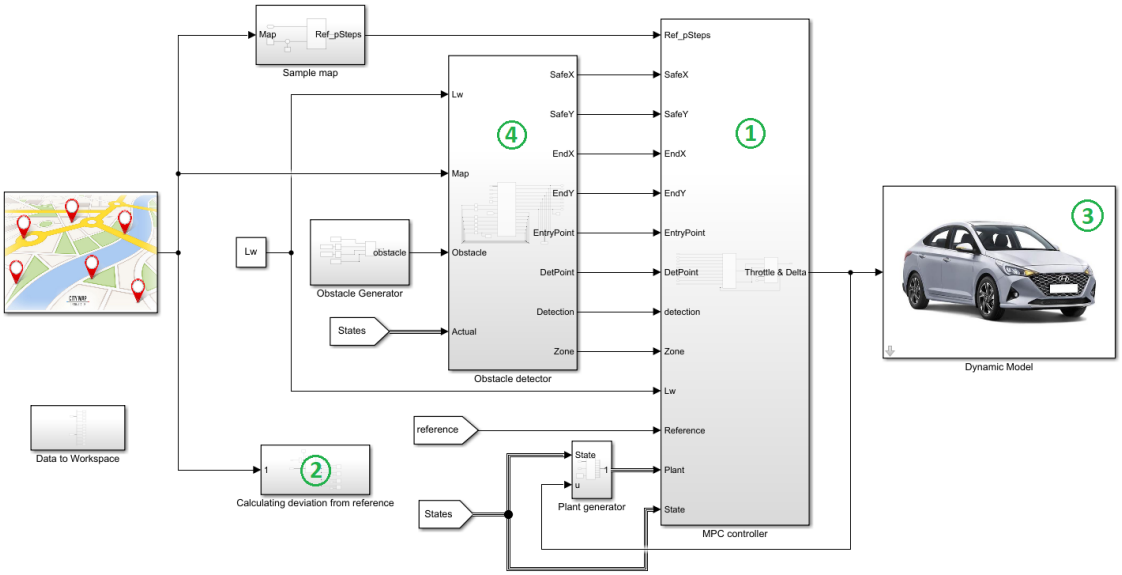


Figure 11: Simulink main

The blocks composing the subsystem *MPC Controller* - subsystem 1 - are showed in Fig.12: a block to call the function *LaneKeepConstraint* and the block *Adaptive MPC Controller*. The latter is itself a subsystem containing the matlab block *Adaptive MPC*; beside the reference input **ref**, there are others eight entry ports used in the adaptive controller block:

- **model**: updated plant model and nominal operating point. At the beginning of each control interval, this signal modifies the controller object.
- **mo**: Measured outputs. The MPC block uses the measured plant outputs to improve its state estimates.
- **E**: Manipulated variable constraint matrix. The F input port along with the E, G ports specify run-time mixed input/output constraints.
- **F**: Controlled output constraint matrix.
- **G**: Custom constraint vector.
- **y.wt**: Output variable tuning weights. These tuning weights penalize deviations from output references.
- **u.wt**: Manipulated variable tuning weights. These tuning weights penalize deviations from MV targets.
- **du.wt**: Manipulated variable rate tuning weights. These tuning weights penalize large changes in control moves.

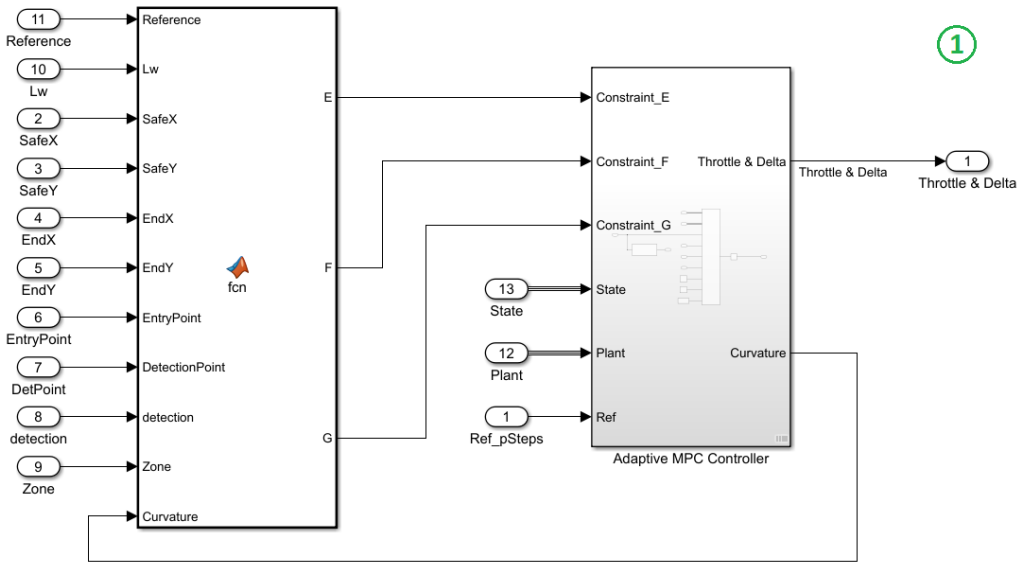


Figure 12: Simulink subsystem: Adaptive MPC controller

The content of the subsystem numbered 2 (see Fig.11) instead, it's displayed in Figure 13: the main purpose of these blocks is to calculate a deviation from the reference in order to test the controller and will be analyzed in details in the Section 8; in order to pass the proper number of reference way-points, a sampling subsystem is used as a input to the MATLAB function block.

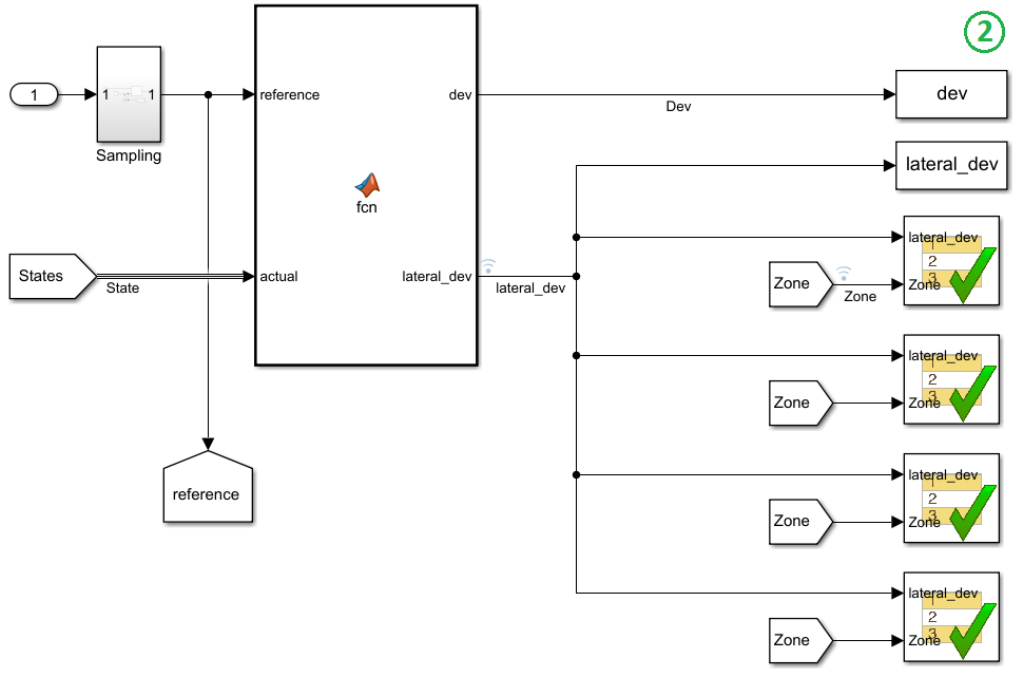


Figure 13: Simulink subsystem to calculate deviations

The subsystem *Dynamic Model* displayed in Fig.11 has as a mask the picture of an Hyundai Azera, the vehicle whose parameters are used in the simulation. Also for these blocks – Fig.14, the main purpose is calling an already designed MATLAB function that

in this case is the *VehicleModelCT_DYN* one, previously described in Section 4.

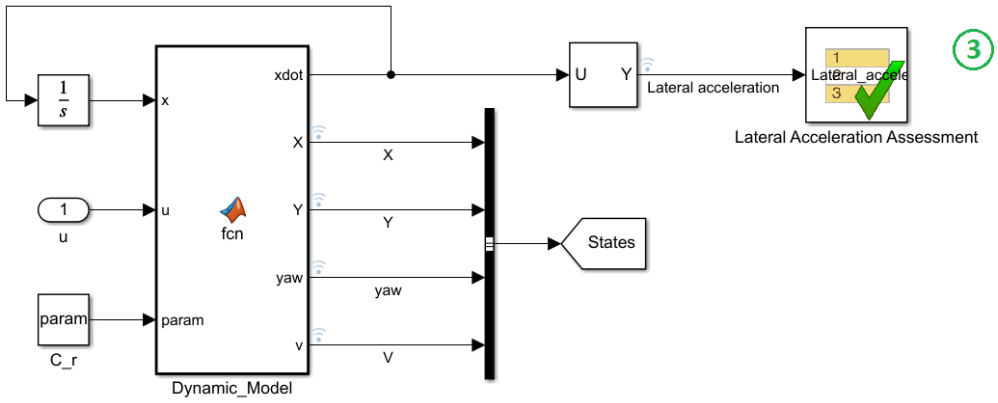


Figure 14: Simulink subsystem: Dynamic Model

Between the blocks of major interest, the last one described is the *Obstacle detector*, whose content is depicted in Figure 15. Again, the main purpose of this subsystem is using the simulink block to call a function. In this case, the *detectionFun* is used to be able to detect multiple obstacles on the road and calculate their relative safe zones.

