Anticipative kinodynamic real-time motion planning for autonomous driving in dynamic road environments

Master's thesis

Double Master in Industrial Engineering and Automatic Control & Robotics

Author: Jordi Pérez Talamino

Advisor: Professor Alberto Sanfeliu Cortés

June 2017

Escola Tècnica Superior d'Enginyeria Industrial de Barcelona

Institut de Robòtica i Informàtica Industrial de Barcelona

Abstract

Nowadays autonomous driving is becoming more and more popular due to its numerous benefits. It is well known that autonomous vehicles can be much safer than human driven ones.

This project presents a novel approach for an anticipative on-road motion planner that can be implemented in real time and can deal with complex environments, such as driving in urban scenarios. The planner first discretizes the plan space and searches for the best trajectory based on a set of cost functions. A suitable low level control framework for a car-like vehicle has also been developed. The main contributions of this thesis are the use of G^2-splines for path generation, a method with 3rd order splines for generating smooth velocity profiles and a longitudinal low level controller for tracking the velocity profiles based on computed-torque control.

The proposed motion planner and control is implemented in MATLAB and tested through simulation in different representative scenarios, involving obstacles and other moving vehicles. Experiments show that the planner outputs high quality trajectories and performs intelligent driving behaviours, and the control system ensures performance and safety.

Keywords: autonomous driving, urban, anticipation, obstacle avoidance, kinodynamic, motion planning, path planning, G^2-splines, velocity profiles, vehicle control, controllers, trajectory tracking.

Table of contents

1 Introduction

The interest in developing autonomous vehicles has been present for a century, and is one of the greatest challenges that have been carried out in Engineering.

At the beginning of the 20th century it began the development of first aircraft and rockets capable of flying autonomously through rudimentary control systems. They were always military, and usually its mission was to be sent to collide against enemy targets. In the 1930s there was talk about cars that could drive alone, however, engineering efforts worldwide focused on developing control systems for aircraft because of their lack of safety and potential in military applications.

It was not until the 80's that the first autonomous cars appeared, the American project ALV (Autonomous Land Vehicle) and the project Eureka-Prometheus with Daimler-Benz. These first autonomous cars were able to perform basic movements and at low speed, reason why their application in a real environment was very limited, in addition to presenting a very high cost.

Since then and until today, interest in this field has been increasing and with it the amount of research and related projects. The technology has improved significantly in the last two decades, both in terms of hardware (sensors, actuators, CPU's) and software (control systems, perception, algorithmic) and it has reduced prices considerably. It is for this reasons that in the last years cars have begun to incorporate systems of assistance to the driving, or ADAS (*advanced driver-assistance systems*). They can be classified into 5 levels of automation, according to the SAE J3016 standard [1]:

- **Level 0. Driver only:** There is no assistance, the vehicle is completely driven by the person.
- **Level 1. Assisted:** Certain point aids restricted to specific cases, such as assisted parking or cruise control systems for maintaining a desired speed.
- **Level 2. Partial automation:** Automatic longitudinal control and steering, but restricted to very specific situations, such as a traffic jam. The driver should be monitoring the situation and other vehicles too.
- **Level 3. Conditional automation:** The driver only needs to be in position to resume control if necessary, the system itself knows its limitations and warns in advance. A clear example is an autopilot for driving in highways.
- **Level 4. High automation:** System totally autonomous and without need of a driver, but within a certain environment.
- **Level 5. Full automation:** Fully autonomous driving system in the whole route, from any starting point to any destination.

Nowadays ADAS systems present in commercial models are, in general, still in level 1 or 2. However, in the last 15 years there have been important achievements reached through investigation and competitions that implemented up to level 4-5 automation level. One of the most known competitions are the DARPA Challenges, starting with the DARPA Grand Challenge, in 2004. There was no human intervention in the whole race of 150-mile off road course and none of the 15 vehicles in the event completed the race. One year later, in 2005, a similar event was held; this time some participants reached the finish line. In 2007 the DARPA Urban Challenge was held, in which vehicles drove autonomously through simulated urban scenarios, demonstrating that fully autonomous urban driving is possible.

Since these challenges, numerous events and testing have been carried out, and research has continued both in the academic setting as well as in industry. Nowadays all the industry is focused in the improvement of this technology, and some companies have developed new models with self-driving capabilities. Tesla Motors with its Autopilot system and the Google self-driving car are some examples receiving considerable media attention.

1.1 Motivation

Every year 1,25 million people die in traffic accidents worldwide, according to the World Health Organization reports. Taking as an example Europe, 26100 people died in 2015 and road traffic injuries are the leading cause of death among young people. In addition, almost all of the accidents are attributed to driver error, legally intoxicated drivers or distractions. Autonomous vehicles have the potential to dramatically reduce the contribution of negligence and human error as the causes of traffic accidents. Furthermore, they can provide personal mobility to people who are unable to drive because of a disability.

Nevertheless, to achieve this objectives autonomous vehicles must reach a human-acceptable driving performance. This performance level is specially related on the software, on developing and improving methods and algorithms that meet the requirements, being crucial motion planning.

1.2 Objectives

The main objective of this project is to contribute to the development of autonomous driving vehicles, presenting a novel approach of an anticipative motion planning algorithm.

For this overall objective to be achieved, it is necessary to fulfil the following more specific points:

- Establish a planning framework.
- Incorporate a new technique of path generation.
- Use a new approach for the velocity profile generation.
- Develop a suitable low level control system.
- Implement all the system in MATLAB and validate its behaviour through simulations.

1.3 Scope

This project starts with the study of the state of the art in autonomous driving, and includes the development of a new approach of motion planning, and also of a suitable low level control framework. The algorithms are implemented in MATLAB and tested in the most relevant scenarios through simulation.

It is beyond the scope of the project the optimization of the implementation of the algorithms in MATLAB, and also the implementation and integration of the systems in a real vehicle.

1.4 Outline of the thesis

Section 2 presents the state of the art of the relevant subjects related with this thesis. In section 3 the developed approach is introduced, detailing each subsystem separately. Section 4 shows the behaviour of the planner in a wide variety of scenarios.

2 State of the art

The following is a brief insight of the state of the art of the fields related to planning and control in autonomous driving. This is useful to expose the previous studies that have been relevant to this work and to explain and define concepts necessary for a better understanding of the proposed solutions.

2.1 Planning in autonomous driving

Driver-less vehicles are essentially autonomous decision-making systems that process a stream of information from the perception system and from other sources such as prior knowledge about the road, traffic rules, vehicle models, etc.

This decision making is the key to autonomy and is achieved through planning algorithms.

The main objective of an autonomous driving system is to achieve transportation from one point to another. The initial step for planning is to find a way to arrive at the destination, starting from the current position, known as *route planning*. There are two different possible scenarios:

- Unstructured environment: when there are no roads or any kind of structured pavement. In this case is needed the spatial information relating to the travelling zones (where one can drive) and the obstacle zones (where driving is not allowed). An example of this scenario is driving in the countryside.

- Structured environment: when the destination can be reached through a road network. In this case, it is usually assumed that the road links are known, and finding an optimal (or close to it) path becomes a well known problem.

Once a route planning is generated, it is needed to provide the vehicle with a safe and collision-free path and velocity profile towards its destination, while taking into account the vehicle dynamics, its manoeuvre capabilities in the presence of obstacles, the traffic rules and road boundaries. This is the task of the *motion planning* algorithms.

2.1.1 Motion planning

Motion planning for autonomous vehicles requires rapidity, coherency, providentness and predictability [2]:

- Rapidity: Decision making needs to be fast. Despite some decisions such as route planning, tactical decisions need to be fast, usually $> 10\ Hz$.
- Coherency: All decisions should align well with a long term goal.
- Providentness: Planning need to have some temporal horizon to predict how the situation will look like.
- Predictability: The taken decisions should be predictable in the sense of acting like a human driver in a certain situation.

The most common adopted approach to fulfil this characteristics is to partition and organize tasks in a hierarchical structure, shown in Figure 1 [3].

For most problems of interest in autonomous driving, exact algorithms with practical computational complexity are unavailable. Some of the existing motion planning algorithms originate primarily from the field of mobile robotics, such as RRT, RRT* and PRM and graph searches. However, some of them present limitations, such as the RRT, that gives jerky paths.

Because of that, nowadays there are a lot of different approaches to the autonomous driving motion planning problem [3] [4]. But in general, the main idea behind the most popular methods is to discretize somehow the space (that also depends on how it is modelled) and search for the best solution.

The most popular techniques used for on-road autonomous driving are the *lattice planers*, where the planner uses a limited horizon both in terms of time and space, and the search space contains a certain geometric curve. Typically, the 2 ways of discretizing the space is to do it in action or state space (Figure 2).

It is interesting to mention that this is the solution adopted by the winners of the DARPA challenges, Stanley in 2005 and BOSS in 2007 [5] [6] [7]. The second classified in 2007, Junior, also presents this approach [8] [9] [10].

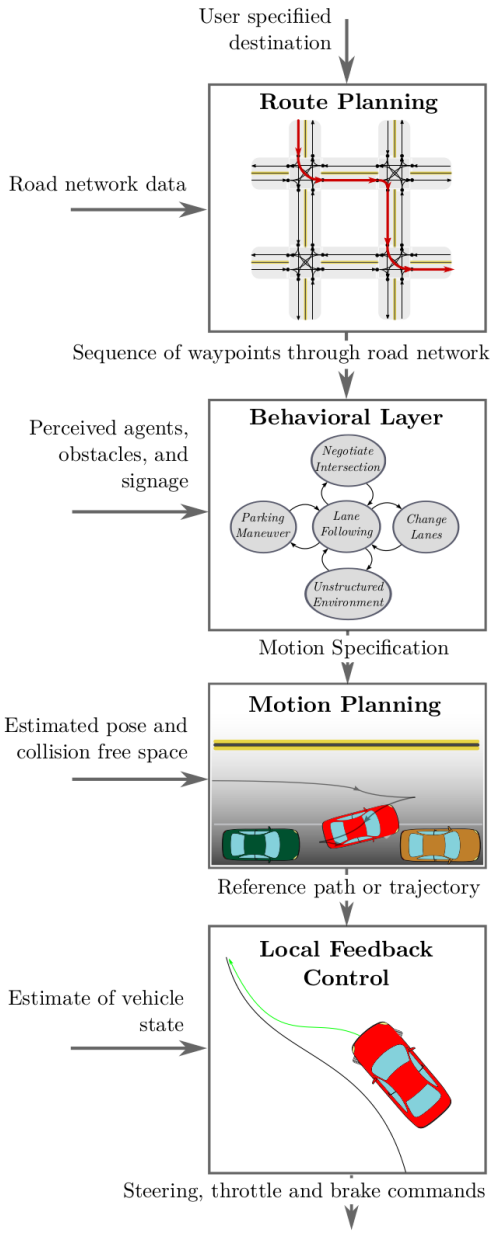


Figure 1: Most common hierarchical structure used in autonomous driving

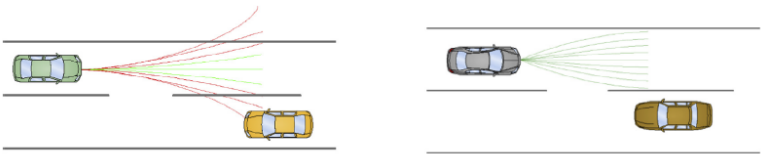


Figure 2: Action (left) and state (right) space discretization

2.1.1.1 Moving obstacles and anticipation

In order to be able to anticipate moving obstacles such as other vehicles, planners need a reliable detection of the obstacles and the future behaviour or movement of these vehicles must be imparted or inferred. In general, this prediction problem is extremely difficult, as there is no control over these other objects, so knowing exactly where they intend to go and how they intend to get there is impossible.

However, when these dynamic obstacles are vehicles operating in urban environments, it is much easier to infer their likely behaviour through exploiting the structure inherent in such environments: vehicles travelling on roads typically follow common rules of the road [11].

As explained in [4], planners takes into account moving obstacles and traffic using different approaches, from Markov Decision Processes, state machines, cost structures, probabilistic models, or Game Theory as an examples.

2.1.2 Path generation algorithms

Lattice planners need to generate candidate geometric paths that will be followed by a low level control system. There exist also a lot of different approaches in path generation algorithms, but paths for autonomous driving need to fulfil kinematic and dynamic constraints. In addition, for driving in road scenarios (structured environments) paths need to follow also the road shape.

These kinodynamic constraints are mandatory to achieve good control performance and also to assure that paths can be exactly followed by the vehicle [12]. More specifically, these constraints are related with the geometric continuity (G^n) of the path.

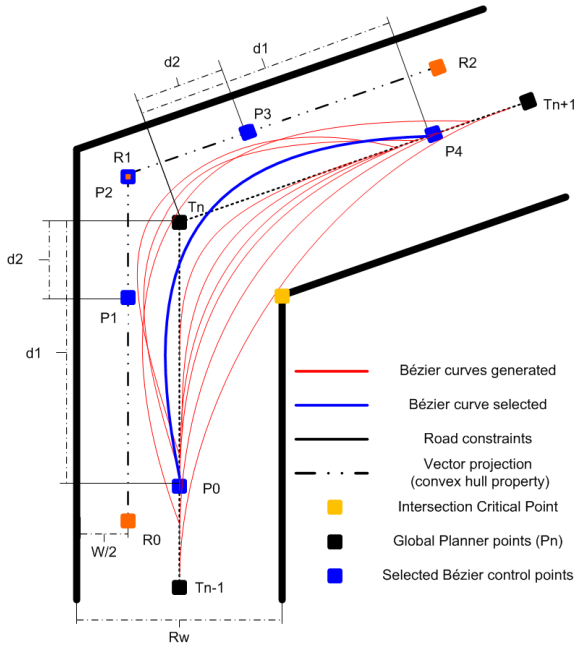
The most popular used techniques in path generation use arcs, clothoids, polynomial spirals, splines and Bézier curves.

They differ in the number of parameters and degrees of freedom, and consequently, in the level of complexity. Curves with, at least, 4th or 5th order polynomials are frequently used, even arriving to 7th order. Generating a path from one state to another and satisfying the mentioned kinodynamic constraints is not an easy problem, and some solutions can be very complex and may need a numerical optimization algorithm.

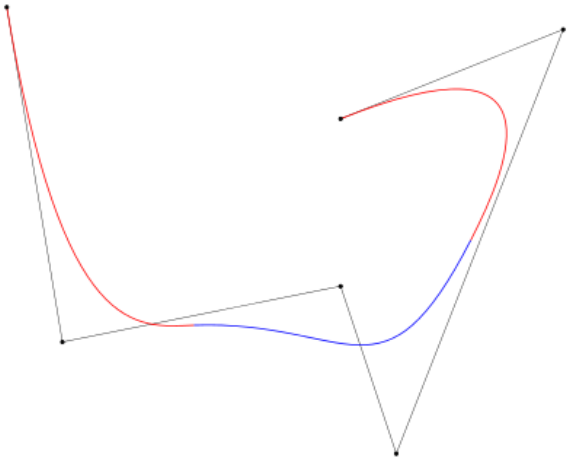
As an example, Figures 3a and 3b show a motion planning algorithm that uses a 4th order Bézier curve and a 5th order B-spline, respectively. Note the complexity of the generation from the road shape, with the presence of non-intuitive geometric waypoints and multitude of parameters.

For this reason, a very useful property for path generation algorithms is the correlation between shape and parameters: if the parameters of the path correspond to its shape on an intuitive level.

As other example, the path generation algorithm [13] used in planner [14], it presents no geometrical



(a) Example of a path generation using 4th order Bézier curves [15]



(b) Example of a path generation using B-splines

Figure 3: Some examples of difficult shape parameters in path generation algorithms

complexity but it requires a numerical optimization to be generated.

2.1.3 Velocity profiles generation algorithms

Regarding the velocity profile generation, its development has been less considerable than in the case of path generation algorithms.

Trapezoidal velocity profiles are widely used due to its simplicity (Figure 4), despite its dynamic limitations because of having discontinuities in the acceleration.

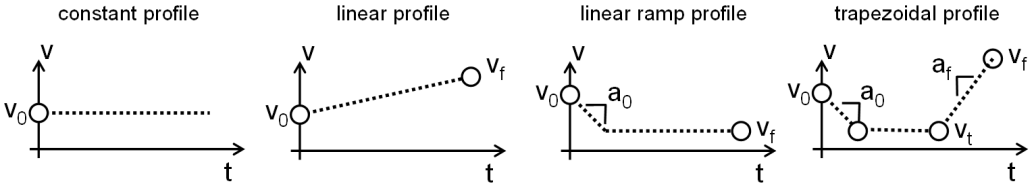


Figure 4: Example of trapezoidal velocity profiles

Nevertheless, in recent years more complex algorithms are being incorporated, using splines. There are also approaches that use MPC (Model Predictive Control) [16].

Generating a spline from a desired state of velocity and acceleration to a final desired state can lead to high order polynomials and complexity, implying high computational cost. In the same way

as in the path generation algorithms, the imposed restrictions are related with the spline parameters, and often this parameters are not easy to adjust and they need some post-optimization.

Other issues are related to the re-launching of the algorithm, that is the capacity of generating a new profile from any point that follows exactly a previous one computed some iterations before without oscillations.

The most common spline methods generate velocity trajectories over time, such as [17] and [18], but there are approaches that use the path length coordinate instead of the time [19].

2.2 Low level control

Most of the current work separates trajectory generation and execution even more strongly and employs real controllers, in the sense that these try to exactly follow the trajectory instead of just being guided by it. This considerably improves and simplifies the predictability of the vehicle's behaviour. In addition, this approach also guarantees higher stability and safety levels.

The majority of the used control systems decouple the longitudinal and the lateral control, although some approaches use a combined control system [20] [21]. However, the increase of complexity of the resulting system in relation to the benefits can be questioned.

Longitudinal controllers that track a velocity profile are based usually on PI or PID control [22]. Sometimes, they are combined with feedforward terms related to the longitudinal vehicle model, in order to compensate the non-linearities [23] [24].

The lateral controllers that follow a path are called *path trackers*. Path tracking refers to a vehicle executing a defined geometric path by applying appropriate steering motions that guide the vehicle along the path. The goal of a path tracking controller is to minimize the lateral distance between the vehicle and the defined path, minimize the difference in the vehicle's heading and the defined path's heading, and limit steering inputs to smooth motions while maintaining stability.

One of the most popular classes of path tracking methods found in robotics and also in self-driving vehicles are the geometric path tracking algorithms. These methods exploit geometric relationships between the vehicle and the path resulting in control law solutions to the path tracking problem. These techniques often make use of a look ahead distance to measure error ahead of the vehicle and can extend from simple circular arc calculations to more complex ones.

There are many other approaches implying more complex solutions based in the kinematic and dynamic models of the car. This kind of solutions can offer slightly better performance in certain situations, but in general they require the knowledge of a lot of car physical parameters that are difficult to adjust, or they require non-intuitive trajectories and control, on-line optimization and computational cost [25].

3 AKRP motion planner

The AKRP (anticipative kinodynamic real-time planner) is a motion planner designed for autonomous on-road driving. It can deal with usual driving manoeuvres, such as adapting the velocity to the road shape, follow and change lanes and also deal with obstacles. However, its biggest potential is its anticipative behaviour with respect other moving vehicles or *dynamic obstacles*. This reason makes it very interesting for driving in urban scenarios, plenty of complex situations. In addition, it is designed to be relaunched at high frequency and assure a high level of reactivity.

3.1 Architecture overview

The AKRP is thought to be working in a whole autonomous driving system, so it needs other systems to operate. Figures 5 and 6 show the needed inputs and outputs, respectively.

It is supposed that a route planner computes the desired route to a certain destination. The AKRP planner needs some information related to the track ahead. First of all, it needs the paths of the track center lanes. These paths can be computed with the same path generation algorithm that AKRP uses, explained in section 3.5, using information from road network data and the perception system also. The planner also needs the information of the number of lanes, the direction of traffic on each one and other traffic rules such as the maximum allowed speed. The planner internally has also traffic rules, that can be adapted to each country. In this case, AKRP is thought to be used with Spanish traffic rules.

From the perception systems, AKRP needs a list with all the detected obstacles, static and dynamic. They are presented and explained in section 3.2.

The internal operation of the AKRP is explained in detail in section 3.3, but basically it has 2 outputs: a path and a velocity profile that the vehicle must follow. These 2 trajectories need some kind of low level control in order to drive autonomously the vehicle. In this project, the control systems have been also developed and they are included in the simulations.

The output path is a geometric trajectory that is fed to a low level controller, the Stanley controller, that follows it precisely. Analogously, the output velocity profile is a trajectory with the vehicle required velocity over time ($v(t)$), that is followed by a longitudinal controller that acts in the throttle and brake. Actually, due to this controller design, the velocity profile is an acceleration profile, $a(t)$.

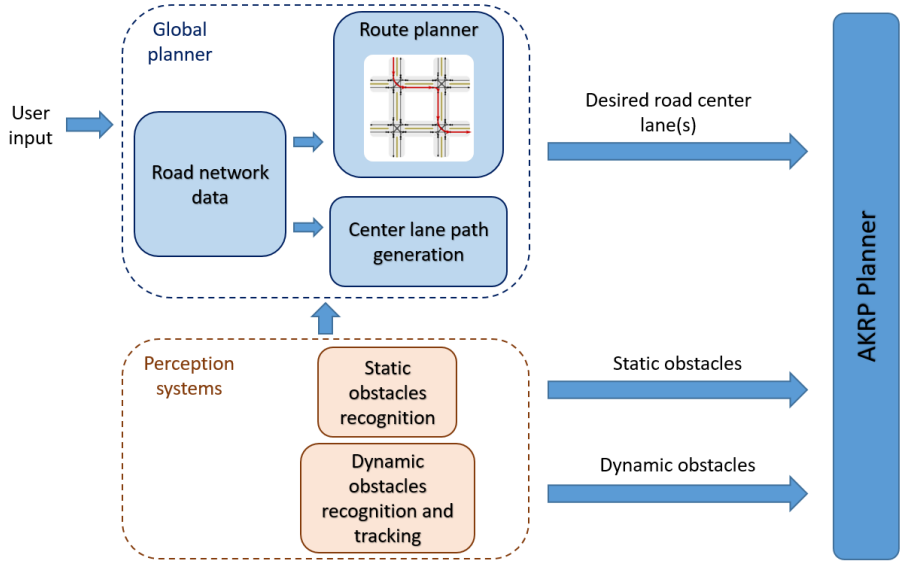


Figure 5: AKRP inputs

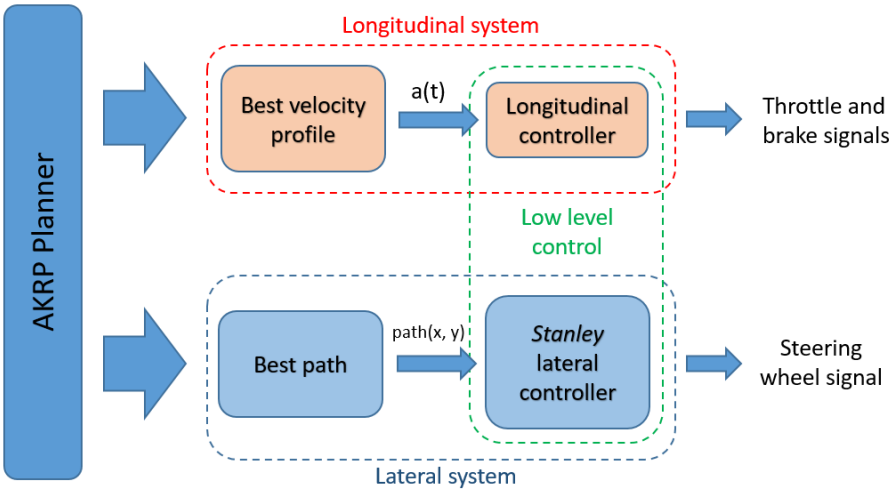
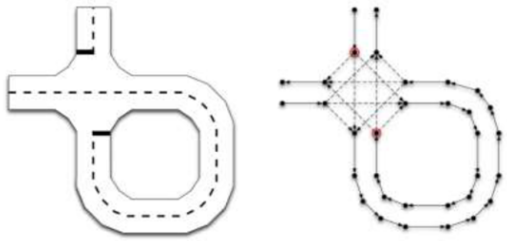


Figure 6: AKRP outputs

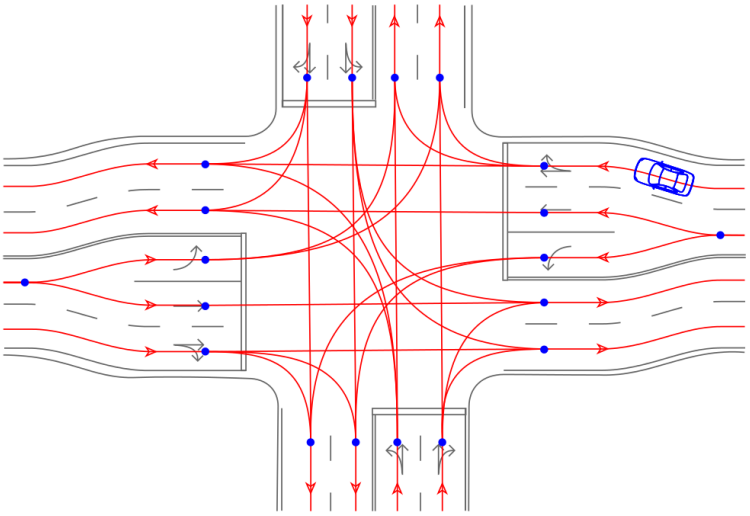
3.2 Environment modelling, assumptions and limitations

3.2.1 Environment

The planner uses a track model based on its center lane paths. A center lane path is the geometric center of a track lane inside a lane, but it is also the desired path that the vehicle would like to follow in intersections and incorporations, or when there is no physical lane painted. It can be shown in Figure 7b. In fact, the center lane paths are the smooth geometrical representation of the lane graphs used to model the road networks (Figure 7a).



(a) Lane graph model example



(b) Example of the center lane paths in a complex intersection

Figure 7

The vehicle is modelled with the dimensions shown in Figure 8. Dimension *car_l* is the length of the car, *car_w* is the width, L is the distance between axles (also called wheelbase) and *car_r_ovh* is the length of the rear overhaul. The rest of dimensions are used to define n circles that are used to check collisions:

$$r = \sqrt{\frac{car\_l}{n^2} + \frac{car\_w^2}{4}}$$

$$d = 2 \cdot \sqrt{r^2 - \frac{car\_w^2}{4}}$$

$$d\_ovh = (car\_l - (n-1) \cdot d)/2$$

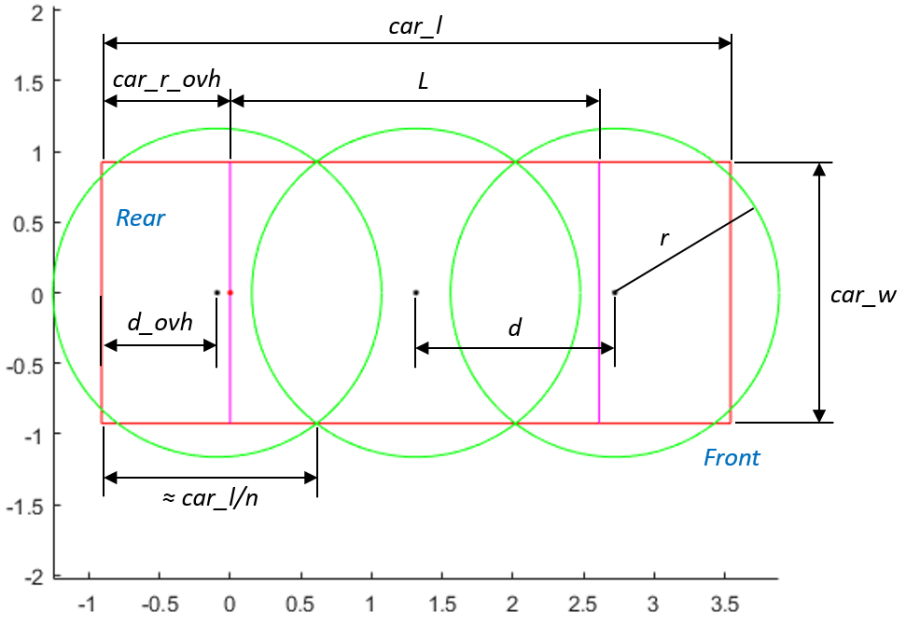


Figure 8: Car-like vehicle dimensions

Increasing the number of chosen circles, the overlapping decreases and increases the number of collision checking. A good number for n is 3, and is conservative with the dimensions.

3.2.2 Static obstacles

Static obstacles are those that are not moving with respect to the ground. They are modelled as circumferences, with a center (x, y) coordinates, a radius and a certain id, for identification. There can be obstacles that have only 1 circumference, like a tree or a paper bin, but there can also exist big obstacles composed by several circles. One example of that could be a large vehicle stopped.

3.2.3 Dynamic obstacles

A dynamic obstacle is anything taken into account that is moving with respect to the ground. They are modelled also with circumferences, but they have a certain velocity and a predicted path.

This thesis makes 2 assumptions for them:

- They travel at constant speed.
- They will follow a known path, primarily following its lane.

These 2 assumptions may seem very strong, but they are not so. Of course that when supposing a constant velocity for a dynamic obstacle, there will be uncertainty, because is very likely that the obstacle will change its velocity over time. But this assumption is compensated with the relaunching. Every time that the planner is relaunched, a dynamic obstacle will have its current speed, so if it is accelerating, in every new iteration its speed will be increased and a new prediction will be taken into account.

The same happens with the path. In order to deal with complex situations in real life, such as a dynamic obstacle that suddenly wants to change its lane or its direction, it would be needed the addition of a complex prediction system, also with uncertainty. However, this is out of the scope of the project, as it is not part of the planner framework. So the assumption of the known path relies on having an external prediction system that gives to the planner the most likely path in every iteration.

3.3 System operation and features

In this section it is explained how the AKRP works. Each subsystem of the planner is then detailed in the following sections. Figure 9 shows the operation flow chart of the AKRP.

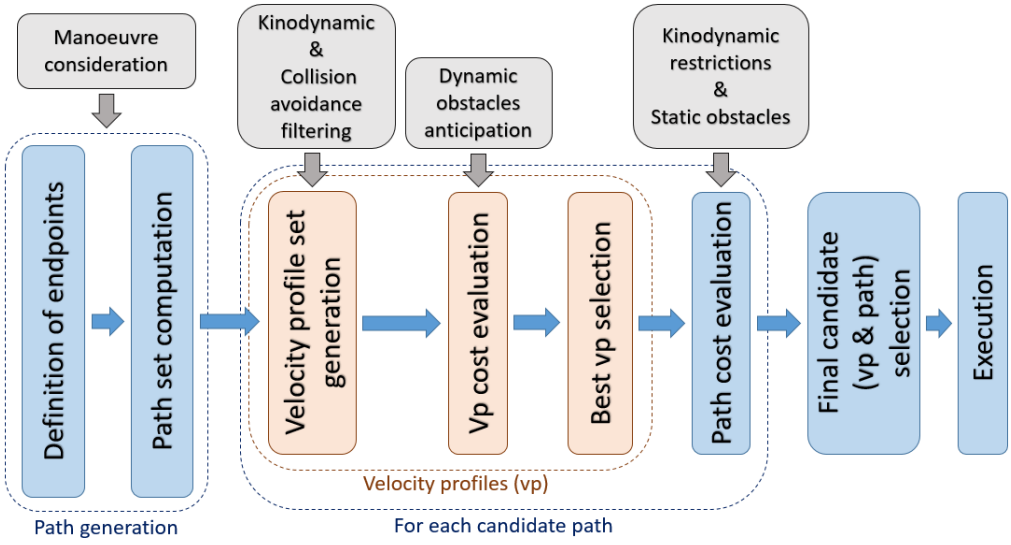


Figure 9: AKRP operation

The AKRP uses some ideas of *state lattice* methods, such as [19]. The states ($\mathbf{X}$) are composed by the following variables:

$$\mathbf{X} = [x, y, \theta, \kappa]$$

Where x and y are the position coordinates with respect to some fixed reference in the ground, θ is the heading and κ is the curvature, which is related with the steering wheel angle (Figure 20).

