POLITECNICO DI TORINO

Master degree course in Automotive Engineering

Master Degree Thesis

Autonomous driving: Model Predictive Control for overtaking manoeuvres

Supervisors
Prof. Carlo Novara
Eng. Francesco Polia

Candidate
Federica Bonanno
enrollment: 234734

April 2019

Abstract

In these last years, the autonomous driving is becoming a very popular trend in the automotive field. Started as a research topic, it has quickly turned into a concern of several companies around the world. Among the considerable technical problems this technology has to face, there's the path planning, key function for an autonomous vehicle. Thus the main objective of this thesis work is to investigate the path planning techniques for autonomous driving vehicles in an overtaking scenario. As first step, a preliminary research on motion planning methods is carried, in order to have an overview of the problem, in particular focusing on the most suitable techniques for automotive purposes. Then, a focus on the Model Predictive Control was done, highlighting its advantages and showing some applications in the automotive field. In order to compare the MPC performance as path planner, a RRT* path planner was set in a preliminary phase of the work. In a second phase, the MPC was tested both as path planner and as controller, working with a kinematic model and a barrier as obstacle. Lastly, the path planning function of the MPC was isolated, resulting in a better performance in comparison with the RRT* planner. An overtaking of a moving vehicle was achieved at high speeds, denoting the already underlined advantages of the MPC. Further improvements can be done using more accurate vehicle models,i.e. dynamic one and a sensor system can be simulated to gather all environmental data.

Acknowledgements

I would like to offer my special thanks to my supervisor prof. Carlo Novara for his support and willingness throughout all the steps of this project.
Besides, I am particularly grateful for the assistance and patience given by my advisor Francesco Polia. Without his kind guidance a part of this project would not have been possible.

I would like also to extent my thanks to my friends and colleagues of Politecnico, with whom I shared this journey and a special acknowledgement to Raquel, Leticia, Shao and Yu for being my international study pals.

Thanks to my oldest friends Alessia, Astra, Beatrice, Irina and Viviana for standing together despite the distance. Thanks to my dear friend Alice for being always there and for sharing Sundays with me at the basketball arena. Thanks also to my ex-colleagues Andrea, Emanuele and Riccardo for showing me sympathy and support in the studying time back in Trieste.

A special thanks to my flatmate Giuseppe for being an excellent cook, a good listener and a wonderful friend.

Last but not least, I would like to express my deep gratitude to my family for all the love and support I had throughout the years. None of this would be possible without them, in particular thanks to my father Carlo for inspiring my curiosity everyday and to my mother Cristiana for being my role model.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the past few years the concept of autonomous vehicles has become more common and discussed both by academic and industrial environment. Public opinion could see the results and problems about this technology and at the moment seems to be divided for this possible revolution in the mobility field. This wide discussion is also due to the fact that this technology is involving not only the automotive engineering but also computer science, robotics, transportation, urban planning, legal and social science. The passage to a complete self-driving car in fact is a difficult task to accomplish and accept not only from the point of view of engineers but also for the users. In any case many automated systems for mobility are already present, such as urban metro subways and trains and recent developments in sensors, actuators and control strategies has brought to big steps ahead. Even if a complete self-driving car still doesn't exist on the market, it seems not so far to happen.

1.1 Background: Autonomous car technology

An autonomous car (also known as a driverless car and a self-driving car) is a vehicle that is capable of sensing its environment and navigating without human input [19]. Autonomous cars combine a variety of techniques to perceive their surroundings, including radar, laser light, GPS, odometry, and computer vision. Advanced control systems interpret sensory information to identify appropriate navigation paths, as well as obstacles and relevant signage. We can sum up all these functions in five main ones:

- Perception
- Localization
- Planning
- Vehicle control
- System management

Already from only these five functions the situation seems pretty complex. A decision-making hierarchy is needed. At the highest level a route is planned through the road network. By representing the road network as a directed graph with edge weights corresponding to the cost of traversing a road segment, such a route can be formulated as the problem of finding a minimum-cost path on a road network graph. This is followed by a behavioral layer, which decides on a local driving task that progresses the car towards the destination and abides by rules of the road. A motion planning module then selects a continuous path through the environment to accomplish a local navigational task. A control system then reactively corrects errors in the execution of the planned motion [33].

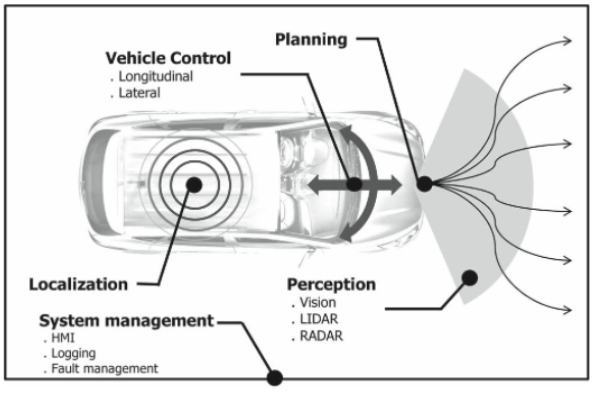


Figure 1.1: Autonomous vehicle's functions

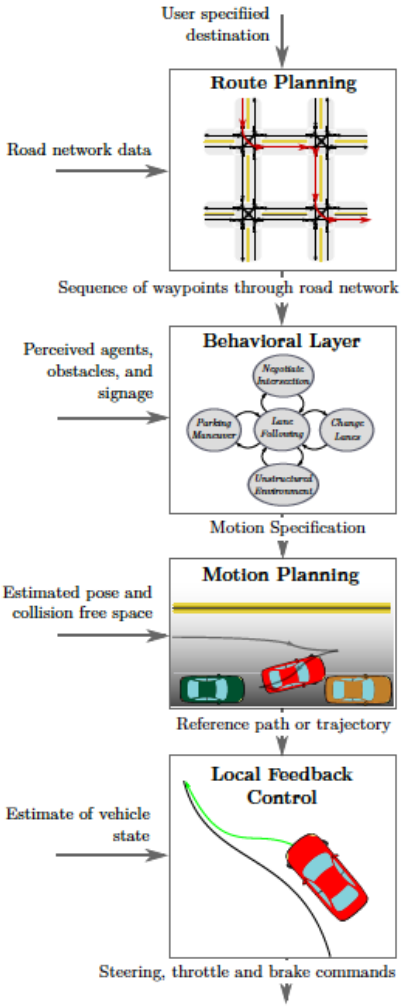


Figure 1.2: Hierarchy of decision-making process

1.1.1 Automation levels

A classification system based on six different levels (ranging from fully manual to fully automated systems) was published in 2014 by SAE International, an automotive standardization body, as J3016, Taxonomy and Definitions for Terms Related to On-Road Motor Vehicle Automated Driving Systems. This includes everything from no automation (where a fully engaged driver is required at all times), to full autonomy (where an automated vehicle operates independently, without a human driver). Also in 2018, SAE updated its classification, called J3016_201806 [18].
The levels are reported as in NHTSA website [1], National Highway Traffic Safety Administration.

Level 0: The human driver does all the driving.
Level 1: The driver and the automated system share control of the vehicle. Examples are Adaptive Cruise Control (ACC), where the driver controls steering and the automated system controls speed; and Parking Assistance, where steering is automated while speed is under manual control. The driver must be ready to retake full control at any time. Lane Keeping Assistance (LKA) Type II is a further example of level 1 self driving.
Level 2: An advanced driver assistance system (ADAS) on the vehicle can itself actually control both steering and braking/accelerating simultaneously under some circumstances. The human driver must continue to pay full attention (“ monitor the driving environment ”) at all times and perform the rest of the driving task. In fact, contact between hand and wheel is often mandatory during SAE 2 driving, to confirm that the driver is ready to intervene.
Level 3: The driver can safely turn their attention away from the driving tasks, e.g. the driver can text or watch a movie. The vehicle will handle situations that call for an immediate response, like emergency braking. The driver must still be prepared to intervene within some limited time, specified by the manufacturer, when called upon by the vehicle to do so.
Level 4: An Automated Driving System (ADS) on the vehicle can itself perform all driving tasks and monitor the driving environment (essentially, do all the driving) in certain circumstances. Self driving in fact is supported only in limited spatial areas (geofenced) or under special circumstances, like traffic jams. Outside of these areas or circumstances, the vehicle must be able to safely abort the trip, i.e. park the car, if the driver does not retake control.
Level 5: No human intervention is required at all. An example would be a robotic taxi.

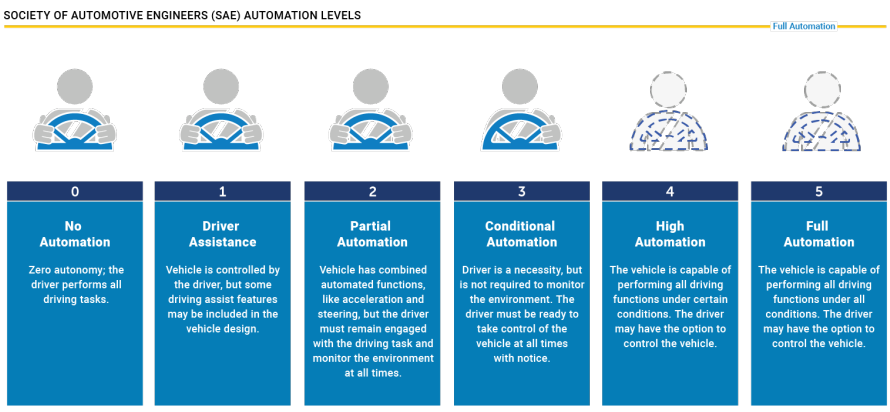


Figure 1.3: Automation levels for SAE

In the formal SAE definition, note in particular what happens in the shift from SAE 2 to SAE 3: the human driver no longer has to monitor the environment. This is the final aspect of the "dynamic driving task" that is now passed over from the human to the automated system. At SAE 3, the human driver still has the responsibility to intervene when asked to do so by the automated system. At SAE 4 the human driver is relieved of that responsibility and at SAE 5 the automated system will never need to ask for an intervention.

1.1.2 Benefits and current situation

The potential benefits are quite a few and pretty clear. First of all safety is increased, that brings to a reduction in traffic collisions, injuries and costs. Then an autonomous car can have no matter who as passenger, so it provides a safe mobility also for children, elderly and disabled people. The news about Steve Mahan, a blind man, in the Google self-driving car spread all over the world in 2012. Moreover the driving style will be smoother and more constant, so also fuel consumption can decrease. Since the cost of a vehicle would be quite high, sharing economy will be the perfect solution, facilitating business models for transportation as a service.
The challenge for driverless car designers is to produce control systems capable of analyzing sensory data in order to provide accurate detection of other vehicles and the road ahead. Modern self-driving cars generally use Bayesian simultaneous localization and mapping (SLAM) algorithms,which fuse data from multiple sensors and an off-line map into current location estimates and map updates. Google is developing a variant called SLAM, with detection and tracking of other moving objects (DATMO), which also handles obstacles such as cars and pedestrians. Simpler systems may use roadside real-time locating system (RTLS) technologies to aid localization. Typical sensors include Lidar, stereo vision, GPS and IMU. Udacity is developing an open-source software stack. Control systems on autonomous cars may use Sensor Fusion, which is an approach that integrates information from a variety of sensors on the car to produce a more consistent, accurate, and useful view of the environment.
As reported in [28] the human decision-making system can be considered the most efficient of all known such systems and can serve as a standard for auto-adaptation. For this reason artificial intelligence,which is any technique that enables computers to mimic human behavior, is the main solution for the decision making system in self driving cars. Autonomous cars are being developed with deep neural networks, a type of deep learning architecture with many computational stages, or levels, in which neurons are simulated from the environment that activate the network. The neural network depends on an extensive amount of data extracted from real-life driving scenarios,enabling the neural network to "learn" how to execute the best course of action.

In May 2018, researchers from MIT together with Toyota Research Institute announced that they had built an autonomous car that can navigate unmapped roads [31]. Researchers at their Computer Science and Artificial Intelligence Laboratory (CSAIL) have developed a new system, called MapLite, which allows self-driving cars to drive on roads that they have never been on before, without using 3D maps. The system combines the GPS position of the vehicle and LIDAR. The LIDAR is used to create a 3D point cloud and the edges of the road are estimated.
As soon as this technology started to be interesting for the market and not only for research groups and DARPA competitions, a lot of car manufactures, BMW, Mercedes Benz, Audi, Tesla, FCA just to name a few, decided to open the research in this field together with new emerging companies dedicating themselves only to this purpose such as Waymo and AID. Also Apple is currently testing self-driven cars, and has increased the number of test vehicles from 3 to 27 in January 2018. This number further increased to 45 in March 2018. Testing vehicles with varying degrees of autonomy can be done physically, in closed environments, on public roads (where permitted, typically with a license or permit or adhering to a specific set of operating principles) or virtually, i.e. in computer simulations, which is preferred especially in the first steps of the project since the great amount of problem in outside testing.
One way to assess the progress of autonomous vehicles is to compute the average number of miles driven between "disengagements", when the autonomous system is turned off, typically by a human driver. In particular, Lex Fridman, MIT researcher, did a review of what has been done in 2018, reporting two main results [16]. Waymo reached 10 million miles, while Tesla, with its Autopilot, reached 1 billion miles.

1.2 Objective and outline of the thesis

As explained in the previous section, the autonomous car has five basic functions to achieve. The planning phase is analyzed in this thesis work, in particular the local navigation task is treated in detail. To this purpose, the objective of this thesis is to design a local path planner for autonomous driving vehicles in the scenario of an overtaking, which in more general terms can be described as obstacle avoidance.
This thesis work was conducted in DAUIN, Department of Control and Computer Engineering of Politecnico of Torino, in particular related to the Prystine project, in which Politecnico of Torino takes part.