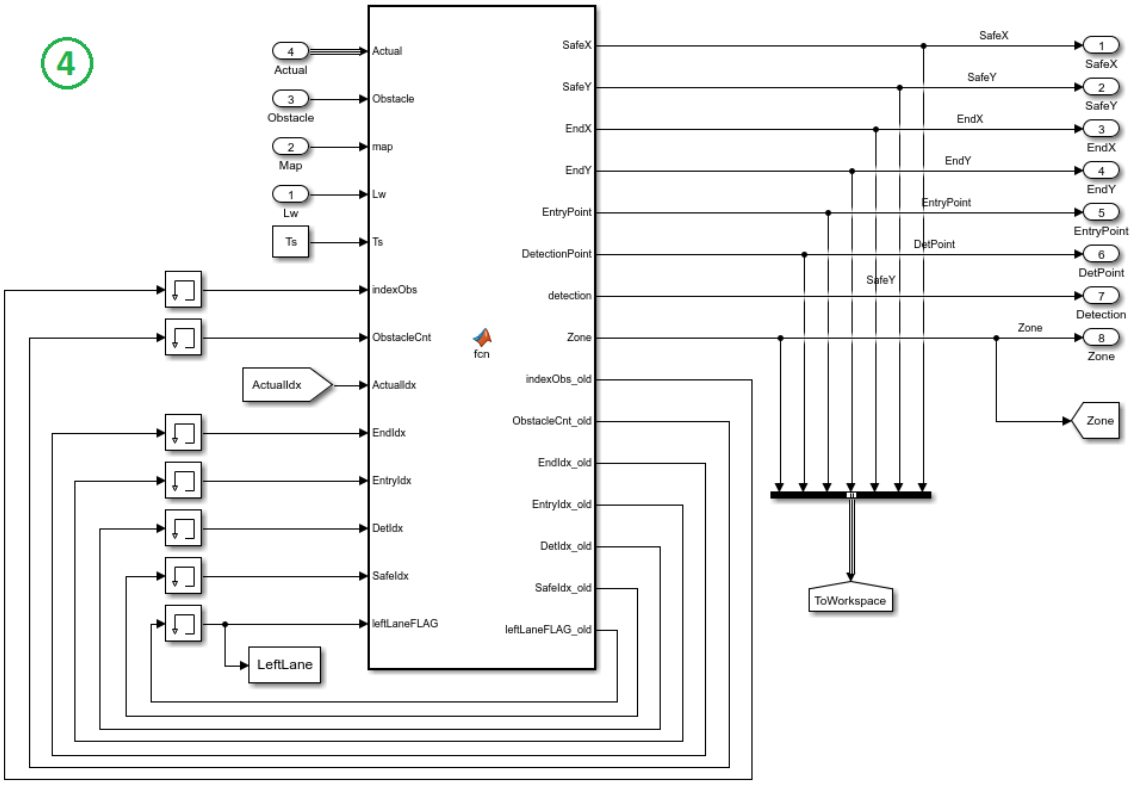


Figure 15: Simulink obstacles

7.3 Partitioning of the MPC functionalities

The developed MPC controller is supposed to carry out three main tasks:

- **Path following**: the car must be able to follow a given reference path without exceeding a predetermined threshold distance;
- **Static obstacle avoidance**: the car must sense and dodge any kind of stationary obstacle present on the road i.e. road construction sites, holes, trees, car accident, etc.;
- **Dynamic obstacle avoidance**: the car has the ability to perform overtaking maneuvers in the event that other moving vehicles, cyclists, pedestrians or animals are present in the same lane. The avoidance maneuver must be accomplished whether they are traveling in the same direction as the vehicle or in the opposite one.

Figure 16 shows how the ego-car (red) performs the overtaking maneuver: firstly it analyze the surrounding thanks to the on-board sensors and if it has enough space to achieve the lane change accordingly to the position and speed of the two moving obstacle (white cars) it plans the trajectory to follow to achieve the avoidance (blue path). Once the ego-car has planned a feasible path it starts to steer and accelerate arriving to the middle lane overtaking the car in the rightmost lane.

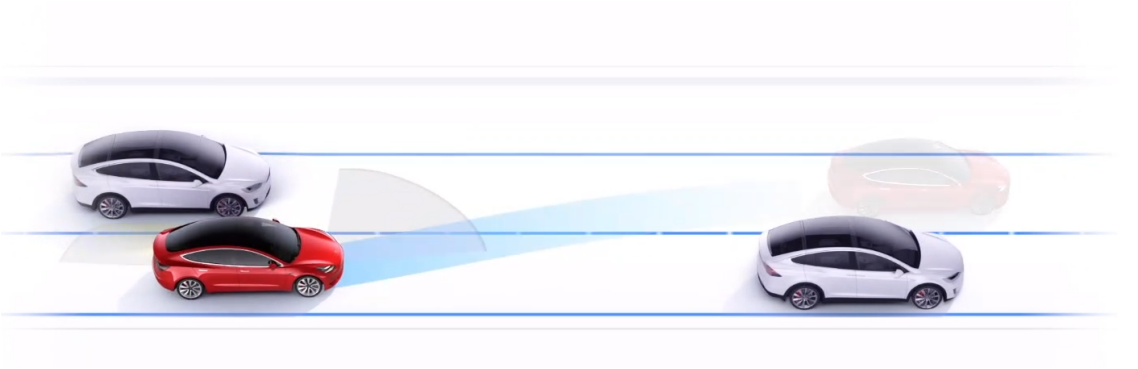


Figure 16: Dynamic obstacle avoidance maneuver on a three lanes road

In order to perform the path following task the MPC collects at each timestep the successive waypoints of the map for the duration of the prediction horizon and elaborates an optimal trajectory and control action to minimize the a cost function of the controller. Things get harder when obstacles are present on the road, either they are static or moving: we assumed to know "a priori" the position of the static obstacle or the trajectory that the moving obstacle is following, in such way thanks to the MATLAB function *"detectionFun"* we augmented the algorithm used for the path following with the capability to detect obstacles within the sensing range of the vehicle, thus allowing to promptly start and execute the avoidance maneuver always being outside the so called *"safe zone"* of the obstacle.
Figure 17 plots an example scenario for static obstacle avoidance, where:

- the **ego car** is represented by the green dot with the black boundary;
- the **horizontal lanes** are represented by the dashed blue lines;
- the **static obstacle** is represented by the red x with the black boundary;
- the **safe zone** is highlighted by the dashed red boundary.

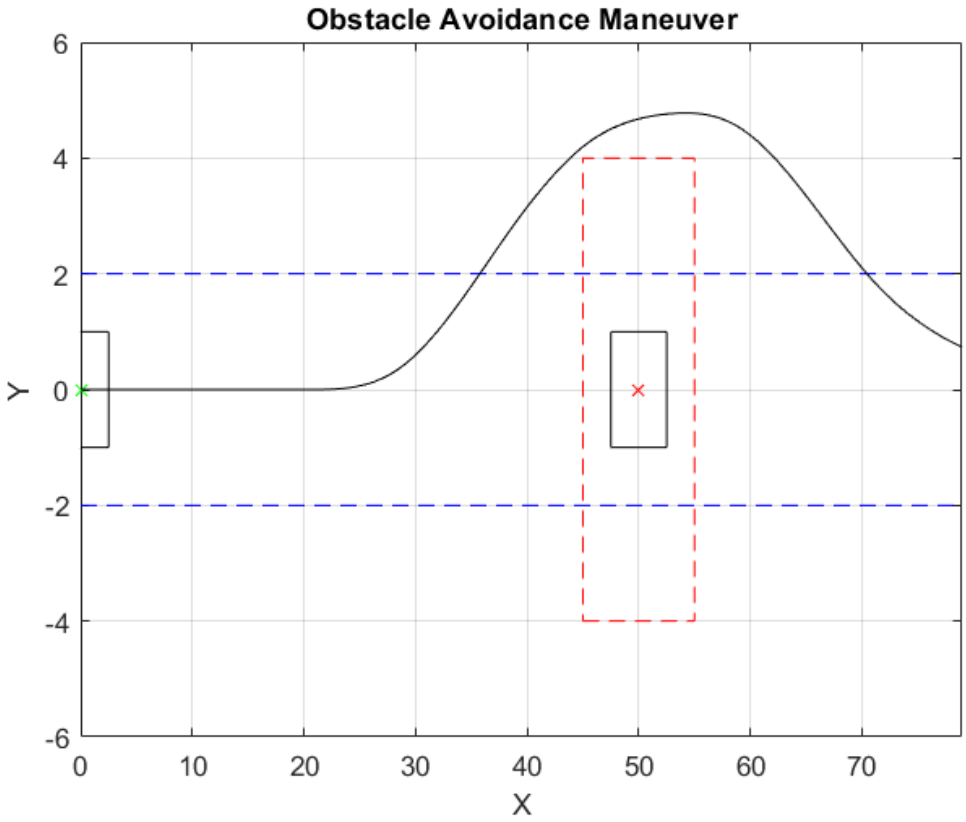


Figure 17: Example of static obstacle avoidance maneuver

8 Path Following

The first task our controller should accomplish is to follow a reference path. How this reference path is given to the controller, is described in the Section 5 - Signal Acquisition, where we assume that we have an high level device which is able to provide to the controller a sampled map composed by a sequence of waypoints reporting x, y, θ, and v for each timestep of the control. We pass to the controller the next p steps in the reference map, where p is the number of steps of the prediction horizon.
Before the final testing process, we have performed several simulations in order to obtain a fine controller tuning regarding its weights, relative to output variables and manipulated variables, and other influential quantities.
The best results for path following are achieved with the following configuration:

Sample Time	0.01 s
Prediction Horizon	15 steps
Control Horizon	5 steps
Output Variables Weights	[30 30 8 30]
Manipulated Variables Weights	[0 0]
Manipulated Variables Rate Weights	[0 0]

Table 6: Configuration parameters adopted for Path Following

For the sake of the Path Following task, only the constraints on the inputs are considered, neglecting any constraint on the states.
Moreover, we have set the initial value for the *covariance matrix* of the internal state estimator to $\begin{bmatrix} 1000 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1000 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1000 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1000 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1000 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1000 \end{bmatrix}$ to avoid oscillations in the starting phase of the control, due to the discrepancies between the plant model given to the controller and the dynamic plant model used for simulation. Covariance matrix is 6×6 because it is related to 4 states and 2 inputs.
This matrix is automatically updated by the MPC algorithm in runtime.

8.1 Path Following - Testing

Path following tests aim to evaluate the performance of a path follower in different scenarios with speed ranging from 10 to 100 km/h as stated in Section 2 - System Requirements. In order to satisfy requirements 1 and 5 regarding respectively the *maximum lateral error* and the *maximum lateral acceleration*, we setup a test-bench to assert these quantities. In detail, to verify the lateral error, we built a function to approximate the distance between the actual point in which the vehicle is and the reference trajectory. This function is based on the assumption that the vehicle is close enough to the reference point considered and the reference, in the close proximity of the point itself, can be approximated as a straight line. These considerations are summarized by the following equation:

$$Lateral\_dev = \frac{\|y_{vehicle} - (mx_{vehicle} + q)\|}{\sqrt{1 + m^2}} \tag{20}$$

where:

$$m = \tan(\theta_{ref})$$

$$q = y_{ref} - mx_{ref}$$

The tests are considered passed if the following conditions are verified:

- lateral deviation always lower than 1.0 m;
- lateral deviation does not exceed 0.75 m for more than 1 s.

For what concerns the lateral acceleration we get it directly from the state of the dynamic model as reported in Section 4, equation 9: $\ddot{y} = -\dot{\psi}\dot{x} + \frac{2}{m}(F_f cos(\delta) + F_r)$.
The test is considered passed if the lateral acceleration does not exceed 2.0 m/s^2 for more than 0.5 s.

Each path following test has been performed with all the vehicles stored in our database, as reported in section 4 Table 2.