The planner discretizes the environment choosing several endpoints, which are state configurations, using the algorithm described in section 3.5.4. This algorithm incorporates some ideas of [19], but adapted to the AKRP.

Then candidate paths are generated from the host vehicle state to the endpoints. AKRP uses a novel approach to generate paths from current motion planners, it uses G^2-splines. These paths are always kinematically feasible for the host vehicle. An example of some candidate path generation is shown in Figure 10.

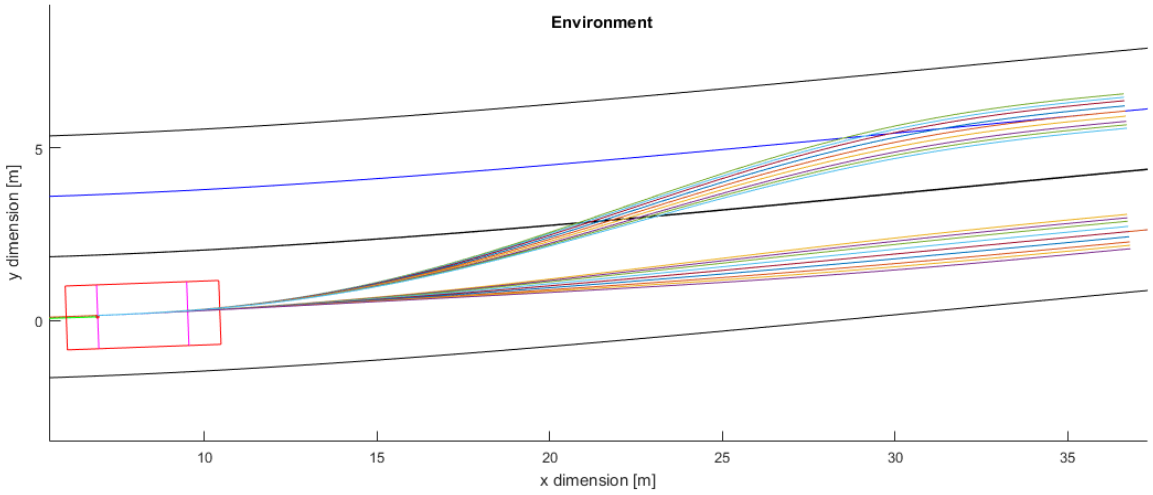


Figure 10: Example of candidate paths

Depending on the situation and the planned manoeuvre, the process of generating candidate paths varies (it is detailed in section 3.5.4). Nevertheless, the main system operation is the same (Figure 9). Once the planner has a set of candidate paths, for each one of them it computes a set of velocity profiles in order to anticipate dynamic obstacles. Each velocity profile has a cost associated and the one with the minimum cost is chosen. So at this point, there is a set of paths, each one with its best velocity profile associated. Finally, all the candidate paths with its velocity profiles are compared with a cost structure and the minimum cost solution (path and velocity profile) is chosen to be the executed one.

The generation of the velocity profiles is detailed in section 3.6. They are computed using a novel approach with 3rd order splines and take into account the dynamic restrictions of the candidate path and also of the center lane path of the road.

All the cost structure is explained in section 3.7.

The output of the AKRP planner is a kinematically and dynamically feasible path with an associated velocity profile for the host vehicle, also fulfilling comfort restrictions for the passengers, such as bounded accelerations and jerk.

The planner outputs are computed up to a certain lookahead horizon, which is a parameter. This horizon can be seen in Figure 11 in terms of the *station*. The station is the longitudinal coordinate of the road.

It is very important to take into account this prediction horizon length. Consider the problem of overtaking some obstacle. Figure 12a shows a planning example of a whole overtaking manoeuvre. In this situation, the planning horizon is assumed to be long enough to allow the complete manoeuvre planning. The planner can anticipate dynamic obstacles up to the manoeuvre completion, so if it initiates the overtaking manoeuvre is because it is safe. Consider that the left lane is an oncoming traffic lane, it is mandatory that the planner can guarantee safety.

Now, consider another situation, shown in Figure 12b. In this case, due to some reason, the perception system range is smaller than the needed horizon for overtaking. What should the planner do? If the other lane has the same way, planning only a lane change manoeuvre to the other lane is safe enough, because the host vehicle only has to be careful with vehicles coming behind. But in the case of oncoming traffic, the manoeuvre cannot be performed. If the obstacle is static, the best solution is to stop the host vehicle safely in its lane.

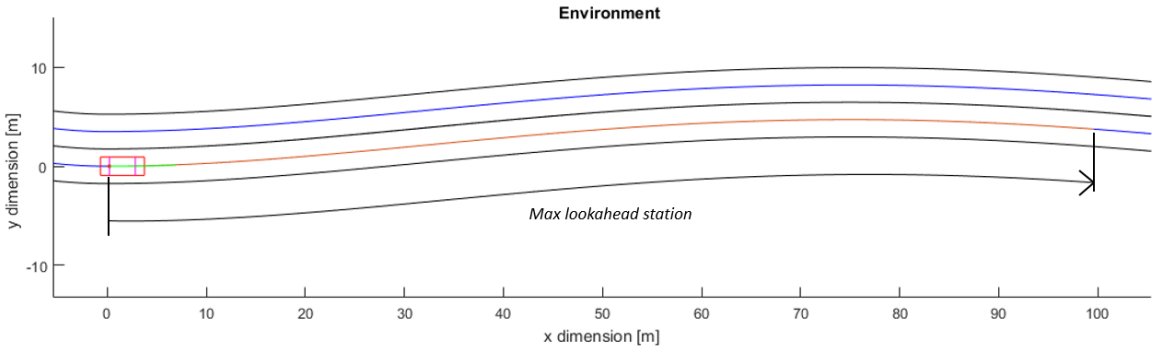
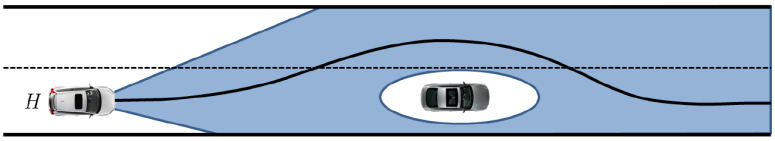
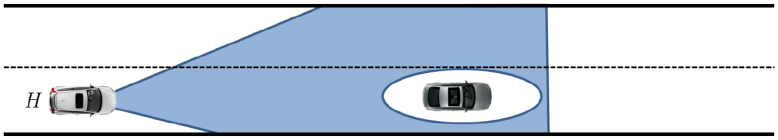


Figure 11: Lookahead station

This kind of restrictions in the planning horizon can be caused by various reasons. The first one is the sensor range: radars, lidars, etc. may have up to 150 m of reliable range. Travelling in a road at 90 km/h this range is covered in 6 s, and this is not so much time to perform some manoeuvres. The other reason is having occlusions between obstacles, that is also related with the perception system.



(a) Enough prediction horizon to plan a complete overtaking manoeuvre



(b) Smaller prediction horizon due to restrictions in the perception range

Figure 12: Planning horizon considerations

3.3.1 On-line relaunching

AKRP planner is thought to be relaunched at a relatively high frequency, in fact it is based on the framework of [19], which is a real-time motion planner. Usual planners are computed at 10-20 Hz, but some of them can not perform all the computations at each iteration due to an excessive computational cost.

It has to be taken into account the refresh frequency of the perception system, such as static and dynamic obstacles, for example. In some conditions it could have no sense to re-plan faster than the information refreshing rate.

3.4 Car-like vehicle model and low level control

The AKRP outputs need to be followed by some low level control system. This control system has been also developed and it is explained in this section. It has been used for validating and simulating the AKRP.

The actuators in a car-like vehicle are the throttle, the brake and the steering. Humans totally decouple longitudinal and lateral control when driving in usual conditions. Nevertheless, this does not mean that one system could not affect the other: when a sharp turn is perceived ahead, the driver will lower the speed, so these 2 systems are somehow related. Racing drivers, on the other hand, perform a much more complex control that takes into account longitudinal and lateral actuators, as well as having knowledge of the vehicle dynamics.

But the AKRP is intended to be used in comfort conditions, and its outputs are already kinematically and dynamically feasible by the vehicle. The chosen architecture has two separate low level controllers: a longitudinal controller following the velocity profile, and a lateral controller following the path. In this way, controllers can be simpler, more robust and can be used at higher frequencies than other approaches, achieving better performance and safety levels.

Both controllers have been implemented first in Simulink models, and once validated they have been implemented in Matlab, for being used with the AKRP planner. The validations in Simulink are shown in Annexes A and B.

3.4.1 Longitudinal car model

A car-like vehicle longitudinal model is shown in Figure 13. Depending on the center of gravity location, on the height of the aerodynamic force (F_{ad}), on the acceleration and on the slope of the road α, the front and rear wheels will have different normals (N_d and N_p).

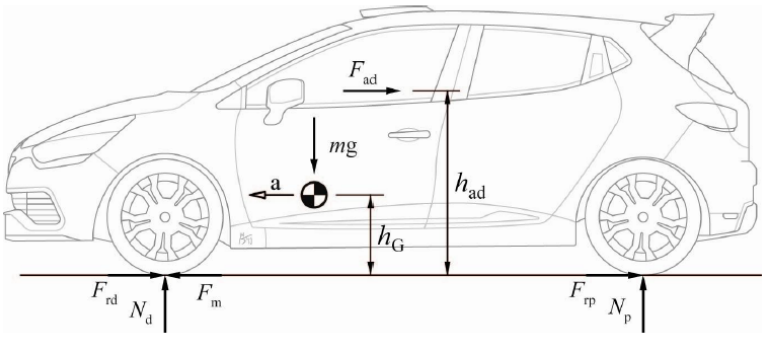


Figure 13: Longitudinal dynamic model of a front traction car

The longitudinal dynamic equation considering an uphill slope α is:

$$F_m - F_{rd} - F_{rp} - F_{ad} - mg\sin(\alpha) = m \cdot a$$

Where F_m is the force generated by the engine, $F_{rd} = N_d \cdot f$ is the front wheels friction due to rolling, $F_{rp} = N_p \cdot f$ is the rear wheels friction due to rolling, $F_{ad} = \frac{1}{2}\rho S c_d v^2$ is the aerodynamic force, a is the current acceleration and v is the velocity. f is the rolling coefficient, ρ is the air density, S is the frontal drag area of the vehicle and c_d is the longitudinal drag coefficient.

A very good simplification is to treat the 2 axles as 1. Then, the rolling friction force is:

$$F_r = mgf\cos(\alpha)$$

In order to simulate and validate the longitudinal controller, the longitudinal behaviour of a car-like vehicle has been modelled in Simulink (Figure 14a) using a more complex model that takes into account much more parameters. This model has been adjusted to fit the real car where the IRI is developing autonomous driving projects, a Seat León Cupra 2.0 TSI 265 cv of 2014.

A car is a very complex system with lots of non-linearities so is mandatory to simulate and validate the control system before they can be tested in the real vehicle, specially in terms of security. This car model is composed by different parts (Figure 14b):

- **Engine**. It's a generic reciprocating internal combustion engine, adjusted with all the parameters to fit our particular spark-ignition engine and its torque curve.

- **Gearbox**. It has been modelled to match an automatic gearbox with its gear change logic very similar to the real one to accurately test the controller. This is a very important system explained later in this section.

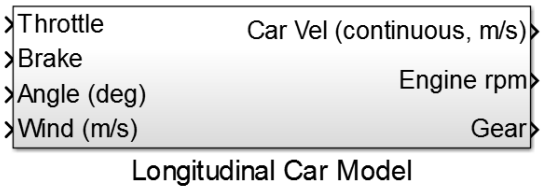
- **Brakes**. Brake system is divided in the front axle and rear axle brakes, with a realistic weighting of its influence, as the real car has (60% front and 40% rear).

- **Inertias**. Realistic inertias have been included of the shafts and other elements present in the real car transmission, as the final drive.

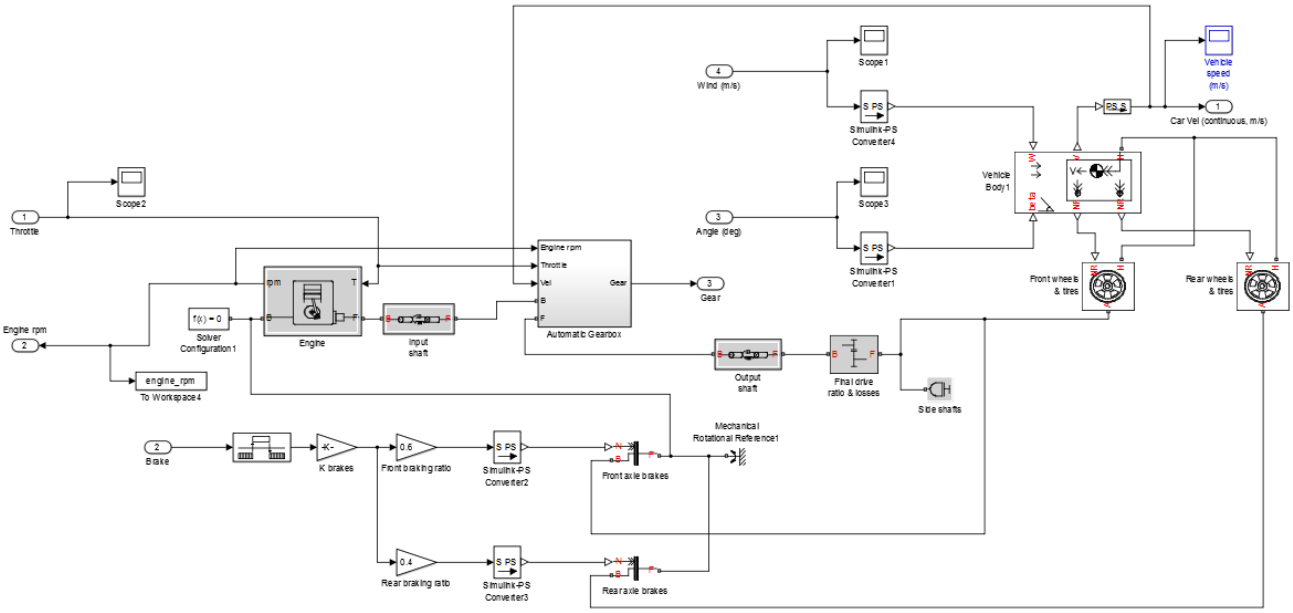
- **Wheels**. Wheels systems are divided in the front pair and the rear pair. They take into account the relative slip against the asphalt.

- **Vehicle body**. It's a generic longitudinal model of a car-like vehicle. It has all the parameters related to its dynamics and also can be perturbed with an arbitrary road inclination and wind to test difficult environments.

The throttle signal $\in [0, 1]$ actuates the engine and the brake signal $\in [0, -1]$ controls the brakes. Other inputs are the perturbations: road tilt angle in degrees and longitudinal wind in m/s. The model has 2 outputs, which are needed by the controller: the current velocity in m/s and the engine rotational speed in min^{-1}. The current gear is not used directly by the controller because in our real car we have no access to this information so we must estimate it from the other 2 values.



(a) Car model block



(b) Detail of the car model block

Figure 14: Longitudinal car model block in Simulink

The automatic gearbox has 6 different gears. The engagement and disengagement of gears has been modelled by adding a mechanical clutch for every gear and building a controller system with a shift logic that controls the clutches such that the overall behaviour is close to the real gearbox. In Figure 15 it can be shown the gearbox model architecture. The behaviour of this model is fully customizable so it can be adapted to almost all cars and conditions. It has the capacity of shifting up and down and is also capable of having neutral position (engine disengagement when the vehicle is stopped) with capability of re-engaging and start moving the vehicle from rest.

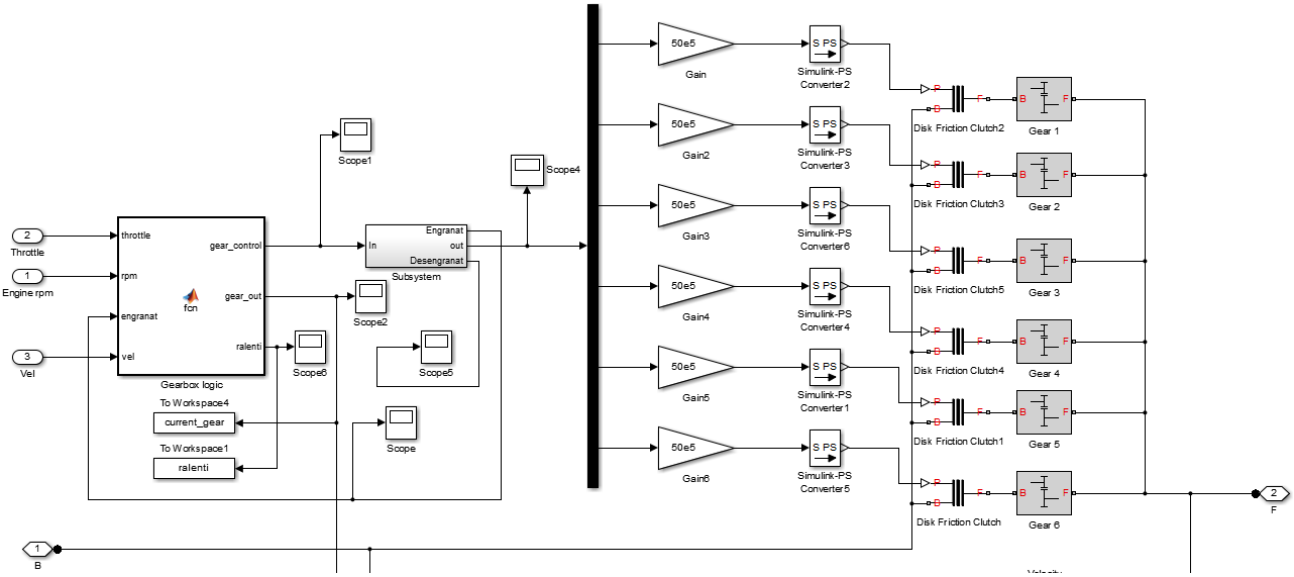


Figure 15: Detail of the gearbox block

3.4.2 Longitudinal controller: Computed-torque controller

The designed controller for tracking the AKRP velocity profiles is based on a computed-torque controller scheme used in robot manipulators (Figure 16). A computed-torque controller is a special application of feedback linearisation of non-linear systems. It uses known plant parameters to compute the inverse dynamics and the required torque in order to track a desired trajectory. It is thought to fulfil a desired acceleration and velocity while it tends to a desired position (each one is the derivation of the other).

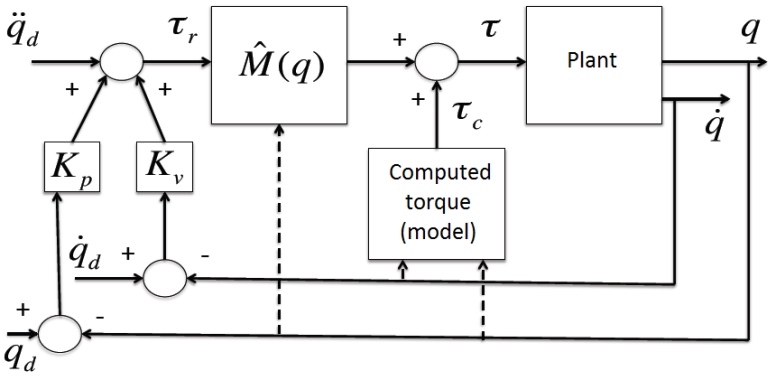


Figure 16: Computed torque control scheme with a 'PD'

The controller, shown in Figure 17, has as input the desired acceleration profile (computed with the velocity profile generation algorithm explained in section 3.6). Then the desired velocity and position inputs are calculated by integrating the desired acceleration. The errors in velocity

($e_{vel} = vel_{desired} - vel$) and position ($e_{pos} = pos_{desired} - pos$) are then computed. This controller has been tested with different linear control types, typically computed torque controllers use a 'PD' or 'PI' control but finally it has been chosen a 'PID' architecture because for this application its performance is much better with no penalty.

At this point it can be computed the acceleration requirement of the car, in m/s^2:

$$a_{req} = a_{des} + K_v \cdot e_{vel} + K_p \cdot e_{pos} + K_i \cdot \int e_{pos}$$

Depending on the different weights, the behaviour of the controller will change:

- 'K_p': This will affect the importance of the position tracking in the controller
- 'K_i': This will reduce the error of the position through its integration. Usually this term is not used in many applications, but it has been checked that this integral part improves a lot the performance without penalty.
- 'K_v': This will affect the importance of the velocity tracking in the controller

The acceleration requirement of the car (a_{req}) is then used to compute the needed force that must be provided to the vehicle. In fact, it is used the required wheel torque (τ_{req}) as it is more practical. In this process, it is used the non-linear compensation of the longitudinal car forces, generating a wheel torque τ_{comp} which contains the friction due to the rolling and the aerodynamic drag:

$$\tau_{req} = m \cdot r \cdot a_{req} + \tau_{comp}$$
$$\tau_{comp} = r \cdot \left[m \cdot g \cdot C_r + \frac{1}{2} \rho \cdot C_x \cdot S \cdot v^2 \right]$$

Where r is the effective wheel radius, m is the vehicle mass, g the gravity constant, C_r is the rolling coefficient, ρ is the density of the air, C_x is the aerodynamic drag coefficient, S is the vehicle frontal surface area and v is the current longitudinal speed of the vehicle, all units in IS.

The wheel torque requirement (τ_{req}) is finally transformed to the throttle and brake signals through the car parameters present in the drivetrain (current gear, final drive ratio, engine maximum torque and maximum braking torque).

In the case of the throttle signal, τ_{req} is converted to the engine torque requirement, that is the torque that must provide the engine. For computing this it would be possible to use the complete torque curve of the engine, which is function of the engine speed, shown in Figure 18. Nevertheless, nowadays engines tend to have a wide work zone where the torque is practically constant, so the engine torque has been approximated as its maximum constant value at WOT (wide open throttle). Then, the throttle signal is the linear fraction required from this maximum torque value, between 0 and 1.

Brake signal is computed in a similar way, but using the value of maximum torque of the brakes. The maximum torque that a braking system delivers can be approximated with the vehicle mean maximum deceleration in the following way:

$$\tau_{max,braking} = (m \cdot g \cdot r) \cdot K$$

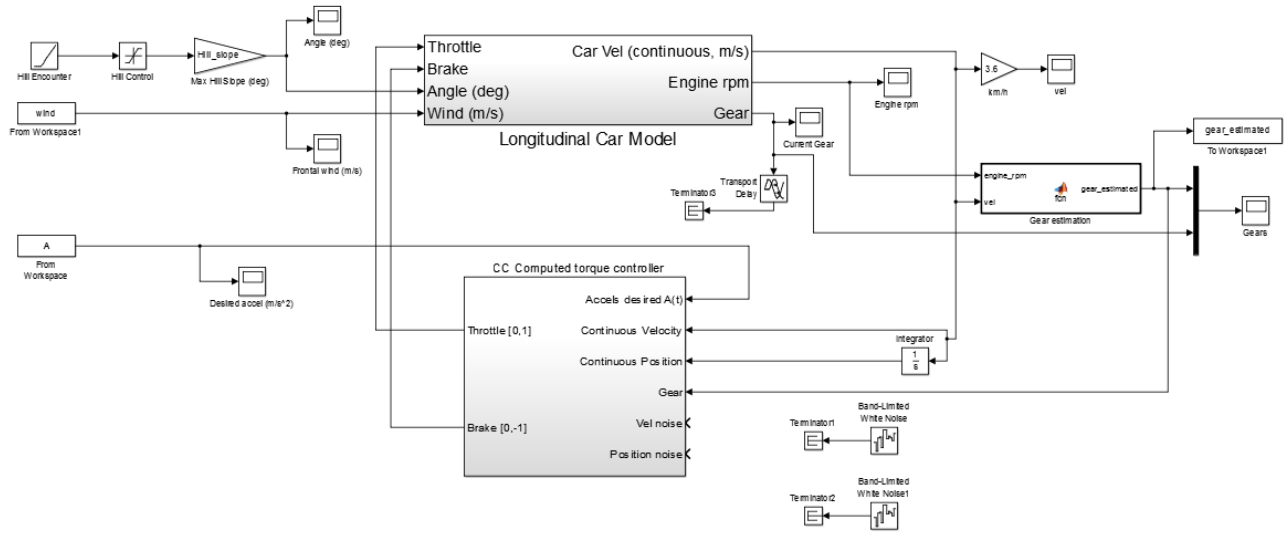
Where K is the mean g force value braking at maximum effort (if a standard car today can brake up to 0,8 times the gravity acceleration, K would be 0,8).

Then, brake signal is the linear fraction required from this maximum torque value, between 0 and -1. This signals are exclusive: they cannot have a different value from 0 at the same time.

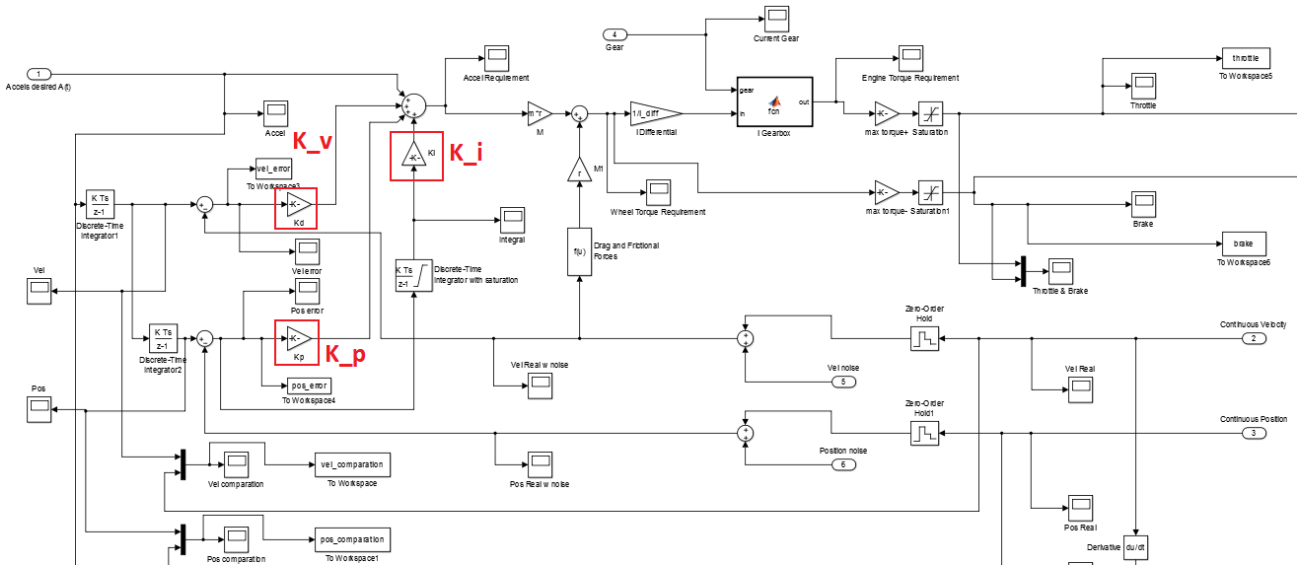
The controller has been tuned to follow rigorously the curves that are given to it in order to minimize the errors. However, it will be maybe too abrupt if sudden large signal changes are given. For this reason the behaviour is perfect to follow smooth pre-computed trajectories (so it will be so smooth like them too).

It has been designed trying to use the minimum number of parameters, and using only the ones that can be easily found or empirically estimated. In addition, it is very generic and can be easily used with any type of vehicle, even with other kind of technologies, like with electrical motors for example.

The resulting controller is very simple, extremely robust and also has a very low computational time, so it can be launched extremely fast and in any embedded system. It has been tested at 100 Hz, but operates correctly in much lower frequencies.



(a) Controller scheme



(b) Detail of the controller block

Figure 17: Longitudinal controller in Simulink

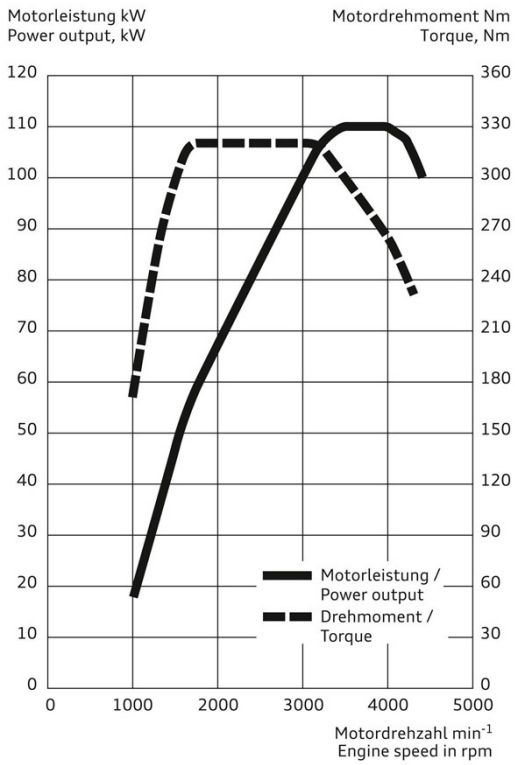


Figure 18: Wide open throttle (WOT) power and torque curves of a similar combustion engine

3.4.3 Kinematic car model

A car-like vehicle uses an Ackermann configuration that can be described by means of the following non-holonomic model, the bicycle model. Although each front wheel rotates a different angle in a real vehicle (Figure 19) when turning, the bicycle model simplifies it and works with a virtual front wheel located in the middle point. This simplification is totally valid, as the Ackermann mechanism has only one degree of freedom, that is the steering wheel turn.

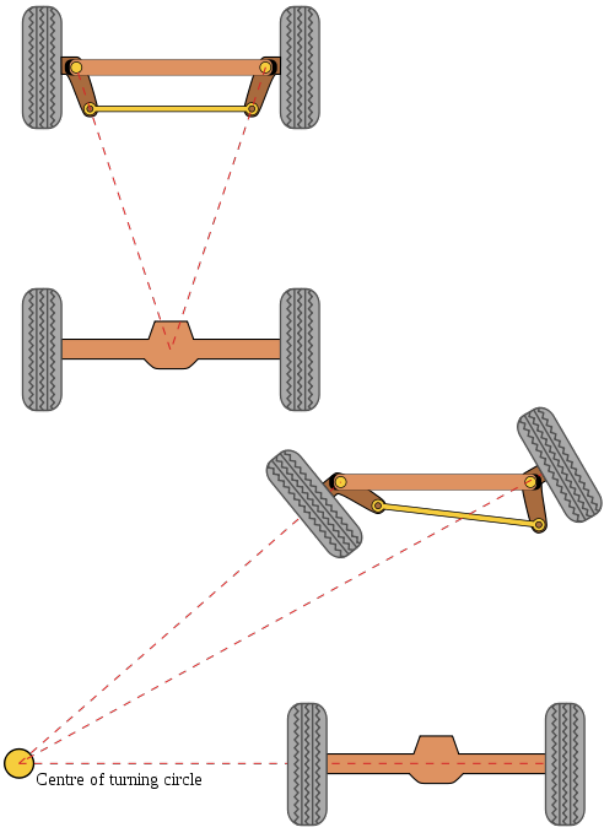


Figure 19: Ackermann steering in a car-like vehicle

The bicycle kinematic model for car-like vehicles is shown in Figure 20, and has the following equations:

$$\begin{cases} \dot{x} = v\cos\theta \\ \dot{y} = v\sin\theta \\ \dot{\theta} = \dfrac{v}{L}\tan\delta \end{cases}$$

where v is the vehicle's velocity, L is the inter-axle distance (or *wheelbase*) and δ is the front wheel steering angle. All angles follow the right hand rule, so the steering angle is defined as follows:

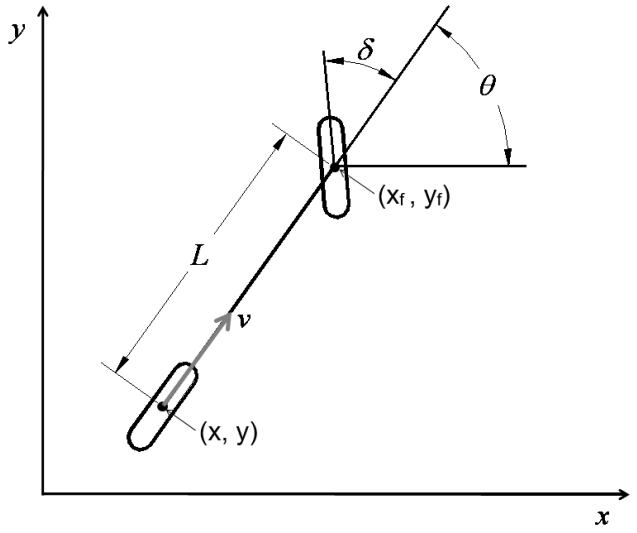


Figure 20: Vehicle's variables of the bicycle kinematic model

$$\delta \begin{cases} < 0 \rightarrow right\ \ turn \\ = 0 \rightarrow go\ \ straight \\ > 0 \rightarrow left\ \ turn \end{cases}$$

The state variables x, y and θ denote respectively the Cartesian coordinates and the orientation of the vehicle frame (the heading), fixed to the midpoint of the vehicle's rear axle. In addition, there are two more variables that gives important information: The radius of turn of the vehicle R and the curvature κ:

$$R = \frac{L}{\tan\delta}$$

$$\kappa = \frac{\tan\delta}{L} = \frac{1}{R}$$

Figure 21 shows the lateral kinematic model implemented in Simulink.