The report is structured in the following way:

- ***Chapter 2***: a state of art regarding the motion planning techniques is reported, starting from basic concepts of robotics, passing to an overview of the possible algorithms dealing with the path planning problem.
- ***Chapter 3***: the model predictive control method is introduced in its general terms to explain its working principle. A brief review of its application in automotive is done, followed by a focus on the computational problem of the MPC itself.
- ***Chapter 4***: the case study scenario is introduced, explaining all the possible methods proposed in literature in order to manage it properly.
- ***Chapter 5***: all the phases of the implementation process are illustrated. The design of the local path planner and related controller is divided in three phases, where RRT* planner, MPC and LQR controller were tested.
- ***Chapter 6***: the results and conclusions are presented, retracing the aforementioned phases of the implementation

Chapter 2

Motion planning: State of Art

In this chapter a state of art regarding the motion planning problem is reported. A brief introduction about the problem is done, explaining all the basic concepts of this wide subject. Then a focus on the principal path planning algorithms is proposed, highlighting the most suitable for automotive applications.

2.1 Basic concepts

Generally speaking, motion planning (also known as the navigation problem or the piano mover's problem) is "a term used in robotics for the process of breaking down a desired movement task into discrete motions that satisfy movement constraints and possibly optimize some aspect of the movement." [40]

Before explaining what a motion planning problem is, let's introduce some useful definitions for dimensioning the problem. The following concepts are taken from the book "Planning Algorithms" of S. M. LaValle [22].

State space S: Planning problems involve a state space that captures all possible situations that could arise. The state could, for example, represent the position and orientation of a robot, the locations of tiles in a puzzle, or the position and velocity of a car; the state space in this last case would be the set of all possible configurations for the car. Then we can introduce the free space and the target space. *Free space* is the set of configurations that avoids collision with obstacles. Often, it is prohibitively difficult to explicitly compute the shape of the free space. However, testing whether a given configuration is in this space is efficient. Strictly in robotic field, at first, forward kinematics determine the position of the robot's geometry, and collision detection tests if the robot's geometry collides with the environment's geometry.

$$\mathrm{S} = \mathrm{S}_{free} + \mathrm{S}_{obs} \tag{2.1}$$

where S_{obs} is the set of configurations of the robot where it is in collision with an obstacle.

Target space is a linear subspace of free space which denotes where we want the robot to move to. At this point the state variable of the robot can be the configuration vector itself or the configuration vector with the velocities; it depends on the application. Both discrete (finite, or countably infinite) and continuous (uncountably infinite) state spaces will be allowed. One recurring theme is that the state space is usually represented implicitly by a planning algorithm. In most applications, the size of the state space (in terms of number of states complexity) is much too large to be explicitly represented. Nevertheless, the definition of the state space is an important component in the formulation of a planning problem and in the design and analysis of algorithms that solve it.

Actions: A plan generates actions that manipulate the state. The terms actions and operators are common in artificial intelligence; in control theory and robotics, the related terms are inputs and controls. Somewhere in the planning formulation, it must be specified how the state changes when actions are applied. This may be expressed as a state-valued function for the case of discrete time or as an ordinary differential equation for continuous time, as written in the equation (2.2):

$$\dot{x} = f(x, u) \tag{2.2}$$

where u is the input.

Initial and goal states: A planning problem usually involves starting in some initial state $s(0) = s_{start}$ being s in function of the time t and trying to arrive at a specified goal state s_{goal} or any state in a set of goal states.

A criterion: This encodes the desired outcome of a plan in terms of the state and actions that are executed. There are generally two different kinds of planning concerns based on the type of criterion:

- **Feasibility**: Find a plan that causes arrival at a goal state, regardless of its efficiency.
- **Optimality**: Find a feasible plan that optimizes performance in some carefully specified manner, in addition to arriving in a goal state.

So a basic motion planning problem is to produce a sequence of control inputs so to drive the robot from its initial configuration in the state space to the target ones, while following the rules of the environment and optimizing the performance. Moreover we can have some variations. We can plan a full trajectory with timing information or just a collision free geometric path. The robot can have a full actuation, meaning we can control all the degrees of freedom or just a few, such as in the car case. The planner can do changes online as it moves in the environment or it can do it in advance before moving (offline).

Properties: A part from all considerations done till now we can mention also these properties:

- *Multiple query vs single query*: the planner may be designed for multiple queries or single query. A multiple query planner is one that invests time in developing a good representation of state space so that future motion planning problems in that space can be solved quickly. If the state space changes often however a single query planner attempts to find the solution to a single motion planning problem as quickly as possible.
- *Completeness*: we say that the planner is complete if it always finds a solution when one exists. A weaker version of this notion is resolution completeness; a planner is resolution complete if it always finds a solution when one exists at the level of discretization employed in the representation of the problem. There is also probabilistic completeness; a planner is probabilistically complete if the chances of it finding a solution,if one exists,go to 100% as the execution time goes to infinity.
- *Computational complexity*: it refers to how much memory a planner will use or how long the planner will take to execute.

The core of the motion planning problem is the algorithm that governs the entire motion, having as input the description of the task and as output the commands to move properly the robot.

2.2 Planning algorithms

In order to generate the movement, during the years, many control strategies and algorithms, based on different theoretical notions, have been created. Thus, we will classify these types of algorithm in two categories, in particular we will focus on the most known and useful in creating a feasible path for our aim.

- Deterministic
- Probabilistic

The first category gives the same output given a certain input. In fact, it computes a mathematical function and during this process, the machine will always pass through the same sequence of states. Instead, we call an algorithm randomized or probabilistic if "its behavior is determined not only by its input but also by values produced by a random-number generator" [11]. This is done with the aim of obtaining a good perfomance in the average case over all possible choices of random bits, when there's no knowledge about the inputs distribution. Probabilistic path

planners provide solutions to problems involving wide, high-dimensional configuration spaces that would be unmanageable and uncontrollable using deterministic approaches. The downside of these methods is that they are only probabilistically complete, i.e., the probability of finding a solution to the planning problem when one exists tends to 1 as the execution time tends to infinity. This means that, if no solution exists, the algorithm will run indefinitely [35].

2.2.1 Deterministic algorithms

Dijkstra's algorithm Dijkstra's algorithm solves the single-source shortest-paths problem on a weighted directed graph. In other words, it finds a path between two nodes in a graph such that the sum of the weights of its constituent edges is minimized. It was conceived by computer scientist Edsger W. Dijkstra in 1956 and published three years later. It is considered a graph search algorithm since, systematically, follows the edges of the graph to visit or discover its nodes in order to find the goal node through a cost minimum path. We report below the steps in the algorithm.

Let the node at which we are starting be called the initial node. Let the distance of node Y be the distance from the initial node to Y. Dijkstra's algorithm will assign initial distance values and will try to improve them step by step.

1. Mark all nodes unvisited. Create a set of all the unvisited nodes called the unvisited set.
2. Assign to every node a tentative distance value: set it to zero for our initial node and to infinity for all other nodes. Set the initial node as current.
3. For the current node, consider all of its unvisited neighbors and calculate their tentative distances through the current node. Compare the newly calculated tentative distance to the current assigned value and assign the smaller one. For example, if the current node A is marked with a distance of 6, and the edge connecting it with a neighbor B has length 2, then the distance to B through A will be $6 + 2 = 8$. If B was previously marked with a distance greater than 8 then change it to 8. Otherwise, keep the current value.
4. When we are done considering all of the unvisited neighbors of the current node, mark the current node as visited and remove it from the unvisited set. A visited node will never be checked again.
5. If the destination node has been marked visited (when planning a route between two specific nodes) or if the smallest tentative distance among the nodes in the unvisited set is infinity (when planning a complete traversal; occurs when there is no connection between the initial node and remaining unvisited nodes), then stop. The algorithm has finished.

6. Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new "current node", and go back to step 3.
When planning a route, it is actually not necessary to wait until the destination node is "visited" as above: the algorithm can stop once the destination node has the smallest tentative distance among all "unvisited" nodes (and thus could be selected as the next "current").

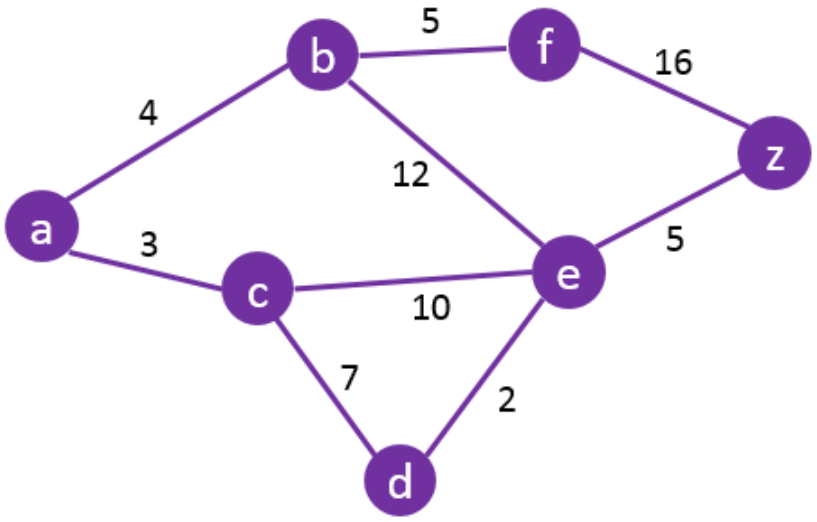


Figure 2.1: Example of Dijkstra algorithm graph

A* Peter Hart, Nils Nilsson and Bertram Raphael of Stanford Research Institute first published the algorithm in 1968 [17]. It can be seen an extension of Edsger Dijkstra's 1959 algorithm. A* achieves better performance by using heuristics to guide its search. This algorithm exploits weighted graphs and it aims to find the optimal path from the starting node to the goal one having the the smallest cost, i.e. least distance traveled. In this way it doesn't cover all the nodes of the graph but at each iteration it decides which branch to extend. In order to do this it evaluates the cost of the path and the cost to extend the path all the way to the goal. The new expanding node is, then, chosen as the node with the minimum cost, reducing hence the number of processed cells and finding an optimal path as long as the heuristic is admissible. Translating in mathematical terms, A* chooses the path that minimizes the following function:

$$f(s) = g(s) + h(s) \tag{2.3}$$

where s is the last node on the path, $g(s)$ is the cost of the path from the start node to s,i.e. the movement cost and $h(s)$ is a heuristic function that estimates the cost of the cheapest path from s to the goal,i.e. the distance from the current checking node to the goal one. The algorithm ends when the path chosen to be extended is from start to goal configuration. However Dijkstra and A* can be slow and memory expensive and moreover usually we don't need to find the optimal path, but just one that is good enough. For these reasons other modifications of A* were proposed such as weighted A*, WA*.

WA* WA* is a variant of A* [27] where an inflation factor η is multiplied by the heuristic function, as reported in the equation 2.4:

$$f(s) = g(s) + \eta h(s) \tag{2.4}$$

where $\eta > 1$ and for $\eta = 1$ we have A*. This inflation factor provides a greedy flavor to the search, which finds a solution in less time at the cost of optimality.
Other variants have been proposed such as: DA*, Hybrid A* and Anytime Repair A*.

D* It's an incremental search algorithm and can be expressed in one of the three following ways:

1 Original D*

2 Focussed D*

3 D* Lite

The first one was ideated by A. Stentz in 1994 and the name D* stands for Dynamic A* Search. Functionally speaking in fact, it's equivalent to A* but the arc

cost parameters can change during the problem solving process, thus dynamic. All three search algorithms have the same assumptions where the robot has to navigate in unknown environment, with the concept of free space and goal position [10]. It starts assuming that in the environment there are no obstacles and finds a shortest path from its current coordinates to the goal coordinates. The robot then follows the path. If during the navigation a new obstacle is detected, this information is added to its map and, if necessary, replans a new shortest path from its current coordinates to the given goal coordinates. It repeats the process until it reaches the goal coordinates or determines that the goal coordinates cannot be reached. When traversing unknown terrain, new obstacles may be discovered frequently, so this replanning needs to be fast. Incremental (heuristic) search algorithms (such as Focussed D*) speed up searches for sequences of similar search problems by using experience with the previous problems to speed up the search for the current one. D* Lite is simply algorithmically different and as the name suggests it has less operations to perform, in order to reduce the time performance. Assuming the goal coordinates do not change, all three search algorithms are more efficient and complete than repeated A* searches.

Artificial potential field The basic idea is that the robot is moving in an abstract artificial force field [38]. The force field is created so that the robot once inside is brought towards its goal position. This is done creating a potential field with two components: an attractive and a repulsive one. The first one will be attached to the goal position, while the second one to the obstacles. The field can be set with different methods, we report for example the quadratic one in the equation (2.5):

$$U_{att}(q) = \frac{1}{2}\xi\rho^2(q, q_{goal}) \tag{2.5}$$

where ξ is a positive scaling factor and $\rho(q, , q_{goal})$ is the distance between robot and goal. The repulsive field instead can be expressed as:

$$U_{rep}(q) = \begin{cases} \frac{1}{2}\eta(\frac{1}{\rho(q,q_{obs})} - \frac{1}{\rho_0}) & \text{if } \rho(q, q_{obs}) \leq \rho_0 \\ 0 & \text{if } \rho(q, q_{obs}) > \rho_0 \end{cases} \tag{2.6}$$

where η is a positive scaling factor, $\rho(q, q_{obs})$ is the shortest distance between robot and obstacle and ρ_0 is the largest distance of the obstacle. These fields act on the robot through forces, so the total motion of the robot is due to the sum of attractive and repulsive forces. As main disadvantages we can mention the fact that if the goal point is really far from the robot, the attractive force will be great and this could lead to move robot too close to the obstacles. Another weak point is that the robot can easily be stuck when attractive and repulsive forces are quite equal.