The scenarios we have selected are the following:

1. **Straight line**: this is the simplest possible scenario, consisting of a straight line starting from $(0, 0)$ up to $(1000, 0)$ in the X-Y reference frame, with 0° orientation. We repeated this scenario twice with speeds respectively of 10 km/h and 100 km/h;

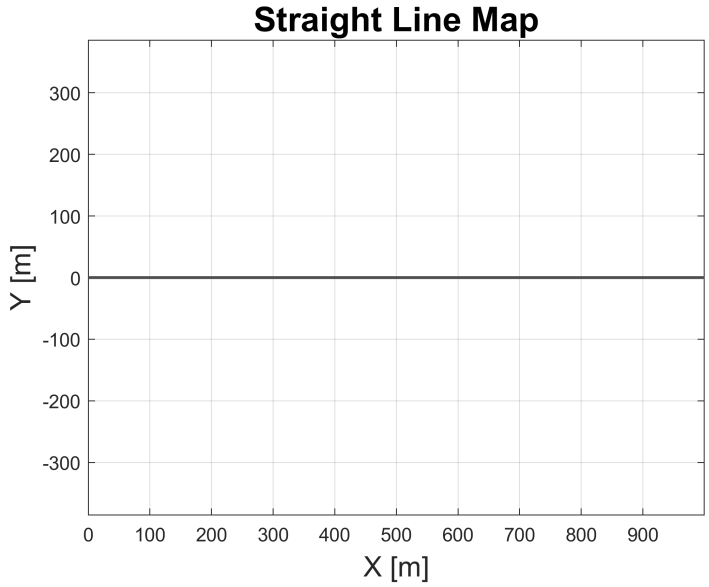


Figure 18: Straight line Map

2. **Tilted straight line - 45°**: this is a simple scenario consisting of a straight line starting from $(0, 0)$ up to $(1000, 1000)$, with an orientation of 45°. We selected a speed of 10 km/h for this scenario;

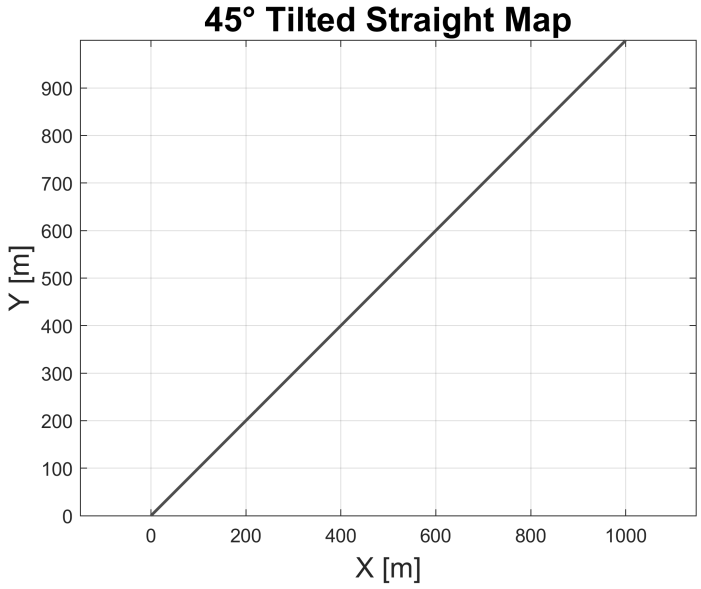


Figure 19: Tilted straight 45° map

3. **Tilted straight line - 135°**: this is a simple scenario consisting of a straight line starting from $(0, 0)$ up to $(-1000, 1000)$, with an orientation of 135°. We selected a speed of 100 km/h for this scenario;

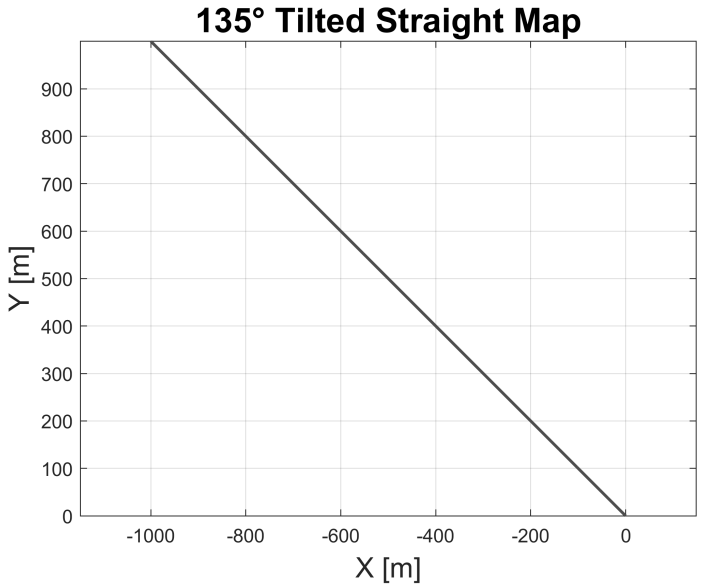


Figure 20: Tilted straight 135° map

4. **100m radius Curve**: this scenario consist of a road with constant curvature with 100 m radius. We repeated this scenario twice with speeds respectively of 10 km/h and 100 km/h;

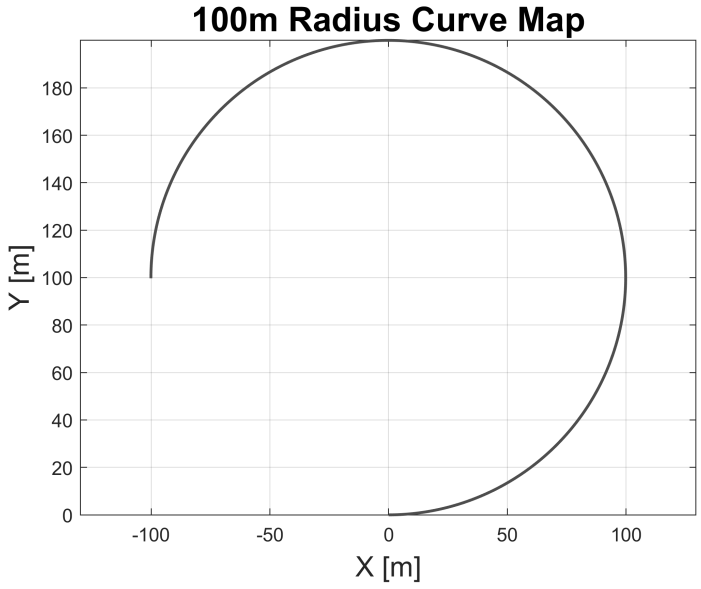


Figure 21: 100m radius Curve Map

5. **1000m radius Curve**: this scenario consist of a road with constant curvature with 1000 m radius. We repeated this scenario twice with speeds respectively of 20 km/h and 100 km/h;

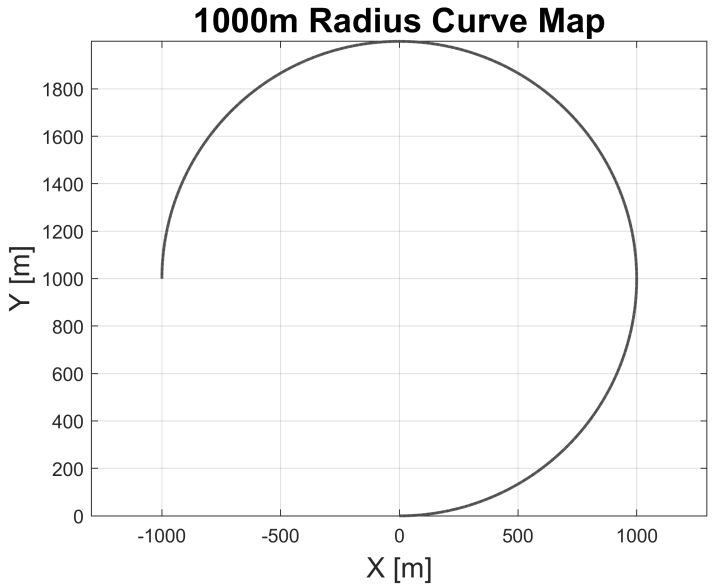


Figure 22: 1000m radius Curve Map

6. **Puglia**: this scenario is taken from a city road and it is straight for the most part with some smooth corners. In this test we try to follow the path with a speed of 40 km/h;

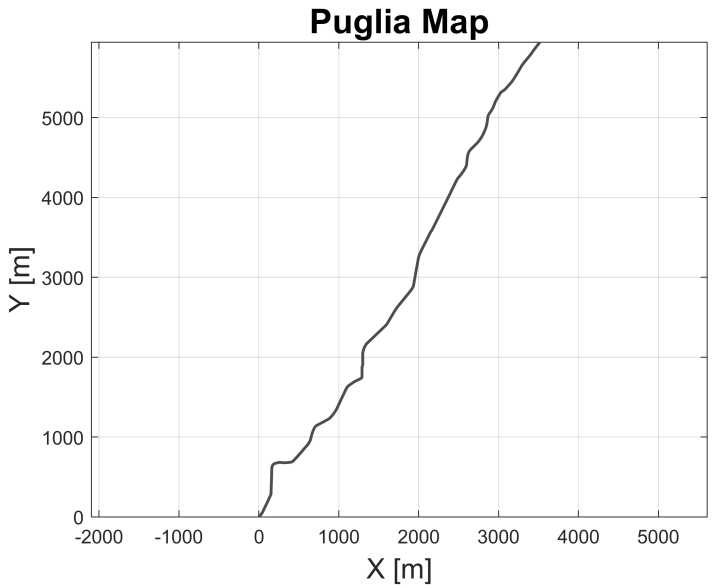


Figure 23: Puglia Map

7. **Switzerland**: this is the slowest scenario considered, with lots of corners one after another. We try to follow this scenario with 15 km/h speed;

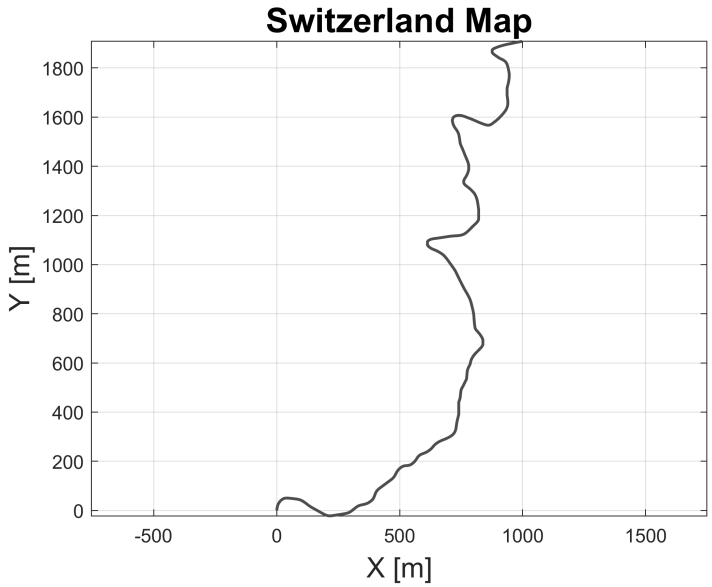


Figure 24: Switzerland Map

8. **Campania**: this scenario is made up by a sequence of smooth corners. We try to follow this path with 30 km/h speed;