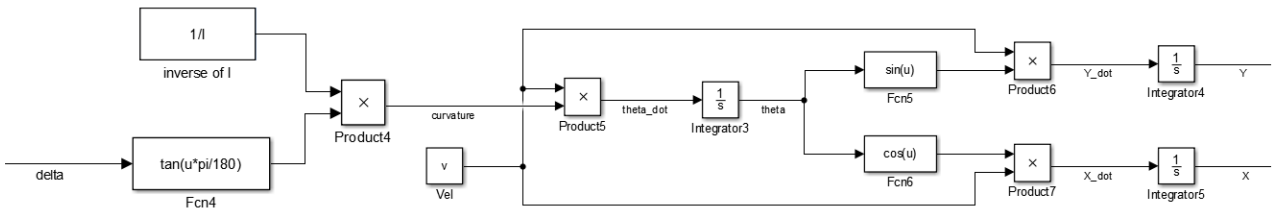


Figure 21: Simulink implementation of the model

3.4.4 Lateral dynamics

On the one hand, there exist the effect of tire slip. However, this behaviour is not relevant when driving at normal conditions and legal speeds. In fact, as it is wanted to preserve comfort in the vehicle as a human driver would do, the car will always be far from this situations. Lateral dynamic models can be very complex and they require knowing a lot of parameters and some of them can be difficult to obtain or estimate in real life (3-d suspension geometries, suspended masses, dampers, slip models...). For these reasons explicit lateral dynamic models have not been considered as necessary and they are not included.

On the other hand, there is one effect that affects lateral dynamics that must be considered: the steering wheel cannot immediately match the commanded angle. Rather, there is a delay, dependent on the inertia of the steering column, the servo system, and communication delays. Experimentally, this system can be approximated through a first order model:

$$\dot{\delta}(t) = \frac{1}{\tau}(\delta_c(t) - \delta(t))$$

The value for the time constant has been set to $\tau = 0,4\ s$ for simulation, which it seems to be a conservative value. This parameter could not be approximated with real data from the real car of the IRI due to external delays to this project, but still this considered value is quite accurate and conservative.

3.4.5 Lateral controller: Stanley controller

The used lateral controller belongs to the geometric path tracking algorithms family, because of their good performance (in fact, similar to the other methods), simplicity and their robustness. Moreover, this kind of control is more robust and adaptable to any vehicle, even if having few data access of the used real plant.

The chosen controller is the Stanley controller, as it offers the best performance, robustness, an intuitive tuning and also takes into account some vehicle dynamics.

The Stanley controller is the path tracking approach used by Stanford University's autonomous vehicle entry in the DARPA Grand Challenge, that was called *Stanley* [5] [26]. The Stanley method is a non-linear feedback function of the cross track error e_{lat}, measured from the center of the front axle to the nearest path point (x_p, y_p), for which it presents exponential convergence.

The fact of using the steered front wheels allows for an intuitive control law. The first term of the control law simply keeps the wheels aligned with the given path by setting the steering angle δ equal to the heading error:

$$\theta_e = \theta - \theta_{traj}$$

where θ is the heading of the vehicle and θ_{path} is the heading of the path at (x_p, y_p). When e_{lat} is different from zero, the second term adjusts δ such that the intended trajectory converges exponentially to the desired path with time constant K. This basic steering control law is given as:

$$\delta(t) = \theta_e(t) + \arctan\left(\frac{K \cdot e_{lat}(t)}{v(t)}\right)$$

where K is a gain parameter and v is the longitudinal car velocity. It can be seen that the desired effect is achieved with this control law: As e_{lat} increases, the wheels are steered further towards the path. Figure 22 illustrates the geometric relationship of the control parameters.

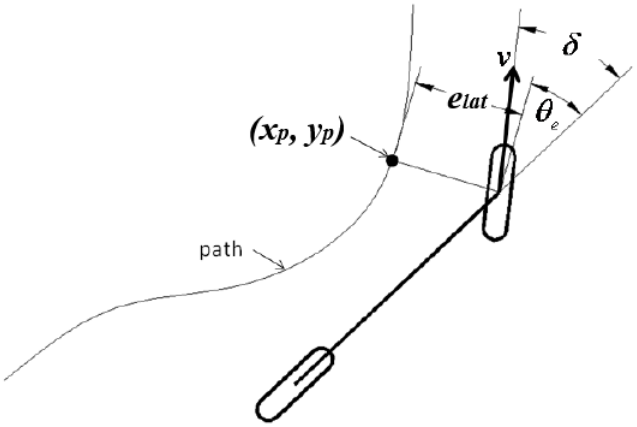


Figure 22: Stanley controller geometry

Using the previous control law, the location of the front wheels is actively controlled, but there are some effects related to vehicle dynamics that must be taken into account in order to guarantee good performance and stability at high speeds. Firstly, the tires act as dampers, providing reaction forces to sideways velocities. At low speeds, this stabilizes the yaw dynamics, however the magnitude of this reaction is inversely proportional to speed. As speed increases, the damping effect diminishes, creating a need for active damping. Thus, a new term that takes into account the yaw rate of the car ($\dot{\theta}_{meas}$) and the trajectory rate ($\dot{\theta}_{traj}$) is added to the steering control law: $K_{d,yaw}(\dot{\theta}_{meas} - \dot{\theta}_{traj})$. Moreover, as the controller commands a steering servo, this actuation system can cause instability due to its dynamics. In order to prevent this, a damping term of the steering wheel response is added to the control law: $K_{d,steer}(\delta_{meas}(i-1) - \delta_{meas}(i))$, where $\delta_{meas}(i)$ is the discrete time measurement of the steering angle at current time step i. The third modification is used for driving at low speeds and prevents the $\frac{e_{lat}}{v(t)}$ term to be over-sensitive to noise on e_{lat} adding a gain K_{soft} to the denominator.

Finally, the complete steering wheel control law is given as:

$$\delta(t) = \theta_e(t) + \arctan\left(\frac{K \cdot e_{lat}(t)}{K_{soft} + v(t)}\right) + K_{d,yaw}(\dot{\theta}_{meas} - \dot{\theta}_{traj}) + K_{d,steer}(\delta_{meas}(i-1) - \delta_{meas}(i))$$

Is important to remark that δ is saturated at $\pm\delta_{max}$, which corresponds to the minimum radius of turn that a vehicle can perform.

The Figure 23 shows the Simulink scheme.

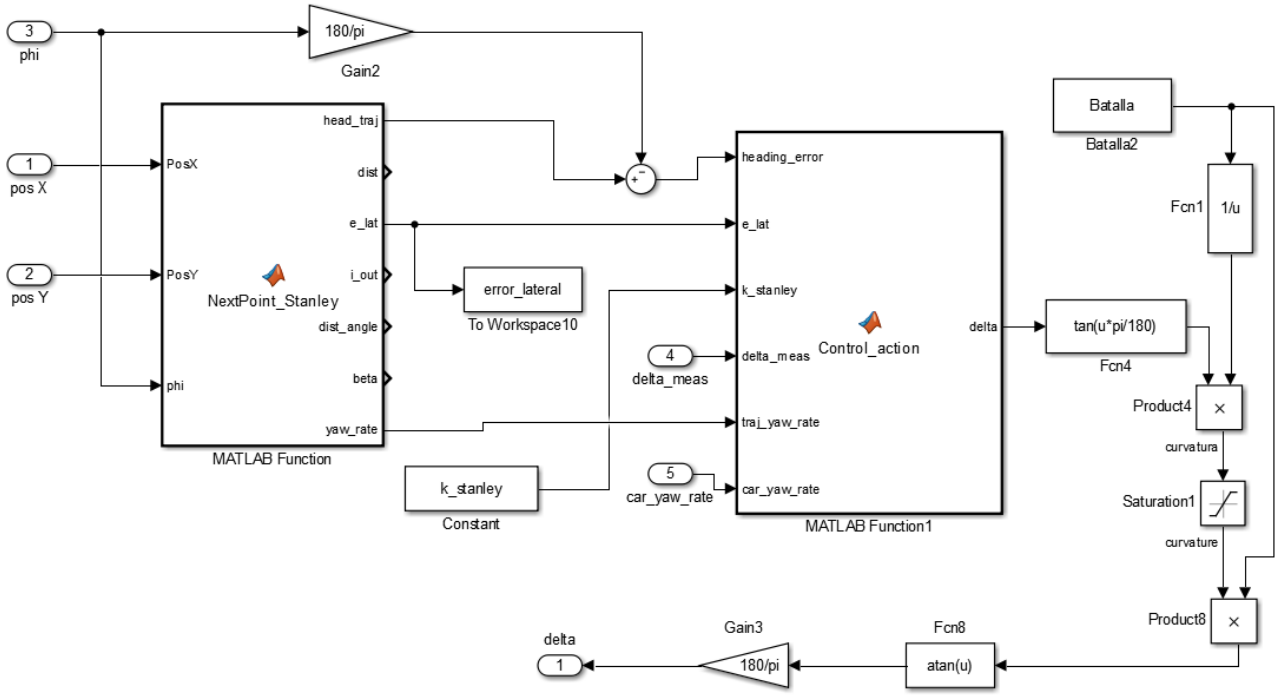


Figure 23: Simulink scheme of the Stanley controller

This lateral controller receives a desired trajectory as a 2-D array of points. For each time instant, the function *NextPoint_Stanley* finds (x_p, y_p). Then it computes the lateral error e_{lat} by terms of geometric relations, the heading of the trajectory θ_{traj} and its yaw rate $\dot{\theta}_{traj}$. Finally, using this data and also the measurements of the current steering angle $\delta_{meas}(i)$ and vehicle yaw rate $\dot{\theta}_{meas}$, the control action is computed and sent to the steering servo.

As the path is discrete, it is important to remark that the resolution affects to the good behaviour of the controller. Having low resolution (points very far) will imply that when (x_p, y_p) changes, the computed trajectory parameters will have big variations that will pass through the controller, so the performance will be penalised. However, this is not a problem because the path generation algorithm seen in 3.5 can provide a very good resolution without issues.

3.5 Path generation

This section explains all the algorithms related with the path generation. First of all, section 3.5.1 details the used approach for the path computation, the G^2-splines. This type of geometric splines was intended to be used as a motion planning primitive for self-driving vehicles, as it has very nice properties, but some of its parameters needed a numerical optimization to be adjusted [27]. Nevertheless, this approach has not been implemented in motion planners. This thesis presents a novel method, shown in section 3.5.2, for finding all the needed parameter values in an easy way and with excellent results, and uses these splines as a path generation for the motion planning.

Section 3.5.3 shows the great potential and examples of paths computed by the G^2-splines algorithm, not only for the AKRP planner. Finally, in section 3.5.4 it is explained the used endpoint generation algorithm, with examples of the different manoeuvres.

3.5.1 General 5th order G^2-splines

In order to build a general path $\mathbf{p}(u)$, understood as a path with arbitrary defined starting and ending states (or endpoints), the quintic G^2-*splines* offers all this needed flexibility with also very good properties. They are geometric polynomials of 5th order and have second order geometric continuity (G^2), so the curvature κ is continuous. Figure 24 exhibits the needed information to generate a path: only the initial and final endpoint states.

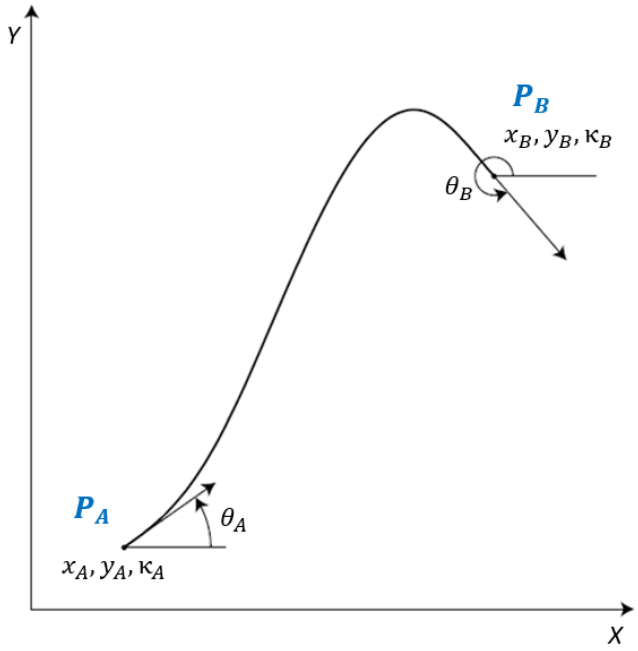


Figure 24: Example of a G^2-spline path connecting P_A with P_B and satisfying endpoint interpolating conditions

Considering the initial state $\mathbf{X}_A = [x_A, y_A, \theta_A, \kappa_A]$, as well as the ending state $\mathbf{X}_B = [x_B, y_B, \theta_B, \kappa_B]$, the equations to define this splines are the following ones:

$$\mathbf{p}(u) = \begin{bmatrix} x(u) \\ y(u) \end{bmatrix} := \begin{bmatrix} x_0 + x_1 u + x_2 u^2 + x_3 u^3 + x_4 u^4 + x_5 u^5 \\ y_0 + y_1 u + y_2 u^2 + y_3 u^3 + y_4 u^4 + y_5 u^5 \end{bmatrix}$$

where $u \in [0, 1]$ and:

$$\begin{aligned}
x_0 &= x_A \\
x_1 &= \eta_1 \cos\theta_A \\
x_2 &= \frac{1}{2}(\eta_3 \cos\theta_A - \eta_1^2 \kappa_A \sin\theta_A) \\
x_3 &= 10(x_B - x_A) - (6\eta_1 + \frac{3}{2}\eta_3)\cos\theta_A \\
&-(4\eta_2 - \frac{1}{2}\eta_4)\cos\theta_B + \frac{3}{2}\eta_1^2\kappa_A \sin\theta_A - \frac{1}{2}\eta_2^2\kappa_B \sin\theta_B \\
x_4 &= -15(x_B - x_A) + (8\eta_1 + \frac{3}{2}\eta_3)\cos\theta_A \\
&+(7\eta_2 - \eta_4)\cos\theta_B - \frac{3}{2}\eta_1^2\kappa_A \sin\theta_A + \eta_2^2\kappa_B \sin\theta_B \\
x_5 &= 6(x_B - x_A) - (3\eta_1 + \frac{1}{2}\eta_3)\cos\theta_A \\
&-(3\eta_2 - \frac{1}{2}\eta_4)\cos\theta_B + \frac{1}{2}\eta_1^2\kappa_A \sin\theta_A - \frac{1}{2}\eta_2^2\kappa_B \sin\theta_B
\end{aligned}$$

$$\begin{aligned}
y_0 &= y_A \\
y_1 &= \eta_1 \sin\theta_A \\
y_2 &= \frac{1}{2}(\eta_3 \sin\theta_A + \eta_1^2 \kappa_A \cos\theta_A) \\
y_3 &= 10(y_B - y_A) - (6\eta_1 + \frac{3}{2}\eta_3)\sin\theta_A \\
&-(4\eta_2 - \frac{1}{2}\eta_4)\sin\theta_B - \frac{3}{2}\eta_1^2\kappa_A \cos\theta_A + \frac{1}{2}\eta_2^2\kappa_B \cos\theta_B \\
y_4 &= -15(y_B - y_A) + (8\eta_1 + \frac{3}{2}\eta_3)\sin\theta_A \\
&+(7\eta_2 - \eta_4)\sin\theta_B + \frac{3}{2}\eta_1^2\kappa_A \cos\theta_A - \eta_2^2\kappa_B \cos\theta_B \\
y_5 &= 6(y_B - y_A) - (3\eta_1 + \frac{1}{2}\eta_3)\sin\theta_A \\
&-(3\eta_2 - \frac{1}{2}\eta_4)\sin\theta_B - \frac{1}{2}\eta_1^2\kappa_A \cos\theta_A + \frac{1}{2}\eta_2^2\kappa_B \cos\theta_B
\end{aligned}$$

The resulting spline depends on some parameters that affect its shape, all included in the parameter vector $\boldsymbol{\eta}$:

$$\boldsymbol{\eta} = [\eta_1, \eta_2, \eta_3, \eta_4]$$

An important characteristic of the resulting spline is its curvature $\kappa(u)$, which can be computed using the following equation:

$$\kappa(u) = \frac{\dot{x}(u)\ddot{y}(u) - \ddot{x}(u)\dot{y}(u)}{(\dot{x}(u)^2 + \dot{y}(u)^2)^{3/2}}$$

3.5.2 G^2-splines optimization algorithm

The $\eta_1, \eta_2, \eta_3, \eta_4$ are free parameters that only affect to the spline shape. In order to generate any kind of path it is mandatory to find a certain parameter vector $\boldsymbol{\eta}$.

Until the present, $\boldsymbol{\eta}$ was obtained through a numerical optimization. This thesis presents an easier method for obtaining a $\boldsymbol{\eta}$ value valid for any situation without a numerical optimization process, also discussing its behaviour.

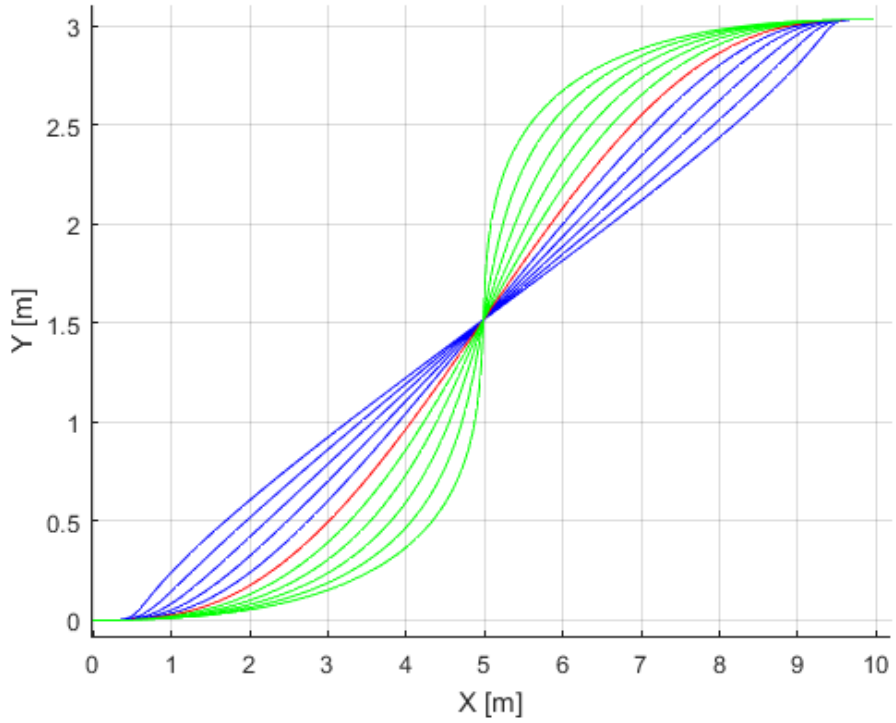
In order to have a symmetrical behaviour, $\boldsymbol{\eta}$ parameters need to be $\eta_1 = \eta_2$ and $\eta_3 = -\eta_4$ so only 2 parameters are needed to be tuned: $\eta_{1\_2} = \eta_1 = \eta_2$ and $\eta_{3\_4} = \eta_3 = -\eta_4$.

$\eta_{3\_4} \in (-\infty, +\infty)$, and it affects to the curvature changes. After several tests, it can be concluded that the best value for this parameter is 0 in all the cases. In Figure 25, different η_{3_4} values are tested performing a lane change manoeuvre. As it can be seen, if $\eta_{3\_4} > 0$ (green trajectories) the curvature changes are concentrated in the center of the trajectory, whereas $\eta_{3\_4} < 0$ (blue trajectories) concentrate it on the extreme initial and final points. The red trajectory represents $\eta_{3\_4} = 0$, and it is the smoothest and more balanced trajectory, so it is the best option.

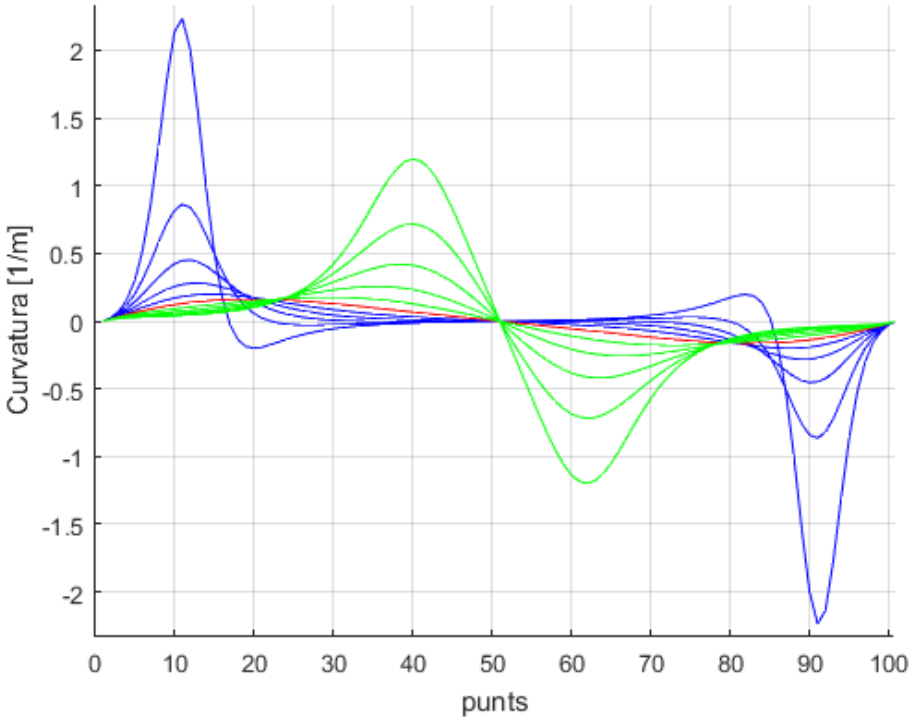
On the other hand, $\eta_{1\_2} \in (0, +\infty)$, and it generally forces θ and κ to stay close to the initial and final values. With this parameter is not trivial to find a good value to perform smooth trajectories in all situations. However, after a lot of testing, it has been found that the smoothest trajectory is reached when η_{1_2} is close to the trajectory length, in meters. For this reason, a simple iterative method is performed in order to converge into the best possible trajectory: Firstly, the Cartesian distance between the initial and ending points is taken as the initialization value for η_{1_2}. Then, the trajectory is computed and the total length of this new trajectory will be used as the new η_{1_2} to compute the next iteration. After a few iterations, usually 3 or 4 depending on the case, all the computed trajectories converge into the smoothest one. In the example of Figure 26, the smoothest path is to follow a circular trajectory, but without curvature in the initial and final points.

Blue curve is the first iteration, and it is pretty smooth, but it tries to 'cut' the circular trajectory. This is translated into having more variations in the curvature (Figure 26b). After 2 more iterations the trajectory converge into the circular one, presenting a curvature shape almost constant in the middle zone.

It can be concluded that this optimization process has the same results as the proposed numerical optimization in [27], which minimizes the curvature variability $\left(\min_{\boldsymbol{\eta}} \frac{d\kappa}{du}\right)$.

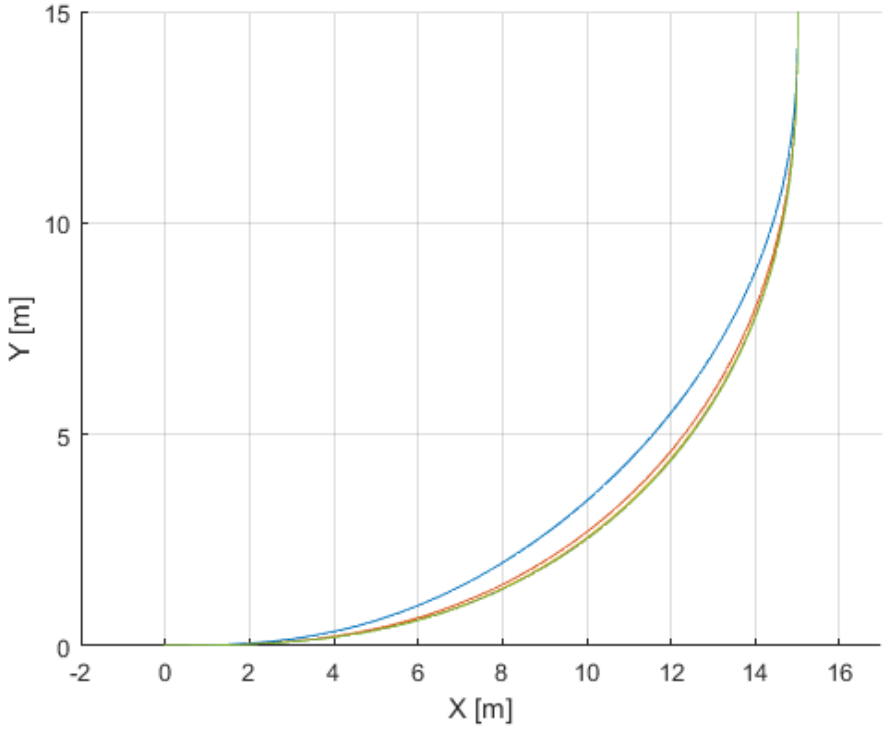


(a) Path X-Y position, in $[m]$

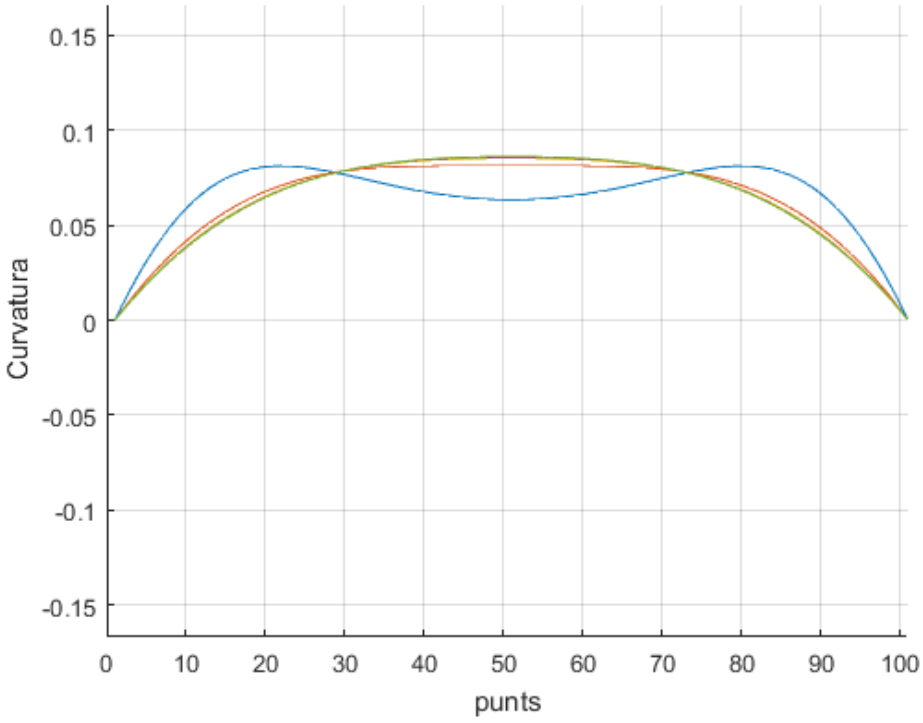


(b) Path curvature, in $[1/m]$

Figure 25: G^2-splines η_{3_4} optimization tests in a lane change manoeuvre



(a) Trajectory X-Y position, in $[m]$



(b) Trajectory curvature along the path, in $[1/m]$

Figure 26: G^2-splines η_{1_2} optimization tests in a circular trajectory

3.5.3 G^2-splines algorithm features and examples

The G^2-splines generation algorithm is used by the AKRP to generate the candidate paths. But this algorithm can be used to generate for example the center lane paths in a very easy way. This is due to the easy and intuitive conditions that it uses: only the basic geometric state for the initial and the final point, as shown in Figure 24. In addition, it also computes the curvature along the path, κ, which is very useful for the AKRP.

As it is a spline, is also very easy to generate a path with any required resolution (distance between consecutive points).

In addition, results show that it can approximate any kind of path shape preserving a smooth curvature continuity and minimizing curvature variability, and this implies kinematic feasibility for a vehicle following it: circular segments, straight lines, clothoids, complete lane changes...

Figure 27 shows a comparison between 1 and 5 iterations of the optimization algorithm in the center lane paths of a complex intersection. Note how the paths are generated only with the geometrical information of the initial and final points. This make this approach very interesting for motion planning applications, as these splines can follow the road shape very easy and intuitively. The execution time of all these paths in MATLAB, running in a 4 GHz CPU, is 20 ms. This time is, at least, 10 times bigger than a compiled C++ code, what it would mean generating all this paths in only 2 ms in the worst case.

Figure 28 is also a center path lane generation in a real roundabout, computed from some waypoints.

However, when generating a path, it has to be present that the algorithm performs an interpolation. If the endpoint is too far away from the initial point, the resulting path may approximate poorly the desired road shape. But this is very easy to solve, simply reducing the endpoint distance.