2.2.2 Probabilistic algorithms

Instead of using an explicit representation of the environment, probabilistic algorithms rely on a collision checking module, providing information about feasibility of candidate trajectories, and connect a set of points sampled from the obstacle-free space in order to build a graph of feasible trajectories. Even though these algorithms are often not complete, they provide probabilistic completeness, i.e., it guarantees that the probability that the planner fails to return a solution, if one exists, decays to zero as the number of samples approaches infinity.

Probabilistic roadmap PRM Presented in 1996 by Kavraki et al. it's one of the main algorithms of the category [37]. It's born mainly for performing a path planning for robots with many dof (more than five) but nowadays it's widely use for many applications, being highly customizable. The method proceeds in two phases: a learning phase and a query phase. In the learning phase a probabilistic roadmap is constructed by repeatedly generating random free configurations of the robot and connecting these configurations using some simple, but very fast motion planner. We call this planner the local planner. The generated roadmap, formed in the free space, is then stored as an undirected graph R. The configurations are the nodes of R and the paths computed by the local planner are the edges of R. The learning phase is concluded by some post-processing of R to improve its connectivity. Following the learning phase, multiple queries can be answered. To process a query the method first attempts to find a path from the start and goal configurations to two nodes of the roadmap. Next, a graph search is done to find a sequence of edges connecting these nodes in the roadmap. Concatenation of the successive path segments transforms the sequence found into a feasible path for the robot. These two phases have not to be sequential. Instead, they can be interwoven to adapt the size of the roadmap to difficulties encountered during the query phase, thus increasing the learning flavor of our method. For instance, a small roadmap could be first constructed; this roadmap could then be augmented (or reduced) using intermediate data generated while queries are being processed. This possibility helps to estimate how much computational time is needed. PRM*, a variant of PRM, has been introduced to improve the PRM performance. It is a batch variable-radius PRM, applicable to multiple-query problems, in which the radius is scaled with the number of samples in a way that provably ensures both asymptotic optimality and computational efficiency.

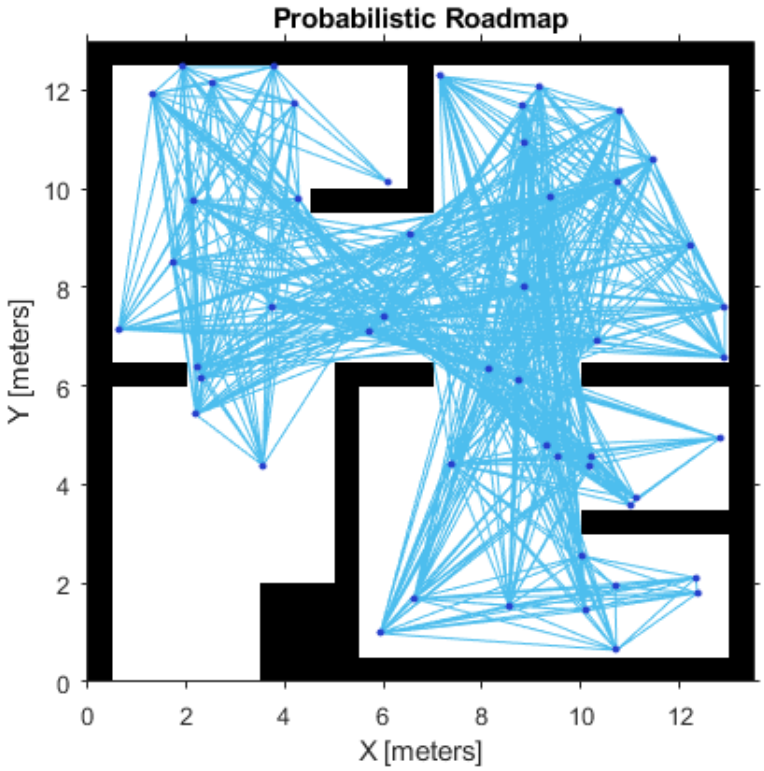


Figure 2.2: Example of PRM in Matlab

Rapid-exploring random trees RRT Developed following the PRM, the rapid-exploring random trees algorithm was presented in 1998 by LaValle [21]. It includes some of the same desirable properties of PRM but it has an unique advantage : it can be directly applied to nonholomic and kinodynamic planning. This is due to the fact that RRT doesn't require any connections to be made between pairs of configurations (or states), while PRM typically requires tens of thousands of connections. In fact the RRT will be constructed so that all its vertices are states in S_{free}, furthermore each edge of the RRT will correspond to a path that lies entirely in S_{free}. For a given initial state s_{start}, an RRT, τ with K vertices is constructed as shown below. This algorithm has several advantages, which make it suitable for a good amount of practical path planning problems; below we list some of the main ones:

- expasion of RRT is heavily biased towards unexplored portions of state space
- an RRT is probabilistically complete under very general conditions
- RRT algorithm is relatively simple, which facilitates performance analysis
- RRT always remains connected even in the case of few number of edges

- it can be considered as a path planning module

Though, there is a drawback: this procedure causes different and unfortunate computationally expensive solution. It means that the algorithm presents long tail in computation time distribution, since it has to save the entire tree, which increases with the time of execution, and an unknown rate of convergence. During last two decades many extensions of the RRT algorithm have been presented in order to reduce its drawbacks. As in the case of the PRM algorithm, at first, a great deal of work has been done on the optimality of the solution and a so called RRT* algorithm has been developed by Karaman et al. in 2011.

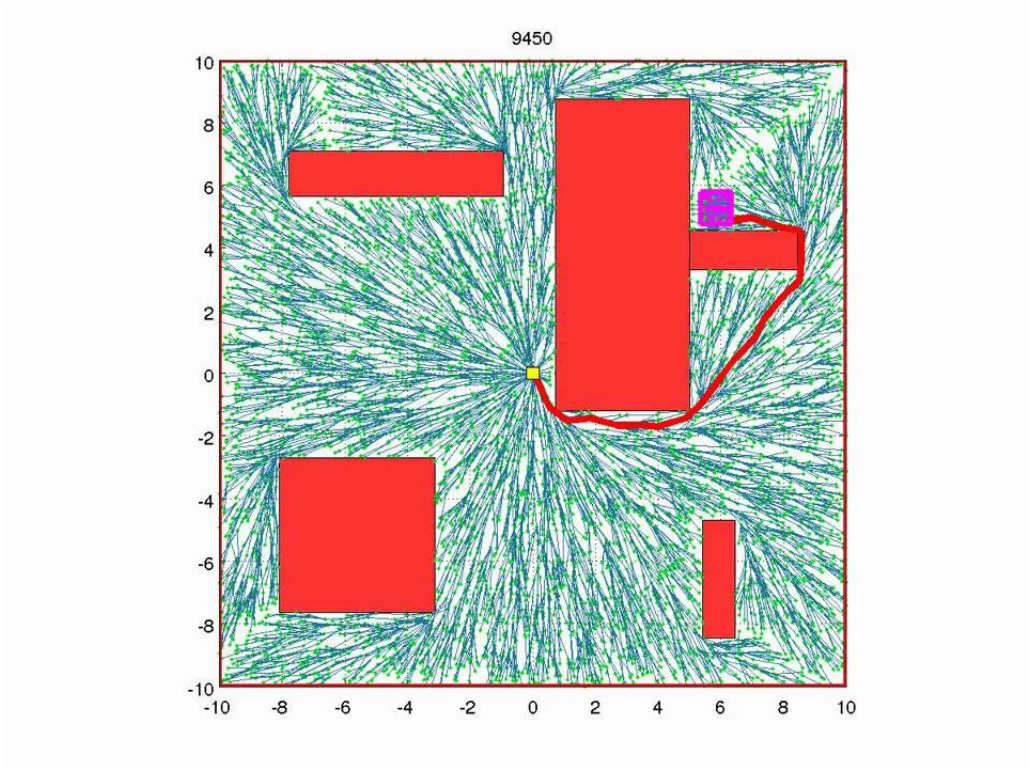


Figure 2.3: Example of RRT in Matlab

RRT* RRT* inherits all the properties of RRT and works similar to RRT. However, it introduces two promising features called near neighbor search and rewiring tree operations, as explained in [30]. Near neighbor operations finds the best parent node for the new node before its insertion in tree. This process is performed within the area of a ball of radius defined by

$$k = \gamma \left(\frac{\log(n)}{n} \right)^{\frac{1}{d}} \tag{2.7}$$

where d is the search space dimension and γ is the planning constant based on environment. Rewiring operation rebuilds the tree within this radius of area k to maintain the tree with minimal cost between tree connections. Space exploration and improvement of path quality is shown. As the number of iterations increase, RRT* improves its path cost gradually due to its asymptotic quality, whereas RRT does not improve its jaggy and suboptimal path. Due to increased efficiency to get less jagged and shorter path, features of rewiring and neighbor search are being adapted in recent revisions of RRT*. However, these operations have an efficiency trade-off. Though, it improved path cost but on the other hand it also slowed down convergence rate of RRT*. The details of the two new features introduced in RRT* are as follows:
Rewire: This function checks if the cost to the nodes in s_{near} is less through s_{new} as compared to their older costs, then its parent is changed to s_{new}.
ChooseParent: This function selects the best parent s_{new} from the nearby nodes.

RRT$^{\mathbf{X}}$ Presented in 2015,it enables real-time kinodynamic navigation in dynamic environments, i.e., in environments with obstacles that unpredictably appear, move, and vanish [32]. In general, replanning algorithms find a motion plan and then repair that plan on-the-fly if/when changes to the obstacle set are detected during navigation. In particular, RRT^X refines, updates, and remodels a single graph and its shortest-path sub-tree over the entire duration of navigation. Both graph and sub-tree exist in the robot's state space, and the tree is rooted at the goal state (allowing it to remain valid as the robot's state changes during navigation). Whenever obstacle changes are detected, e.g., via the robot's sensors, rewiring operations cascade down the affected branches of the tree in order to repair the graph and remodel the shortest-path tree. The expected runtime is achieved, despite rewiring cascades, by using two new graph rewiring strategies:
1) rewiring cascades are aborted once the graph becomes epsilon-consistent, for a predefined $\epsilon > 0$ 2) graph connectivity information is maintained in local neighbor sets stored at each node, and the usual edge symmetry is allowed to be broken. These features significantly decrease reaction time without hindering asymptotic convergence to the optimal solution. Indeed, reaction time is the most important metric for a re-planner in dynamic environments.

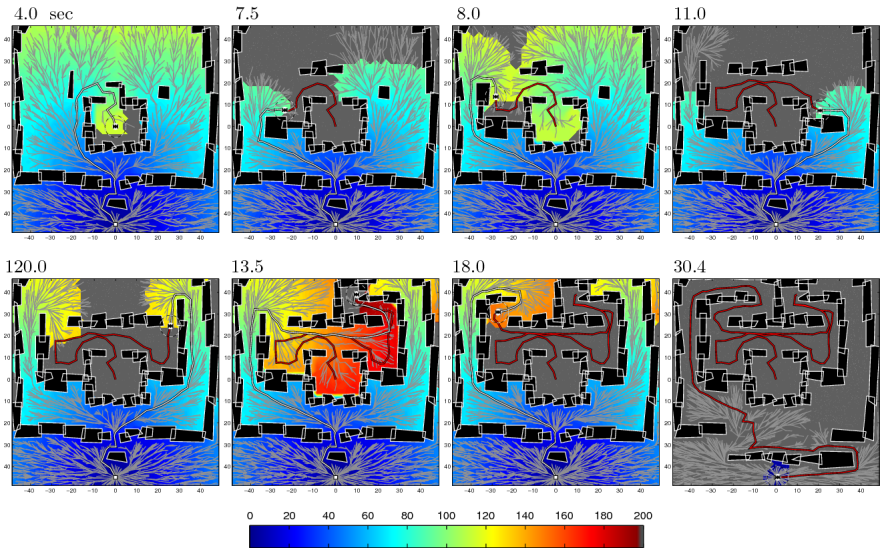


Figure 2.4: Example of RRT^{X} at different simulation steps

Chapter 3

Model predictive control

In this chapter the Model Predictive Control is presented. A general introduction is done to explain the major principle behind this technique, together with all the mathematical elements. Some applications in automotive are presented and a brief focus on the optimization problem is done, since it's a crucial passage for the MPC. The entire chapter is based on [7].