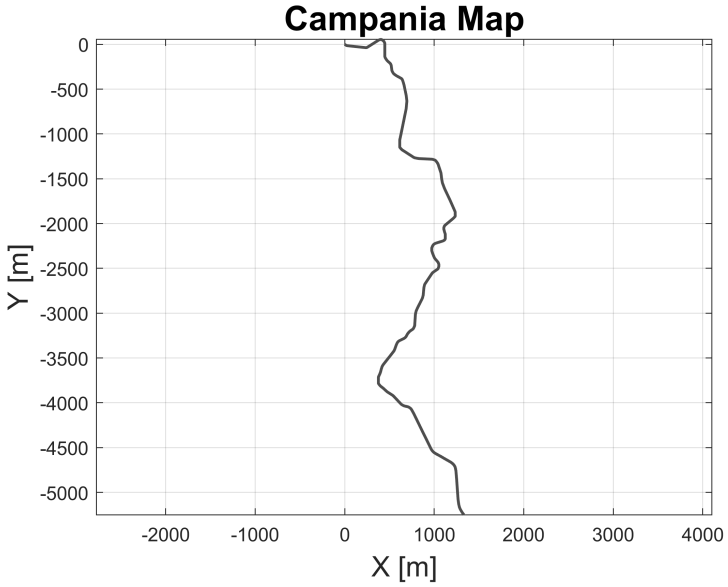


Figure 25: Campania Map

9. **Adriatic Highway - A14**: This scenario is taken from the A14 highway which is a straight road for the most of it, with some highspeed corners. We simulate this scenario at 100 km/h;

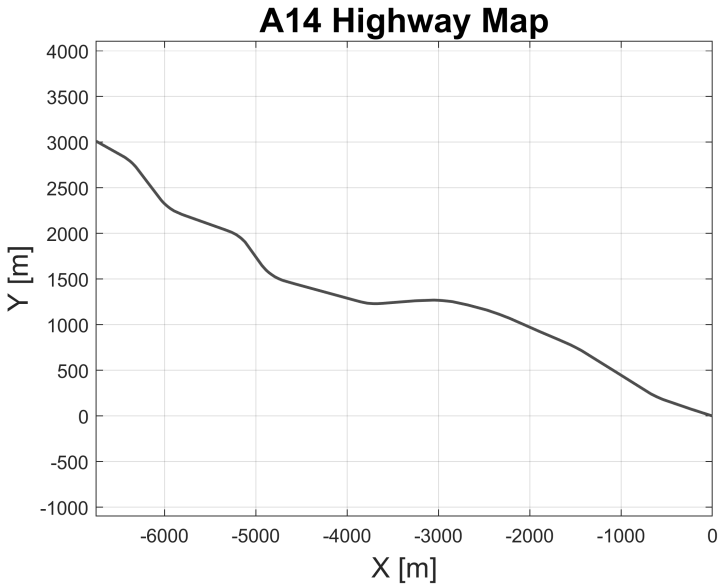


Figure 26: Adriatic Highway - A14 Map

10. **Indianapolis Speedway**: This scenario is taken from the Indianapolis Speedway. We simulate it at 100 km/h.

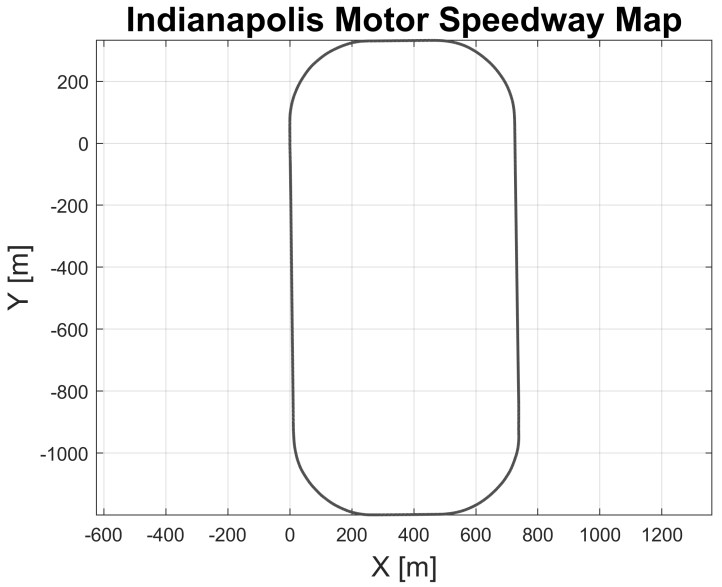
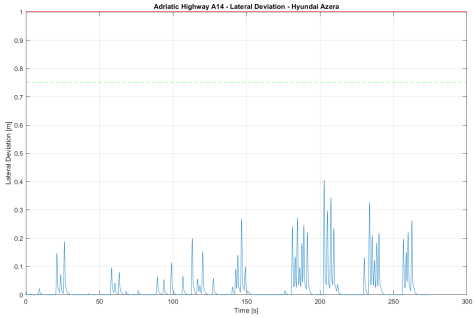
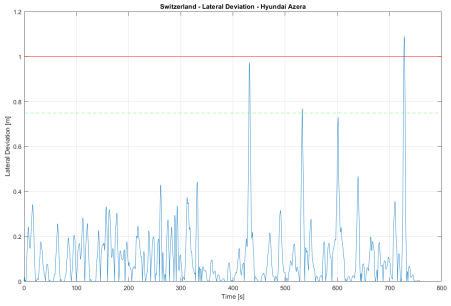


Figure 27: Indianapolis Speedway Map

Details about the path-following tests performed can be found in the documents included in the repository[3].
Performing the tests we have found that in simple scenarios, such as straight lines and Highway, the performances of our controller are very good, with little deviation from the reference path, while in winding scenarios, such as the *Switzerland* one, we found out that the vehicle is not able to keep its position inside the road boundaries in all the corners. Figures 28a and 28b show the lateral deviation found in the test performed on the Adriatic Highway A14 and the one found in Switzerland respectively. As shown below, the lateral deviation in the former never exceeds the limits imposed (described at the beginning of this section), while in the latter these limits are exceeded a few times resulting in the failure of the test.



(a) Lateral Deviation Plot (Adriatic Highway A14)



(b) Lateral Deviation Plot (Switzerland)

Figure 28: Lateral Deviation Comparison

8.1.1 Failed Tests Analysis

In the "Path_Following-Test_Report - Failed Tests" we have included the results of the failed tests. In this section we will describe in more detail the issues we have faced during these tests. What we want to point out is that in the failed tests, the lateral deviation condition is violated only in some specific points, where the curvature of the reference road is, maybe, too high to be performed at constant speed by the vehicle. We encountered this problem in the *Switzerland* scenario with all vehicles and in the *Campania* scenario only with the vehicle Ford E150, which is a heavy van, and it suffers, as expected, of a higher lateral slip compared with other vehicles. In the following figures, are reported some details about the test performed with the Hyundai Azera in the *Switzerland* Scenario. The figure 29 shows how the vehicle behaves in paths with smooth corners, satisfying the lateral deviation requirement, while the figure 30 refers to one of the most critical sections of the map, where can be appreciated how the vehicle goes outside the lane in the apex of the hairpin turn.

[3]The "Path_Following-Test_Report", the "Path_Following_Test-Annex_A" and the "Path_Following-Test_Report - Failed Tests" files generated by Simulink Test are included in the /Documentation/Test Reports/ file path.

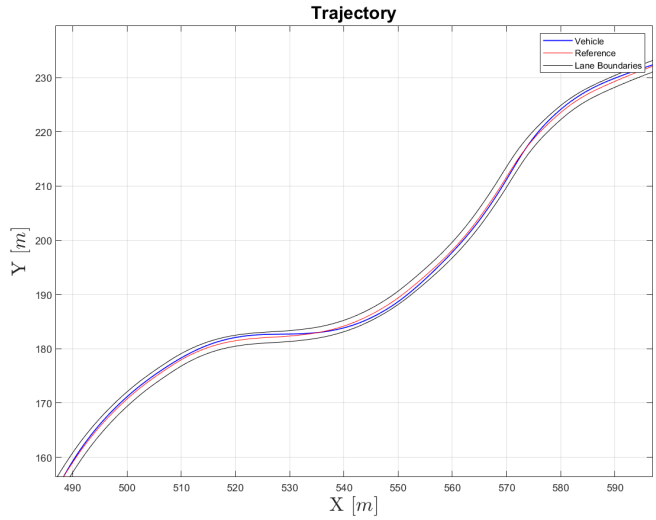


Figure 29: Trajectory of the vehicle in some corners, where all requirements are satisfied

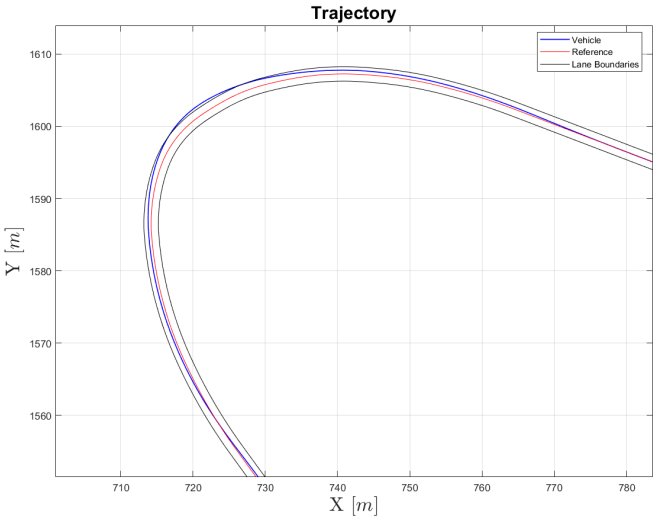


Figure 30: Particular where trajectory of the vehicle is out of the lane boundaries, causing the "fail" of the test

Although these failed tests, we have decided to keep the same controller setup for the obstacle avoidance tests as well, excluding the *Switzerland* Scenario from further simulations.

9 Static Obstacle Avoidance

For the static obstacle avoidance task, the controller must drive the vehicle in order to avoid collision with any obstacle detected by a set of sensors.
Before implementing the algorithm for obstacle avoidance, we have made a set of assumptions to simplify this task:

- we have a set of sensor able to determine the position of the obstacle and to place it on the reference map, but the detection is performed only when the distance from the obstacle is less than 200 m;
- it is always possible to avoid the obstacle by moving in the left lane;
- the obstacle is never placed at less than 200 m from the starting point, therefore it is always possible to avoid it without violating the zones;
- dimensions of the obstacle are not relevant.

To be sure that the controller tries to avoid obstacles, we decided to implement the avoidance algorithm by giving to the controller a set of constraints on the output, in order to define a *"forbidden zone"*, as described in subsection 6.2.1.
To be precise, we have defined 5 zones nearby each obstacle:

1. **Zone 1**: the obstacle is detected, but the vehicle is still too far from it, so it stays in its lane;
2. **Zone 2**: the vehicle is close enough to the obstacle to start the overtaking maneuver, so it can start to change lane;
3. **Zone 3**: the vehicle is in the left lane, in the so called *"Safe Zone"* for the obstacle;
4. **Zone 4**: the obstacle is passed and the vehicle can come back to the right lane;
5. **Zone 5**: the vehicle is in its lane and the obstacle has been already passed, so we are in the same condition of no obstacle detected.