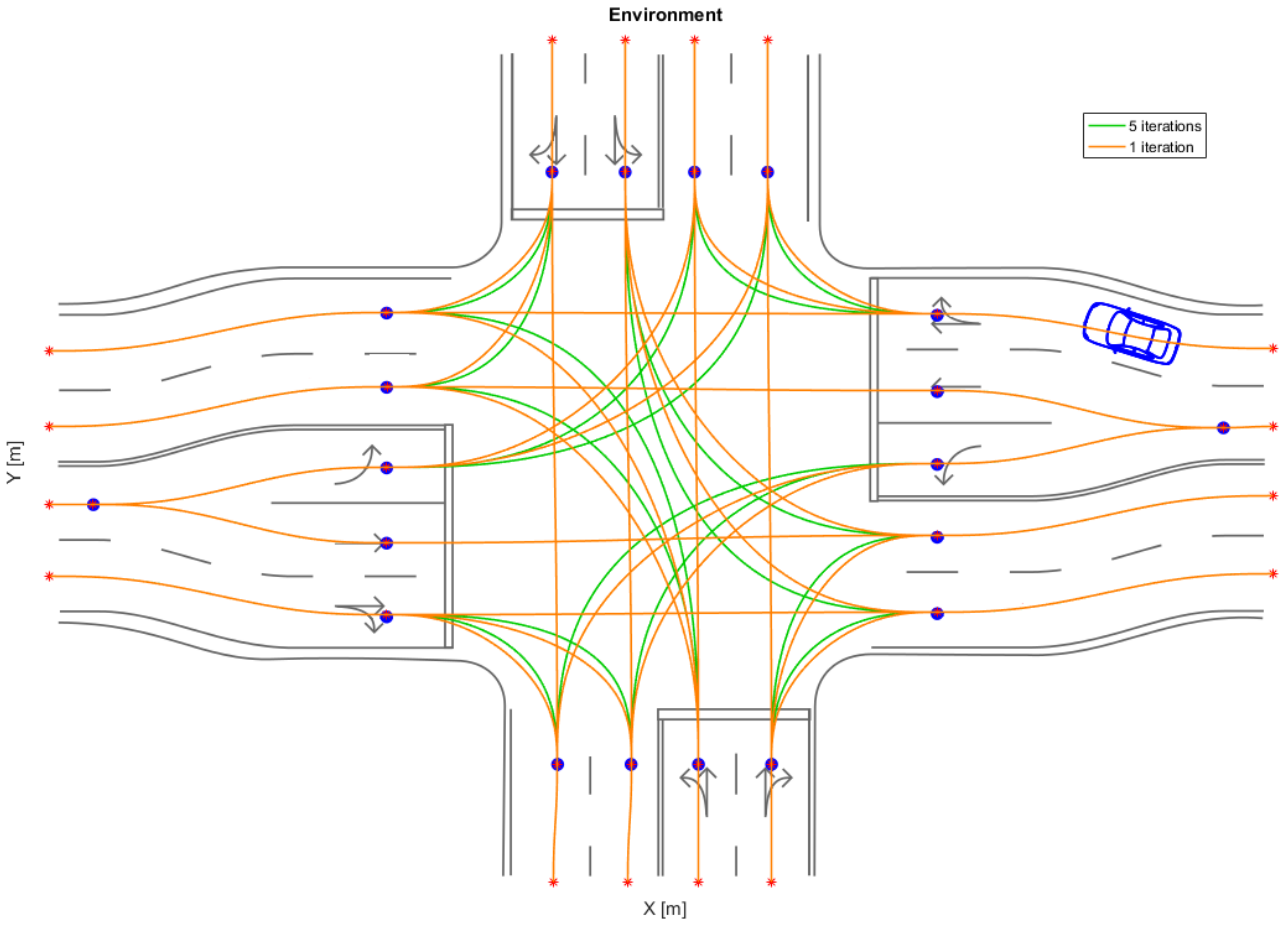


Figure 27: Center lane paths generated with the G^2-splines algorithm in a complex intersection, with different iterations. The only used information is the endpoints x, y and θ, as all the endpoints curvatures are considered as 0

Figure 28: Center lane paths generated with the G^2-splines algorithm in a real roundabout. The only used information is the endpoints x, y, θ and κ

3.5.4 Endpoints generation algorithm and manoeuvring

In order to calculate the paths with the G^2-splines it is needed to discretize the space first and generate the ending points.

Each ending point is composed by an state $\mathbf{X} = [x, y, \theta, \kappa]$. Expressing the function of a center lane path $\mathbf{L}$:

$$\mathbf{L}(s) = [x_L(s), y_L(s), \theta_L(s), \kappa_L(s)]$$

Where s is the station (or longitudinal offset). An arbitrary endpoint $\mathbf{P}$ can be defined, at a given station s and lateral offset from the center lane d:

$$\mathbf{P}(s, d) = [x_P(s, d), y_P(s, d), \theta_P(s, d), \kappa_P(s, d)]$$

and:

$$\begin{aligned} x_p(s, d) &= x_L(s) + d \cdot \cos(\theta_L(s) + \pi/2) \\ y_p(s, d) &= y_L(s) + d \cdot \sin(\theta_L(s) + \pi/2) \\ \theta_P(s, d) &= \theta_L(s) \\ \kappa_P(s, d) &= \left(\kappa_L(s)^{-1} - d\right)^{-1} \end{aligned}$$

Depending on the manoeuvre, the endpoints are generated slightly different, but the main issue that must be taken into account is the maximum interpolation distance between an initial point and an endpoint of a path generation. This is because if this distance is too high, the generated path can differ from the lane shape. This parameter can be seen in Figure 30.

Empirically, this parameter has been adjusted to be function of velocity, related with the needed distance for stopping the vehicle at full braking, $dist_{full\_braking}$. Approximating $dist_{full\_braking}$ with an uniformly accelerated linear motion with $a_{braking}$ acceleration:

$$dist_{full\_braking} = \frac{1}{2} \frac{v_0^2}{|a_{braking}|}$$

$$dist_{interpolation} = \lceil dist_{full\_braking} + C \rceil$$

C is a constant value, used to shift the function. Empirically, C has been set to 20, and $a_{braking}$ to -7 m/s^2.

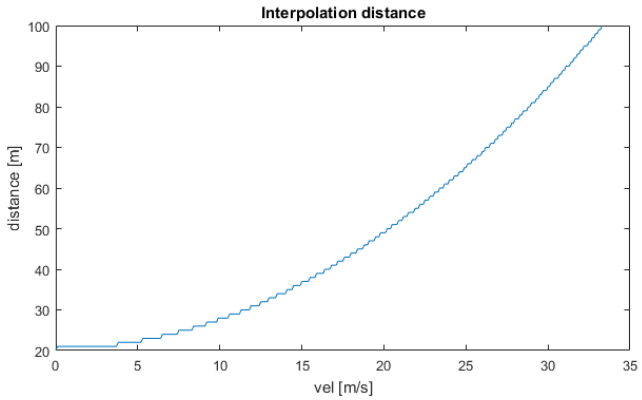


Figure 29: Interpolation distance function of velocity, with $C = 20\ m$ and $a_{braking} = -7\ m/s^2$

3.5.4.1 Lane following

This is the basic manoeuvre. In the case without obstacles, the sampling output is shown in Figure 30.

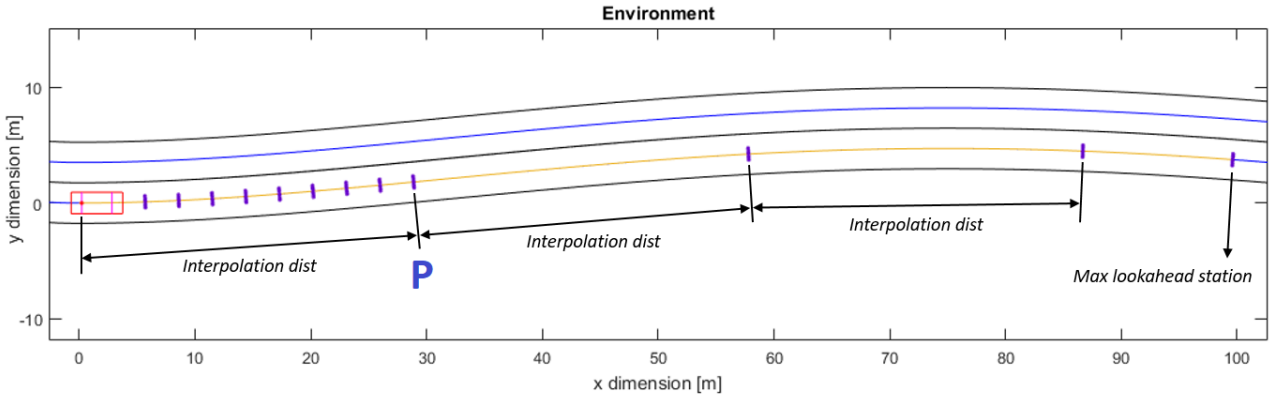


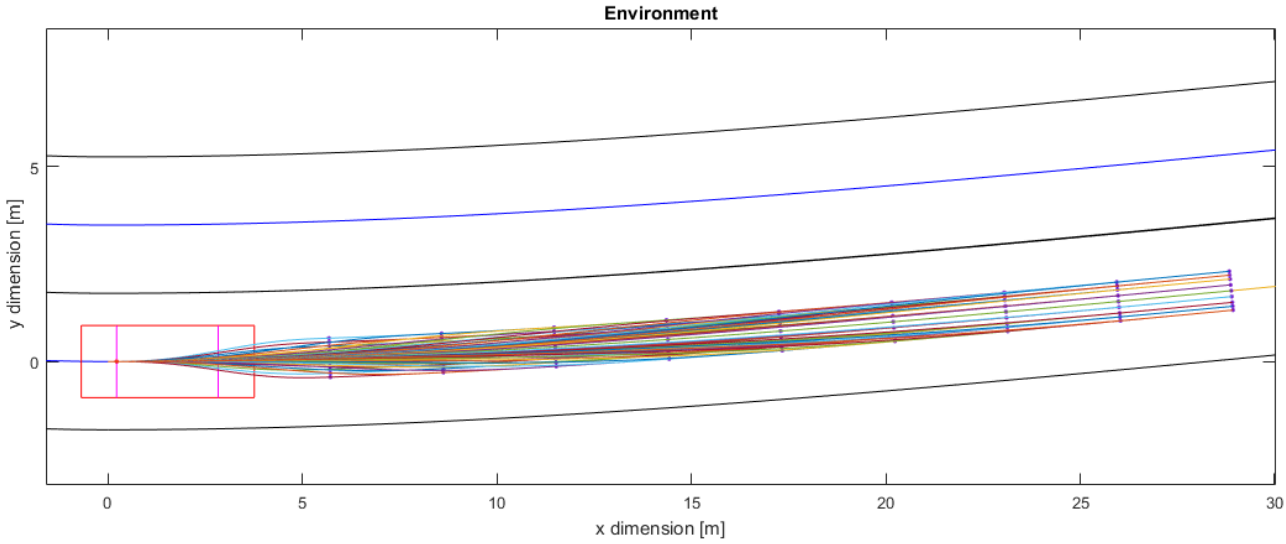
Figure 30: Endpoint generation dimensions in a lane following example

Up to the first interpolation distance (point P) there is a finer discretization. This has been used to test the reactiveness of the sampling. First of all, paths are generated from the current host vehicle state to each set of different endpoints situated in the same station, but with different lateral offsets. Then, the path is extended up to P station if needed, maintaining the current offset. This generation can be shown in Figure 31a. The other option tested is generating a path directly from the current host vehicle state to P station, shown in Figure 31b.

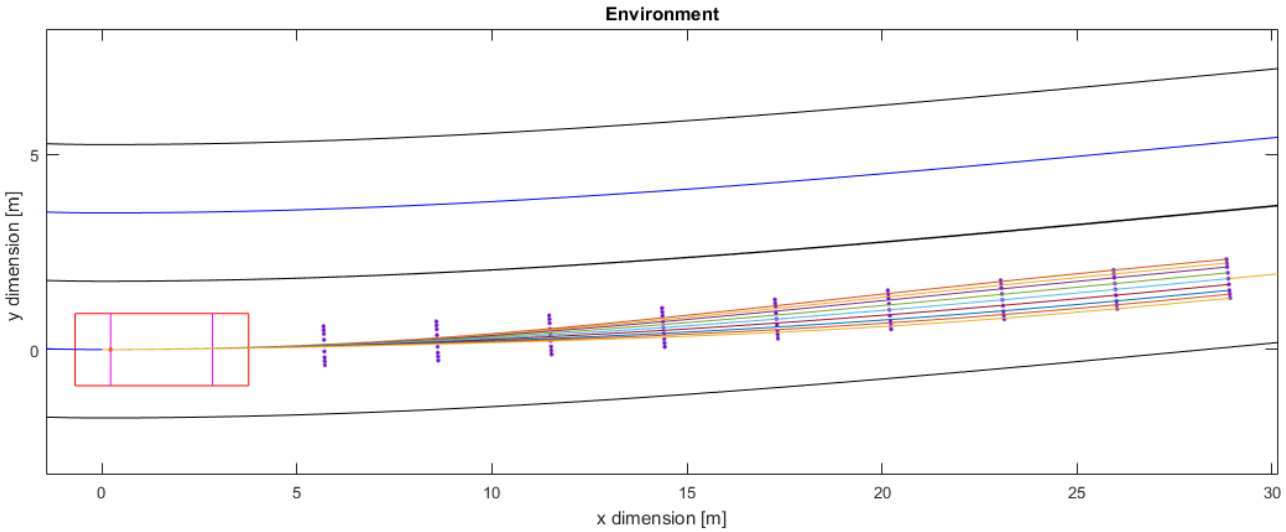
Once in the first interpolation station P, each candidate path is extended (with a new computation of the G^2-splines algorithm) up to the maximum lookahead station, in station increments equal to the interpolation distance and maintaining its lateral offset (Figure 31c).

The lateral offset is maintained up to the end station. This is because of the consideration of static obstacles in the sampling, that is explained below. Performing a lot of lateral offsets combinations between stations can lead to an exponential growing of the candidate path set, resulting in being impossible to compute a planning iteration in a reasonable time.

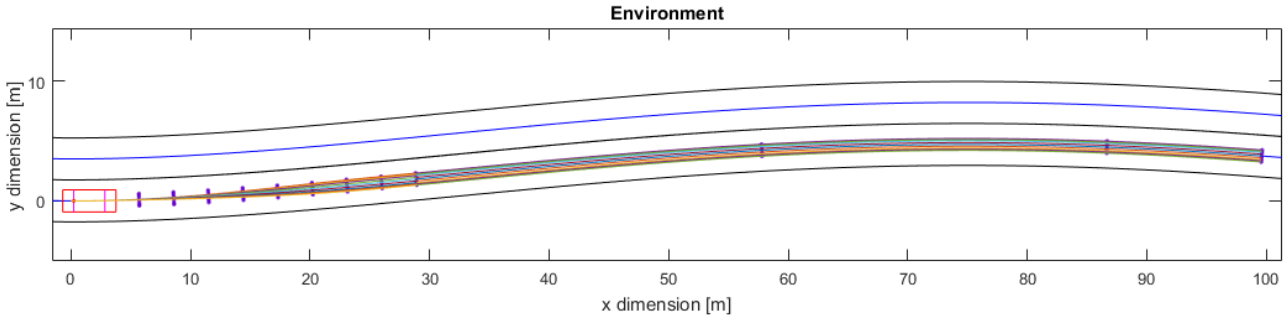
As it can be seen easily between figures 31a and 31b, when applying the static costs to the candidate paths, the optimal paths for each lateral offset will be always (in this case without static obstacles) the ones of Figure 31b. In the first option, there will be 9 stations x 9 lateral offsets, so 81 candidate paths. Instead, the second option uses only 9 candidate paths, corresponding to the number of lateral offsets.



(a) Path generation using a finer discretization up to the first interpolation distance (81 candidate paths)



(b) Used path generation up to the first interpolation distance (9 candidate paths)



(c) Used path generation up to the maximum lookahead distance (9 candidate paths)

Figure 31: Path generation following a lane without static obstacles

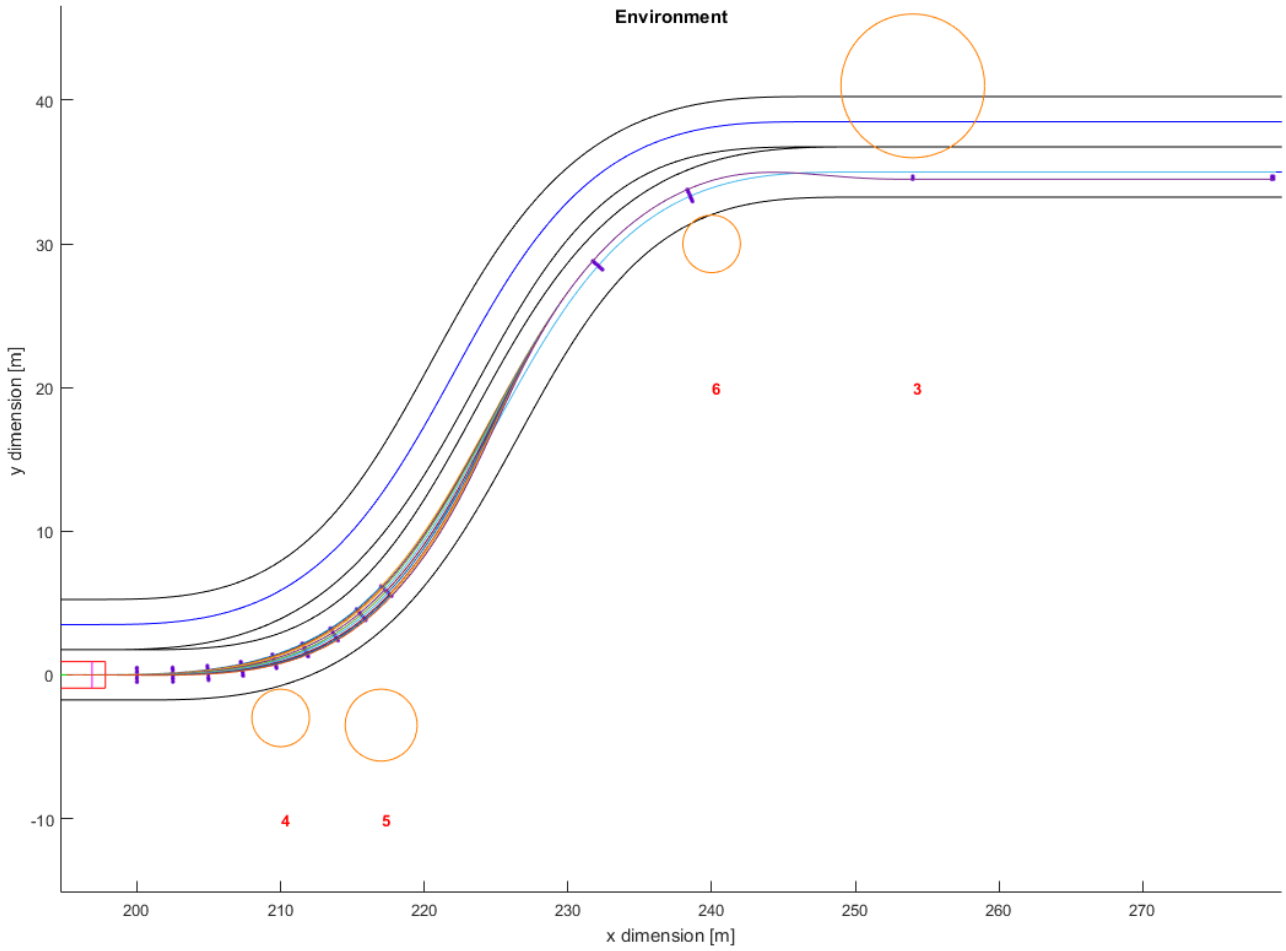
In presence of static obstacles, the endpoint generation algorithm takes them into account in order to discretize the lane smartly. Some examples are shown in Figure 32.

The algorithm works the same way as the case without static obstacles, but when a static obstacle is inside an interpolation interval, a fast collision checking is performed. In the same station as the obstacle, a provisional set of endpoints are generated (with different lateral offsets) and for each one a collision test is performed. If at least one endpoint collides, then this station is finally chosen only with its feasible endpoints, and the process is repeated looking ahead up to arriving to the end (the maximum lookahead station). In addition, in order to reduce the number of combinations, only one endpoint is considered when encountering an obstruction, and it is the farthest endpoint from the obstacle (that corresponds to the minimum cost one). Moreover, from the last obstruction up to the end, only the considered lateral offset is maintained; and the same behaviour from the first encountered obstruction to P.

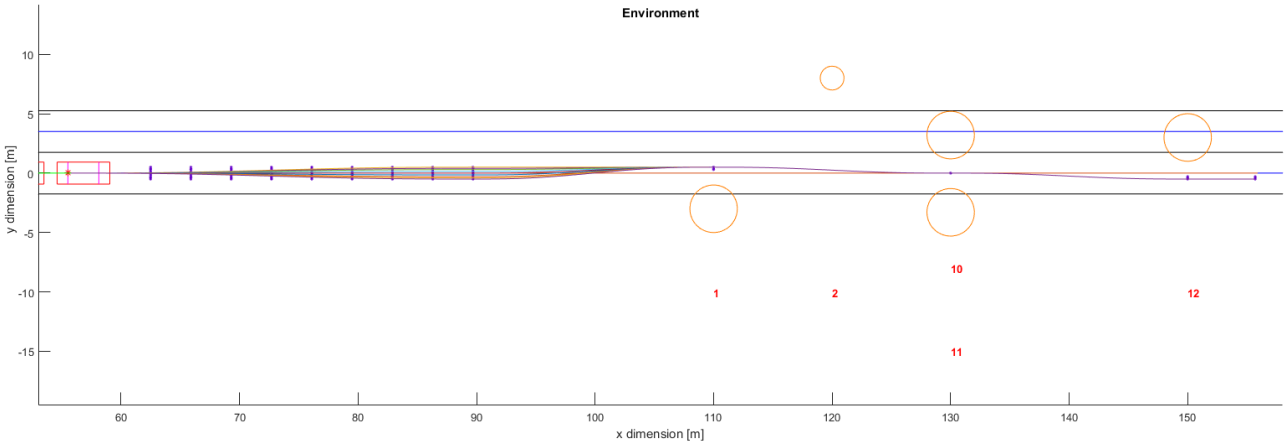
In the example of Figure 32a, there are 2 static obstacles in the first interpolation distance (up to P), but when checking possible collisions, they do not affect, so they are not taken into account. The next interpolation distance has no obstacles, but the third one encounters obstacle number 6, so its station is considered. In this case, only 8 lateral offsets are feasible. Then, starting in the station of obstacle 6, a new obstacle is encountered (obstacle 3), and in its station only the 3 lateral offsets more to the right are feasible. Finally, from the last obstruction, only the feasible lateral offsets are considered.

If there is no obstruction up to P, then there will be n_{paths} paths, equal to the number of desired lateral offsets. Then, if some obstruction is encountered ahead, paths will be extended starting in each endpoint of P station up to the next endpoint station, and from there the n_{paths} paths will be common.

In the case when the first interpolation distance is obstructed, P is the obstruction station and n_{paths} will be the number of feasible endpoints in P.



(a) Path generation example (9 candidate paths)



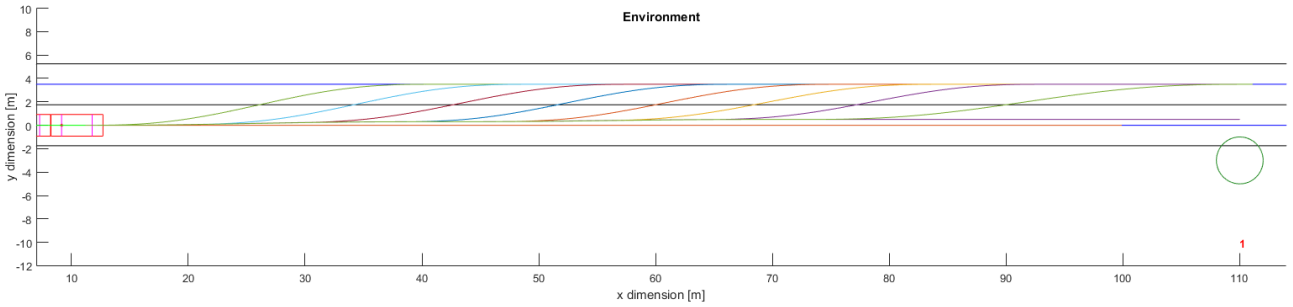
(b) Path generation example (9 candidate paths)

Figure 32: Path generation following a lane with static obstacles

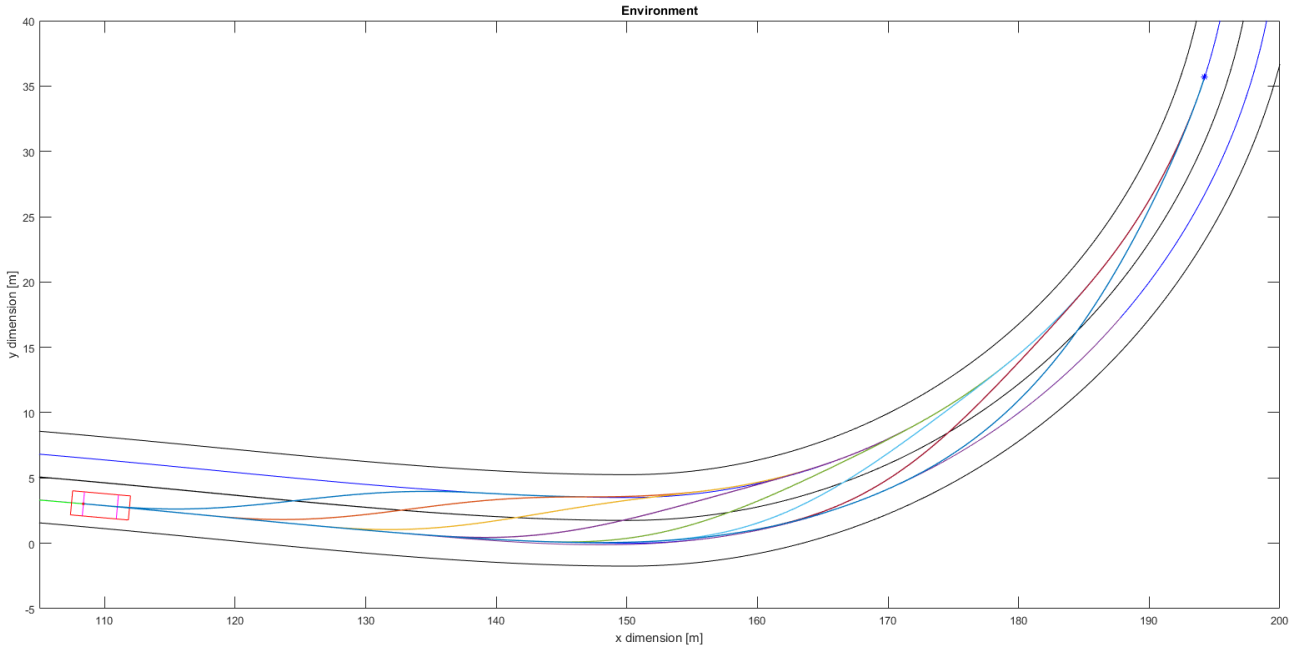
3.5.4.2 Lane changing

The lane changing manoeuvre is based on performing 2 lane following path generations: one in the current lane of the host vehicle and the other in the target lane. In this way, both lanes are collision-checked. Then, $n_{changes}$ candidate paths are generated from the best path of the current lane to the best path of the other lane, equally spaced. This manoeuvre is seen in Figure 33.

Note that static obstacles are already taken into account in the lane change paths (Figure 33a).



(a) Lane change with $n_{changes} = 8$



(b) Lane change with $n_{changes} = 8$

Figure 33: Lane change path generation examples

3.5.4.3 Obstacle passing

This manoeuvre is done when the current lane is blocked by a static obstacle, and the only way to advance is to pass the obstacle through an adjacent lane. It is composed by 2 lane change manoeuvres, one for going to the target lane, and other to return to the starting lane.

Regarding the Figure 12, the whole manoeuvre must be feasible before starting it, so if the current lane is totally blocked ahead the second lane change will not finish, so the manoeuvre will not be feasible.

Some examples can be seen in Figure 44.

3.5.4.4 Overtaking

The overtaking manoeuvre consists on passing a slower vehicle, a dynamic obstacle. Is the same manoeuvre as the obstacle passing, but in this case, the returning lane change must be predicted ahead with the movement of the dynamic obstacle being passed.

Figure 34 shows an example manoeuvre. It is predicted until the lookahead horizon, denoted as the blue point. Then several changing lane paths are generated ($n_{changes}$), to and from the other lane in the half of the horizon. The half-paths are then combined to reach the maximum lookahead station, so there are $n^2_{changes}$ candidate paths.

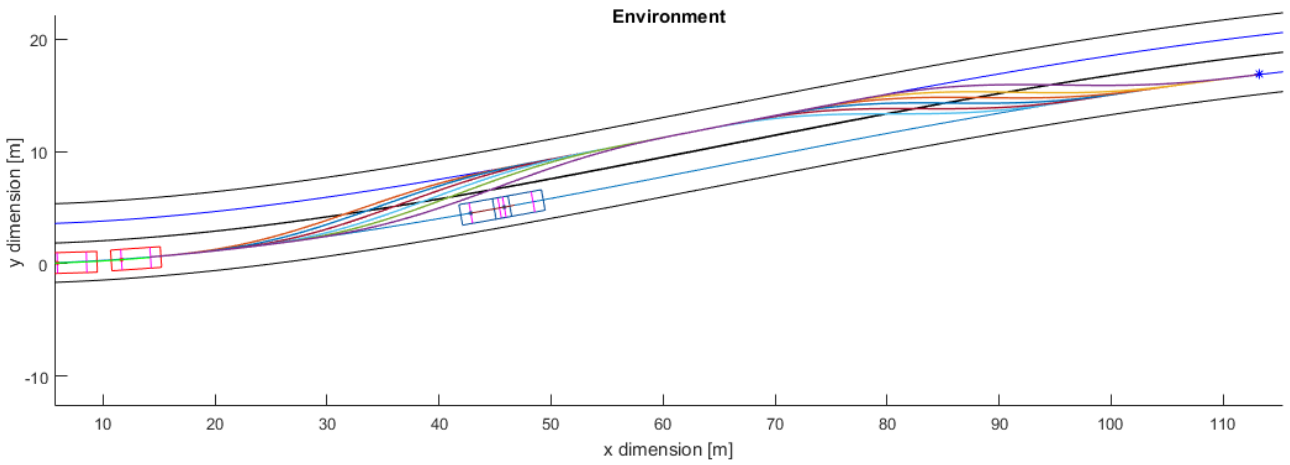


Figure 34: Generation of a set of candidate manoeuvres for a dynamic obstacle avoidance

3.6 Velocity profiles generation

To generate the velocity profiles, AKRP uses a 3rd grade polynomial spline in velocity, $v(t)$. This spline gives a smooth transition between two different velocities, and it has a parabolic acceleration profile fixing the maximum (or minimum) acceleration. This smooth and comfortable transition between velocities is caused by the derivative of the acceleration, the jerk, which is continuous.

In addition, the possibility of adjusting directly this parameter of maximum desired acceleration in the trajectory is very useful for this application. It is directly related with the time needed for the trajectory and also with the comfort of the passengers and its feasibility.

The core of the velocity profile generation algorithm is based in the case without initial acceleration, consisting in starting from initial constant velocity ($a_0 = 0$) to a different final constant velocity. In this situation, the acceleration parabolic profile is symmetric. In the next sections it is explained the rest of the equations that can start at any different initial acceleration so the whole algorithm could be relaunched at any time when necessary.

The algorithm returns the acceleration required over time, the velocity over time of the trajectory and the position over time that will follow the vehicle (each one is the integral of the previous, so all of them can be easily obtained).

The splines have always a full analytical solution, to reduce computational cost and to be relaunched without issues.