3.1 Introduction

Model Predictive Control (MPC) has developed considerably over the last two decades, both within the research control community and in industry. This success can be attributed to the fact that Model Predictive Control is, perhaps, the most general way of posing the process control problem in the time domain. Model Predictive Control formulation integrates optimal control, stochastic control, control of processes with dead time, multivariable control and future references when available. Another advantage of Model Predictive Control is that general nonlinear processes, which are very frequent in industry can be handled.
The term Model Predictive Control does not designate a specific control strategy but rather an ample range of control methods which make explicit use of a model of the process to obtain the control signal by minimizing an objective function. These design methods lead to controllers which have practically the same structure, in the sense that the algorithm underneath is the same, while the model changes in every application and present adequate degrees of freedom. The ideas, appearing in greater or lesser degree in the predictive control family, are basically:

- explicit use of a model to predict the process output at future time instants (horizon)

- calculation of a control sequence minimizing an objective function

- receding strategy, so that at each instant the horizon is displaced towards the future, which involves the application of the first control signal of the sequence calculated at each step

The various MPC algorithms only differ among themselves in the model used to represent the process and the noises and cost function to be minimized. Then also the optimization strategy, definition and how it's solved, depends on the type of application. The computational complexity and theoretical properties depend on chosen model,objectives or constraints.
Consequently it finds a very wide application since:

- it can deal easily with MIMO system which can have interactions among inputs and outputs
- it has a preview capability
- the resulting controller is an easy-to-implement control law
- it intrisically compensates for dead times
- it can be used as controller for a great variety of processes, also for very complex dynamically speaking, including systems with long delay times or nonminimum phase or unstable ones
- it permits to explicitly incorporate the constraints on the states and outputs of the system so to formulate the problem as a constrained optimization problem

Nevertheless it has also a series of drawbacks. One of these is that although the resulting control law is easy to implement, its derivation is more complex than that of the classical PID controllers. If the process dynamic does not change, the derivation of the controller can be done in advance, but in the adaptive control case all the computation has to be carried out at every sampling time. If we consider also the constraints, the amount of computation required is even higher. So a powerful and fast processor with a large memory is needed. In any case nowadays it's not anymore a big issue since the computing power is increased. The greatest drawback is in fact the need for an appropriate model of the process to be available. The design algorithm is based on prior knowledge of the model and is independent of it, but it is obvious that the benefits obtained will be affected by the discrepancies existing between the real process and the model used.

3.2 Strategy and elements

Usually in a control problem the goal of the controller is calculating the input of the plant so that the output of the plant is as close as possible to the reference.

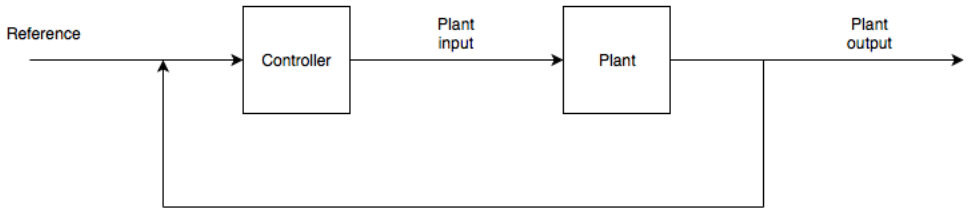


Figure 3.1: Example of a general controller scheme

For the MPC controller is slightly different because we're predicting the future in order to evaluate the plant input. Inside the MPC controller in fact we've a plant model, that is needed to make proper predictions, based on past and current values and an optimizer, which makes sure that the plant output follows the reference as close as possible, taking into account the cost function and constraints.

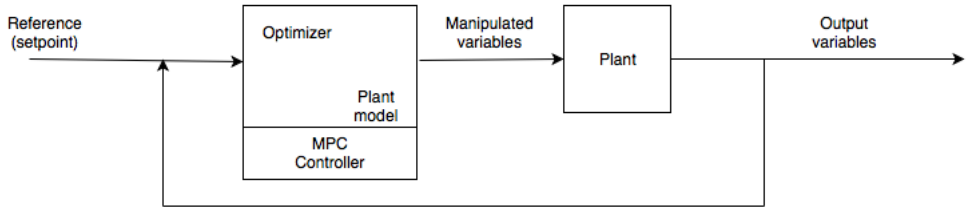


Figure 3.2: MPC controller scheme

The methodology of all the controllers belonging to the MPC family is characterized by the following strategy:

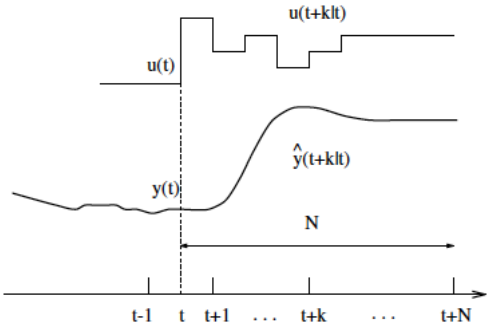


Figure 3.3: MPC strategy

1 The future outputs for a determined horizon N, called the prediction horizon, are predicted at each instant t using the process model. These predicted outputs $y(t+k|t)1$ for $k = 1...N$ depend on the known values up to instant t (past inputs and outputs) and on the future control signals $u(t+k|t), k = 0...N_1$, which are those to be sent to the system and calculated.

2 The set of future control signals is calculated by optimizing a determined criterion, or better a cost function, to keep the process as close as possible to the reference trajectory $w(t+k)$ (which can be the setpoint itself or a close approximation of it). The cost function usually takes the form of a quadratic function of the errors between the predicted output signal and the predicted reference trajectory. The control effort is included in the objective function in most cases. An explicit solution can be obtained if the cost function is quadratic, the model is linear, and there are no constraints; otherwise an iterative optimization method has to be used. Some assumptions about the structure of the future control law are also made in some cases, such as that it will be constant from a given instant.

3 The control signal $u(t|t)$ is sent to the process whilst the next control signals calculated are rejected, because at the next sampling instant $y(t+1)$ is already known and step 1 is repeated with this new value and all the sequences are brought up to date. Thus the $u(t+1|t+1)$ is calculated (which in principle will be different from the $u(t+1|t)$ because of the new information available) using the receding horizon concept.

All MPC algorithms have in common three basic elements:

- the prediction model
- objective function
- obtaining the control law

Prediction model The prediction model is the core of the MPC controller. Depending on the application it can be more or less detailed but in any case it should describe the dynamics of the process in order to do right predictions in the behavior and also intuitive and simple, so as to simplify the theoretical analysis. It can be divided in process model and disturbances model, that takes into account the effect of unmeasurable inputs, noises and model errors. As process model we can use a transfer function or a state space representation, both discrete and continuous.

Objective function We can have different cost functions for obtaining the control law. In general we want that the future output y follows as close as possible the reference in the considered horizon and that the control effort should be penalized. The generalized expression of this objective function is:

$$J(N_1, N_2, N_u) = \sum_{j=N_1}^{N_2} \delta(j)[\hat{y}(t+j|t) - w(t+j)]^2 + \sum_{j=1}^{N_u} \lambda(j)[\triangle u(t+j-1)]^2 \quad (3.1)$$

where N_1 and N_2 are the minimum and maximum prediction horizon and N_u is the control horizon. The limit values set the time interval in which the output should follow the reference. $\delta(j)$ and $\lambda(j)$ are sequences which consider the future behavior, expressed usually in constant or exponential way.
As said previously we introduce constraints in order to obtain the desired results and to set eventual physical limits of the model on input and output signals. These constraints can be inserted already in the objective function, making the minimization of the function J more complex. The optimization problem deals with this kind of functions and it will be treated with more details in the next section.

Obtaining control law In order to obtain values $u(t+k|t)$ it is necessary to minimize the function J. To do this the values of the predicted outputs $\hat{y}(t+k|t)$ are calculated as a function of past values of inputs and outputs and future control signals, making use of the model chosen and substituted in the cost function, obtaining an expression whose minimization leads to the looked for values. An analytical solution can be obtained for a quadratic function if the model is linear and there are no constraints, otherwise an iterative method of optimization should

be used. Whatever the method, obtaining the solution is not easy because there will be $N_2 - N_1 + 1$ independent variables, a value which can be high (on the order of 10 to 30). In order to reduce this degree of freedom a certain structure may be imposed on the control law, improving also the robustness of the system and its general behavior.

Design parameters This paragraph is based on the documentation about MPC of Mathworks [23]. Each MPC controller has the following parameters on which we can act in the design phase:the controller sample time, prediction and control horizons, constraints and weights. Choosing proper values for these parameters is important as they affect not only the controller performance but also the computational complexity of the MPC algorithm that solves an online optimization problem at each time step.
By choosing the *sample time* T_s, we determine the rate at which the controller executes the control algorithm. If it is too big, when a disturbance comes in, the controller won't be able to react to the disturbance fast enough. On the contrary, if the sample time is too small, the controller can react much faster to disturbances and setpoint changes, but this causes an excessive computational load. To find the right balance between performance and computational effort, the recommendation is to fit 10 to 20 samples within the rise time T_r of the open-loop system response.

$$\frac{T_r}{20} \leq T_s \leq \frac{T_r}{10} \tag{3.2}$$

As we've discussed previously, at each time step, the MPC controller makes predictions about the future plant output and the optimizer finds the optimal sequence of control inputs that drives the predicted plant output as close to the setpoint as possible. As said before the *prediction horizon* N shows how far the controller predicts into the future. Of course we should choose a prediction horizon that will cover the significant dynamics of the system, in order to face the unexpected. At the same time we don't want to waste computational power in planning, having a too long prediction horizon. The ideal situation would be an infinite prediction horizon, but the solution of such optimization problem wouldn't be sufficiently fast. As mentioned before, the recommendation for choosing the prediction horizon is to have 20 to 30 samples covering the open-loop transient system response.
Another design parameter is the *control horizon* N_u, that is the number of control moves to time step which brings the future control action to the predictive future output. The rest of the inputs are held constant. Each control move in the control horizon can be thought of as a free variable that needs to be computed by the optimizer. So, the smaller the control horizon, the fewer the computations.The most logical choice then would be that of put the control horizon equal to 1 but it might not give us the best possible maneuver. And by increasing the control horizon, we

can get better predictions but at the cost of increasing the complexity. We can even choose to make the control horizon the same as the prediction horizon. However, note that usually only the first couple of control moves have a significant effect on the predicted output behavior, while the remaining moves have only a minor effect. Therefore, choosing a really large control horizon only increases computational complexity. A good rule of thumb for choosing the control horizon is setting it to 10 to 20% of the prediction horizon N and having minimum 2-3 steps.

$$0.1N \leq N_u \leq 0.2N \tag{3.3}$$

MPC can incorporate *constraints* on the inputs, the rate of change of inputs, and the outputs. These can be either soft or hard constraints. Hard constraints cannot be violated, whereas soft constraints can be violated. A good balance among constraints on inputs and outputs is needed. In fact we cannot have hard constraints both on inputs and outputs because they may conflict with each other leading to an unfeasible solution for the optimization problem. So we have to use quite a number of soft constraints and manage the violation, keeping it a small as possible. Note that to keep the violation of the soft constraint small, it is being minimized by the optimization problem. The recommendation is to set output constraints as soft and avoid having hard constraints both on the inputs and the rate of change of the inputs. Lastly, MPC can have multiple goals with different priority. Consequently we need *weights*. We want the outputs to track as close as possible to their setpoints, but at the same time we want to have smooth control moves to avoid aggressive control maneuvers. The way to achieve a balanced performance between these competing goals is to weigh the input rates and outputs relative to each other. What matters in fact is the ratio between outputs weights and input weights. The larger this ratio is, the controller will be more performing but also more aggressive, decreasing instead its robustness.

3.2.1 Application in automotive

Being highly flexible, MPC started to be implemented also in automotive field for active safety purposes. It can manage in fact not only the path planning problem but also threat assessment and hazard avoidance. As shown in [4] Sterling J. Anderson, Steven C. Peters and colleagues started with the assumption that road lane data is available and that road hazards have been detected, located, and mapped into a 2-dimensional corridor of travel. This is possible thanks to a proper system of sensors, such as LIDAR, radar and camera and an optimal sensor fusion. These should provide lane, position and environmental information needed for the application. Thanks to these data we can obtain constrained vectors in the prediction horizon, in order to evaluate the minimum threat pose. Another interesting application was done by Cesari, Schildbach and colleagues in [9], where they account for

the uncertainty in the traffic environment by a small number of future scenarios, which is intuitive and computationally efficient. These scenarios can be generated by any model-based or data-based approach, resulting in a good performance for highways' scenarios. As explained in section 2.2.1 the artificial potential field can be an optimal method to deal with different types of constraints. An impressive application using the MPC and the APF is done in [41] and [36]. The main idea behind is to associate at every constraint a proper potential field, so that the minimum values are in the center of the vehicle's lane. An explanatory image of this method is reported in figure 3.4, where the peaks of the APF correspond to the obstacles, i.e. other vehicles. Furthermore the MPC can manage also low friction road conditions such as demonstrated by Frash, Gray et al. in [15]. In order to deal with these limit conditions, they take into account highly nonlinear models, which consider both wheel dynamic and load transfer, resulting in a high computational request. For this reason an auto-generated tailored NMPC is implemented thanks to the ACADO Code Generation tool. Lastly, according to a review did by Fitri Yakub and Yasuchika Mori [42], MPC has been implemented not only for active safety systems but also for vehicle dynamics,driver modeling and integrated chassis control systems. Nonetheless the major focus is in active safety with many possible applications such as active steering [5], active braking [13], active traction and active differentials or suspension to coordinate and improve the vehicle handling, the stability and the ride comfort while avoiding collisions.