These zones are defined, starting from the obstacle position, in the following way:

- **Zone 1** starts when the obstacle is detected, so when the distance between the vehicle and the obstacle is less than 200 m, as specified in the assumptions made at the start of this section;
- **Zone 2** starts 40 meters before the Zone 3. We decided to define this zone in this way to be symmetric with respect to the Zone 4, where the other change lane maneuver is performed;
- **Zone 3** is defined to be compliant with the Safety Distance described in 6.2.1, according to equation 17: $SafetyDistance[m] = \left(\frac{v[km/h]}{10}\right)^2$, so it is proportional to the square of the speed. Zone 3 ends 10 meters after the obstacle, to be compliant with the requirement n° 4 in section 2;
- **Zone 4** is defined to be compliant with requirement n° 4 in section 2, so it starts 10 meters after the obstacle and ends 50 meters after it, resulting in a space of 40 meters for the re-entering maneuver, thus it is symmetric with respect to the maneuver performed in Zone 2 as previously stated;

- **Zone 5** starts right after Zone 4 and it is kept until the distance from the obstacle becomes greater then 200 m.

The distance of 200 m from the obstacle is evaluated as the Euclidean distance between the vehicle and the obstacle, while the other distances involved in the zone definition are evaluated as projection on the reference path, and assuming constant speed, they result in fixed points on the reference map.

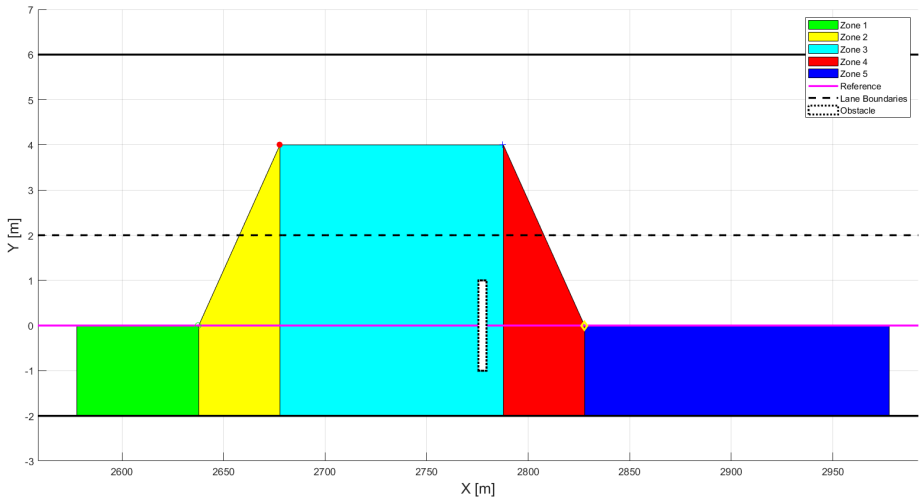


Figure 31: Zones defined for obstacle avoidance

In Figure 31 the aforementioned zones are depicted on a straight line with a reference speed of 100 km/h. In zones 1 and 5, as previously said, the obstacle is too far from the vehicle to be a problem, so no procedure is applied to the controller. In zone 2, the yellow one in the figure, the vehicle should pass in the left lane. To do so, we defined two points, represented in the figure with a green $\circ$, which is called the *Detection Point*, and the red circle that is the *Safe Point*. The latter is defined as the projection on the left lane of the point in the reference trajectory at a distance equal to the safety one. To let the controller move on the path defined by these two points, we have exploited the *Custom Constraint* of the MPC block in Simulink.
As briefly explained in Section 7, linear combination of inputs and states can be set as constraints for the controller. To accomplish our task, we have defined 2 constraints on the x and y states, which result in two lines representing an upper and a lower bound for the vehicle position during the overtaking maneuver. Looking at the example in the Figure 31, the lower bound becomes the border of the polygon defined by zones 2, 3 and 4, while the upper bound is the left limit of the road, that is to say the left border of the left lane.
For what concerns the lower bound for constraints in zone 2, it is completely defined by the line passing from the two aforementioned points.
In zone 3, the vehicle should stay in the left lane. To define constraints in this zone, we have set as lower bound the projection of the reference trajectory on the left lane, instead of the straight line connecting points from the start (red circle) to the end (blue cross), since the former method allows to track both straight and curved paths. One limitation of the *linear* constraints is that they can only define *linear* relation between states, thus when the path is curved, we have to approximate it to a straight line, but this can cause errors

because the controller tries to satisfy the constraint for all the timesteps in the prediction horizon, so resulting in a reduced or increased distance with respect to the points in the reference map. To limit this effect, we have inserted a correction factor to the constraint generated.

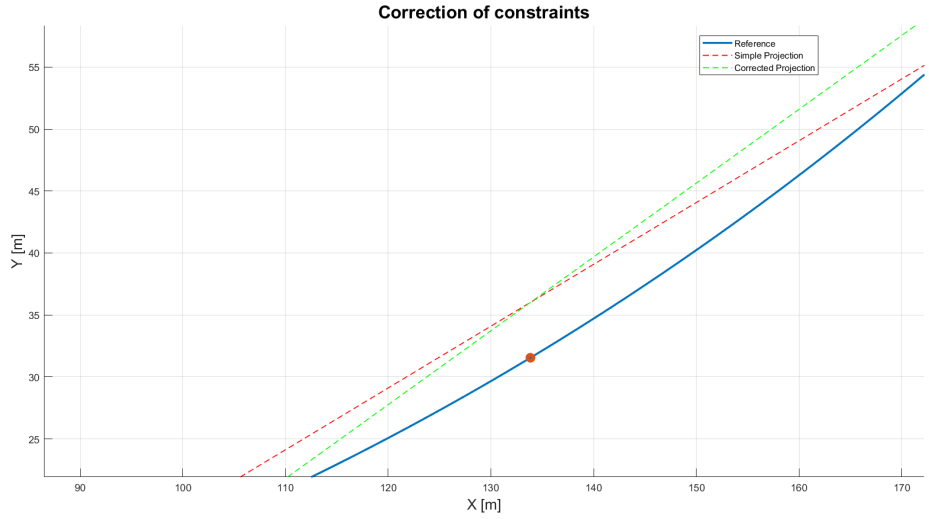


Figure 32: Constraints generated in a curved path with 300 m radius at 50 km/h, with and without the correction factor

In figure 32 is represented a case where the reference road is a left curve with 300 meters radius. The dashed red line is the projection of the road approximated with the slope of the reference in the point where the controller is called, while the green dashed line is the projection evaluated with the correction factor. As can be seen, the original projection "collapse" on the reference path before the corrected one, resulting in a reduction of the distance from the reference during the predicted horizon and a consequent reduction of distance between the trajectory and starting lane, occupied by the obstacle.
This corrected constraint is evaluated according to the following procedure:

1. An approximation of the *curvature* of the road is evaluated as the mean value of the reference angle in the prediction horizon: $curvature = \frac{\sum_{i=0}^{p} \theta_i - \theta_0}{p}$;

2. The correction factor is: $416 \cdot \frac{|curvature|}{V}$, where the coefficient 416 has been found empirically by simulations;

3. The *CorrectedSlope* is given by the road slope ($tan(\theta_r)$) plus the correction factor;

4. The intercept for the constraint line is given by:
$CorrectedIntercept = y_r + Lw/cos(\theta_r) - CorrectedSlope \times x_r$
where x_r, y_r, and θ_r are the reference trajectory values at the first timestep of the prediction horizon, and Lw is the lane width;

5. The constraint is applied as $y > CorrectedSlope \times x + CorrectedIntercept$.

Obviously, when the reference path is straight, the correction function becomes 0 and the constraint is the simple projection of the road.
In zone 4, the situation is analogous to the one in zone 2, with the only difference that

here we are *"relaxing"* the constraint, allowing the vehicle to come back to its original position on the right lane. The bound here is defined by the blue cross point, called *End Point* and the yellow diamond point, called *Entry Point*.
To define these bounds in a suitable form for the controller, we have to transpose information in a matrix form, describing two line equations:

$$y > m_{lower}x + q_{lower} \tag{21}$$

$$y < m_{upper}x + q_{upper} \tag{22}$$

where m is the slope of the constraint and q is the intercept of the constraint. Slope of the constraint can be evaluated by calculating the tangent of the road orientation when we are considering the upper bound or the zone 3, while it is evaluated as the slope between the points defined above in zones 2 and 4.
The intercept is given by the projection of the reference trajectory, on the left lane center line for zone 3 and on the left road limit for the upper bound, or it is evaluated by inverting the line equation in the changing lane maneuvers.
To let our algorithm work on all the quadrants, we make a check of the reference trajectory orientation and we adapt the constraint generation to the case we are in. This check projects the constraints on the x axis rather than y if the road inclination is closer to the vertical line in the X-Y plane, moreover it defines which are the relations between the bounds and the reference trajectory, intended as *"lower than"* or *"greater than"*, according to the direction of the vehicle and its left.

9.1 Multiple obstacles avoidance

The procedure described until now shows how the controller sets constraints when an obstacle is detected, but when multiple obstacles are present, things can become harder.
If two obstacles are very far one from the other, at least 400 m, they can be treated as single obstacles in sequence, but if they are close, a new definition of the zones is needed.
In detail, we decided to control the position of the next obstacle when we reach the *End Point* of the previous. In this case, we can decide whether to come back in the right lane or to stay in the left lane and continue the overtaking maneuver. The procedure follows the rules below:

- if the next obstacle is more than 400 m far from the previous one, it is treated as an independent single obstacle;
- if the next obstacle is in the range 400 m and 210 m from the previous obstacle, it is not detected when the vehicle is in the *End Point* of the previous obstacle. Here vehicle simply comes back to the right lane and when the next obstacle is detected, it is treated as a single one;
- if the next obstacle is closer than 210 m from the previous one, it is detected when the vehicle is in the *End Point* of the latter, so here a decision is taken:
    1. the next *Detection Point* is after the previous *Entry Point* $\rightarrow$ the vehicle reenters in the right lane and starts the new overtaking maneuver when it reaches the next *Detection Point*;
    2. the next *Detection Point* is before the previous *Entry Point* $\rightarrow$ the vehicle stays in the left lane and sets as target the next obstacle, so then it repeats this procedure when the next *End Point* is reached.

To implement this control strategy in the simulation environment, we have used a counter on the obstacle, which takes into account if an obstacle has already been passed and in that case the detection algorithm neglects it, considering directly the next obstacle.
In reality, of course, a counter is not a feasible solution since it is impossible to know how many obstacles will be detected during a trip, but advanced navigation and sensor systems can detect and classify obstacles, and then can understand whether they have been passed or not, so our assumption is justified.

10 Static Obstacle Avoidance Tests

In order to evaluate the behaviour of our controller when it comes to avoiding static obstacles, we have set up a set of tests using *Simulink Test*. These tests have been performed on different sample scenarios at different reference speeds using the same MPC configuration as in the Path Following tests (Table 6). In particular we have simulated 22 scenarios, similarly to the first 5 scenarios in the Path Following tests (from figure 18 to 22), considering 16 straight lines with different slopes (exploring all four quadrants) and 6 curves with different radii and directions. The scenarios we have selected are the following:

- **straight lines** with:
  - ◇ 0° slope
  - ◇ 20° slope
  - ◇ 45° slope
  - ◇ 70° slope
  - ◇ 90° slope
  - ◇ 110° slope
  - ◇ 135° slope
  - ◇ 160° slope
  - ◇ 180° slope
  - ◇ -20° slope
  - ◇ -45° slope
  - ◇ -70° slope
  - ◇ -90° slope
  - ◇ -110° slope
  - ◇ -135° slope
  - ◇ -160° slope
- **curves** with:
  - ◇ 1000 m radius clockwise
  - ◇ 500 m radius clockwise
  - ◇ 300 m radius clockwise
  - ◇ 300 m radius counterclockwise
  - ◇ 500 m radius counterclockwise
  - ◇ 1000 m radius counterclockwise

In the following subsections we describe in details the test performed.