3.6.1 3rd order spline without initial acceleration

The 3rd order spline in velocity has 4 variables: a, b, c, d. The fifth unknown is the total time of the trajectory, T. The equation system is obtained through applying initial and final conditions, and also using the fact that the velocity spline is symmetric, so the maximum acceleration is achieved at time equal to $T/2$:

$$
\begin{aligned}
x(t) &= \tfrac{a}{4}t^4 + \tfrac{b}{3}t^3 + \tfrac{c}{2}t^2 + dt + x_0 \\
v(t) &= at^3 + bt^2 + ct + d \\
a(t) &= 3at^2 + 2bt + c
\end{aligned}
$$

$$
\begin{cases}
v(0) = v_0 \\
a(0) = 0 \\
v(T) = v_f \\
a(T) = 0 \\
a(\frac{T}{2}) = a_{max}
\end{cases}
$$

The analytical solution is the following:

$$\begin{cases} b = \dfrac{4a_{max}^2}{3(v_f - v_0)} \\ a = -\dfrac{b^2}{3a_{max}} \\ c = 0 \\ d = v_0 \\ T = \dfrac{3(v_f - v_0)}{2a_{max}} \end{cases}$$

3.6.2 3rd order spline with arbitrary initial acceleration

The 3rd order spline in velocity is the same as in the case without initial acceleration. The equation system is a bit different because the spline is not symmetric and the maximum acceleration is reached in a certain time t_1. The 6 unknowns are the spline variables a, b, c, d, the total time of the trajectory T and time t_1:

$$\begin{aligned} x(t) &= \tfrac{a}{4}t^4 + \tfrac{b}{3}t^3 + \tfrac{c}{2}t^2 + dt + x_0 \\ v(t) &= at^3 + bt^2 + ct + d \\ a(t) &= 3at^2 + 2bt + c \\ a'(t) &= 6at + 2b \end{aligned}$$

$$\begin{cases} v(0) = v_0 \\ a(0) = a_0 \\ v(T) = v_f \\ a(T) = 0 \\ a(t_1) = a_{max} \\ a'(t_1) = 0 \end{cases}$$

The analytical solution is the following:

$$\begin{cases} 0 = \left[\dfrac{4a_0^2}{a_0 - a_{max}} - 3a_0\right] T^2 + \left[6(v_f - v_0) - 12\dfrac{(v_f - v_0)a_0}{a_0 - a_{max}}\right] T + \left[\frac{9(v_f - v_0)^2}{a_0 - a_{max}}\right] \\ b = \dfrac{1}{T}\left[\frac{3(v_f - v_0)}{T} - 2a_0\right] \\ a = \dfrac{b^2}{3(a_0 - a_{max})} \\ c = a_0 \\ d = v_0 \\ t_1 = -\dfrac{b}{3a} \end{cases}$$

First of all is needed to find T_1 and T_2, solutions of the 2nd order equation. Not in all cases there exists a solution, due to the denominators. a_{max} must be always greater than a_0 if accelerating or lower when decelerating.

When only one of the T_i is positive (that means that the spline has one valid solution), this is the unique solution. The case when both T_i are positive corresponds to the two possible solutions: increase acceleration up to a_{max} and finish faster the manoeuvre or start decreasing acceleration from a_0 and finish the manoeuvre in more time. In this case, always is chosen the fastest manoeuvre, so $T = min(T_1, T_2)$.

The cases when the spline has no solution correspond to situations when the vehicle is accelerating in the opposite direction of the desired final speed, and with close velocity values. Actually the spline can solve these 'contradictions' but fails when there is not much difference in the current velocity and the desired one. As it is wanted a full-case analytical solution, this problem is solved by adding a linear section that starts in a_0 and ends in 0 acceleration, with a desired slope that guarantees comfort. Finally, after this section the spline without initial acceleration can be launched normally.

3.6.3 Velocity profile generation algorithm

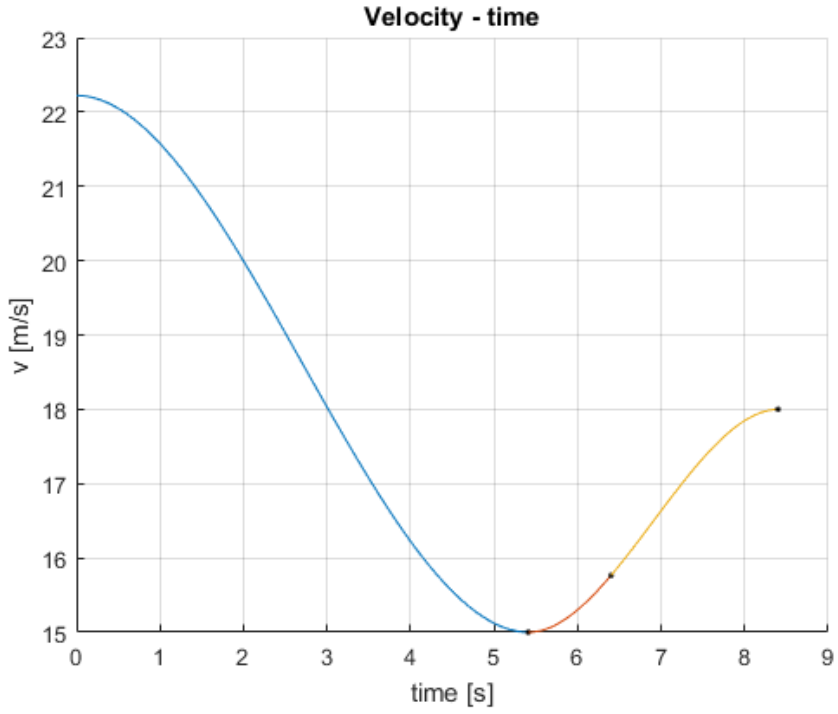
Now that all the equations of the splines are detailed, the whole velocity profile generation algorithm can be defined. It is explained in Algorithm 1. A *contradictory case* is a situation when the final velocity demands accelerating, but a_0 is a braking acceleration, and the same with contrary signs.

Algorithm 1 Generate a velocity profile

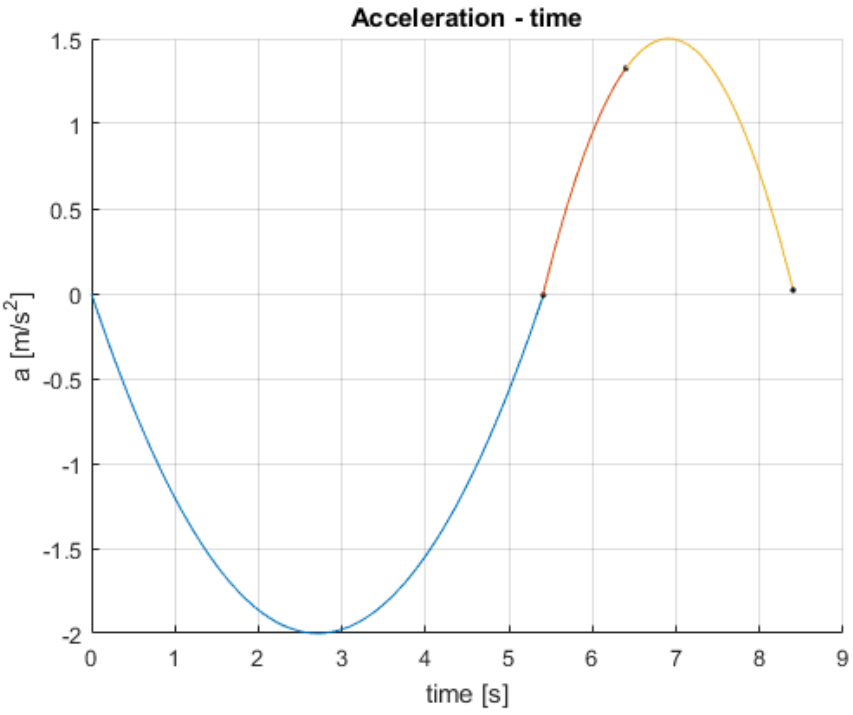
```
if a_0 = 0 then
    Spline case without a_0
else
    if v_f = v_0 then
        Add linear acceleration from a_0 to 0
        Spline case without a_0
    else
        if contradictory case then
            Add linear acceleration from a_0 to 0
            Spline case without a_0
        else
            Spline case with arbitrary a_0
        end if
    end if
end if
```

Figure 35 shows an example of operation. Each colour is a different launch of the algorithm. This example starts at 80 km/h (22.22 m/s) and slows down to 15 m/s, having a maximum deceleration of -2 m/s^2 (blue curve). This is the spline without initial acceleration. Then, it is computed a new velocity profile to 18 m/s (in red) but in a certain instant, the algorithm is re-launched to the same final speed (in orange). As it can be seen, the algorithm can follow the same initial computed trajectory in any state. This orange solution corresponds to the spline with arbitrary initial acceleration, as a_0 is approximately 1.4 m/s^2.

In the contradictory cases, as explained before, the algorithm performs a linear section to 0 acceleration, shown in Figure 36, also preserving comfort and feasibility.

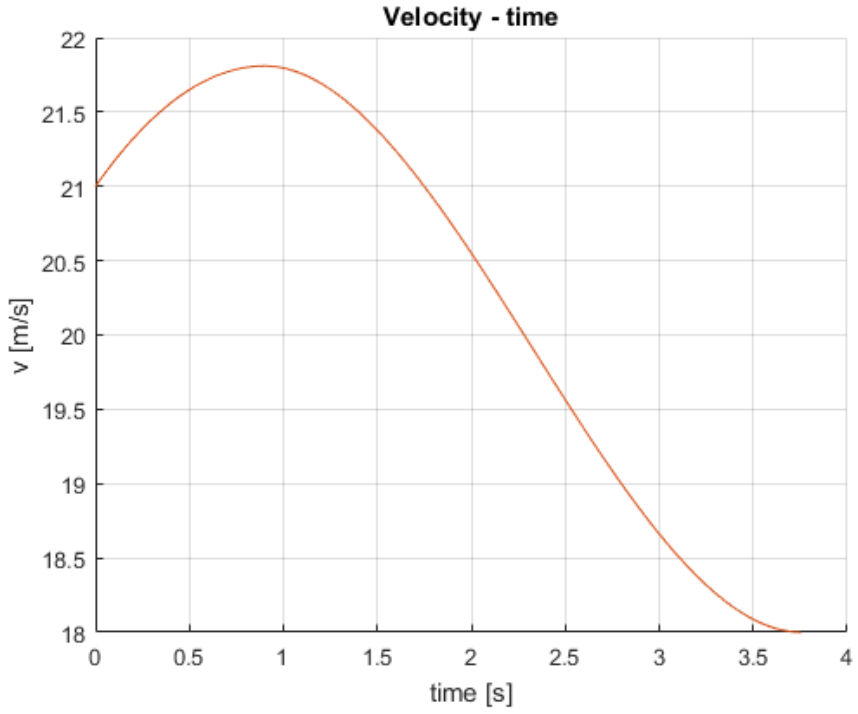


(a) Velocity over time $[m/s]$

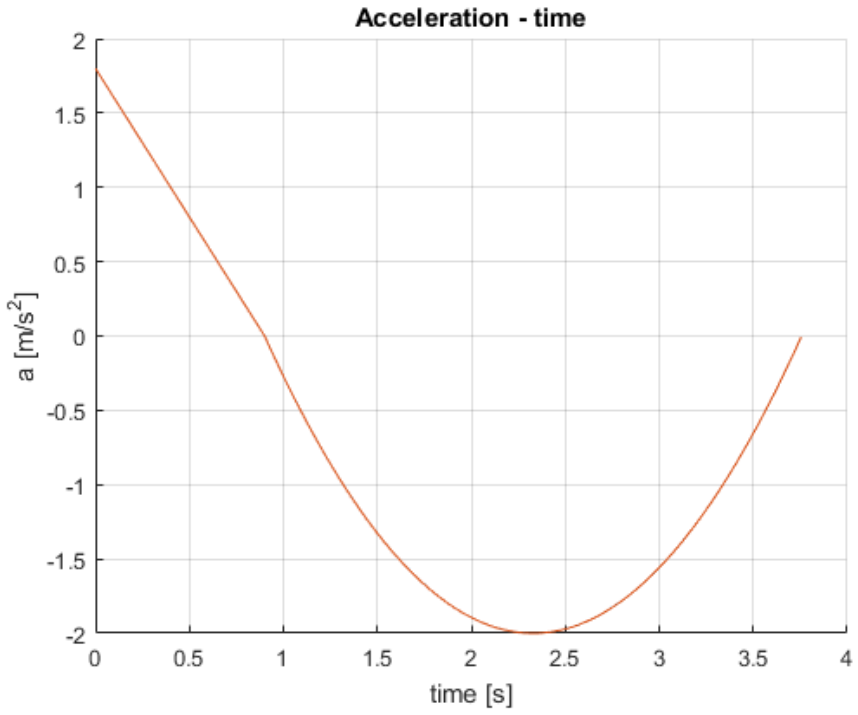


(b) Acceleration over time $[m/s^2]$

Figure 35: Different launches of the velocity profiles generation, in different colours



(a) Velocity over time $[m/s]$



(b) Acceleration over time $[m/s^2]$

Figure 36: Contradictory launch of the trajectory generation splines

3.6.4 3rd order spline without initial acceleration and station (auxiliary)

One last velocity profile generation algorithm is used by the AKRP, but this spline is not used to generate the velocity profiles. It is used in the candidate filtering, explained in section 3.6.5. It is exactly the same spline as the case without initial acceleration, but with one of the degrees of freedom substituted: the maximum desired acceleration is changed by the desired final station.

The 3rd order spline in velocity has 4 variables: a, b, c, d. The fifth unknown is the total time of the trajectory, T. The equation system is obtained through applying initial and final conditions:

$$
\begin{aligned}
x(t) &= \tfrac{a}{4}t^4 + \tfrac{b}{3}t^3 + \tfrac{c}{2}t^2 + dt + x_0 \\
v(t) &= at^3 + bt^2 + ct + d \\
a(t) &= 3at^2 + 2bt + c
\end{aligned}
$$

$$
\begin{cases}
v(0) = v_0 \\
a(0) = 0 \\
v(T) = v_f \\
a(T) = 0 \\
x(T) = x_f
\end{cases}
$$

The analytical solution is the following:

$$
\begin{cases}
T = \dfrac{2 \cdot x_f}{v_f + v_0} \\
b = \dfrac{3(v_f - v_0)}{T^2} \\
a = -\dfrac{2}{3}\dfrac{b}{T} \\
c = 0 \\
d = v_0
\end{cases}
$$

As in the case without initial acceleration, the resulting profile is also symmetric, and the maximum (or minimum) value of the acceleration is located also in $t = T/2$.

3.6.5 Generation of the set of velocity profile candidates

As it has been seen in Figure 9, for each candidate path the AKRP computes a set of velocity profile candidates, discretizing the final velocities and also the accelerations. Starting in the current state of the host vehicle, a set of final velocities is chosen from 0 to the maximum road velocity allowed in that moment. Then, for each final velocity, a set of different accelerations is also chosen, from the maximum deceleration limit to the desired acceleration value, which are parameters. These sets can be seen in Figures 37, 38 and 39.

In order to optimize the algorithm and reduce the computation, not always all the profile candidates are computed. Before computing the velocity profiles, some filtering is done to reject the non-feasible candidates and restrict the search space.

The filtering process uses 3 restrictions that contain:

1. The presence of a collision with a static obstacle, and its station (the distance in length of the track from the host vehicle to the obstacle). If a collision is detected, the only allowed final velocity is 0. Then, from the acceleration discretization, it will be selected the best velocity profile. In this case, this will correspond to the minimum deceleration that safely stops the host vehicle before the detected collision station.

2. The maximum allowed velocity due to the planned path curvature, and its station.

3. The maximum allowed velocity due to the center lane path curvature, and its station. This restriction assures safety, in order to be more conservative and check a static property of the track geometry, because the planned path could be smoothed when re-launching the AKRP planner.

For each one of the restrictions, if braking is needed, then the discretization in accelerations is restricted only to accelerations that safely reach the allowed velocity before its station. The computation of the softer feasible braking acceleration is done with the auxiliary spline of section 3.6.4. As an example, if a braking is needed and the braking acceleration that fulfils the restriction in the limit is -1.5 m/s^2, the possible accelerations will be restricted to be inside the interval:

$[maximum\_deceleration\_limit = -5, -1.5]$.

With the velocity and acceleration discretizations that fulfil all 3 restrictions are computed the set of velocity profiles.

For example, consider a urban situation with a maximum allowed velocity of $50km/h$. The host vehicle is turning in a street at only $30km/h$, because of the turn. Suppose that the filtering process checks the curvature of the planned path and restricts the maximum velocity to $32km/h$, so there is no need to compute velocity profiles up to $50km/h$ if we already know that they are not feasible.

Now consider the same scenario, but suddenly a static obstacle is detected 25 m ahead inside our planned path. In this situation the velocity profile candidates will be only those with final velocity 0, but with different decelerations. In addition, when filtering the decelerations, we will only have candidates that stop before the obstacle (in this situation, a solution with a very soft braking has no sense because implies collision). This will allow to the AKRP to choose the optimal deceleration. Note that in this case, supposing that we only have one allowed lane, the best solution is braking to a stop, so there is no need to compute the whole velocity profile set. This example is shown in Figure 39.

The execution time of the complete set of Figure 37 is 45 ms, in MATLAB running in a 4 GHz CPU.

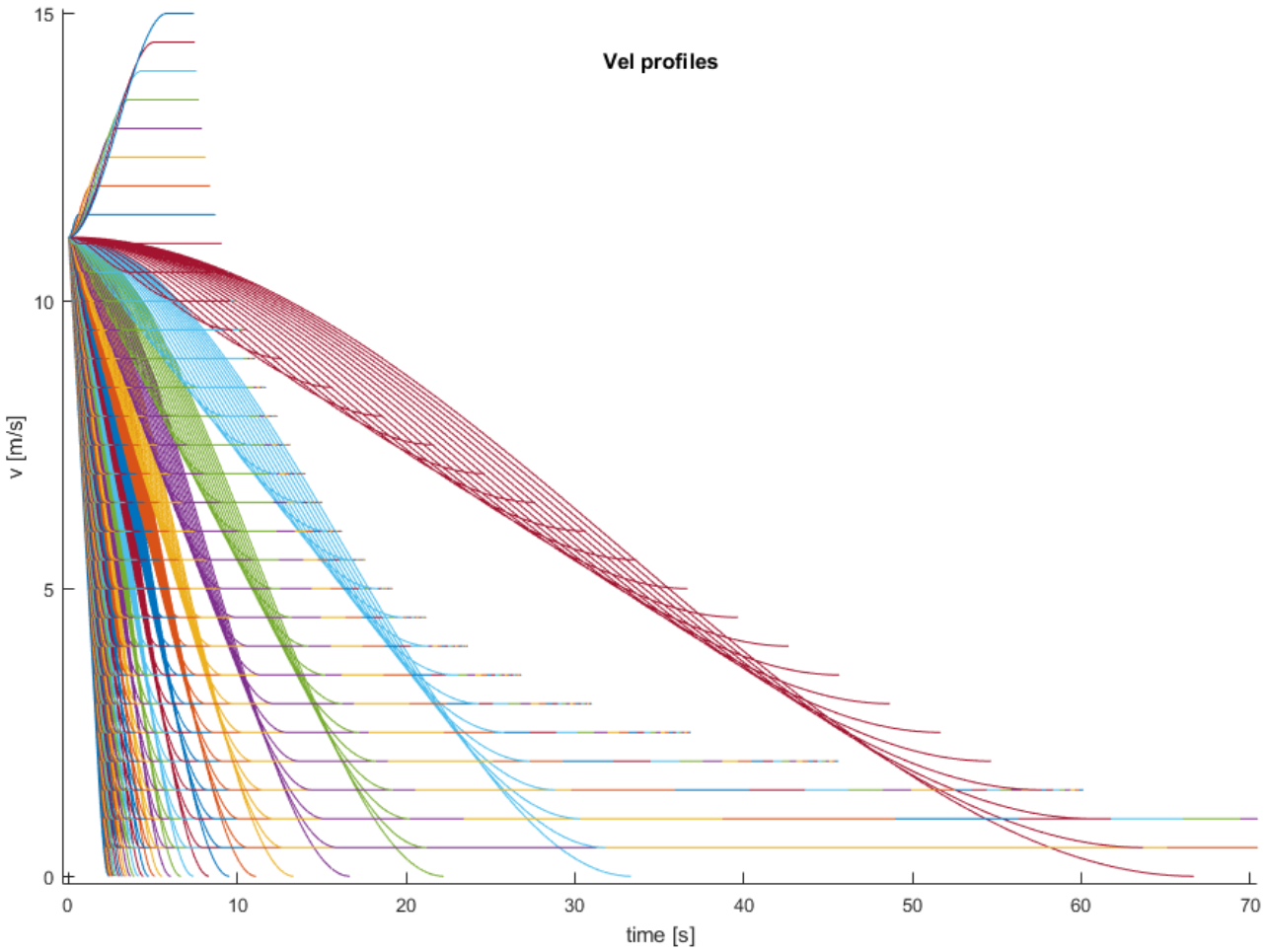


Figure 37: Set of velocity profiles. In this example, there is no active restrictions, so these are all the possible velocity profiles, with a discretization of 0.5 m/s and different accelerations

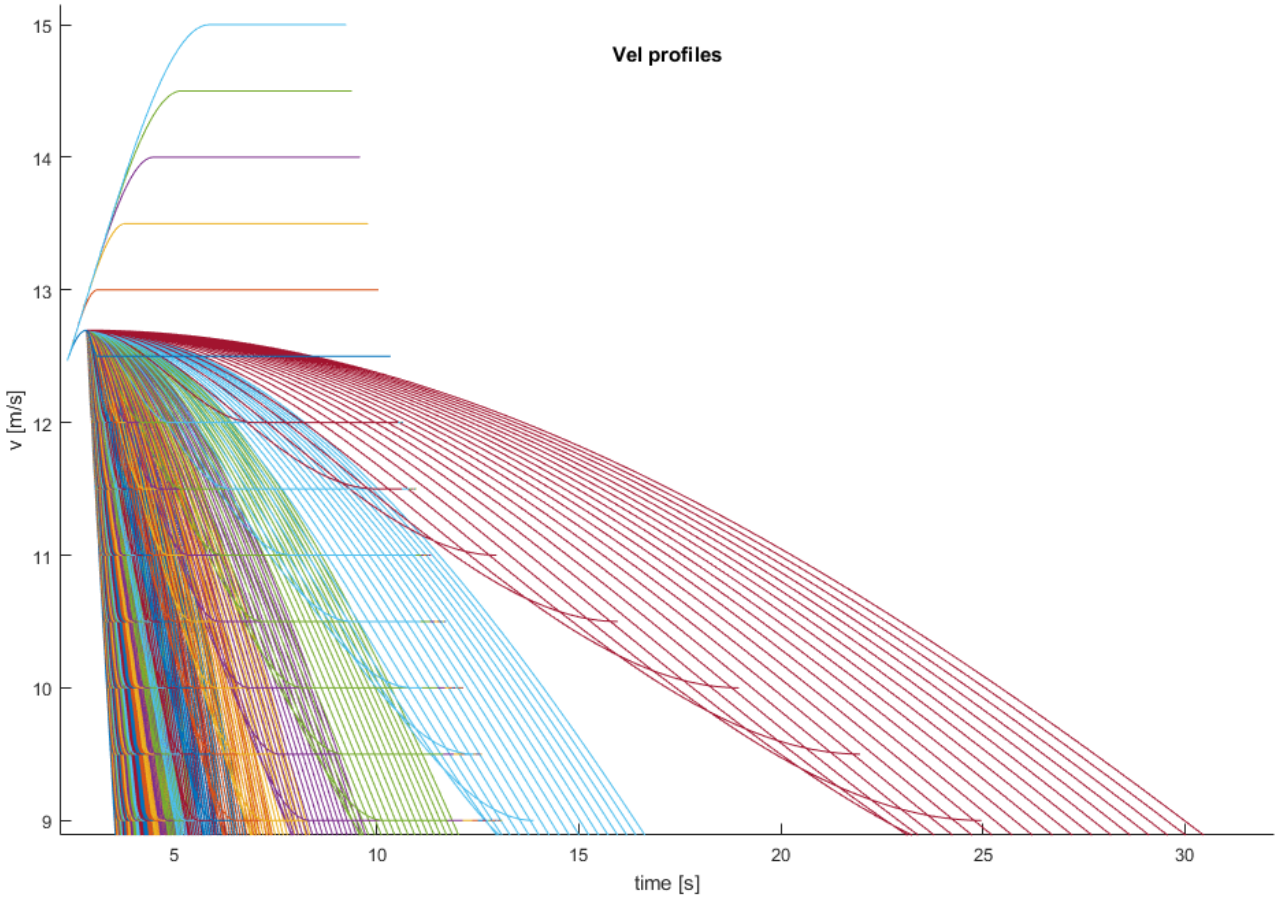
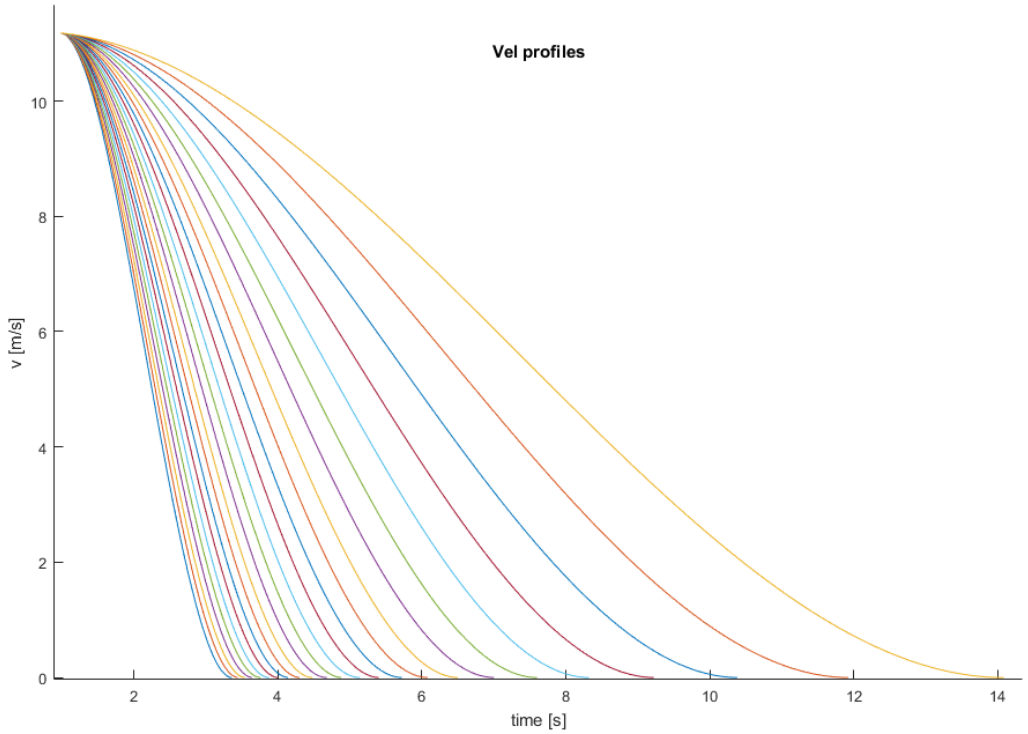
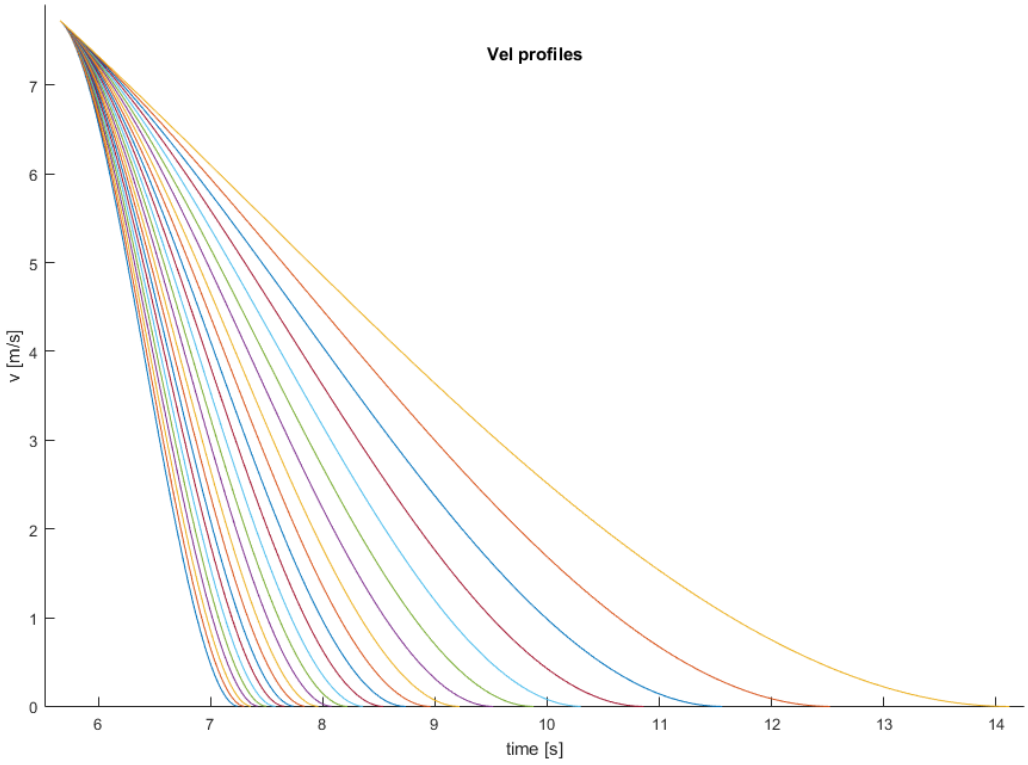


Figure 38: Detail of a set of velocity profiles, with a discretization of 0.5 m/s and different accelerations. Note the initial acceleration, different from 0. The cases with a lower final velocity are *contradictory cases*



(a) Example without initial acceleration



(b) Example with initial acceleration, the vehicle is already braking

Figure 39: Set of velocity profiles candidates. In this example, there is an obstacle ahead, so these are the feasible velocity profiles, all with a final velocity of 0 m/s and different decelerations

3.7 Costs

The selection of the AKRP output is made in each iteration by choosing the minimum cost solution.

There are different cost terms to evaluate several criteria, such as safety, comfort, efficiency, energy consumption and behaviour.

In the same way as most of the lattice planners, the total cost function, J, is a weighted sum of all the cost terms:

$$J = \sum_i w_i \cdot c_i$$

Where w_i is the weight of the cost i. Depending on the weights and also in the used terms, the planner can be adjusted to a desired behaviour.

The AKRP uses two kind of cost terms: static and dynamic. Static costs are the terms related to path geometry and static obstacles. They are associated to a candidate path. In the other hand, dynamic costs are related to the temporal dimension and they are associated to a velocity profile and include the dynamic obstacles. These two groups are shown in Tables 1 and 2, and the terms are detailed below. The costs are based on the method proposed in [19], but adapted to fit the AKRP framework.

Cost	Formula	Normalized term	Physical interpretation	Impact
c_l	l	$l/s_{manoeuvre}$	path length	Efficiency
c_κ	$max(\kappa)$	$max(\kappa) \cdot r_{min}$	maximum path κ	Comfort & Kinematic feasibility
$c_{\dot{\kappa}}$	$max(\dot{\kappa})$	$max(\dot{\kappa}) \cdot r_{min}$	maximum path $\dot{\kappa}$	Comfort & Kinematic feasibility
c_{off}	o	o/o_{max}	lateral offset from centerline	Behaviour
$c_{obs,s}$	$f \cdot e^{(-1/\lambda)\cdot d}$	-	static obstacles repulsion	Safety

Table 1: Static costs

Cost	Formula	Physical interpretation	Impact
c_v	$1 - v_{f,vp}/v_{max,desired}$	velocity	Behaviour
c_a	$abs(a_{max,vp}/a_{max,braking})$	acceleration	Comfort & Efficiency
$c_{obs,d}$	$f \cdot e^{(-1/\lambda)\cdot d}$	dynamic obstacles repulsion	Safety & Behaviour

Table 2: Dynamic costs

All the terms with the exception of the obstacle terms are normalized between 0 and 1. In this way, the weighting can be more clear and intuitive. Each cost term is detailed below:

- c_l: l is the length of the candidate path. $s_{manoeuvre}$ is the station of the ending point of the path, from the host vehicle.
- c_κ: κ is the curvature value of a specific point in the path. r_{min} is the minimum turning radius of the host vehicle.
- $c_{\dot{\kappa}}$: $\dot{\kappa}$ is the rate of change of the curvature in a specific point of the path.
- c_{off}: o_{max} is the lateral distance from the farthest endpoint to the center of the lane (the point with more lateral offset).
- c_v: $v_{f,vp}$ is the ending velocity of the candidate velocity profile. $v_{max,desired}$ is the maximum allowed velocity at the current moment.
- c_a: $a_{max,vp}$ is the maximum acceleration value of the candidate velocity profile. $a_{max,braking}$ is the acceleration value (always < 0) of the maximum allowed braking.
- c_{obs}: f and λ are parameters of the exponential function, f is a scale value and λ is the decay. d is the distance from a obstacle (static or dynamic) to the host vehicle (the distance from the host vehicle circle to the obstacle circle). If d is smaller than a threshold, then the cost is penalized:

$$c_{obs} = \begin{cases} f \cdot e^{(-1/\lambda)\cdot d} + Penalization, & if \;\; d < threshold \\ f \cdot e^{(-1/\lambda)\cdot d}, & otherwise \end{cases}$$

  The function, without the penalization term, is shown in Figure 40.