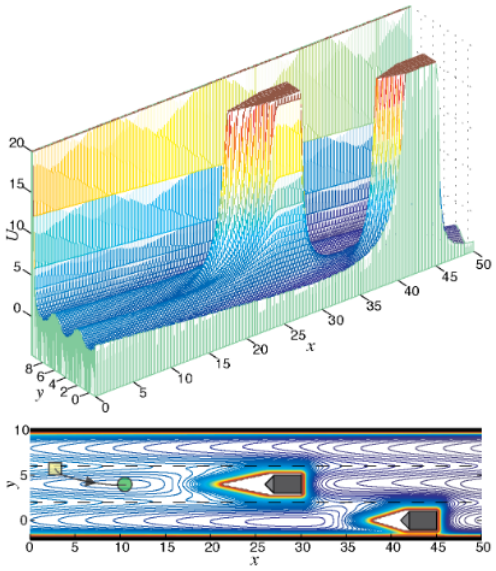


Figure 3.4: Examples of road and obstacles APF application

3.3 Optimization problem

As mentioned in the previous section, the MPC controller in order to evaluate the proper output and manipulated variables solves an optimization problem, which can be defined as finding the minimum of the objective function, considering eventual constraints. In other words it corresponds to find the best solution among all the feasible ones. The subject of the optimization problem is far wider than the simple application inside the model predictive control and it has a huge importance in many technological and scientific fields. In this section the main issues and characteristics of the problem are briefly described with a focus on our application in controls. The definitions in this section are based on [6]. The function in the optimization problem is called cost or objective function and it describes the targets we want to get. In mathematical terms we can define the optimization problem in the standard form like this:

$$\begin{aligned} &minimize \quad f_0(x) \\ &subject\ to \quad f_i(x) \leq b_i \quad i = 1, ..., m \end{aligned} \tag{3.4}$$

where the function $f_0 : R^n \to R$ is the objective function, the function $f_i : R^n \to R, i = 1, ..., m$ are the (inequality) constraint functions and the constraints $b_1, ..., b_m$ are the limits or bounds for the constraints. A vector x^* is called optimal or a solution of the problem previously described if it has the smallest objective value among all vectors that satisfy the constraints: for any z with $f_1(z) \leq b_1, ..., f_m(z) \leq b_m$ we have $f_0(z) \geq f_0(x^*)$. Here we are considering the variable continuous, but we can define the same problem also for a discrete variable. In this case we are talking about combinatorial optimization problem. Depending on the form of the objective function and of the constraint functions, we have different characterizations of the problem, among which we cite:

- **Linear program**: the objective function and constraint function $f_0, ..., f_m$ are linear that is to say:

$$f_i(\alpha x + \beta y) = \alpha f_i(x) + \beta f_i(y) \tag{3.5}$$

  for all $x, y \in R^n$ and all $\alpha, \beta \in R$.
  If the optimization problem is not linear, it's called nonlinear program.

- **Convex optimization**: the objective function and constraint functions are convex, which means they satisfy the inequality

$$f_i(\alpha x + \beta y) \leq \alpha f_i(x) + \beta f_i(y) \tag{3.6}$$

  for all $x, y \in R^n$ and all $\alpha, \beta \in R$ with $\alpha + \beta = 1, \alpha \geq 0, \beta \geq 0$.

From these definitions it's clear that it's convenient to talk about convex optimization since it's a generalization of the linear programming. Nevertheless it was important to outline this difference because methods of solution can be classified in the same way. Moreover, it's relevant to underline the category of convex optimization because it deals with convex functions, which,by definition, are convex if their level curve define convex sets. They're important because for all of them every local minimum is a global minimum. In this way a numerical algorithm can find a global minimum. In figures 3.5 and 3.6 two examples of convex and non-convex functions are shown.

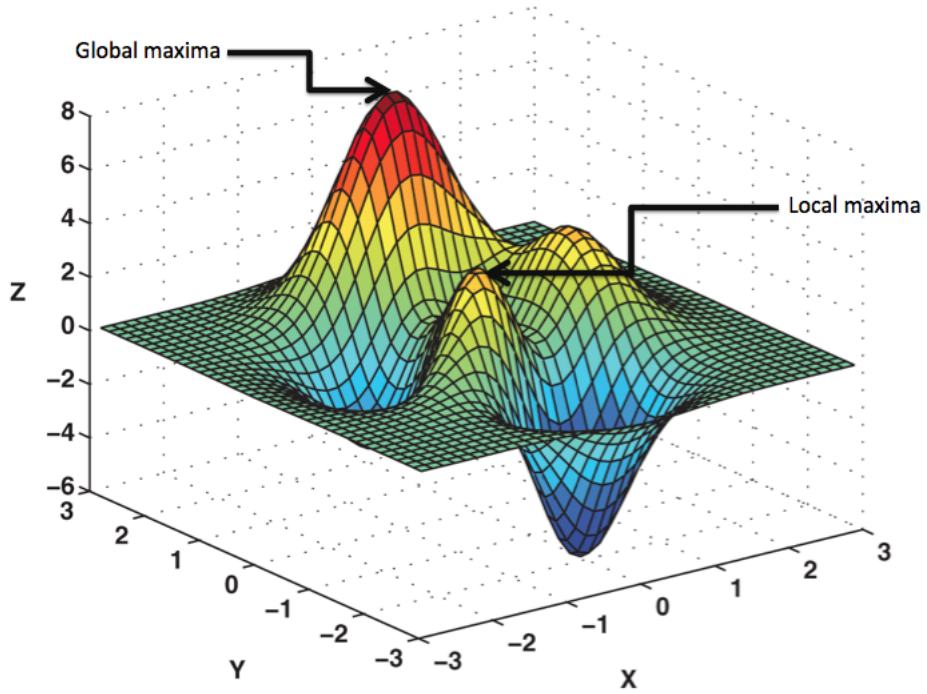


Figure 3.5: Example of non-convex function

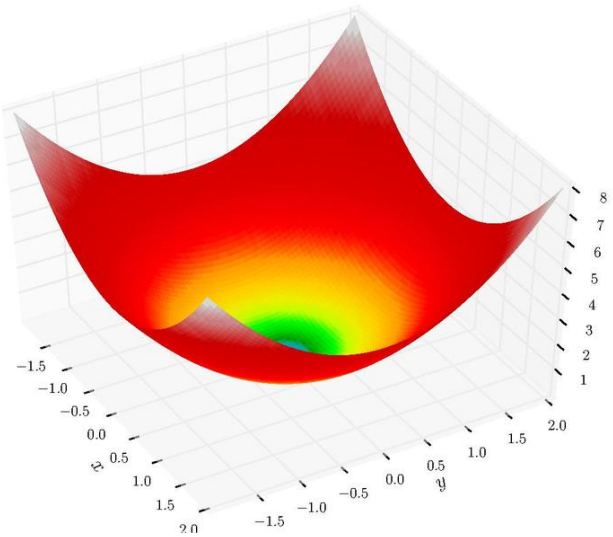


Figure 3.6: Example of convex function

Regarding the solution methods, during the years, different algorithms have been created in order to solve the problem considering the specific characteristics, such as type of involved functions and number of constraints. For example for the linear program we can apply the graphical method, consisting in representing the problem in the Cartesian plane and detecting the area of the feasible solutions, or the simplex method, where adjacent vertices of the feasible set (which is a polytope) are tested in sequence so that at each new vertex the objective function improves or is unchanged. Generally in the MPC controller we have a quadratic programming, that is a particular case of the nonlinear program. It's in any case a convex optimization. The objective function in fact is quadratic and it's subjected to linear constraints and in mathematical terms it's defined as:

$$\begin{aligned} &\textit{minimize} && \frac{1}{2}x^T Px + q^T x + r \\ &\textit{subject to} && Gx \leq h,\ Ax = b \end{aligned} \tag{3.7}$$

where P is an Hessian matrix.

The methods we can apply to solve the quadratic programming are several, some are reported in the list below as done in [39]

- Interior point
- Active set
- Augmented Lagrangian
- Conjugate gradient
- Gradient projection
- Extensions of the simplex algorithm

As the name can suggest, the first method reaches the best solution by traversing the interior of the feasible region, which is the set of all possible points that satisfy the problem's constraints. It comprehends a set of algorithms such as Karmarkar's algorithm, which appeared to be very efficient in practice.
For active set instead are meant a set of constraints which satisfy certain requests, in fact , given a point x in the feasible region a constraint $g_i(x) >= 0$ is called active at x if $g_i(x) = 0$ and inactive at x if $g_i(x) > 0$. The active set is relevant in the optimization theory since it determines which constraints will influence the final result of optimization, resulting in a reduction of the complexity of the solution's search.
In the augmented Lagrangian method, the optimization problem is replaced by a series of unconstrained problems and a penalty term is added to the objective, together with a term designed to mimic a Lagrange multiplier.
The conjugate gradient method, applied as solver for unconstrained optimization problem such as energy minimization, was developed by M. Hestenes and E. Stiefel. It's an algorithm for the numerical solution of linear systems whose matrix is simmetric and positive-definite. The gradient projection was developed to face criticisms of active set methods. It permits in fact to have large changes in the working set at each iteration, while for the active set these changes happen slowly [29]. Lastly extensions of the simplex algorithm are employed as QP solver. The simplex method is designed to solve linear programming and to explain it in geometrical terms, it considers the feasible region as a convex polytope.

3.4 Typology of Model Predictive Control

Adaptive MPC MPC control predicts future behavior using a linear-time-invariant (LTI) dynamic model. In practice, such predictions are never exact, and a key tuning objective is to make MPC insensitive to prediction errors. In many applications, this approach is sufficient for robust controller performance. If the plant is strongly nonlinear or its characteristics vary dramatically with time, LTI prediction accuracy might degrade so much that MPC performance becomes unacceptable. Adaptive MPC can address this degradation by adapting the prediction model for changing operating conditions. Adaptive MPC uses a fixed model structure, but allows the models parameters to evolve with time. Ideally, whenever the controller requires a prediction (at the beginning of each control interval) it uses a model appropriate for the current conditions. At each control interval, the adaptive MPC controller updates the plant model and nominal conditions. Once updated, the model and conditions remain constant over the prediction horizon.

Nonlinear MPC As the name suggests, the main advantage of this type of MPC is that it can deal with nonlinear dynamics. In this way we can simulate closed-loop control of nonlinear plants under nonlinear costs and constraints ($m < p$ control horizon shorter than prediction horizon) and plan the optimal trajectories by solving an open-loop constrained nonlinear optmization problem ($m = p$). A good theoritical analysis of this instrument is done in [2]. Dealing with nonlinear dynamics can be also a drawback if the problem is not well tuned. First of all the possibility to directly use a nonlinear model is advantageous if a detailed first principles model is available. In this case often the performance of the closed loop can be increased significantly without much tuning. This is a major issue when the model derived directly from a set of data of a system, whose behavior is unknown for us. In the automotive case, this problem is widely solved by several possible models of the vehicle's motion. On the other side, if no first principle model is available, it is often impossible to obtain a good nonlinear model based on identification techniques. In this case it is better to fall back to other control strategies like linear MPC. Moreover using a nonlinear model changes the control problem from a convex quadratic program to a nonconvex nonlinear problem, which is much more difficult to solve.

Linear MPC In most of the cases real processes are describe by nonlinear systems, but limiting the operating range, they can often be approximated to linear systems. Linear systems, in fact, simplify a lot the control problem, which results in a more fast and robust control scheme. Especially in the preliminary phases of a project, a linear approximation is fundamental to have an idea about the behavior of the system. In particular, linear MPC approaches are used in the majority of applications with the feedback mechanism of the MPC compensating for prediction errors due to structural mismatch between the model and the process. In model predictive controllers that consist only of linear models, the superposition principle of linear algebra enables the effect of changes in multiple independent variables to be added together to predict the response of the dependent variables.

Chapter 4

Case study: overtaking

For this thesis work we choose to see in details the overtaking manoeuvre for an autonomus driving application. In this chapter a brief introduction of this specific situation is done and several solutions in order to manage it are reviewed, based on [12]. Overtaking is one of the most common driving maneuver and any vehicle which can be considered autonomous has to determine if, when, and how to perform this driving task. The task isn't so simple as it can seem, since it's not a standardized maneuver. A lot of external factors are involved such as traffic situation, speed, surrounding obstacles and road legislation. Both longitudinal and lateral dynamics of the vehicle have to be comprehended, together with sensor data for a correct interpretation of the surrounding environment. Two important functions have to be considered in this maneuver: the trajectory planning and tracking. Also the speed factor is important in this scenario,since,for high speed trajectory, the knowledge of both vehicle dynamics and environment has to be as accurate as possible.
The overtaking maneuver can be divided in three phases and a schematic representation can be see in the figure 4.1:

- lane change
- pass front vehicle
- lane change back to original lane

Of course the last phase depends on the traffic situation and on the speed of both the subject vehicle and others. This is why the maneuver cannot be standardized, since this last part can be covered by many scenarios.

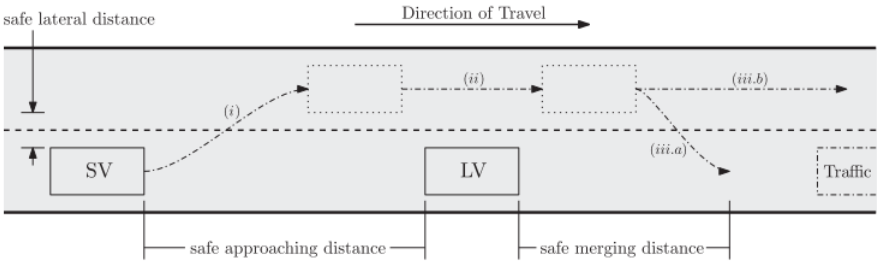


Figure 4.1: Overtaking maneuver: three phases

In this chapter we will focus on trajectory planning and tracking techniques, considering that all the information we need are coming from a proper sensor system, integrated in the vehicle. In these last few years also advanced communication systems such as V2V and V2X can be considered as an integration of a sensor system. A schematic explanation of the system can be seen in figure 4.2, where the V2X is inserted as optional.