10.1 Single Static Obstacle

We have started off from the easiest case of obstacle avoidance possible, one static obstacle on the whole reference map. This is the easiest case possible to implement because once detected an obstacle, the controller does not need to search for another; moreover, since the obstacle is static, the controller does not need to know its speed in order to calculate its trajectory either. In particular we have placed the obstacle always exactly in the middle of the simulated map so that the detection of the obstacle and consequent overtaking maneuver could be executed correctly by the controller in any circumstance. Similarly to the tests performed to assert the path following task, we have performed the obstacle avoidance tests taking into account equations 9 and 20 on lateral acceleration and lateral deviation respectively, in order to monitor whether or not requirements 1 and 5 (Section 2) on maximum lateral error and maximum lateral acceleration were satisfied. Regarding the other requirements described in the same section, we suppose that the controller and the set of sensors installed take care of them. Regarding requirements 1 and 5, tests are considered passed if the following conditions are verified:

- lateral deviation always greater than or equal to 2 m and lower than or equal to 6 m when performing the overtaking maneuver;

- lateral deviation must not exceed 5 m and be lower than 3 m for more than 1 s when performing the overtaking maneuver;
- lateral deviation lower than 6 m at any point of time;
- lateral acceleration does not exceed 2.0 m/s^2 for more than 0.5 s at any point of time.

We have chosen those limits because we assumed that a lane is 4 m wide and a lateral deviation of 0 m means that we are exactly on the reference trajectory, always considered to be on the right lane. The 2 m bound represents, instead, the boundary between the right and the left lane while the upper bound of 6 m is the leftest boundary of the left lane. In order to verify these conditions, we have set up a test bench to monitor them. Details about the single static obstacle avoidance tests performed can be found in the documents included in the repository[4].
Performing the tests we have noticed that our controller performs very well at medium/low speeds (such as 40-50 km/h) while it has some stability issues at lower speeds. The figures below show a comparison between the overtaking maneuver performed respectively at 50 km/h (Figure 33) and at 10 km/h (Figure 34), highlighting how the performance of the MPC degrades with lower speeds.

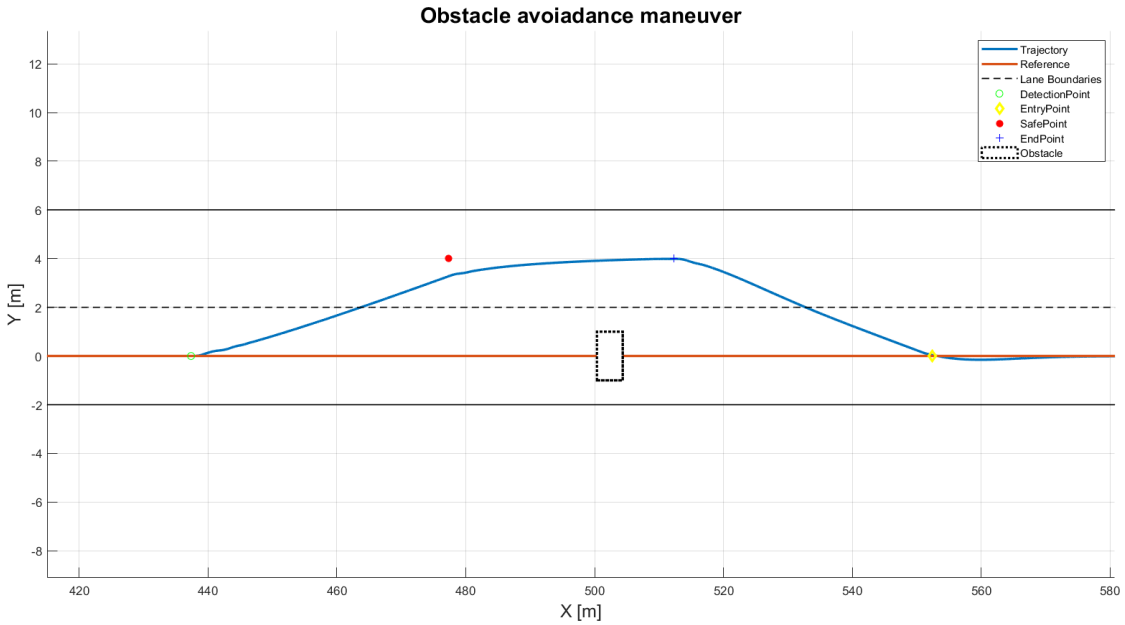


Figure 33: Overtaking maneuver at 50 km/h

[4]The "Single_Static_Obstacle_Avoidance-Test_Report" and "Single_Static_Obstacle_Avoidance-Test_Specification_Report" files generated by Simulink Test are included in the /Documentation/Test Reports/ file path.

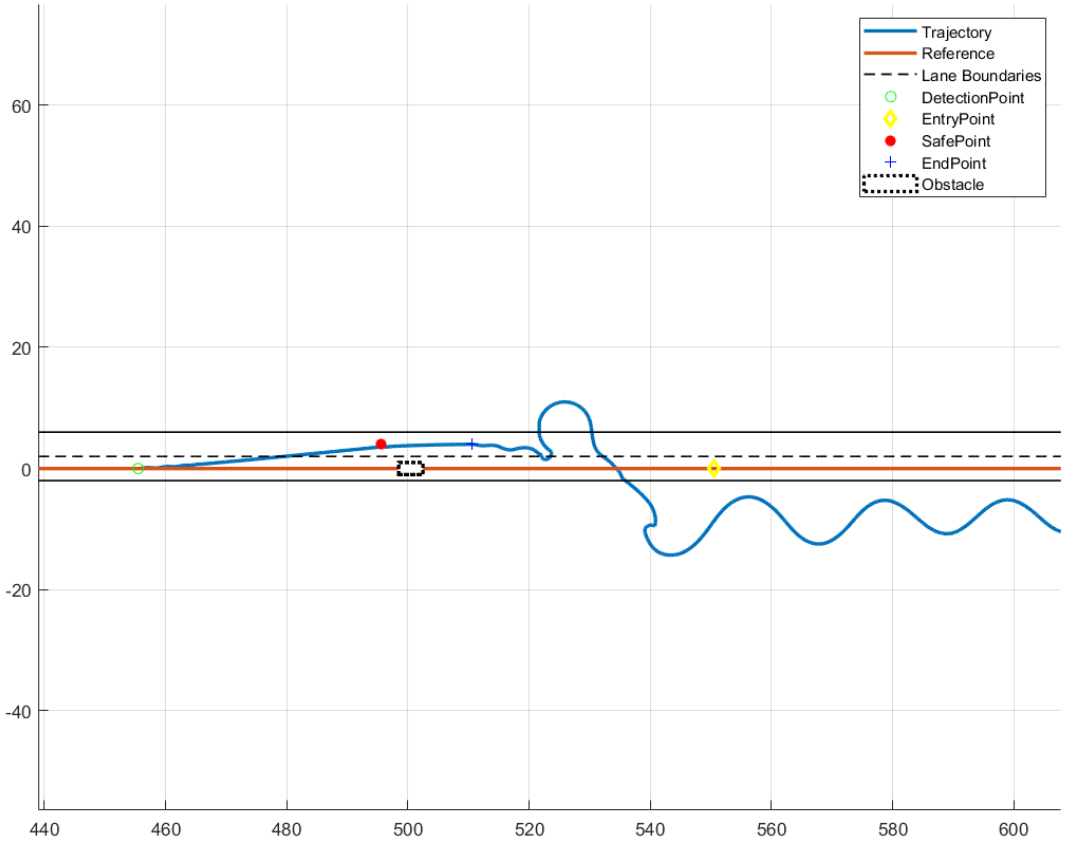


Figure 34: Overtaking maneuver at 10 km/h

The tests have also highlighted how the performance of our MPC is still fairly good at higher speeds (such as 100 km/h) with some issues only on the curved scenario with curvature radius of 300 m when the vehicle travels counterclockwise.
More details about the failed tests and the subsequent considerations are reported in the next subsection.

10.2 Failed tests analysis and important considerations

The "Single_Static_Obstacle_Avoidance-Test_Report" shows all the issues that have arisen during the testing phase. In this section we describe the most important ones and we make some considerations. As shown in Figure 35, our controller is not fully able to be compliant with the first condition described in the previous subsection when we are travelling at 100 km/h on a curve with 300 m radius with counterclockwise direction. In this case, the ego vehicle is not able to perform in time the overtaking maneuver because of the high speed.

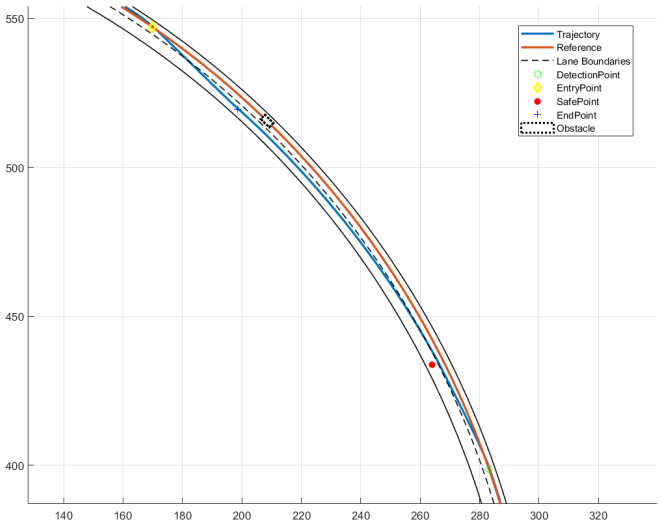


Figure 35: Failure during the overtaking maneuver at 100 km/h on a curve with 300 m wide radius with counterclockwise direction

Despite this anomaly, we recognize that this is not a major problem since breaking this particular assessment does not fully compromise the ability of the controller to avoid the obstacle since, at this reference speed, the safety distance is 100 m and so, going slightly over the so-called "*SafePoint*", does not represent a dangerous situation. Moreover, it only occurs on curves with quite small radius when the vehicle is steering towards the left side.

The main issue we have found is that our controller is not able to perform a proper overtaking maneuver at low speed. Critical speeds range from 10 to about 30 km/h. An example of this behaviour on a curved scenario is shown in Figure 36.