  Unlike other approaches that penalize collisions with an infinite cost, as [19], it is preferred to preserve the value. Then, in a situation where all the candidate solutions present high cost (for instance, an unavoidable obstacle), the planner will give always the minimum cost solution, so it will be the 'less dangerous' one.

The used weighting of the costs has been adjusted through simulation and it is detailed in Table 3.

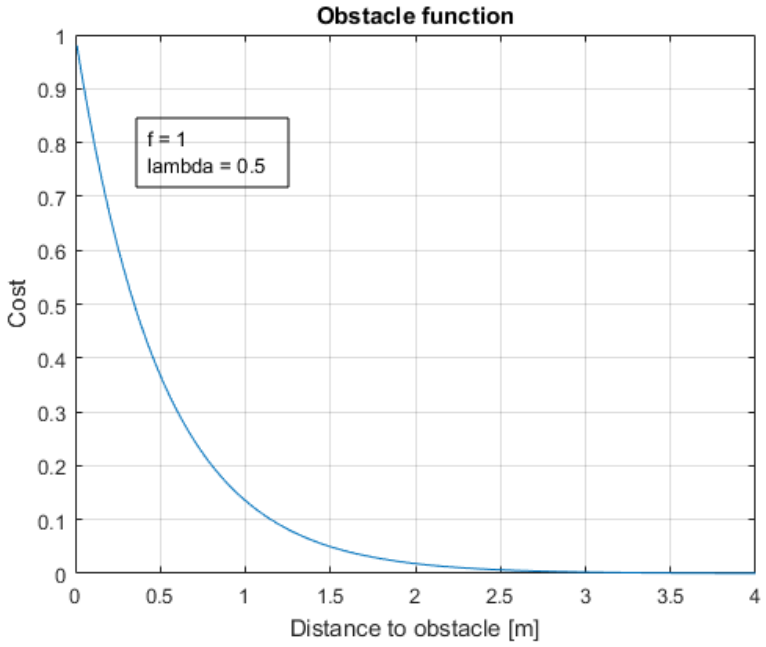


Figure 40: Obstacle cost function c_{obs}, without the penalization term for plotting

Cost	Weight	Parameters
c_l	1	-
c_κ	1	-
$c_{\dot{\kappa}}$	1	-
c_{off}	1	-
$c_{obs,s}$	1	$f = 10,\ \lambda = 0.5$
c_v	10	-
c_a	1	-
$c_{obs,d}$	1	$f = 1,\ \lambda = 0.5$

Table 3: Used weighting and parameters of the costs

4 Simulations and performance

This section shows examples of simulations of the AKRP in several scenarios and situations. AKRP has been implemented in MATLAB, but also the low level controllers in order to simulate the host vehicle. Controllers have been tuned with the values of Annexes A and B (Tables 4 and 5), to fit the real car that IRI uses as a test platform (a Seat León Cupra 2.0 TSI 265 cv of 2014). As explained in section 3.4.4, the simulations include the model of the steering column actuator. However, the complex longitudinal dynamic model used to validate the longitudinal controller (shown in section 3.4.1) has not been used, as the controller tracks perfectly any desired velocity profile when requested accelerations are feasible by the car.

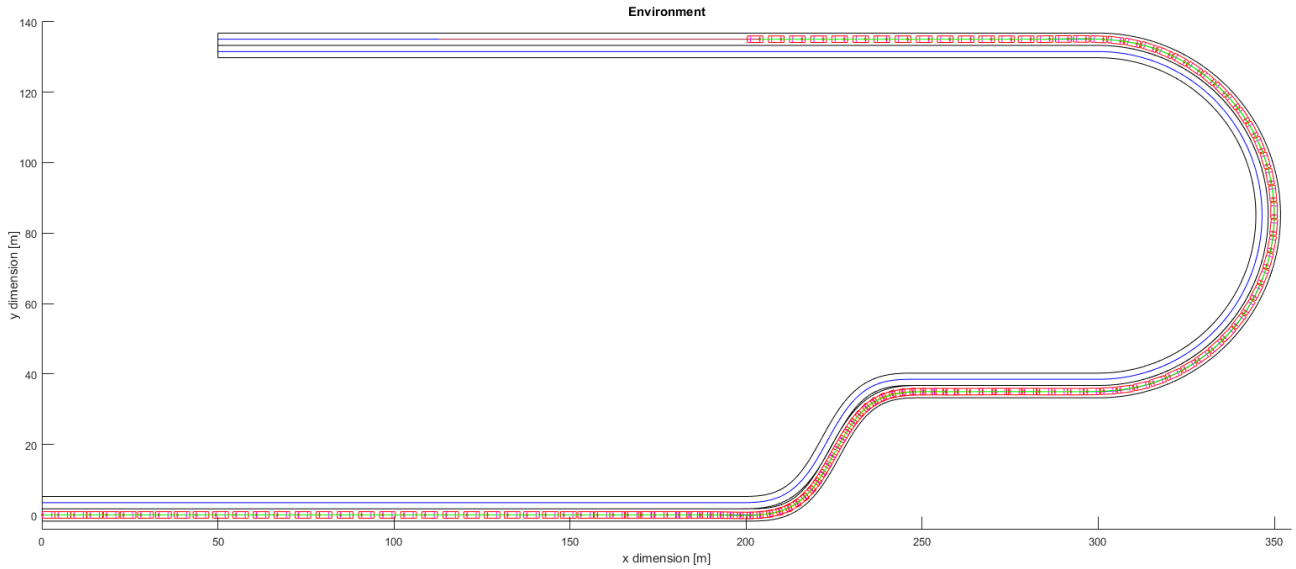
Static obstacles are detected by the AKRP planner when they are closer than a certain threshold that simulates the perception range. This threshold has been set to 100 m. Detected static obstacles are plotted in green colour and undetected ones in orange.

Dynamic obstacles appear in a certain time instant, and have a constant velocity.

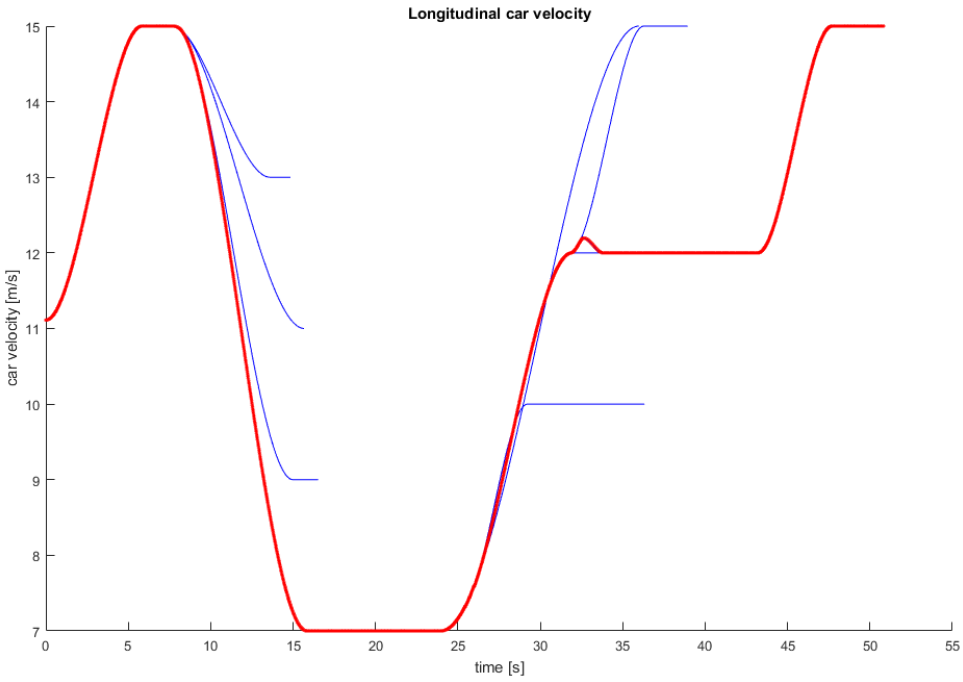
The vehicles are plotted every a fixed time interval, to facilitate the perception of its velocities in the figures.

4.1 Driving without obstacles

Figure 41 shows a simulation in an environment without any obstacles. The maximum allowed velocity is 15 m/s, and the host vehicle starts at 11.11 m/s (40 km/h). At the beginning, it accelerates to the maximum speed, until encountering the "S" turn, which is passed at 7 m/s. Then, speed is incremented up to 12 m/s because of the left turn, and in the final straight section the vehicle accelerates again to the maximum allowed speed.



(a) Environment, dimensions in $[m]$



(b) Host vehicle velocity over time (in red) and planned (in blue), in $[m/s^2]$

Figure 41: Simulation without any obstacle

4.2 Driving with static obstacles

Figure 43 shows the same environment as in section 4.1 but full of static obstacles. This simulates a urban scenario, for instance with vehicles parked in double row.

The maximum allowed velocity is 15 m/s, and the host vehicle starts at 11.11 m/s (40 km/h). At the beginning, it tries to accelerate to the maximum speed, but then some static obstacles that obstruct the lane are encountered. Due to this, the planned path curvature forces the host vehicle to reduce speed to 7 m/s. The 'S' turn is passed at 7 m/s, but suddenly, obstacle 6 is detected blocking the current lane and a new planned velocity profile to 0 m/s is followed. This is because at this current moment, the planning horizon cannot finish the avoidance manoeuvre, as explained in previous sections. In addition, the manoeuvre cannot be executed because of obstacle 5, which blocks the left lane.

Once the host vehicle passes obstacle 5, the avoiding manoeuvre to avoid obstacle 6 can be performed, so immediately velocity increases again ($t \approx 28$ s).

When the 'S' turn is behind, speed is incremented up to 12 m/s because of the left turn, but then the lane is totally blocked ahead by obstacle 20.

Finally, when the planning horizon allows the avoidance, it is performed at 7 m/s, and in the final straight section the vehicle accelerates to the maximum allowed speed.

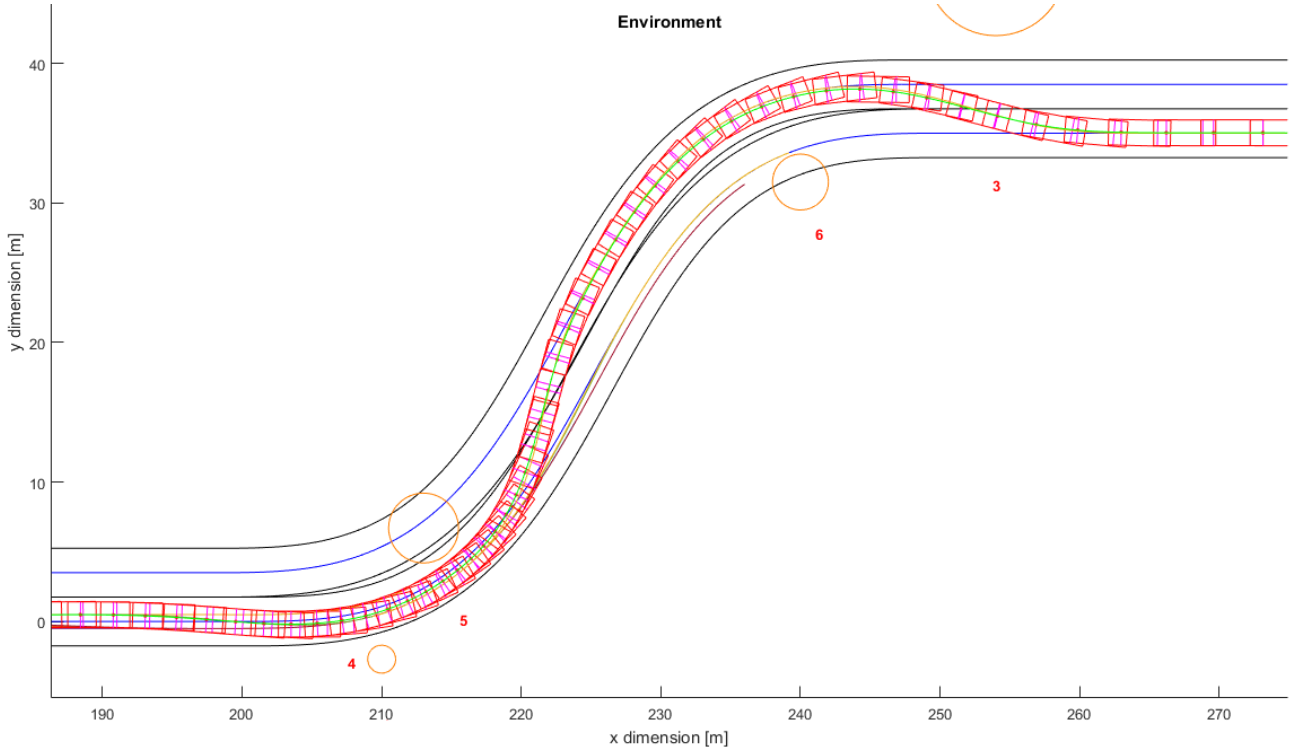
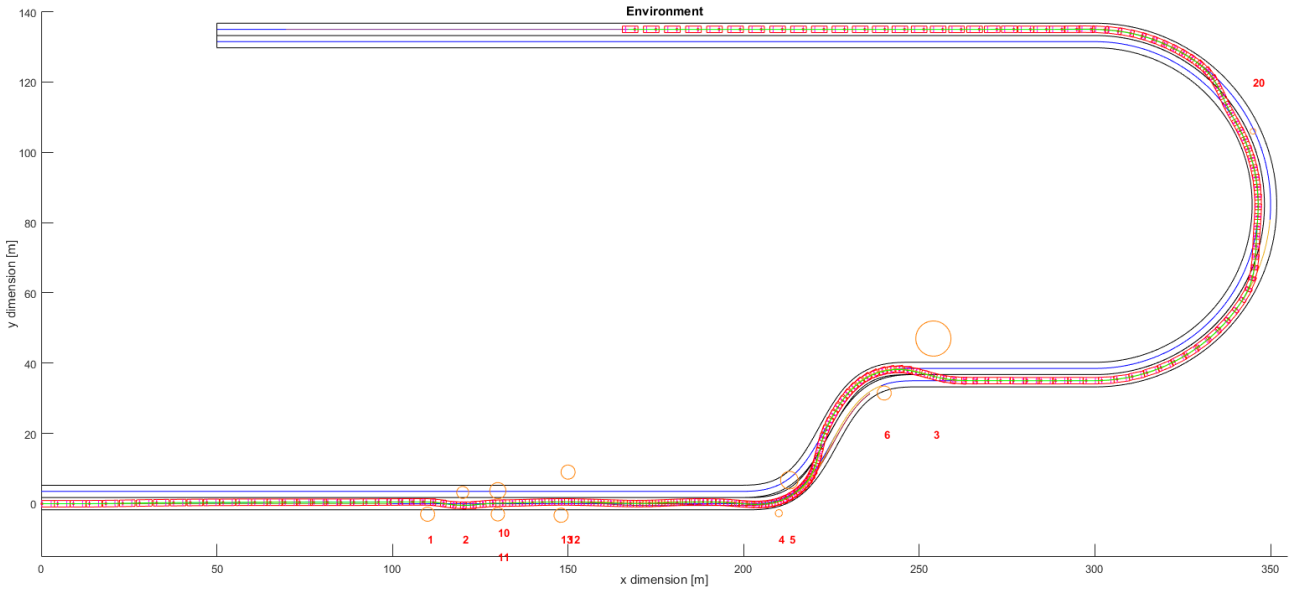
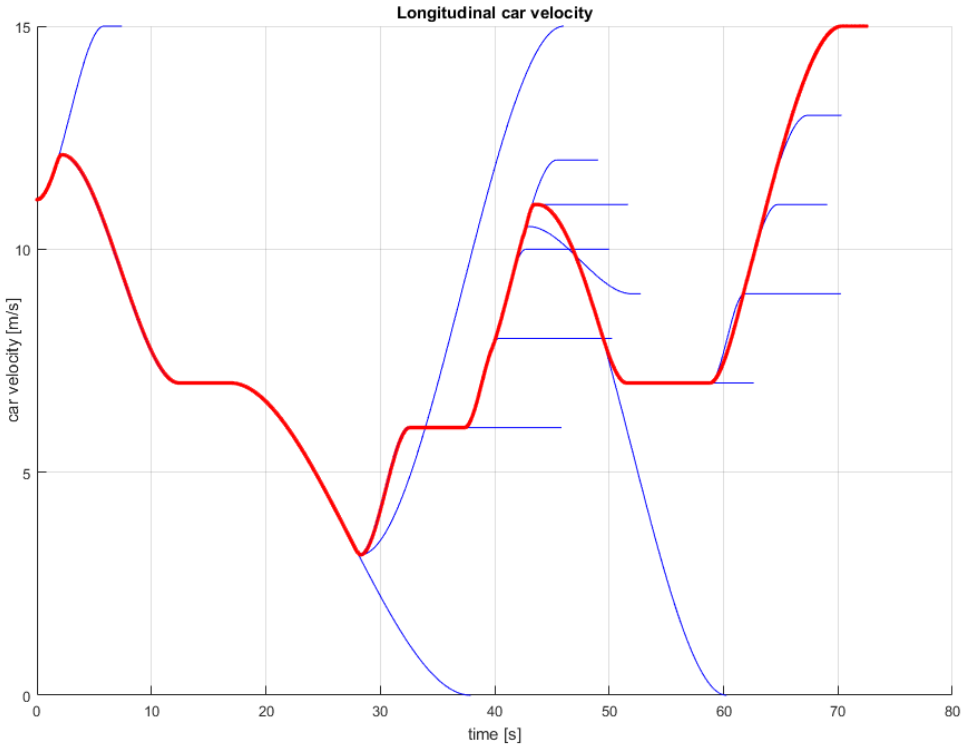


Figure 42: Detail of the first static obstacle avoidance manoeuvre



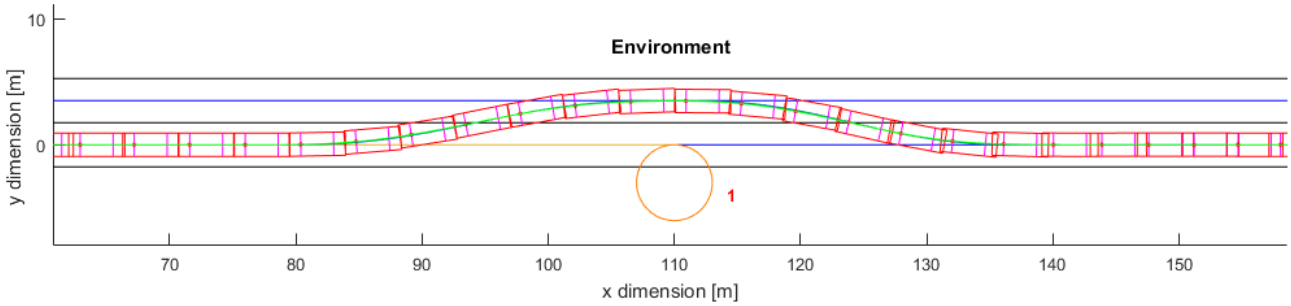
(a) Environment, dimensions in $[m]$



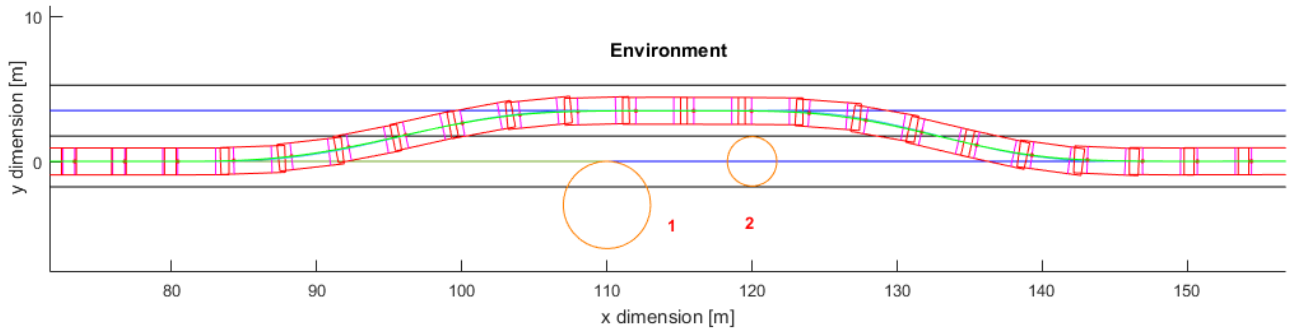
(b) Host vehicle velocity over time (in red) and planned (in blue), in $[m/s^2]$

Figure 43: Simulation with static obstacles

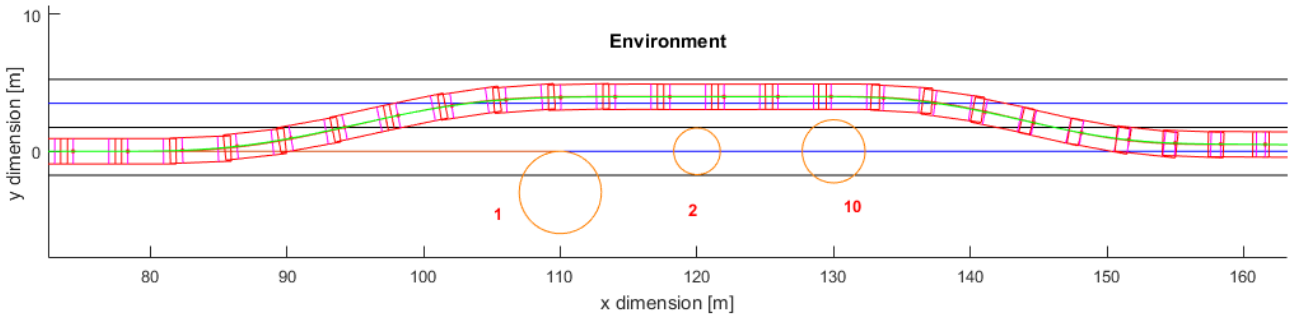
Figure 44 shows different obstacle avoidance situations, from an easy one to an unfeasible manoeuvre.



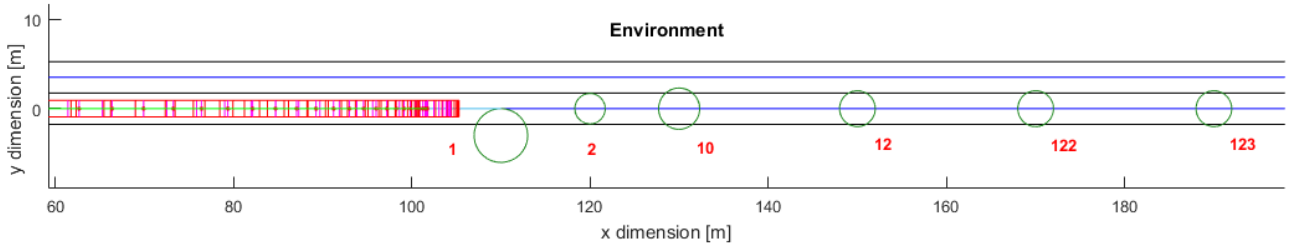
(a) Avoidance of 1 obstacle



(b) Avoidance of 2 obstacles



(c) Avoidance of 3 obstacles



(d) Avoidance of several obstacles that cannot be performed due to having lane blocking in all the planning horizon

Figure 44: Static obstacle avoidances, dimensions in $[m]$

4.3 Anticipation with dynamic obstacles

The simulations present in this section take into account the influence of the dynamic obstacles in the future.

4.3.1 Dynamic obstacle incursion

Figure 46 shows the effect of a dynamic obstacle which comes from one side and intersects the planned path.

The maximum allowed velocity is 15 m/s, and the host vehicle starts at 11.11 m/s (40 km/h). At the beginning, it tries to accelerate to the maximum speed, but all of a sudden a dynamic obstacle is detected. It goes at a constant speed of 20 km/h.

If the host vehicle continues with the same planned speed, it will collide with the dynamic obstacle. Therefore, it slows down a little to 8,5 m/s, allowing the dynamic obstacle to cross the road ahead. Finally, the planner accelerates again as the road is clear ahead.

Figure 45 is the detail of the modelling of the dynamic obstacle. Note the circles, much bigger than the real vehicle itself. This is equivalent to modify the parameters of the repulsion function of the dynamic obstacles. This is done in order to force a conservative behaviour, to maintain safety.

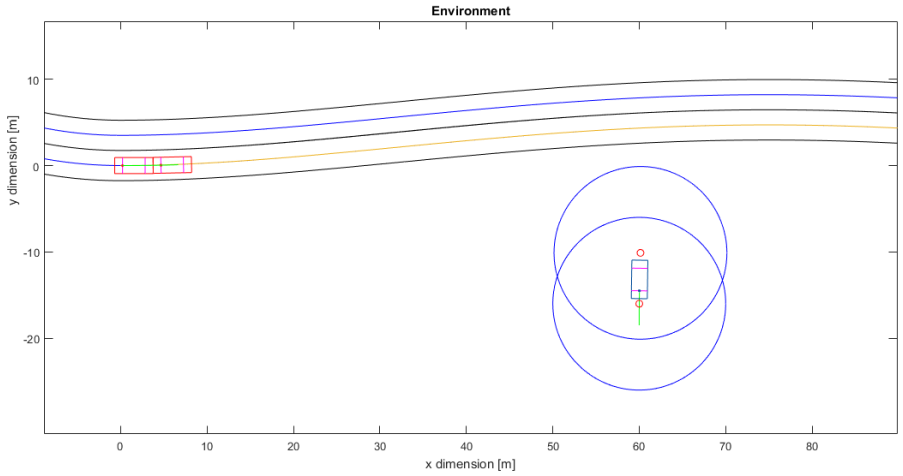


Figure 45: Detail of the dynamic obstacle modelling. The blue circles are the obstacle itself, with its centers in red. It is also shown the vehicle

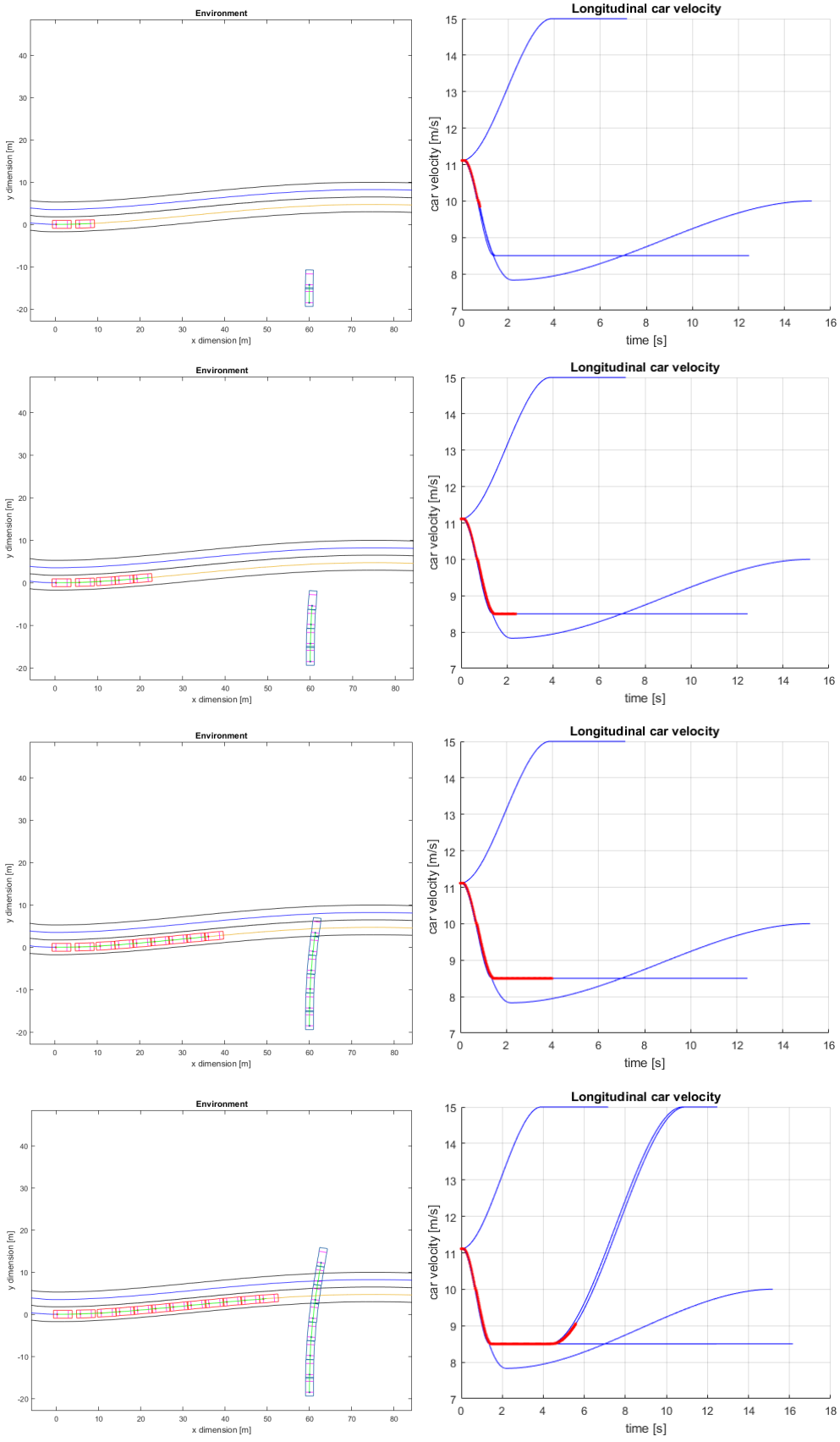


Figure 46: Dynamic obstacle avoidance, in several instants

4.3.2 T-intersection

To enter in a T-intersection is mandatory to anticipate the oncoming vehicles. The following simulations show the most relevant scenarios that can be encountered in this kind of intersection, with oncoming vehicles in both directions.

- Slow down without stopping, shown in Figure 47. In this example, the host vehicle slows down to 2 m/s and lets the 2 oncoming vehicles to pass before.
- Stopping before entering and wait, shown in Figure 48. In this case, the oncoming vehicles force the host car to stop for a while before entering to the intersection.
- Take advantage of a gap, shown in Figure 49. Here, the host vehicle continues as if there were no obstacles and enters to the intersection at its maximum allowed speed due to the lateral comfort acceleration, that is 8 m/s.

The dynamic obstacles have been modelled with a circle ahead, for ensuring that the AKRP does not choose a solution that passes too close to them. The radius of the circles has been set to be a little smaller than the vehicles itself. Putting a big radius like in Figure 45 implies having a lot of repulsion when driving in a parallel lane, and for the dynamic obstacle coming from the top, this effect forces the planner to wait and let the obstacles pass. This behaviour is good and can be desired also, but it cannot be used for simulating the 3rd situation of taking advantage of a gap.