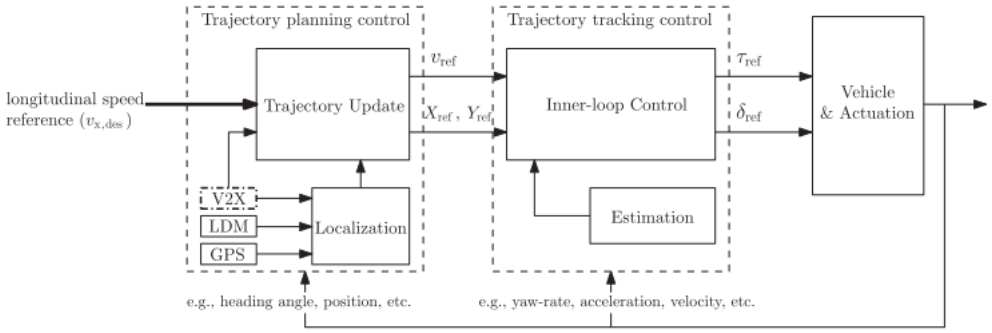


Figure 4.2: Control architecture for trajectory planning and tracking

4.1 Trajectory planning

Among the possible trajectory planning techniques mentioned in the chapter 2,for the purpose of an overtaking manoeuvre we can implement the following :

- potential fields
- cell decomposition (RRT)
- interdisciplinary methods
- optimal control

To have more details about the first two techniques, see chapter 2. As interdisciplinary methods we mean techniques derived from the robotics and missile guidance systems, where motion primitives, intended as combination of steady-state equilibrium trajectories and pre-specified maneuvers, are employed. An other example for this category can be a system of virtual points positioned at priori at known distance from the front vehicle, being tracked by the subject vehicle. As optimal control instead we indicate all the methods were a performance index such as lateral acceleration or change of kinetic energy are minimized. In this last category we can insert also the Model Predictive Control, used recently in many applications as local planner. In this case the performance index to be minimized is the cost function where all kind of constraints can be easily managed. The MPC technique is treated in more detail in chapter 3. The major problems concerning all these methods are computational complexity and following robustness in creating feasible trajectories, critical issues which increase at high speed due to necessity of a more accurate system.

4.2 Trajectory tracking

As control techniques for trajectory tracking, a lot of choices are possible. First of all, we'll mention the required characteristics for this kind of application and correlated scenario:

- Real-time capability
- Robustness
- Operating range
- Controller parameter tuning

The controller needs to work in real-time,the control laws have to be implemented in the ECU of the vehicle and thus be very efficient in terms of calculation time. The speed range should be ideally 0-120 km/h and robustness is needed in order to manage system nonlinearities, model parameter variations or external disturbances. Lastly also an organized tuning procedure of the controller simplifies the design.
A brief summary of all control strategies is reported below, as outlined in [12]:
-*Geometric and kinematic*. These types of controllers do not consider the dynamics of the vehicle, for this reason the performances in high speed trajectory tracking are pretty poor, due to the inaccuracy of the model in regions of tire saturation. The Stanley method and pure-pursuit are two main examples of geometric controllers, which are easy to implement and are efficient at low speeds. The first one, as the name itself suggests, is a method where the vehicle is in constant pursuit of a virtual moving point in front of it, while the Stanley controller considers the heading and the

lateral error in order to computer a proper steering angle, based only on geometric evaluations. Kinematic controllers instead are simple feedback controllers designed considering only the vehicle kinematic.
-*Classical*. Examples : PID, which turns out to have a difficult tuning procedure in such applications and Sliding Mode Control (SMC)
-*Dynamic state feedback*. This kind of controllers can comprehend in the control law also the dynamic of the vehicle, resulting in a better performance. An example can be the LQR, Linear Quadratic Regulator, which obtains good results together with an easy design. It may present some issues when trajectories of varying curvature are considered, but the problem can be fixed adding a feedforward control.
-*Neural network and fuzzy logic.* A human-like behavior is tried to be achieved from these kinds of system, still in the early stages of studies. A large amount of real data are necessary and this brings several issues regarding management of the data themselves, failure explanations and system tuning.
-*MPC.* See chapter 3

Chapter 5

Implementation

In the previous chapters the basic concepts of path planning and control were introduced; in this one details about implementation will be explained. The chosen case study is an overtaking maneuver, already analyzed in chapter 4. The main idea of this thesis work is set a path planner and a controller for this scenario in an autonomous driving application, so as first attempt, we implemented a RRT* planner, working on a costmap and creating a reference path for an LQR controller. We will consider this first application as a standard method for path planning, in order to distinguish it with the MPC technique. Then we passed to the core of the project, that is to say the setting of an MPC controller working with a kinematic model as prediction model, managing both path planning and control phases. Lastly we decided to see how an MPC can work only as path planner due to industrial reason, so once evaluated the optimal path by the MPC, we set it as reference for a standard LQR controller.

5.1 Development environment and assumptions

As development environment we chose Matlab, due to its simplicity to write mathematical models and its easiness in managing different types of data. Moreover, it permits to carry simulations in Simulink, which allows an easy design of the control scheme in the block diagram form. Lastly, since we're interested in automotive applications, Matlab offers different toolboxs proper for this kind of work; in particular Automated driving system toolbox, Vehicle Dynamics Blockset and the Optimization toolbox were used. In all the following applications we assumed that all data coming from the environment such as road and obstacle information were known, thanks to a proper sensor system and data elaboration.

5.2 Car model

Regarding the motion planning problem, a model of the car is required, in order to know how it will move and two different aspects can be taken into account. First we are dealing with a *non-holomonic system*, whose state depends on the path taken in order to achieve it. Such a system is described by a set of parameters subject to differential constraints, such that when the system evolves along a path in its parameter space (the parameters varying continuously in values). Then also the computational complexity is an important factor. Since it's a real time application the complexity should be low or the computational power high in order to get results very fast. In general for all type of study, from vehicle's stability to driver assistance systems, a set of models with increasing complexity are employed. In a preliminary phase simple vehicle point-mass models are adopted, which consider the vehicle as a particle with a mass. Then kinematic models are used, where the speed is considered constant and we can get mathematical relationships between angles and geometrical parameters of the model. Finally, dynamic models are employed, where we have a direct connection between the vehicle's behavior and applied forces, coming from inside and outside the vehicle. In this last case the model is highly flexible depending on the data we have and the analysis we want to carry. In this thesis project we will use two types of car model: a kinematic one for the path planning phase and a dynamic one to simulate the real car and get the results from it. In the next section the second one is reported.

5.2.1 3 DOF Single track model

Regarding the car model for the simulation of the car itself, we took as reference the Vehicle Body 3DOF block, which implements a rigid two-axle vehicle body model to calculate longitudinal, lateral, and yaw motion. The block accounts for body mass, aerodynamic drag and weight distribution between the axles due to acceleration and steering. As coordinate systems we used the standard ones, presented by SAE in 1992. The bicycle model is a planar model, while to calculate the F_z we use the simplest model in (y, z) plane.

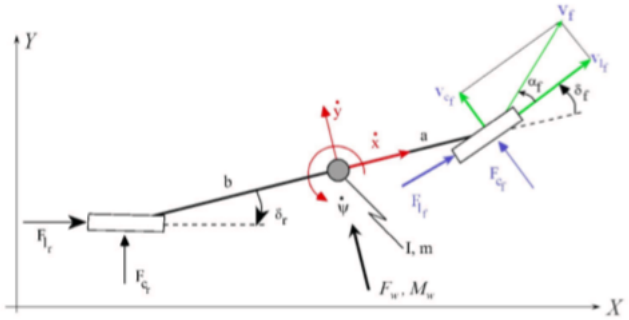


Figure 5.1: Scheme of the bicycle model

The block we will use, uses the following equations, as reported in [25], using the reference system shown in figure 5.1

$$\begin{aligned}
\ddot{x} &= \dot{y}r + \frac{F_{xf} + F_{xr} + F_{x,ext}}{m} \\
\ddot{y} &= -\dot{x}r + \frac{F_{yf} + F_{yr} + F_{y,ext}}{m} \\
\dot{r} &= \frac{aF_{yf} - bF_{yr} + M_{z,ext}}{I_{zz}} \\
r &= \dot{\psi}
\end{aligned} \tag{5.1}$$

where r is the yaw rate, being ψ is the yaw angle. As external forces we can evaluate the drag and the external force inputs, which, we consider, acting on the CG of the model.

$$\begin{aligned}
F_{xyz,ext} &= F_{d,xyz} + F_{input,xyz} \\
M_{xyz,ext} &= M_{d,xyz} + M_{input,xyz}
\end{aligned} \tag{5.2}$$

Depending on the choice in the setting of the block, which is shown in figure 5.2, we can have the following values for the forces if we choose external longitudinal forces:

$$\begin{aligned} F_{xft} &= F_{xfinput} \\ F_{yft} &= -Cyf\alpha f\mu_f \frac{F_{zf}}{F_{znom}} \\ F_{xrt} &= F_{xrinput} \\ F_{yrt} &= -Cyr\alpha r\mu_r \frac{F_{zr}}{F_{znom}} \end{aligned} \tag{5.3}$$

where $C_{yf/r}$ are the rear or front wheel cornering stiffness, $\alpha_{f/r}$ is the front or rear slip angle, while $\mu_{f/r}$ is the friction coefficient. If instead we set a value of external longitudinal velocity, meaning that the acceleration is zero, the F_{xt} force will be zero:

$$\begin{aligned} F_{xft} &= 0 \\ F_{xrt} &= 0 \end{aligned} \tag{5.4}$$

In order to evaluate F_{zf} and F_{zr} we use the roll and pitch equilibrium:

$$\begin{aligned} F_{zf} &= \frac{bmg - (\ddot{x} - \dot{y}r)mh + hF_{x,ext} + bF_{z,ext} - M_{y,ext}}{a+b} \\ F_{zr} &= \frac{amg + (\ddot{x} - \dot{y}r)mh + hF_{x,ext} + aF_{z,ext} + M_{y,ext}}{a+b} \end{aligned} \tag{5.5}$$

where a, b and h are geometric values of the model, representing the distances from front and real wheels to the CG, and the height of CG from ground respectively. To determine the slip angles we use the following expressions:

$$\begin{aligned} \alpha_f &= \arctan\left(\frac{\dot{y} + ar}{\dot{x}}\right) - \delta_f \\ \alpha_r &= \arctan\left(\frac{\dot{y} - br}{\dot{x}}\right) - \delta_r \end{aligned} \tag{5.6}$$

Once we have these values we can calculate the tire forces:

$$\begin{aligned} F_{xf} &= F_{xft}\cos(\delta_f) - F_{yft}\sin(\delta_f) \\ F_{yf} &= -F_{xft}\sin(\delta_f) + F_{yft}\cos(\delta_f) \\ F_{xr} &= F_{xrt}\cos(\delta_r) - F_{yrt}\sin(\delta_r) \\ F_{yr} &= -F_{xrt}\sin(\delta_r) + F_{yrt}\cos(\delta_r) \end{aligned} \tag{5.7}$$

If we are setting external forces these last values are considered as inputs forces.

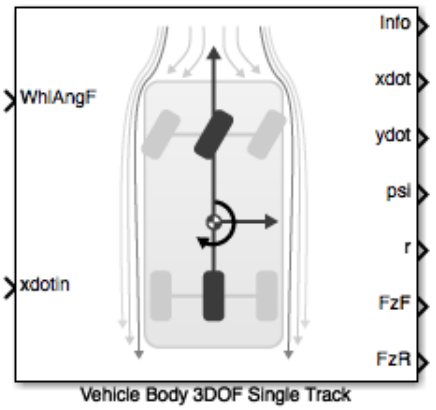


Figure 5.2: Vehicle body 3DOF block on Simulink

5.2.2 Vehicle data

As set of data of the car, the ones proposed in the 3DOF Single Track Block in Simulink were taken into account. They are representative of a segment C car.

Vehicle data		
Mass	m	2000 kg
Moment of Inertia	I_z	4000 $kg \cdot m^2$
Wheelbase	w_b	3 m
Front track	l_f	1.4 m
Center of mass height	h_g	0.35 m
Drag coefficient	C_d	0.30
Longitudinal front area	S	2 m^2
Friction coefficient	$\mu_{f/r}$	1
Front cornering stiffness	$C_{\alpha f}$	$12 \cdot 10^3$ N/rad
Rear cornering stiffness	$C_{\alpha r}$	$11 \cdot 10^3$ N/rad

Table 5.1: Vehicle data

5.3 RRT* planner and LQR controller as first attempt

As shown in chapter 4 there are quite a few possibilities for the path planning and tracking techniques. In a preliminary phase of the work we separated the two tasks and designed an RRT* path planner together with a LQR controller for tracking. In this section both these methods are explained, first showing the general terms of these instruments and then the major details for our implementation.