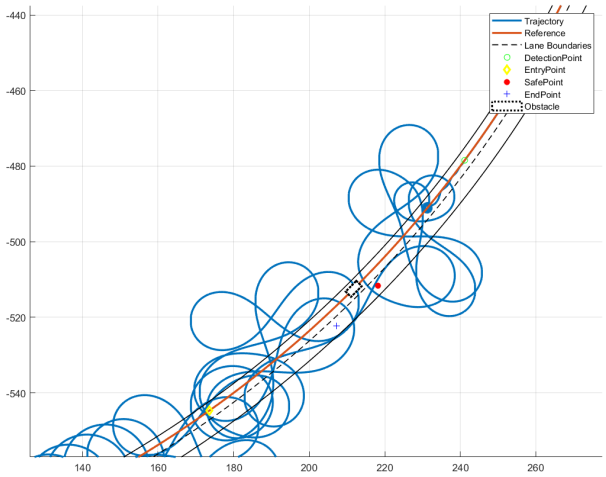


Figure 36: Example of controller stability issues at 10 km/h

As you might guess, this issue implies that we are no longer able to comply with the sixth requirement imposed in the starting phase of our project described in Section 2. Having based the development of our whole project on the V-model, as already stated in the Introduction of this report, going back to fix this issue in order to make the MPC work properly at low speeds would imply modifying the weights in the cost function and repeating all the testing phases. We have decided to do so only for a certain speed range which means that we are actually implementing a new MPC with different weights in order to manage this speed range while keeping the current configuration for the set of speeds already tested on which our MPC has been already validated. This decision implies a higher modularity of our project, that from now on will be based on two sub-problems, consisting of two different speed ranges which will be managed by two different MPCs:

- **MPC-H** (the current one) which will manage speeds ranging from 40 km/h to 100 km/h;
- **MPC-L** (a new one) which will be developed to be in charge for speeds in the range from 10 km/h to 40 km/h.

We believe that this is the optimal solution in order to be able to continue the tests on the current MPC, considering those already performed as valid in the proper speed range and to have a faster validation phase for the second MPC that will be developed afterwards.

10.3 Multiple Static Obstacles (MPC-H)

After performing the tests on single static obstacle avoidance, we have decided to simulate the presence of various obstacles on the maps in order to evaluate the behaviour of our MPC in the range from 40 km/h to 100 km/h (MPC-H). We have considered a total of 5 static obstacles placed on the maps as follows:

- Obstacle 1: placed at 20/100 of the total length of the map;
- Obstacle 2: placed at 40/100 of the total length of the map;
- Obstacle 3: placed at 65/100 of the total length of the map;
- Obstacle 4: placed at 68/100 of the total length of the map;
- Obstacle 5: placed at 80/100 of the total length of the map.

Figure 37 graphically represents the lateral deviation of the ego vehicle and the position of the obstacles (in orange) on a straight sample map according to what has been described above.

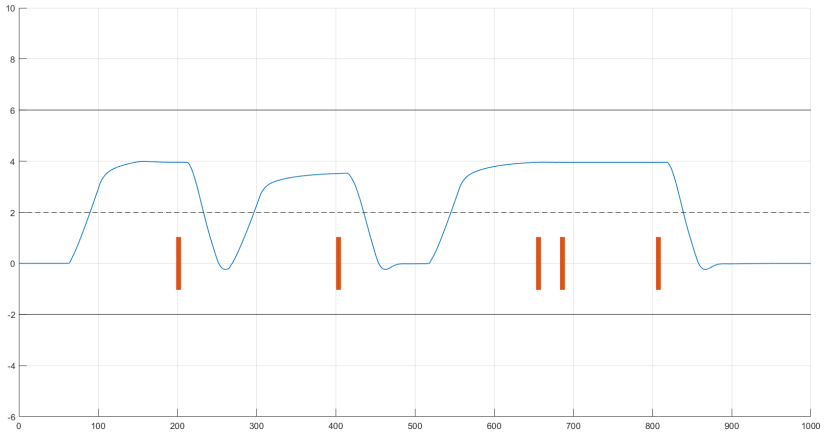


Figure 37: Example of static multiple obstacles avoidance

In particular we have positioned obstacles 1 and 2 well spaced from the others, while obstacles 3, 4, and in some instances obstacle 5, are quite close to each other in order to test the capability of our MPC to lengthen the overtaking maneuver when two or more close consecutive obstacles are detected. Indeed obstacle 4 has been placed in a position that, depending on the speed and the total length of the map, it results to be so close to obstacle 3 that our vehicle is obliged to lengthen the maneuver instead of performing two separate ones. The only exception to this can be found in the curved scenarios with 1000 m curvature radius when the vehicle is travelling at 40 km/h, due to the fact that the safety distance is considerably lower at this speed than it is at higher ones and that the length of the map is substantial. These considerations might apply also to obstacle 5 in some scenarios at high speeds because it has been positioned relatively close to obstacle 4, potentially resulting in a single overtaking maneuver in order to avoid obstacles 3, 4 and 5 all at once.

Regarding the assessments, we have imposed the same four conditions as in the single static obstacle avoidance tests described in Section 10.1. Details about the multiple static obstacle avoidance tests performed can be found in the documents included in the repository[5].

The tests have been performed at the maximum and minimum speeds of the range suitable for this MPC (MPC-H). Performing the simulations we have noticed that our controller responds coherently with the tests on single static obstacle avoidance. In particular, at 40 km/h all the tests are passed while at 100 km/h the only issue is on the curved scenario with curvature radius of 300 m when the vehicle travels counterclockwise. The considerations made for the failed test on the curved scenario with curvature radius of 300 m in the single static obstacle tests at 100 km/h (described in Section 10.2), also apply to these tests with multiple obstacles; hence the reader is referred to that part of the subsection for further information. Figure 38 shows the behaviour of this minor failure during the multiple static avoidance test.

[5]The "Multiple_Static_Obstacle_Avoidance_Test_Report" and "Multiple_static_obstacle_avoidance_test_specification_report" files generated by Simulink Test are included in the /Documentation/Test Reports/ file path.

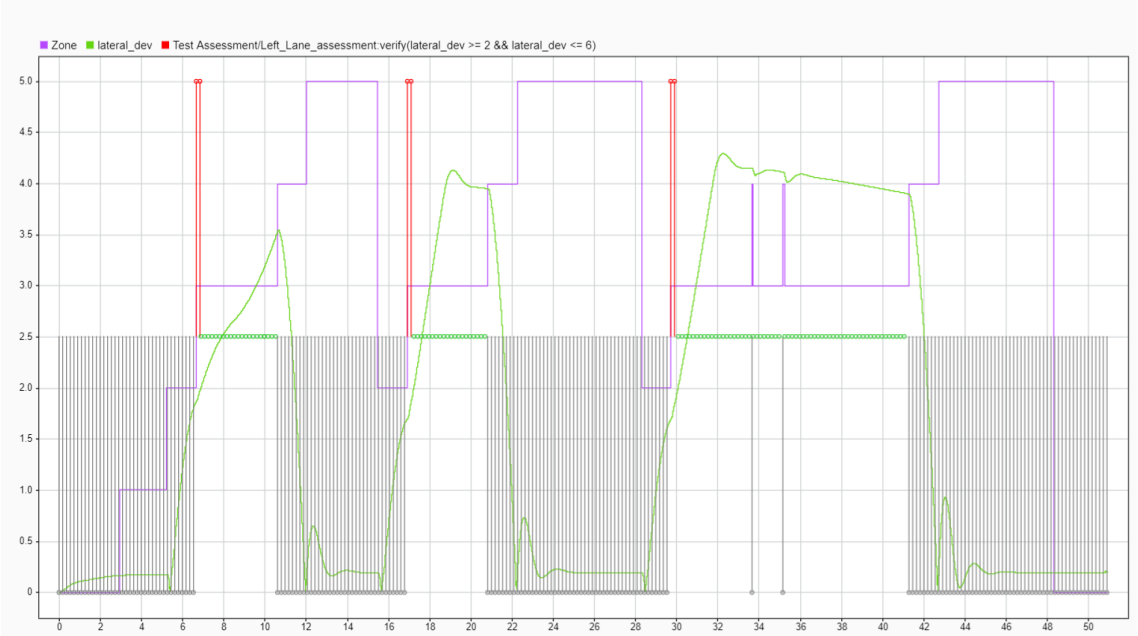


Figure 38: Assessment failure during the overtaking maneuver at 100 km/h on a curve with 300 m wide radius with counterclockwise direction

The Figure above shows the superimposition of the *Zone* and *lateral deviation* plots together with the violation points of the first assessment (2 $m \leq$ *lateral deviation* $\leq$ 6 m when performing the overtaking maneuver). The failure occurs because, at the inception of Zone 3, the vehicle has a lateral deviation slightly lower than 2 m. As discussed before, this does not represent a critical behaviour because of the safety distance considered.

11 Dynamic Obstacles Avoidance (MPC-H)

For the dynamic obstacles avoidance task, we have considered the same four assumptions already made for the static avoidance (as reported at the beginning of the Section 9) with 3 additional ones:

- the dynamic obstacle has a constant speed throughout the whole scenario;
- the dynamic obstacle follows the same trajectory of the ego vehicle but it stays always on the right lane;
- the distance between two consecutive dynamic obstacles is in any case at least equal to twice the safety distance evaluated considering the speed of the ego vehicle.

Thanks to these additional assumptions, once the ego vehicle detects the obstacle in motion, it calculates the 5 zones needed to perform the overtaking maneuver correctly. As a matter of fact, we have kept the same 5 zones nearby each obstacle with some adjustments to make them work also for dynamic obstacles. In case a dynamic obstacle is detected, the zones are evaluated in a different way with respect to those evaluated in case of a static obstacle detection. These are the steps followed in order to calculate the 5 zones:

1. as soon as the obstacle is detected, its position and speed are acquired by the system;
2. the trajectory of the dynamic obstacle is predicted;
3. the safety distance is evaluated considering only the speed of the ego vehicle;
4. Zones 1, 2 and 3 are evaluated as for the static obstacle case, but they are relative to the position of the obstacle in the future; to find the right points on the map, the following relations are used:

$$V_{relative} = V_{ref} - V_{obs} \tag{23}$$

$$DetectionTime = \frac{d - 40m - SafetyDistance}{V_{relative}} \tag{24}$$

$$SafeTime = \frac{d - SafetyDistance}{V_{relative}} \tag{25}$$

$$EndTime = \frac{d + 10m}{V_{relative}} \tag{26}$$

$$DetectionStep = DetectionTime/Ts \tag{27}$$

$$SafeStep = SafeTime/Ts \tag{28}$$

$$EndStep = EndTime/Ts \tag{29}$$

$$DetIdx = ActualIdx + DetectionStep \tag{30}$$

$$SafeIdx = ActualIdx + SafeStep \tag{31}$$

$$EndIdx = ActualIdx + EndStep \tag{32}$$

$$EntryIdx = EndIdx + \frac{max(SafeDistance, 40)}{V_{ref}Ts} \tag{33}$$

where d is the distance from the obstacle when it is detected and the parameters containing Idx in their name are the indexes in the reference map, corresponding

to the points where the zones are defined (following the procedure described in the static obstacle section [9]). In this case we have used indexes in the map because they are the easiest way to represent the predicted trajectory of the obstacle. However, this "trick" is only feasible in a simulation environment while in reality a different prediction algorithm is needed, but this goes beyond the purposes of our project.

5. as shown in the previous equations, the $EntryPoint$ is no longer always placed 40 meters after the $EndPoint$, but it is chosen equal to the maximum value between the $SafetyDistance$ and 40 m.