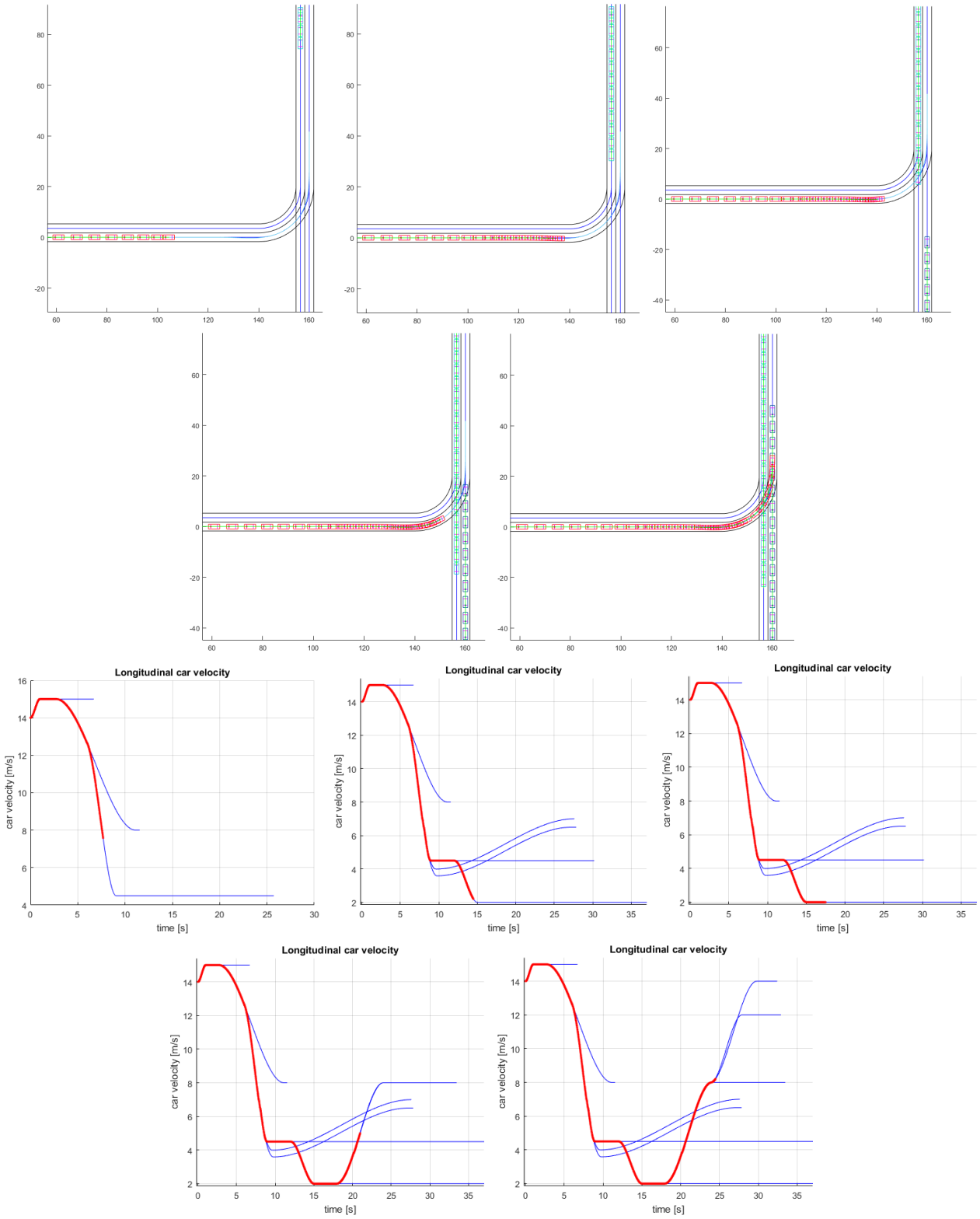


Figure 47: Incorporation in a T-intersection, in several instants. Slowing down case

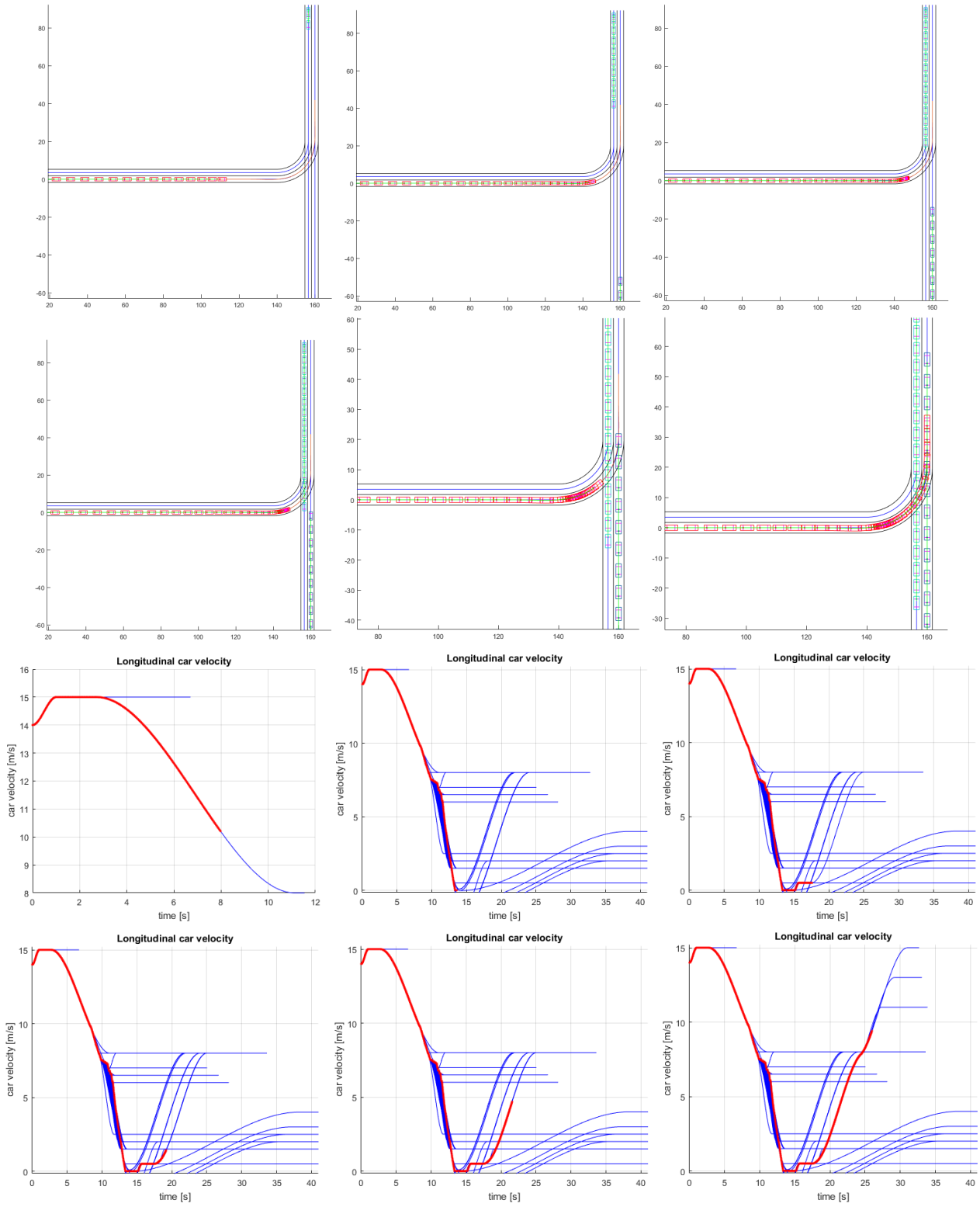


Figure 48: Incorporation in a T-intersection, in several instants. Unfavorable case

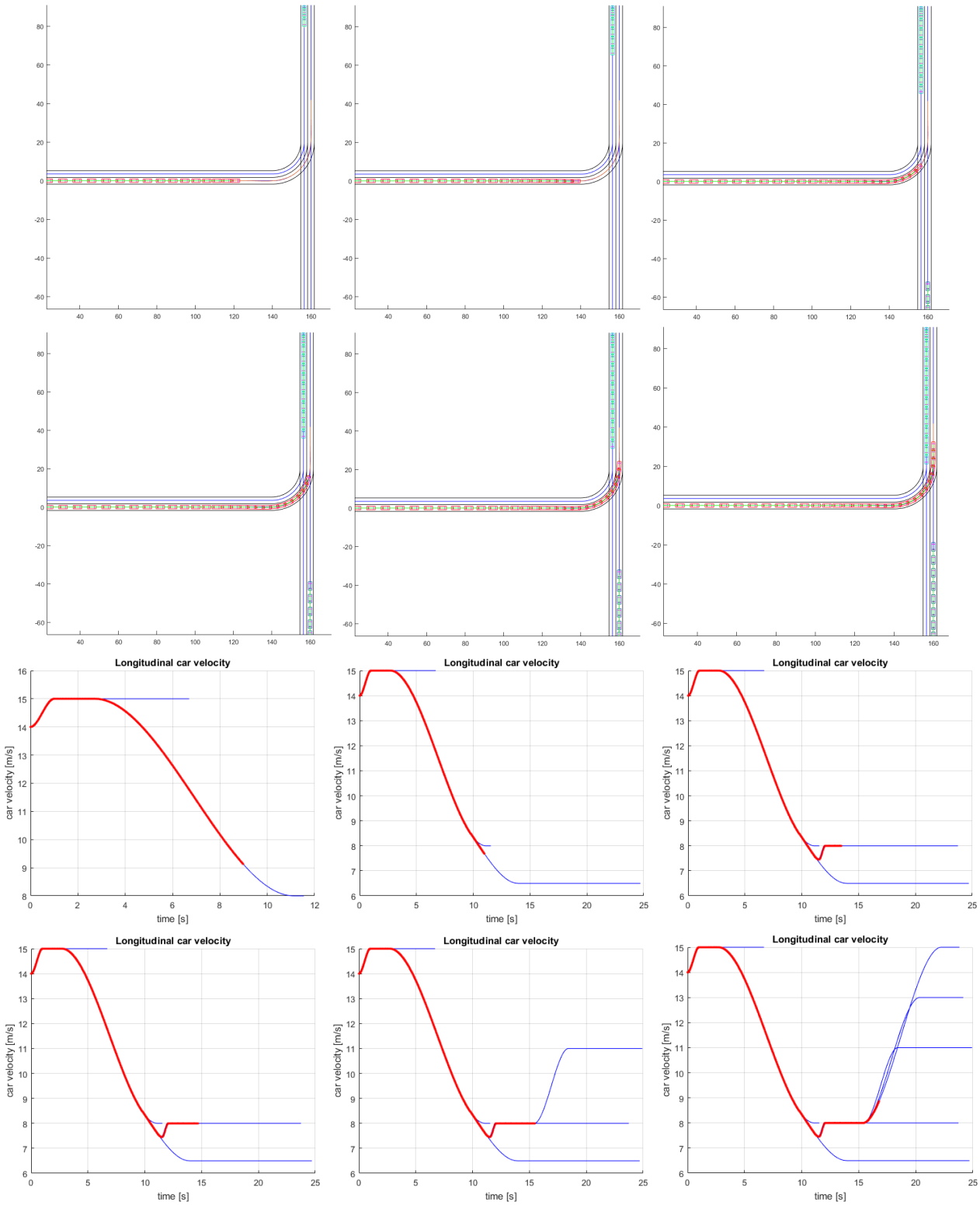


Figure 49: Incorporation in a T-intersection, in several instants. Optimal case

4.3.3 Passing a static obstacle with oncoming traffic

In Figure 51 the host vehicle is travelling in a 2-way track. It encounters the lane blocked ahead by some static obstacles, and there is an oncoming vehicle in the other lane. The velocity profiles associated with each one of the time stamps are shown in Figure 50.

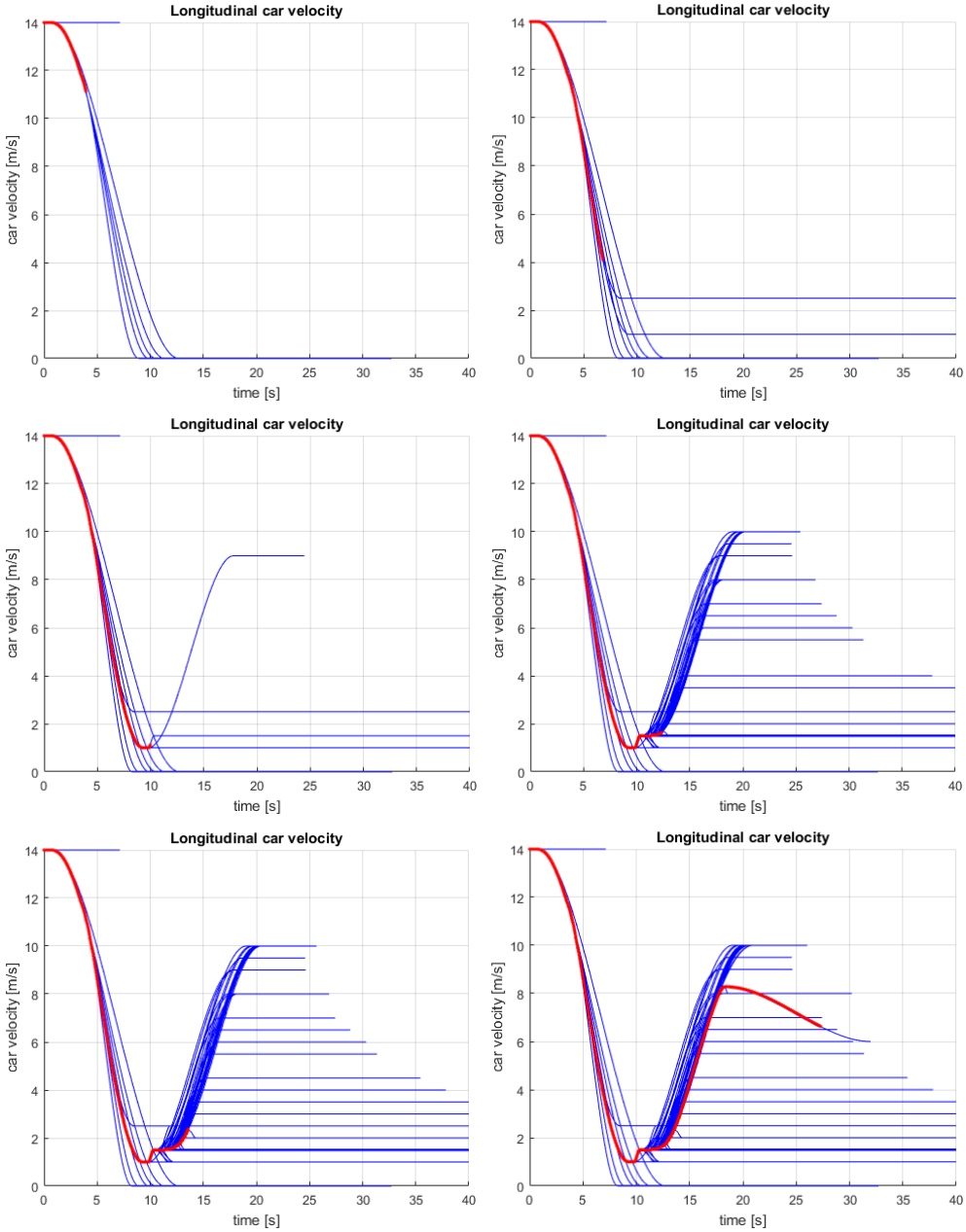


Figure 50: Planned (blue) and followed (red) velocity of the simulation shown in Figure 51, in $[m/s]$

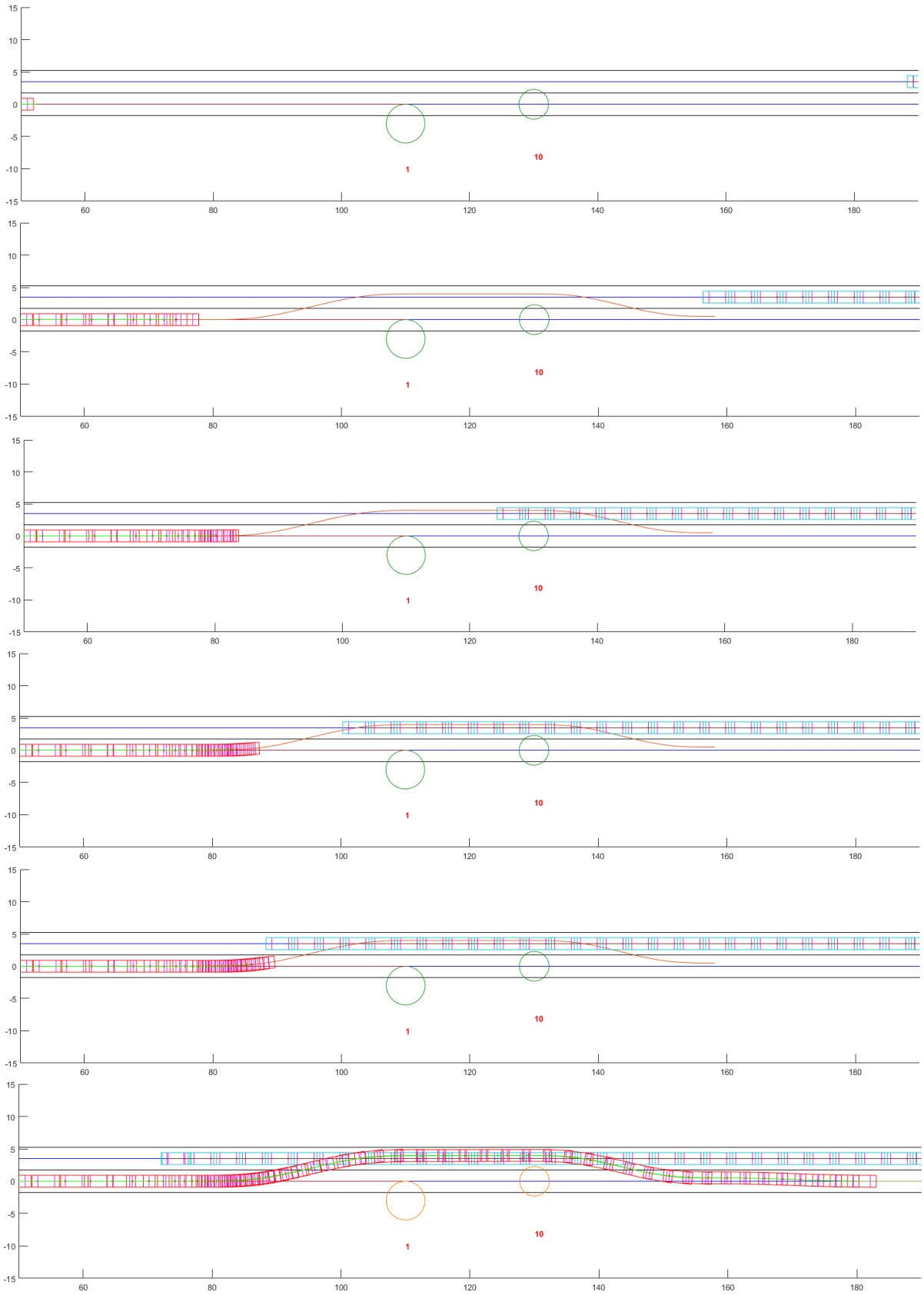


Figure 51: Overtaking of a static obstacle in a two way track

4.3.4 Overtaking in one way track

Figure 53 shows an overtaking. The host vehicle starts at 14 m/s and tries to reach the maximum speed of 15 m/s, but then it encounters a slower vehicle ahead going at 5 m/s. At first, the AKRP plans a velocity profile to 6,5 m/s because the complete manoeuvre until de desired horizon of 100 m ahead cannot be done. But when the host vehicle gets closer to the ahead vehicle, the AKRP fins a feasible overtaking solution.

For allowing the planner to overtake, the repulsion cost of the obstacle in its laterals cannot be too high, depending on this adjustments the planner will behave different: it can be conservative or aggressive (Figure 52).

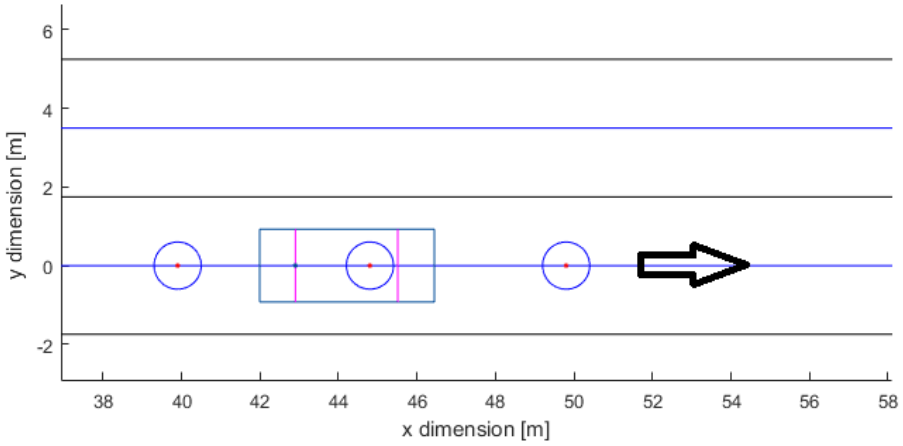


Figure 52: Detail of the dynamic obstacle modelling. The blue circles are the obstacle itself, with its centers in red. It is also shown the vehicle

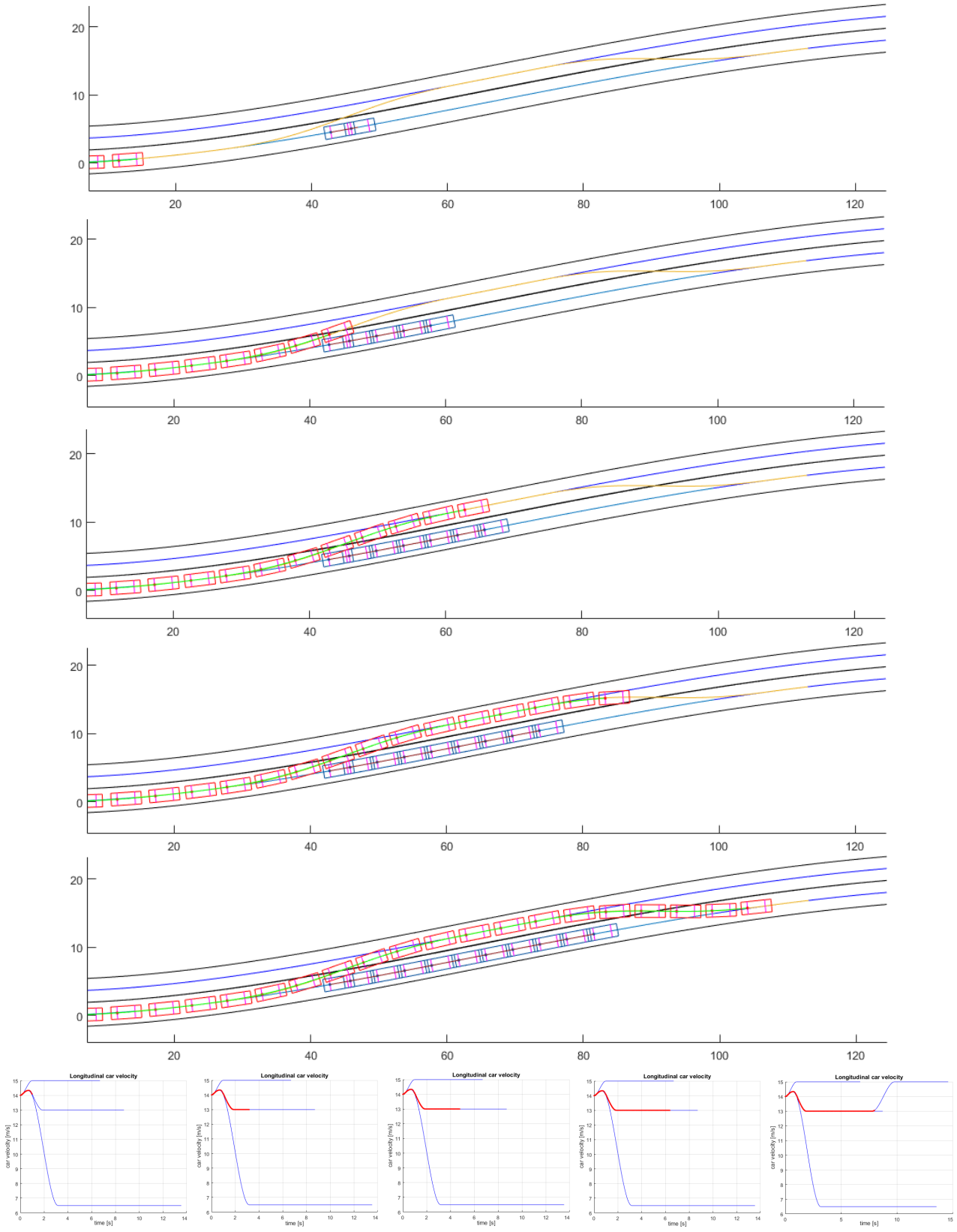


Figure 53: Overtaking of a slower vehicle in a one way track

4.3.5 Overtaking with oncoming traffic

This manoeuvre can be done only if it is predicted to its end, otherwise it is discarded and the host vehicle slows down and follows the dynamic obstacle. The distance between vehicles when following the ahead vehicle can be adjusted positioning the circles of the dynamic obstacle.

Figures 54 and 55 show a simulation where the host vehicle can take advantage of its initial higher velocity and return to the right lane before the oncoming traffic.

However, if the oncoming traffic prevents the manoeuvre, the host vehicle will wait behind. This behaviour can be seen in Figures 57 and 56.

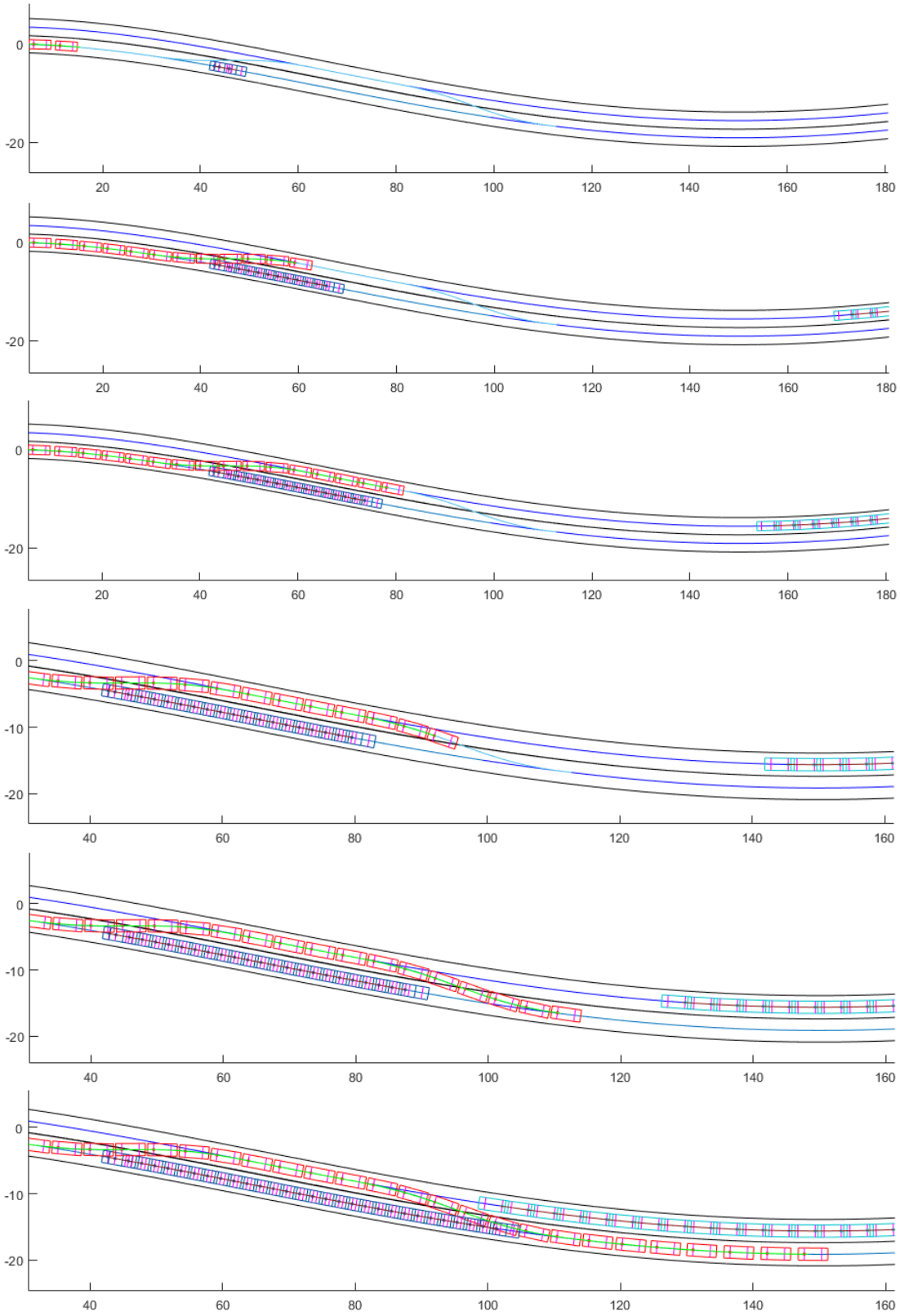


Figure 54: Overtaking of a slower vehicle in a two way track. Aggressive case

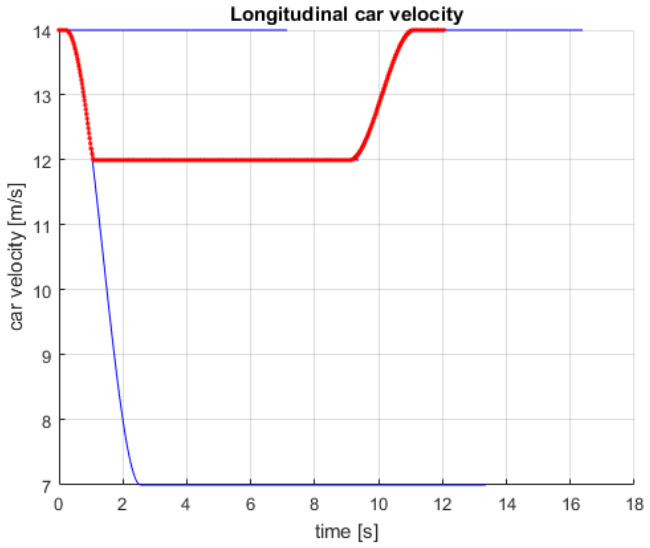


Figure 55: Planned (blue) and followed (red) velocity of the simulation shown in Figure 54, in $[m/s]$

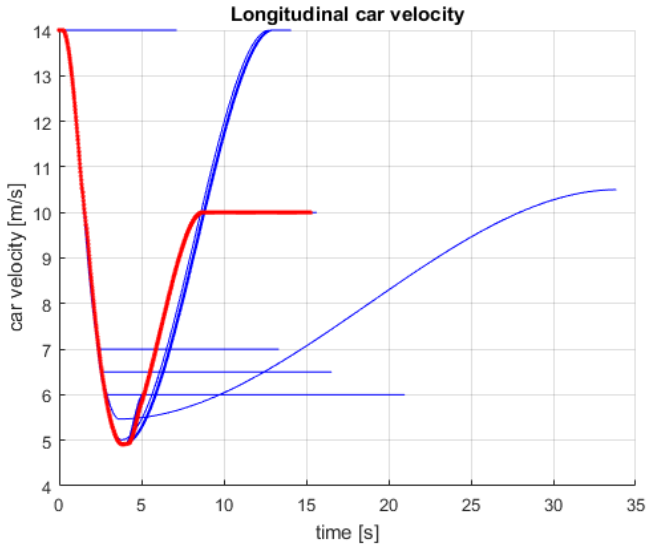


Figure 56: Planned (blue) and followed (red) velocity of the simulation shown in Figure 57, in $[m/s]$

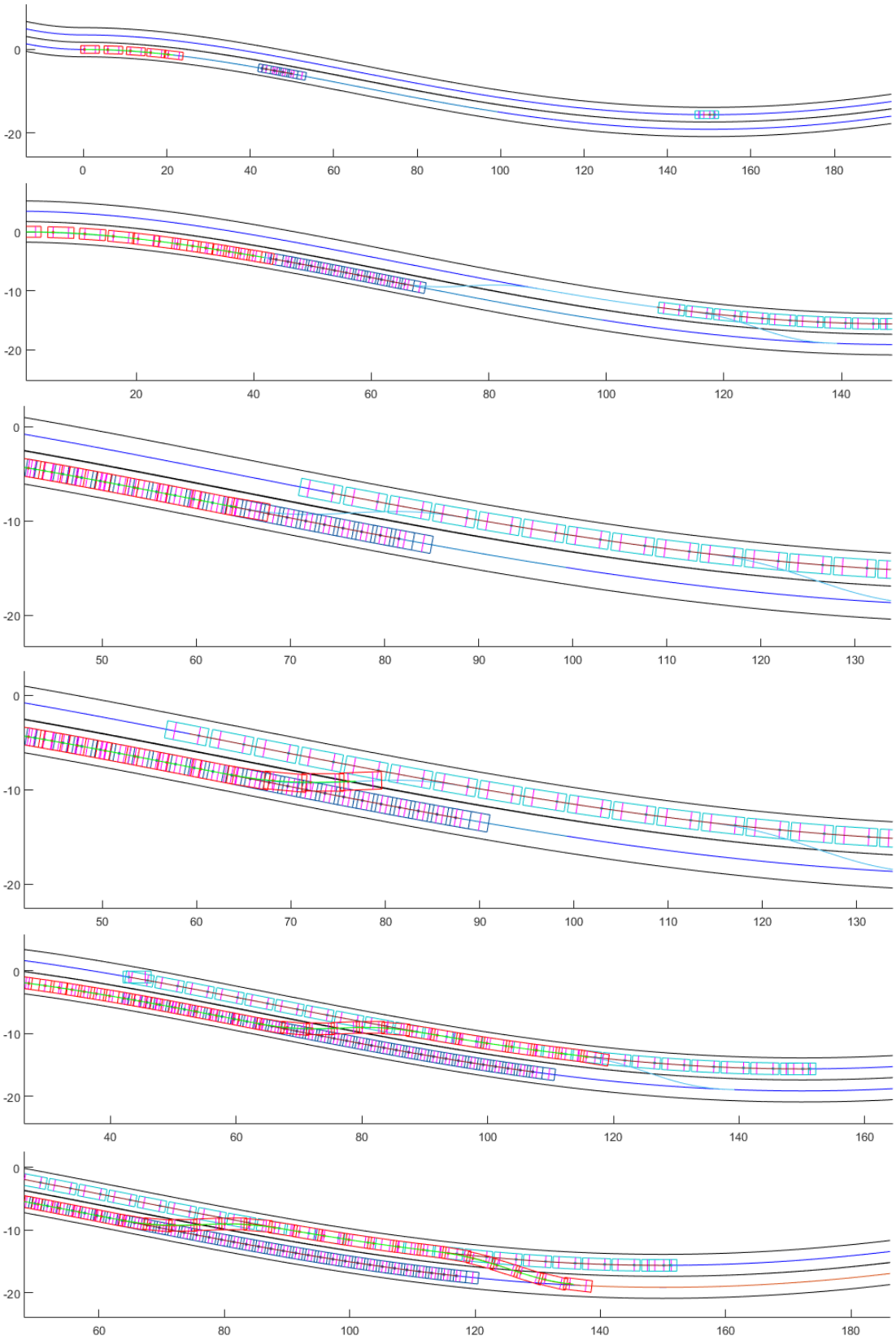


Figure 57: Overtaking of a slower vehicle in a two way track. Conservative case

5 Conclusions and future work

After analysing the simulations, it is verified that the proposed method of the AKRP behaves correctly and it can overcome successfully all the presented situations, which correspond to representative cases that occur in on-road driving.