5.3.1 RRT* path planner

An RRT* path planner explores the environment around the vehicle by constructing a tree of random collision-free poses. To have more details about the algorithm, see chapter 2. We exploit the `pathPlannerRRT` function in Matlab [24], that calculates the optimal path through another object called `vehicleCostmap`,which represents the planning search space around the vehicle [26]. It contains information about the environment like obstacles or areas forbidden to the main vehicle and it's stored as a 2-D grid of cells. Each grid cell in the costmap has a value between 0 and 1, which is the cost of passing through that grid cell. An example of a possible cost map with costs and grid cell states is shown in figure 5.3:

Figure 5.3: Example of costmap

The map has a size, which can be expressed in length and width (ex. 50x50 meters),each cell is square and its size can be specified by the length of the side. Moreover, to simplify checking for whether a vehicle pose is in collision, `vehicleCostmap` considers a safe area around the obstacles, thanks to the setting of an inflation radius. Since this radius creates a circle area, this one is converted in a number of

corresponding grid cells, once the map is created. Having set the costmap, we can pass to the planner itself, where quite a few parameters can be tuned. In particular we decide to set the connection distance intended as the maximum distance between two consecutive poses. Bigger this value is, longer is the segment connecting two poses, resulting in a smaller number of poses composing the path.
Considering it's a map object build on purpose for vehicle application, we can specify the minimum turning radius of the vehicle, which corresponds to the radius of the turning circle at the maximum steering angle. Smaller values result in sharper turns, so, since in our case scenario a smooth trajectory is needed, we increased its value. Once the path planner is set, with a start and goal pose, it's possible to compute the optimal trajectory, which is composed by a series of point coordinates through which the vehicle should pass. In particular the point coordinates are $[x, y, \psi]$, where x and y are the positions of the vehicle on the map and ψ is the heading angle, which is evaluated with the following formula thanks to the current and start positions:

$$\psi = \arctan \frac{y_c - y_0}{x_c - x_0} \tag{5.8}$$

At this point we have the reference path, evaluated with all the information gathered in the map. In table 5.2 the values of the aforementioned parameters are reported, while in figures 5.4 and 5.5 the costmap with and without optimal trajectory are shown.

Cost map and RRT* planner parameters	
Map lenght	100 m
Map width	100 m
Cell size	1
Inflation radius	1.5 m
Road cost	0.15
Offroad cost	0.7
Obstacle cost	0.9

Table 5.2: Cost map and RRT* planner parameters

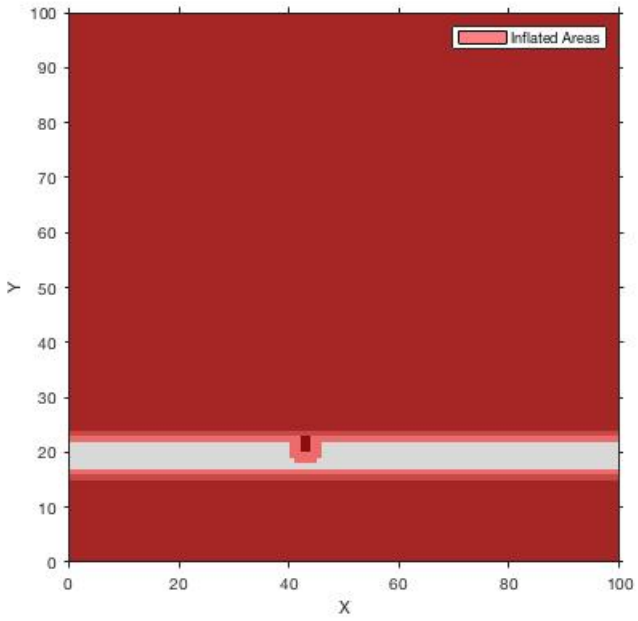


Figure 5.4: Setting of costmap: road with barrier

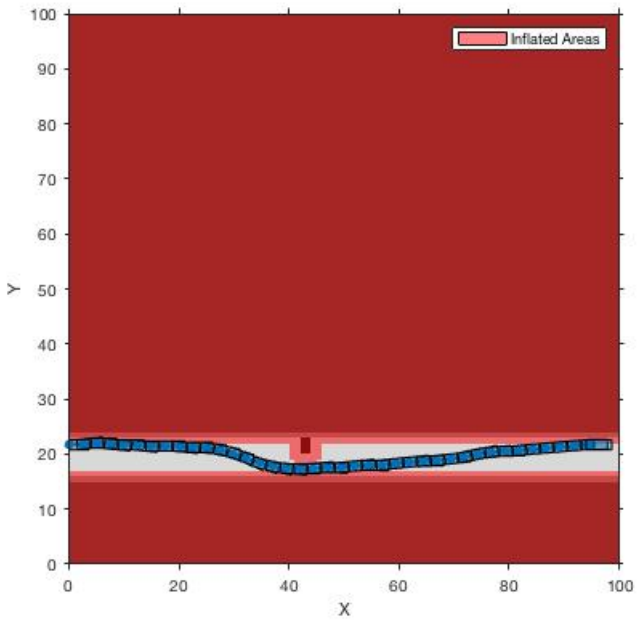


Figure 5.5: Costmap with optimal path

5.3.2 LQR controller: theory

Optimal control is one particular branch of modern control, which aims to obtain the best possible performance from a system, hence the word optimal. The problem consists in finding a control law such that a certain optimality criterion should be achieved. Linear optimal control is a subcategory of optimal control [3]. The plant that is controlled is assumed linear, and the controller, the device that generates the optimal control, is constrained to be linear. The advantages of linear optimal control are:

- Quite all linear optimal control problems have readily computable solutions
- If the plant states of a linear optimal control design are measurable, often the system presents good phase margin, gain margin and tolerance of nonlinearties, which are the basic properties of the classic control methods
- Linear optimal control can be applied to nonlinear systems operating on a small signal basis,i.e. the system will start in a certain initial state

Linear controllers are achieved by working with quadratic performance indices. These are quadratic in the control and regulation/tracking error variables. Such methods that achieve linear optimal control are termed Linear-Quadratic (LQ) methods. In particular the LQR (Linear Quadratic Regulator) is an important part of the solution to the LQG (Linear Quadratic Gaussian) problem, where linear systems with additive white Gaussian noise are taken into account. Output measurements are assumed to be corrupted by Gaussian noise and the initial state, likewise, is assumed to be a Gaussian random vector.
For a continuous time system, whose dynamics is described by:

$$\dot{x} = Ax + Bu \tag{5.9}$$

where $x \in \mathrm{R}^n$, $u \in \mathrm{R}^n$ and x_0 is given and with a quadratic cost function described as:

$$J = \int_0^\infty (x^T Q x + u^T R u + 2x^T N u)\, dt \tag{5.10}$$

the feedback control law that minimizes the value of the cost is given by:

$$u = -Kx \tag{5.11}$$

where K is obtained through the following equation $K = R^{-1}(B^T S + N^T)$ and S is found solving the associated Riccati differential equation (5.12)

$$A^T + SA - (SB + N)R^-1(B^T S + N^T) + Q = 0 \tag{5.12}$$

5.3.3 LQR vehicle model

In our application case we have to deal with lateral dynamics, in particular we have to find an optimal steering action, since we will consider the speed constant. In order to achieve this, we've to re-define the variables of the model in terms of position and orientation error with respect to the road. The definition of this model is based on Rajamani [34] and the error variables are e_1 and e_2, defined respectively as the lateral distance of the center of gravity of the vehicle from the center line of the lane and the orientation error of the vehicle with respect to the road. To complete the model's variables also their derivatives are considered. The reference system is reported in figure 5.6.

$$\begin{aligned} e_1 &= y - y_{lane} \\ e_2 &= \psi - \psi_{des} \\ \dot{e_1} &= \dot{y} + V_x e_2 \\ \dot{e_2} &= \dot{\psi} - \dot{\psi}_{des} \end{aligned} \tag{5.13}$$

where $\dot{\psi}_{des}$, the desired yaw rate, is evaluated as

$$\dot{\psi}_{des} = \frac{V_x}{R} \tag{5.14}$$

being R the curvature radius of the road, that will be consider large ($R >> 500m$) since it's a similar value of the real curvature radius in motorways. The equations of the model in state-space are :

$$\begin{aligned} \frac{d}{dt}\begin{bmatrix} e_1 \\ \dot{e_1} \\ e_2 \\ \dot{e_2} \end{bmatrix} &= \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -\frac{2C_{\alpha f} + 2C_{\alpha r}}{mV_x} & \frac{2C_{\alpha f + 2C_{\alpha r}}}{m} & \frac{-2C_{\alpha f}l_f + 2C_{\alpha r}l_r}{mV_x} \\ 0 & 0 & 0 & 1 \\ 0 & -\frac{2C_{\alpha f}l_f - 2C_{\alpha r}l_r}{I_z V_x} & \frac{2C_{\alpha f}l_f - 2C_{\alpha r}l_r}{I_z} & -\frac{2C_{\alpha f}l_f^2 + 2C_{\alpha r}l_r^2}{I_z V_x} \end{bmatrix} \begin{bmatrix} e_1 \\ \dot{e_1} \\ e_2 \\ \dot{e_2} \end{bmatrix} + \\ &\begin{bmatrix} 0 \\ \frac{2C_{\alpha f}}{m} \\ 0 \\ \frac{2C_{\alpha f}l_f}{I_z} \end{bmatrix} \delta + \begin{bmatrix} 0 \\ -\frac{2C_{\alpha f}l_f - 2C_{\alpha r}l_r}{mV_x} - V_x \\ 0 \\ -\frac{2C_{\alpha f}l_f^2 + 2C_{\alpha r}l_r^2}{I_z V_x} \end{bmatrix} \dot{\psi}_{des} \end{aligned} \tag{5.15}$$

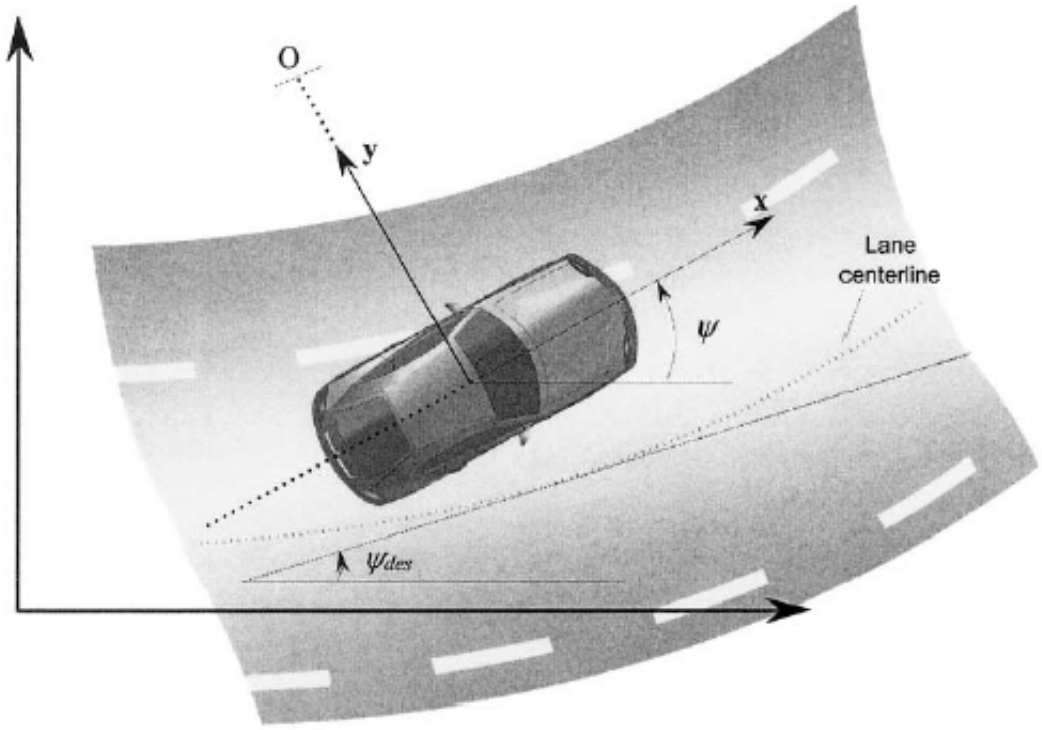


Figure 5.6: Reference system of lateral dynamic error model

5.3.4 Feedforward term

Till now we have described a model for lateral closed-loop dynamics. As well as in any closed-loop system, the controller will react to the inputs minimizing the error, but in our case the presence of the term $B_2\dot{\psi}_{des}$ will not permit the convergence to zero of the tracking errors, as shown in the equation (5.16):

$$\dot{x} = (A - B_1K)x + B_2\dot{\psi}_{des} \tag{5.16}$$

where even in the case where the matrix $(A - B_1K)$ is asymptotically stable, $\dot{x}$ cannot be zero and lower is the curvature radius R, bigger will be the error. In other words, this problem is highlighted when the car is traveling on a curve. Since it's a kind of information we can know a priori, in fact it's sufficient to know the curvature radius R through a sensor system, we can exploit it adding a feedforward term, whose value is independent from the error states. Indeed it's evaluated from the model we've described in the previous section, as reported in [34].

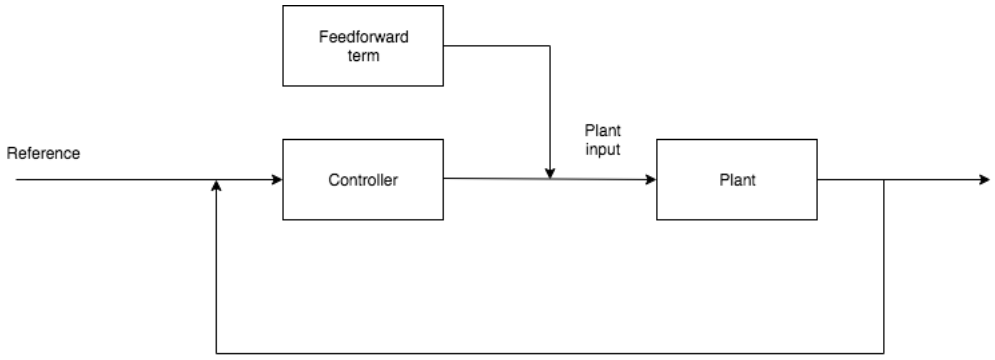


Figure 5.7: Control scheme with feedforward block

At the end what we obtain is a steering action result of the sum of the state feedback and a feedforward term:

$$\delta = -Kx + \delta_{ff} \tag{5.17}$$

where the feedforward steering angle is chosen as:

$$\delta_{ff} = \frac{mV_x^2}{RL}\left[\frac{l_r}{2C_{\alpha f}} - \frac{l_f}{2C_{\alpha r}} + \frac{l_f}{2C_{\alpha r}}k_3\right] + \frac{L}{R} - \frac{l_r}{R}k_3 \tag{5.18}$$

where k_3 is calculated from the state feedback as $\delta = -Kx = -k_1e_1 - k_2e_2 - k_3e_3 - k_4e_4$. In general this modification of the control scheme has to be done in order to have a better control of the vehicle in case of low curvature roads. In our case scenario we have simply to overtake an obstacle, so the effect of the feedforward term can be seen through a more stable steering action.