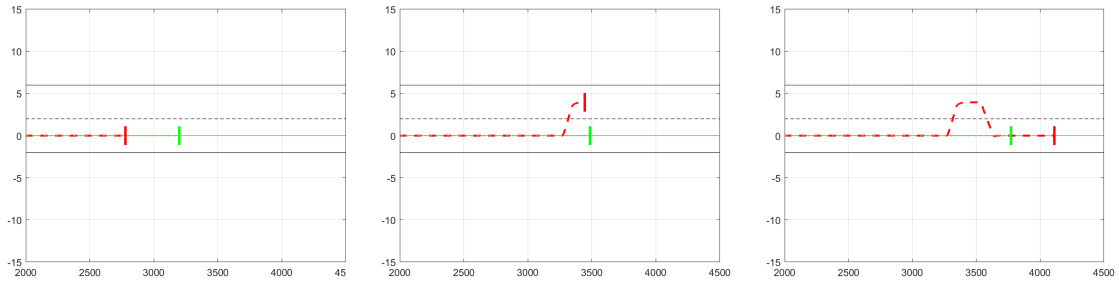
Once found, these points are passed to the same constraint generator function used for the static obstacle avoidance task.

11.1 Dynamic Obstacle Avoidance Tests

In order to test the performance of our controller when dynamic obstacles are involved, we have set up 2 simulations exploiting the same 22 scenarios used for the Static Obstacle Avoidance tests (Section 10). Both simulations have been performed at the maximum and minimum speed of the range suitable for the MPC-H ($40 \div 100$ km/h).

11.1.1 Single Dynamic Obstacle

Firstly, we have implemented a simple simulation considering a single obstacle in motion to verify the correctness of the overtaking algorithm. In particular, we have placed an obstacle with a starting position equal to the 40% of the total length of the considered scenario with a constant speed that depends on the speed of the ego vehicle. Indeed, when the simulation is performed with 100 km/h speed for the ego vehicle, the obstacle will have a constant speed of 50 km/h while it will have a constant speed of 10 km/h when the ego vehicle travels at 40 km/h. In Figure 39 the passing maneuver performed at 100 km/h is shown.



(a) Approaching the obstacle (b) Overtaking phase (c) Back to the right lane

Figure 39: Overtaking maneuver performed at 100 km/h with an obstacle travelling at 50 km/h in a straight line scenario

Further details about the single dynamic obstacle avoidance tests performed can be found in the documents included in the repository[6].

[6]The "Single_Dynamic_Obstacle_Avoidance-Test_Report" and "Single_Dynamic_Obstacle_Avoidance-Test_Specification_Report" files generated by Simulink Test are included in the /Documentation/Test Reports/ file path.

The tests performed have given great results at 40 km/h while there have been 2 fails at 100 km/h. The fails have been found in the curved scenarios travelled at 100 km/h in a counterclockwise direction with a curvature radius of 300 m and 500 m. The same considerations made in Section 10.2 and at the end of Section 10.3 are still valid here. The problem is that, when the vehicle is at the beginning of Zone 3, the lateral deviation is not grater of or equal to 2 m yet. As said for the other fails of the same kind, these can be considered minor failures.

11.1.2 Static and Dynamic Obstacles

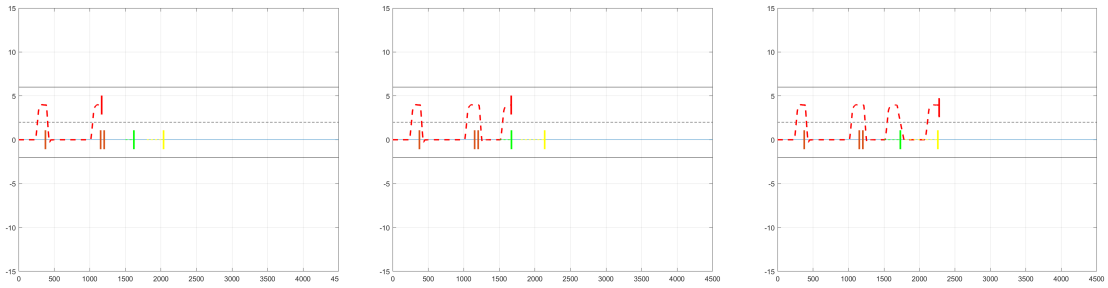
After performing the test with a single dynamic obstacle, we have decided to integrate the dynamic obstacles avoidance algorithm with the static one, so we have built a test-bench with both static and dynamic obstacles. For this test we have considered a total of 5 obstacles (4 static and 1 dynamic) placed on the maps as follows:

- Obstacle 1 (static) : placed at 7.5/100 of the total length of the map;
- Obstacle 2 (static) : placed at 23/100 of the total length of the map;
- Obstacle 3 (static) : placed at 24/100 of the total length of the map;
- Obstacle 4 (dynamic) : starting position at 30/100 of the total length of the map with a constant speed of 10 km/h;
- Obstacle 5 (dynamic) : starting position at 36/100 of the total length of the map with a constant speed of 20 km/h.

This obstacles configuration is not valid for testing purposes when considering shorter scenarios, as in case of curved scenarios with 500 m and 300 m of curvature radius. In these situations, we have tuned the positioning of a total of 4 obstacles as follows:

- Obstacle 1 (static) : placed at 15/100 of the total length of the map;
- Obstacle 2 (static) : placed at 16/100 of the total length of the map;
- Obstacle 3 (dynamic) : starting position at 43/100 of the total length of the map in case of 500 m curvature radius and starting position at 45/100 of the total length of the map in case of 300 m curvature radius (constant speed of 10 km/h);
- Obstacle 4 (dynamic) : starting position at 56/100 of the total length of the map in case of 500 m curvature radius and starting position at 70/100 of the total length of the map in case of 300 m curvature radius (constant speed of 20 km/h).

As partially explained in Section 10.3, we have tried to place the obstacles in such a way to test the capability of our MPC-H to lengthen the overtaking maneuver when two close consecutive obstacles are detected, and to come back to the right lane after passing an obstacle in motion and to complete another passing maneuver after a while when detecting the following obstacle.
Figure 40 shows the steps of the avoidance maneuvers performed considering both static and dynamic obstacles.



(a) Static obstacles (b) Slower moving obstacle (c) Faster moving obstacle

Figure 40: Overtaking maneuvers performed at 100 km/h with (from left to right) three static obstacles and two dynamic obstacles travelling respectively at 10 km/h (green) and 20 km/h (yellow) in a straight line scenario

Further details about the tests performed can be found in the documents included in the repository[7].
Again the only failed result of the test is the one corresponding to the curved scenario with a curvature radius of 300 m when the ego-vehicle is travelling at 100 km/h counter-clockwise.

[7]The "Dynamic_and_static_obstacle_avoidance-test_report" and "Dynamic_and_static_obstacle_avoidance-test_specification_report" files generated by Simulink Test are included in the /Documentation/Test Reports/ file path.

12 System Integration (MPC-H)

After testing the functioning of both static and dynamic obstacle avoidance together on the sample scenarios, we have proceeded with the integration of the path following task and the obstacle avoidance task. To do so, we have performed the tests with both static and dynamic obstacles exploiting three real scenarios among those already used in the path following tests (Section 8). These tests represent the Model-In-the-Loop, which will be described in detail in Section 12.1. Subsequently, the code has been automatically generated and Software-In-the-Loop tests have been performed (Section 12.2). Lastly, the code obtained has been flashed in an evaluation board in order to perform the Processor-In-the-Loop tests (Section 12.3).

12.1 MIL

For the Model-In-the-Loop task we have decided to tests our model in three different real scenarios considering the five vehicle models described in Table 2. In particular, the three real maps are:

- Adriatic Highway - A14 : travelled at 100 km/h;
- Puglia : travelled at 40 km/h;
- Indianapolis Speedway : travelled at 100 km/h.

Regarding the obstacles positioning, we have placed them as follows:

- Obstacle 1 (static) : placed at 7.5/100 of the total length of the map;
- Obstacle 2 (static) : placed at 23/100 of the total length of the map;
- Obstacle 3 (static) : placed at 24/100 of the total length of the map;
- Obstacle 4 (dynamic) : starting position at 30/100 of the total length of the map in case of A14 and Puglia and starting position at 50/100 of the total length of the map in case of Indianapolis (constant speed of 10 km/h);
- Obstacle 5 (dynamic) : starting position at 36/100 of the total length of the map in case of A14 and Puglia and starting position at 56/100 of the total length of the map in case of Indianapolis (constant speed of 20 km/h).

To integrate the path following test and the dynamic obstacle avoidance test we have set as assessments both the ones used for the path following (as described in Section 8.1) and the ones used for the obstacle avoidance (as described in Section 10.1), resulting in a total of 6 different assessments that have to be verified in order to considered pass the MIL.

12.2 SIL

12.3 PIL

References

[1] Roberta Frisoni, Andrea Dall'Oglio, Craig Nelson, James Long, Christoph Vollath, Davide Ranghetti, Sarah McMinimy, and Steer Davies Gleave. Research for tran committee–self-piloted cars: The future of road transport? *European Parliament, Directorate-General for internal policies, policy department B: Structural and Cohesion Policies, Transport and Tourism*, 2016.

[2] MathWorks Inc. Understanding model predictive control. `https://it.mathworks.com/videos/series/understanding-model-predictive-control.html`.

[3] Kim R. Fowler. Chapter 1 - introduction to good development. In Kim R. Fowler and Craig L. Silver, editors, *Developing and Managing Embedded Systems and Products*, pages 1–38. Newnes, Oxford, 2015.

[4] MathWorks Inc. Obstacle avoidance using adaptive model predictive control. `https://it.mathworks.com/help/mpc/ug/obstacle-avoidance-using-adaptive-model-predictive-control.html`.

[5] J. Kong, M. Pfeiffer, G. Schildbach, and F. Borrelli. Kinematic and dynamic vehicle models for autonomous driving control design. In *2015 IEEE Intelligent Vehicles Symposium (IV)*, pages 1094–1099, 2015.

[6] Gary J. Heydinger, Ronald A. Bixel, W. Riley Garrott, Michael Pyne, J. Gavin Howe, and Dennis A. Guenther. Measured vehicle inertial parameters-nhtsa's data through november 1998. *SAE Transactions*, 108:2462–2485, 1999.

[7] MathWorks Inc. Simulink test. `https://www.mathworks.com/help/sltest/index.html?s_tid=srchtitlel`.

[8] Katie Burke. "How Does a Self-Driving Car See?". `https://blogs.nvidia.com/blog/2019/04/15/how-does-a-self-driving-car-see/`, 2019.

[9] Qian Luo, Yurui Cao, Jiajia Liu, and Abderrahim Benslimane. "Localization and navigation in autonomous driving: Threats and countermeasures". *IEEE Wireless Communications*, 26(4):38–45, 2019.

[10] Openstreetmap. `https://www.openstreetmap.org`.

[11] Garrick J Forkenbrock and Devin Elsasser. An assessment of human driver steering capability. *National Highway Traffic Safety Administration DOT HS*, 809:875, 2005.

[12] MathWorks Inc. Model predictive control toolbox. `https://www.mathworks.com/products/model-predictive-control.html`.