The proposed approach for the path generation, using the G^2-splines, performs above expectations. Additionally, it is very easy and intuitive to adjust it to follow any road shape, and it has proved to be computationally fast.

The proposed method for the velocity profile generation behaves smoothly and provides a fully analytic solution that can deal with any arbitrary situation.

The used low level control framework is very robust and it can track precisely the desired trajectories. More specifically, the Stanley lateral controller tracks accurately the G^2-splines paths and the designed longitudinal controller follows smoothly any velocity profile. In addition, the controllers require low computation effort and they can be fitted easily to any car-like vehicle.

Regarding the future work, it may be interesting to optimize the current implementation in MATLAB of the algorithms and the simulations. This would make it easier to carry out complex simulations involving several dynamic obstacles. Moreover, the AKRP could be implemented and tested in a real vehicle to prove that it maintains the performance in real environments.

Bibliography

[1] S. O.-R. A. V. S. Committee *et al.*, *Taxonomy and definitions for terms related to on-road motor vehicle automated driving systems*, 2014.

[2] S. Ulbrich, M. Maurer, *Probabilistic Online POMDP Decision Making for Lane Changes in Fully Automated Driving*. 16th International IEEE Conference on Intelligent Transportation Systems (ITSC), pp. 2063-2067, 2013.

[3] B. Paden, M. Cáp, S. Zheng Yong, D. Yershov, E. Frazzoli, *A Survey of Motion Planning and Control Techniques for Self-driving Urban Vehicles*. IEEE Transactions on Intelligent Vehicles, vol. 1, no. 1, pp. 33-55, 2016.

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

[5] S. Thrun, M. Montemerlo *et al.*, *Stanley: The Robot that Won the DARPA Grand Challenge*. Journal of Field Robotics, vol. 23, issue 9, pp. 661-692, 2006.

[6] D. Ferguson, T. M. Howard, M. Likhachev, *Motion Planning in Urban Environments*, Part I and II. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 1063-1069, 1070-1076, 2008.

[7] C. R. Baker, J. M. Dolan, *Traffic Interaction in the Urban Challenge: Putting Boss on its Best Behavior*. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 1752-1758, 2008.

[8] M. Montemerlo, J Becker, S. Thrun *et al.*, *Junior: The Stanford Entry in the Urban Challenge*. Journal of Field Robotics, vol. 25, issue 9, pp. 569-597, 2008.

[9] D. Dolgov, S. Thrun, M. Montemerlo, J. Diebel, *Practical Search Techniques in Path Planning for Autonomous Driving*. Proceedings of the First International Symposium on Search Techniques in Artificial Intelligence and Robotics (STAIR), 2008.

[10] D. Dolgov, S. Thrun, M. Montemerlo, J. Diebel, *Path Planning for Autonomous Vehicles in Unknown Semi-structured Environments*. The International Journal of Robotics Research, vol. 29, issue 5, pp. 485-501, 2010.

[11] D. Ferguson, M. Darms, C. Urmson, S. Kolski, *Detection, Prediction, and Avoidance of Dynamic Obstacles in Urban Environments*. IEEE Intelligent Vehicles Symposium, pp. 1149-1154, 2008.

[12] C. Sprunk, *Planning Motion Trajectories for Mobile Robots Using Splines*. Albert-Ludwigs-Universität Freiburg, 2008.

[13] A. Kelly, B. Nagy, *Reactive Nonholonomic Trajectory Generation via Parametric Optimal Control*. The International Journal of Robotics Research, vol. 22, issue 7-8, pp. 583-601, 2003.

[14] T. Gu, J. M. Dolan, *On-Road Motion Planning for Autonomous Vehicles*. International Conference on Intelligent Robotics and Applications (ICIRA), vol. 3, pp. 588-597, 2012.

[15] D. G. Bautista, J. P. Rastelli, R. Lattarulo, V. Milanés, F. Nashashibi, *Continuous curvature planning with obstacle avoidance capabilities in urban scenarios*. 17th International IEEE Conference on Intelligent Transportation Systems (ITSC), pp. 1430-1435, 2014.

[16] X. Qian, I. Navarro, A. de La Fortelle, F. Moutarde, *Motion Planning for Urban Autonomous Driving using Bézier Curves and MPC*. IEEE 19th International Conference on Intelligent Transportation Systems (ITSC), pp. 826-833, 2016.

[17] J. Ziegler, C. Stiller, *Spatiotemporal state lattices for fast trajectory planning in dynamic on-road driving scenarios*. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 1879-1884, 2009.

[18] M. Werling, J. Ziegler, S. Kammel, S. Thrun, *Optimal Trajectory Generation for Dynamic Street Scenarios in a Frenét Frame*. IEEE International Conference on Robotics and Automation (ICRA), pp. 987-993, 2010.

[19] W. Xu, J. Wei, J. M. Dolan, H. Zhao, H. Zha, *A Real-Time Motion Planner with Trajectory Optimization for Autonomous Vehicles*. IEEE International Conference on Robotics and Automation (ICRA), pp. 2061-2067, 2012.

[20] C. Olsson, *Model Complexity and Coupling of Longitudinal and Lateral Control in Autonomous Vehicles Using Model Predictive Control*. KTH Royal Institute of Technology, 2015.

[21] R. Attia, R. Orjuela, M. Basset, *Combined longitudinal and lateral control for automated vehicle guidance*. Vehicle System Dynamics, International Journal of Vehicle Mechanics and Mobility, vol. 52, issue 2, pp. 261-279, 2014.

[22] K. Soltesz, *Trajectory Tracking Control of an Autonomous Ground Vehicle*. Lund University, 2008.

[23] J. E. A. Dias, G. A. S. Pereira, R. M. Palhares, *Longitudinal Model Identification and Velocity Control of an Autonomous Car*. IEEE Transactions on Intelligent Transportation Systems, vol. 16, issue 2, pp. 776-786, 2015.

[24] K. Jo, J. Kim, D. Kim, C. Jang, M. Sunwoo, *Development of Autonomous Car – Part II: A Case Study on the Implementation of an Autonomous Driving System Based on Distributed Architecture.* IEEE Transactions on Industrial Electronics, vol. 62, issue 8, pp. 5119-5132, 2015.

[25] J. M. Snider, *Automatic Steering Methods for Autonomous Automobile Path Tracking.* Robotics Institute, Carnegie Mellon University, 2009.

[26] G. M. Hoffmann, C. J. Tomlin, M. Montemerlo, S. Thrun, *Autonomous Automobile Trajectory Tracking for Off-Road Driving: Controller Design, Experimental Validation and Racing.* American Control Conference (ACC), pp. 2296-2301, 2007.

[27] C. G. Bianco, A. Piazzi, *Optimal trajectory planning with quintic G^2- splines.* Proceedings of the IEEE Intelligent Vehicles Symposium, pp. 620-625, 2000.

Annex A. Longitudinal controller validation and testing with Simulink

After a lot of testing and simulations with the longitudinal vehicle model presented in section 3.4.1, the controller gains have been adjusted and tuned with the values shown in Table 4.

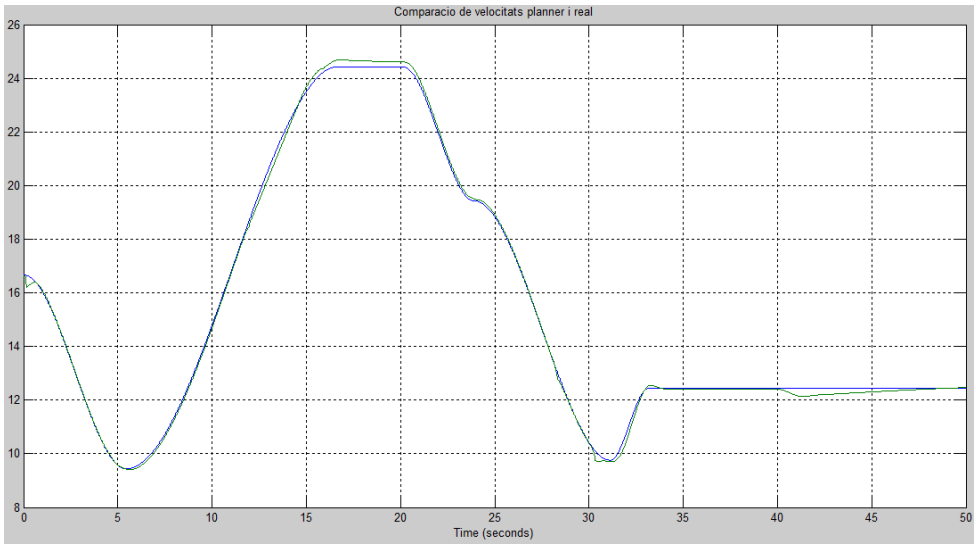
Gain	value
K_p	0,5
K_i	0,1
K_v	4,0

Table 4: Adjusted gains for the longitudinal controller

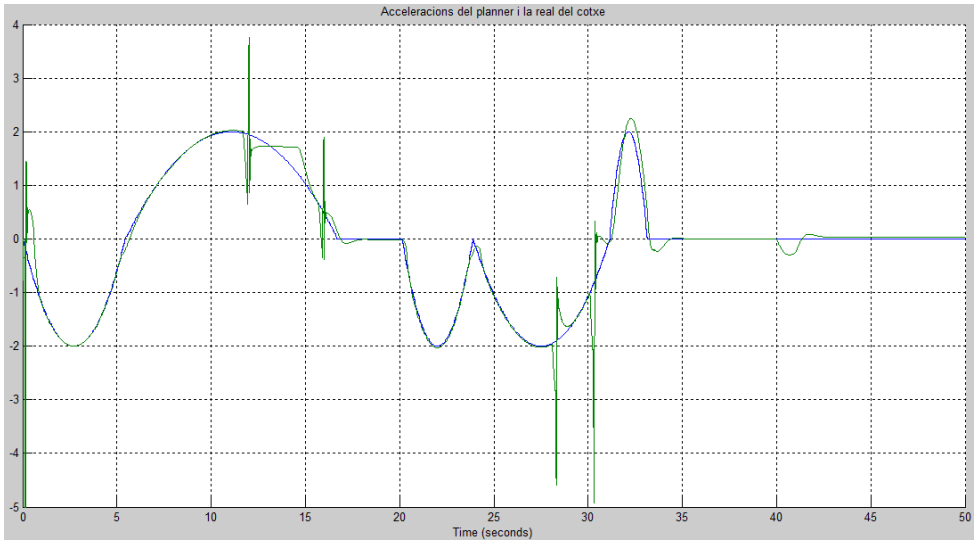
In this section, the longitudinal controller is tested following a mixed trajectory with wide velocity variations to prove its performance against gear changes. It has been tested at 100 Hz, but operates correctly in much lower frequencies. In the 40th second there is also an uphill with constant 5 degrees slope until the end in order to test the controller against other external perturbations.

As it is seen in Figure 58a, the desired (in blue) and real velocity (in green) are very close. Note that in the 40th second there is a slight deviation (corresponding to the uphill) that is corrected by the integral part of the controller. In the Figure 59b it can be seen that the maximum velocity error is around $\pm$ 0,3 m/s so it achieves very good performance. The Figure 59a shows a maximum position error around 1,1 m that, taking into account the magnitude of the velocity, is also a very good result.

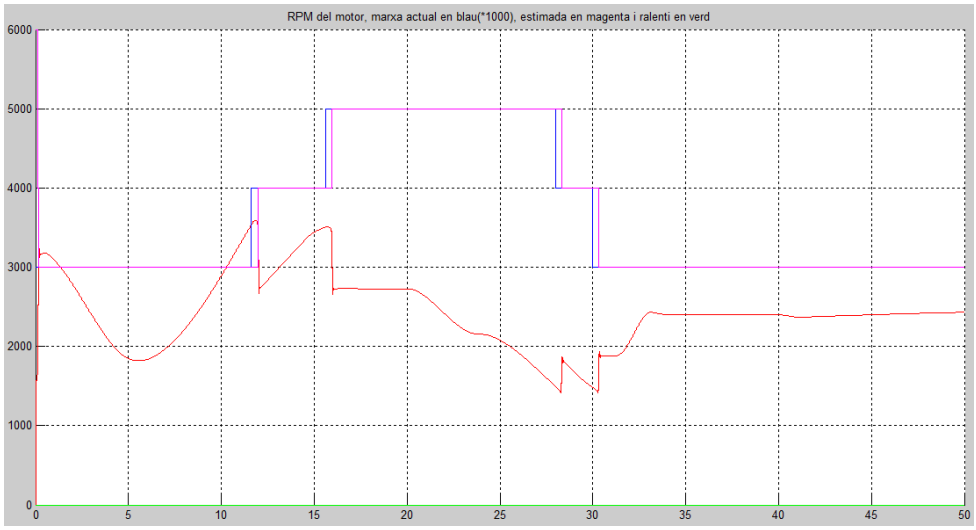
In the Figure 58b the accelerations can be seen. Note that both lines fits very well, but there are some peaks: These correspond to every gear change that the model realizes. This peaks are not important for the real plant as they are numerical issues of the simulation due to the used model. In the Figure 58c the engine speed in min^{-1} and the estimated gear are shown.



(a) Desired (in blue) and real velocity (in green), in $[m/s]$

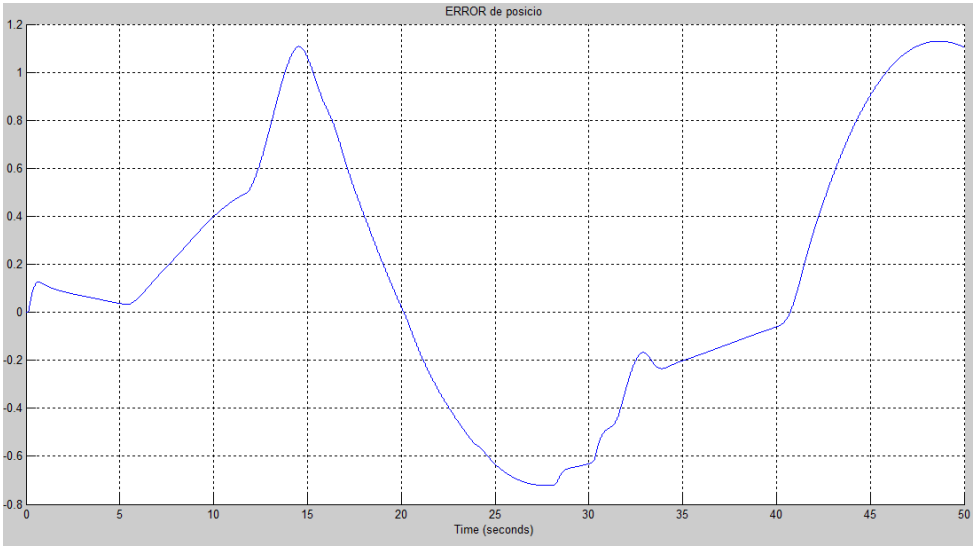


(b) Desired (in blue) and real accelerations (in green), in $[m/s^2]$

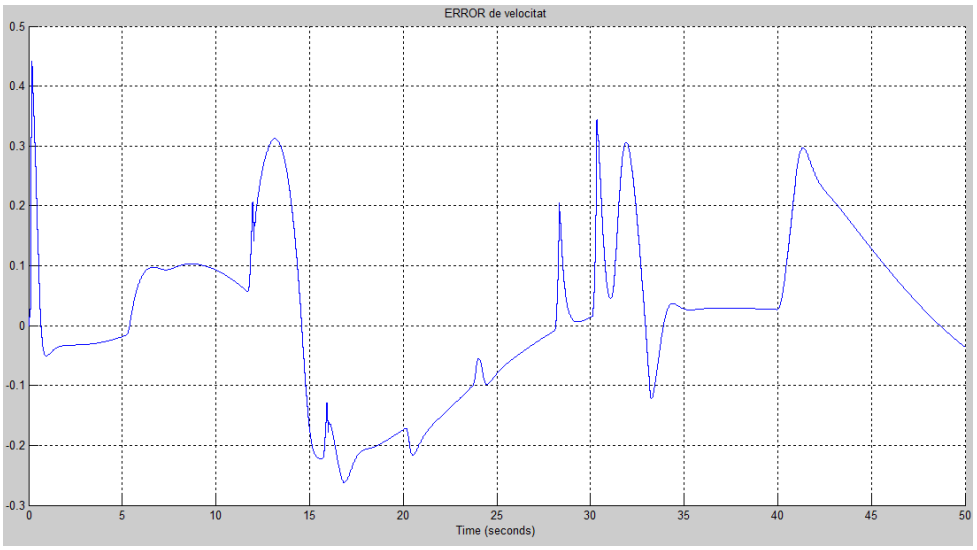


(c) Engine speed in $[min^{-1}]$ (red) and estimated gear (magenta)

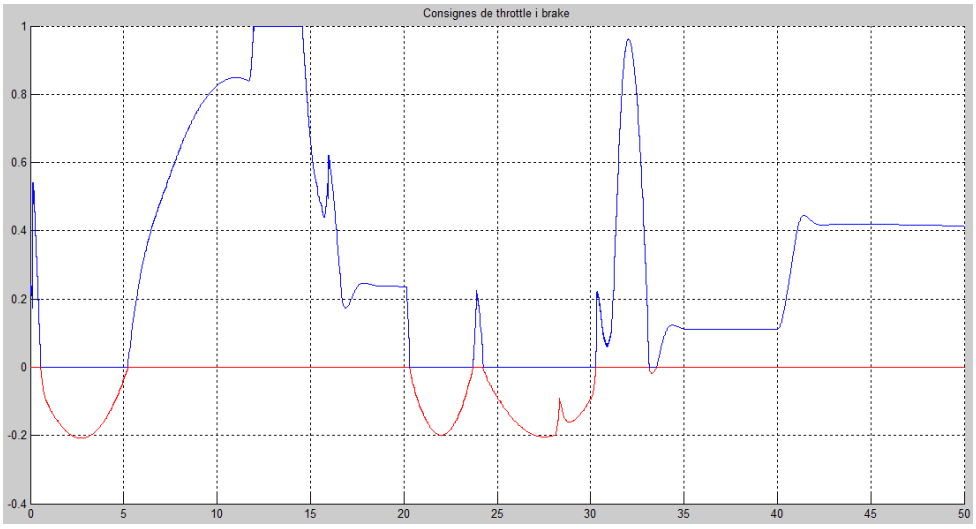
Figure 58: Longitudinal controller performance over time, in $[s]$ (1)



(a) Position error, in $[m]$



(b) Velocity error, in $[m/s]$



(c) Throttle signal (in blue) and brake signal (in red)

Figure 59: Longitudinal controller performance over time, in $[s]$ (2)

Annex B. Lateral controller validation and testing with Simulink

After a lot of testing and simulations, the Stanley controller gains have been adjusted and tuned with the values shown in Table 5.

Gain	Value
K	2,0
K_{soft}	1,0
$K_{d,yaw}$	0,5
$K_{d,steer}$	10,0

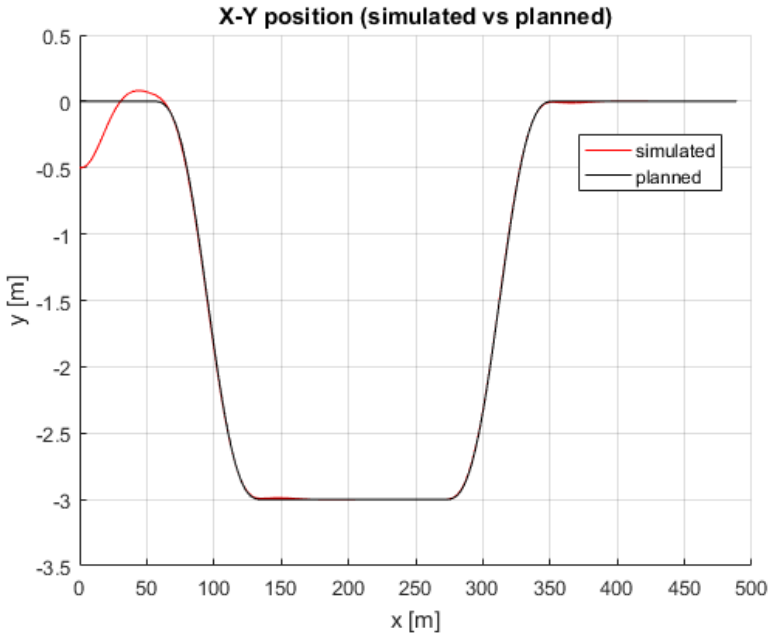
Table 5: Adjusted gains for the Stanley controller

The controller is affected by the control loop frequency, having a low frequency can lead to bad performance or even instability. At a reasonable rate of 50 Hz the controller works perfectly.

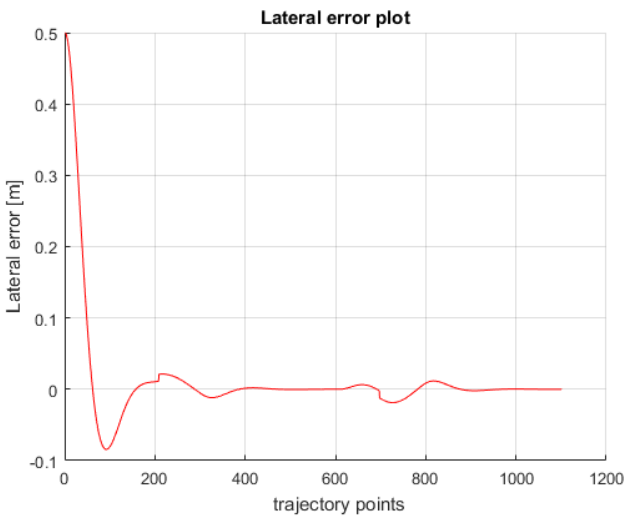
It has been tested in a standard lane change and a simple circuit made with circumferences. This controller has an intuitive control law, and this allows an easy tuning process. As it can be seen in Figures 60 and 61, the simulated path is almost all the time overlapped with the desired one, and looking to the scale of the lateral error, it can be observed that the maximum errors have a value of less than 3 cm.

This lateral controller does not present overshoot, neither it cuts the corners of the path, and this is translated in an exceptional performance. In addition, the damping terms ensure stability and testing even with higher velocities, up to 150 km/h, the performance is similar.

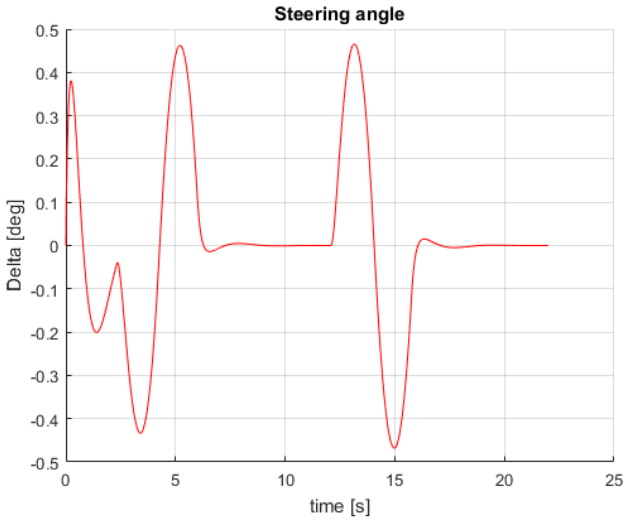
Looking at the shape of the steering angle in the lane change trajectory (Figure 60c) it can be observed how it is very smooth.



(a) X-Y trajectories, in $[m]$

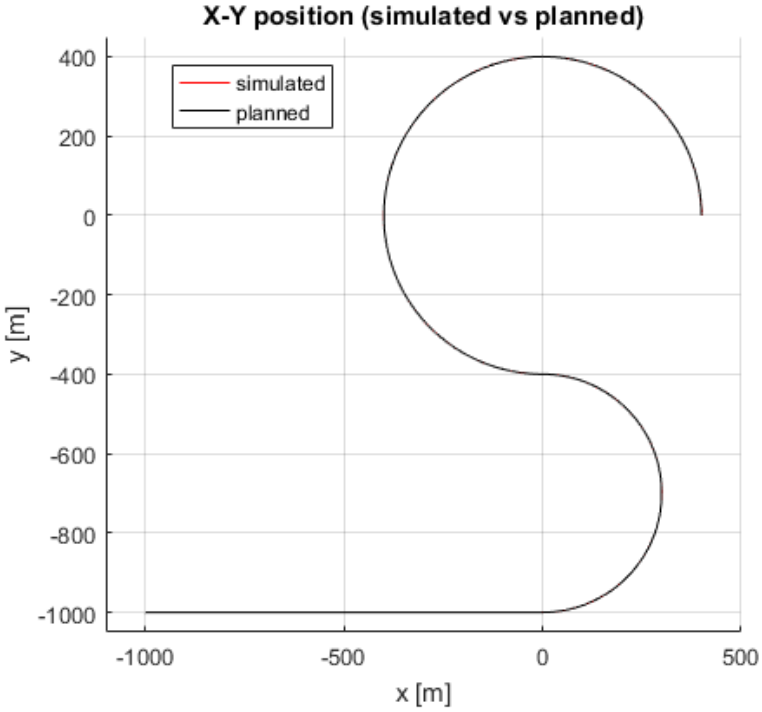


(b) Lateral error (e_{lat}) over time, in $[m]$

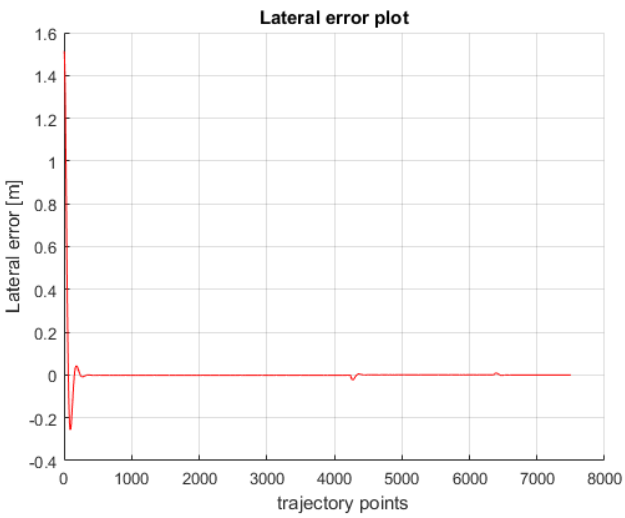


(c) Steering angle (δ) over time, in $[deg]$

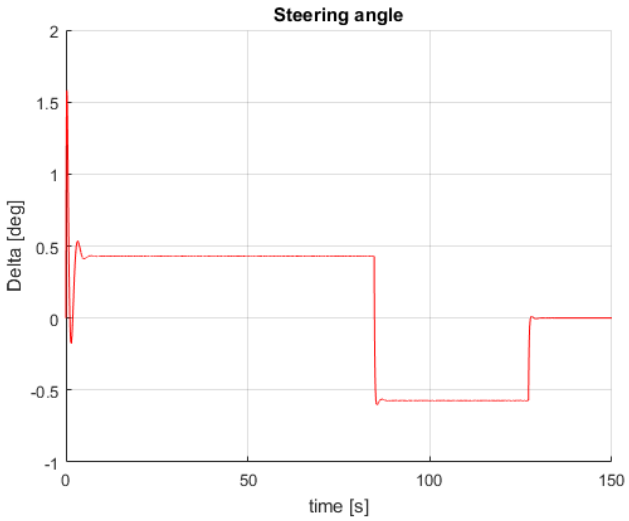
Figure 60: Stanley controller in a dual lane change example at 80 km/h, starting with an offset of 0,5 m to the right. The path has been built with the G^2-splines. The manoeuvre has a maximum lateral acceleration of 2 m/s^2



(a) X-Y trajectories, in $[m]$



(b) Lateral error (e_{lat}) over time, in $[m]$



(c) Steering angle (δ) over time, in $[deg]$

Figure 61: Stanley controller in a generic road example at 80 km/h, starting with an offset of 1,5 m to the right. First circle has a turning radius of 400 m and the second of 300 m