5.3.5 Setup and simulation

In the previous subsections the main elements of the first application were described. The control scheme we designed in Simulink can be seen in figure 5.8.
The block called *followPath*, which interprets a Matlab function, whose details can be seen in the Appendix, acts as reference for the entire scheme. In fact it provides the next point to reach in the reference path, calculating the longitudinal distance d from the actual point of the vehicle (x, y, ψ) to the goal position $(x_{ref}, y_{ref}, \psi_{ref})_2$ as reported in the equation (5.19). When this distance is equal to zero, the next goal position can be pursued.

$$d = \sin(\psi_{ref})(y_{ref} - y) + \cos(\psi_{ref})(x_{ref} - x) \tag{5.19}$$

The next block generates the model used by the LQR described in section 5.3.3. Its output is the vector $x = [\mathbf{e_1}, \dot{\mathbf{e}}_\mathbf{1}, \mathbf{e_2}, \dot{\mathbf{e}}_\mathbf{2}]$ at which a gain K is applied in order to

calculate the steering angle. The vector K is calculated through the `lqr` command in Matlab. From the figure 5.8 is also clear the presence of the feedforward term, explained in section 5.3.4. The steering action acts on the 3DOF Single Track block, described in section 5.2.1, while, since the speed was considered constant for this first application, a constant block stands for the speed input.

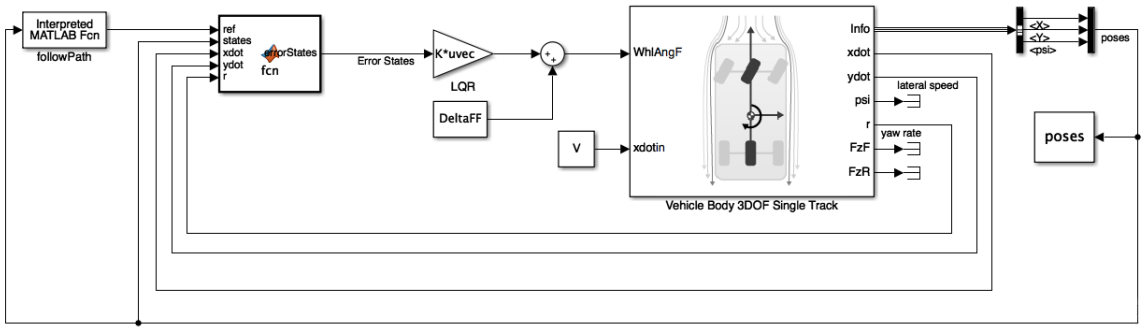


Figure 5.8: Control scheme of RRT* and LQR

5.4 Kinematic MPC as path planner and controller

In chapter 3 the Model Predictive Control technique was introduced. As already underlined the advantages of the MPC are many. First of all the two functions of path planning and vehicle control are managed together, keeping simpler the design phase as well as the overall system's complexity. Moreover all types of constraints can be inserted easily in the objective function. In our application we decided to use a NMPC, built in Matlab, whose main elements are the same cited in chapter 3, i.e. a prediction model, an objective function with constraints linked to an optimization problem and several parameters to be set. All these elements will be explained in the following sections.

5.4.1 Prediction model: kinematic bicycle model

As prediction model we decided to use a simple kinematic model. According to [20], even if less precise and complex, the kinematic model shows a good performance in model-based design, especially for its low computational efficiency, which, since we're dealing with a real-time application, is quite an important and relevant aspect.

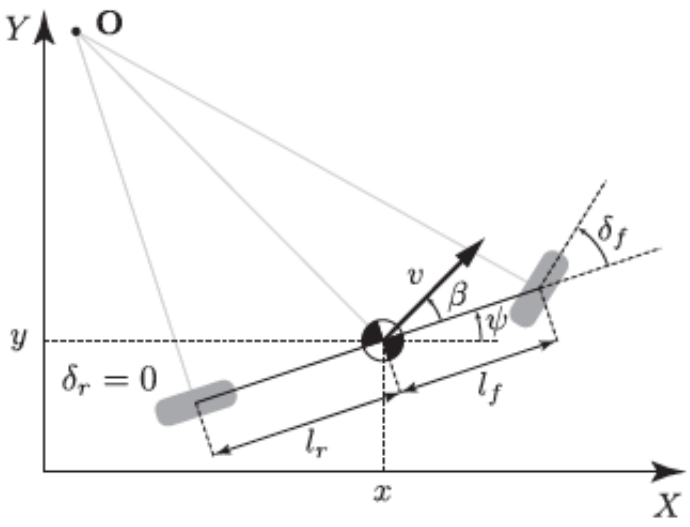


Figure 5.9: Scheme of the kinematic bicycle model

In this model we will take into account the following assumptions:

- Wheel's slip angles equal to zero
- The motion is planar
- Only the front wheel is steering

In this model three coordinates are used to describe the motion: x,y and ψ. (x, y) are the inertial coordinates of the location of the c.g. of the vehicle in the (X, Y) frame, while ψ describes the orientation of the vehicle. The velocity at the c.g. of the vehicle is denoted by V and makes an angle β with the longitudinal axis of the vehicle, being the side slip angle of the vehicle. These four variables are the state of the system:

$$x = [\mathbf{x}, \mathbf{y}, \psi, \mathbf{V}]^T \tag{5.20}$$

The distance between the c.g. of the vehicle and respectively, the rear wheel and the front one, are called l_r and l_f. As inputs we have the steering angle in the front δ_f and the acceleration a, which will be the commands of our controller:

$$u = [\delta_{\mathbf{f}}, \mathbf{a}]^T \tag{5.21}$$

Exploiting trigonometric properties we can obtain the following equations:

$$\begin{aligned} \dot{x} &= V cos(\psi + \beta) \\ \dot{y} &= V sin(\psi + \beta) \\ \dot{\psi} &= \frac{V cos(\beta)}{l_r + l_f} tan(\delta_f) \\ \dot{v} &= a \end{aligned} \tag{5.22}$$

Moreover we can also evaluate the side slip angle of the vehicle β:

$$\beta = arctan\left(\frac{l_r tan(\delta_f)}{l_r + l_f}\right) \tag{5.23}$$

Considering the output instead, we choose the longitudinal and lateral positions x and y:

$$y = [\mathbf{x}, \mathbf{y}]^T \tag{5.24}$$

The model we described is clearly a MIMO nonlinear system.

5.4.2 Objective function and constraints

Now that we have a model of the system in the form:

$$\begin{aligned} \dot{x} &= f(x,u) \\ y &= h(x,u) \end{aligned} \tag{5.25}$$

we have to evaluate a proper input $u \in R^{n_u}$ so that the output $y \in R^{n_y}$ is as close as possible to our reference. It is assumed that the state is measured in real-time, with a sampling time T_s, so that the measurements are

$$x(t_k), \quad t_k = T_s k, \quad k = 0,1,... \tag{5.26}$$

At time t a prediction of the state and output over an interval $[t, t+T_p]$ is obtained by integration of the model, where T_p is the prediction horizon. In order to simplify calculations the input signal is assumed constant over a certain time interval T_c, called control horizon:

$$u(\tau) = u(t+T_c) \quad \tau \in [t+T_c, t+T_p] \tag{5.27}$$

where $0 < T_s < T_c < T_p$. At each time $t = t_k$ we look for an input signal $u^*(t:\tau)$ such that the prediction $\hat{y}(\tau, x(t), u^*(t:\tau)) \equiv \hat{y}(u^*(t:\tau))$ has the desired behavior for $\tau \in [t, t+T_p]$. The objective function defines the desired behavior we want for the system and it's defines as:

$$J(u(t:t+T_p)) \doteq \int_t^{t+T_p} \left(||\tilde{y}_P(\tau)||_Q^2 + ||u(\tau)||_R^2 \right) d\tau + ||\tilde{y}_P(t+T_p)||_P^2 \tag{5.28}$$

where $\tilde{y}_P \doteq r(\tau) - \hat{y}(\tau)$ is the predicted tracking error, $r(\tau) \in R^{n_y}$ is the reference to track, while $||\cdot||_X$ are weighted vector norms and their integrals are square signal norms. The input signal $u^*(t:t+p)$ is chosen as the one minimizing the objective function $J(u(t:t+T_p))$. In particular we want to minimize the tracking error square norm $||\tilde{y}_P(\tau)||_Q^2$ over a finite time interval, while the term $||\tilde{y}(t+T_P)||_P^2$ gives further importance to the final tracking error. Lastly, the term $||u(\tau)||_R^2$ permits us to manage the trade-off between performance and command activity [8]. We remember to the reader that the square weighted norm of a vector $v \in R^n$ is

$$v||_Q^2 \doteq v^T Q v = \sum_{i=1}^{n} q_i v_i^2, \quad Q = diag(q_1, ..., q_n) \in R^{n \times n} \tag{5.29}$$

where $q_i \geq 0$ are the weights for the matrix Q. We have the same definition also for the matrices P and R. The values of the weights for all three matrices are fundamental for the setting of the NMPC, since they regulate the optimization process.

The minimization of $J(u(\cdot)))$ is subjected to constraints

$$\begin{aligned} \dot{\hat{x}}(\tau) &= f(\hat{\mathrm{x}}(\tau), u(\tau)), \quad \hat{\mathrm{x}}(t) = x(t), \quad \tau \in [t, t+T_p] \\ \tilde{y}(\tau) &= h(\tilde{x}(\tau), u(\tau)) \end{aligned} \tag{5.30}$$

It's possible to define constraints also on predicted state/output and on the input. In our case we put constraints simply on some states of the system corresponding to:

-Road: in order to transform the road limits in constraint for the objective function we gave two limits on the state corresponding to the lateral position y, imaging that the vehicle starts its manoeuvre on the right lane and has to overtake the obstacle on the left lane.

$$\begin{aligned} y &> y_{right,road} \\ y &< y_{left,road} \end{aligned} \tag{5.31}$$

-Obstacle: the presence of an obstacle is transformed in a circle in the objective function's constraints. The center $CC = (x_C, y_C)$ is positioned on the road, in the right lane, and it comprehends a safety radius CR, in order to obtain a safe overtaking manoeuvre, without the possibility of the two cars touching.

$$CR - \sqrt{(x - x_C)^2 + (y - y_C)^2} \tag{5.32}$$

-Velocity: being the velocity a state of the system, we can set a constraint to limit its value.

$$V < V_{limit} \tag{5.33}$$

This last constraint can be an option depending on the situation, i.e. constant speed or not. The chosen values for the constraints are reported in the table 5.3 down below.

MPC Constraints	
Right road limit	8 m
Left road limit	16 m
Obstacle position CC	(400 m,10 m)
Safety radius CR	3.5 m
Speed limit	20 m/s

Table 5.3: Kinematic MPC Constraints

5.4.3 Optimization problem and NMPC algorithm

As explained in section 3.3, once an objective function is defined, an optimization problem has to be solved. In our case we have:

$$\begin{aligned} & u^*(t:t+T_p) = arg \quad \min_{u(\cdot)} J(u(t:t+T_p)) \\ & subject \quad to: \\ & \dot{\tilde{x}}(\tau) = f(\tilde{x}(\tau), u(\tau)), \quad \tilde{x}(t) = x(t) \\ & \tilde{y}(\tau) = h(\tilde{x}(\tau), u(\tau)) \\ & \tilde{x} \in X_c, \quad \tilde{y}(\tau) \in Y_c, \quad u(\tau) \in U_c \\ & u(\tau) = u(t+T_c), \quad \tau \in [t+T_c, t+T_p] \end{aligned} \tag{5.34}$$

This is the general formulation of the optimization problem, which involves the minimization of a functional $J(\cdot)$. The problem must be solved on-line, thus the decision of the solver is fundamental. Matlab in his Optimization Toolbox offers several possible solvers; in our application `fmincon` was employed. It's a nonlinear programming solver aimed to find the minimum of constrained nonlinear multivariable function. But before setting the solver, it has to be noticed that the input signal $u(t:t+T_p)$ can be seen as a vector with an infinite number of elements. This means that the optimization involves an infinite number of decision variables and so in order to overcome this problem, the input signal can be parametrized in the following way:

$$u(\tau) = \sum_{i=1}^{m} c_i \phi_i(\tau) = c\phi(\tau) \tag{5.35}$$

where c are parameters such as $c = [c_1, ..., c_m] \in R^{n_u \times m}$, while $\phi(\tau)$ are basic functions $\phi(\tau) = [\phi_1(\tau), ..., \phi_m(\tau)]^T \in R^{m \times 1}$, which can be:

- Rectangular functions:

$$\phi_i(\tau) = \begin{cases} 1, & \tau \in [t + (i-1)T_s, t + iT_s] \\ 0, & otherwise \end{cases} \tag{5.36}$$

- Polynomial functions:

$$\phi_i(\tau) = (\tau - t)^{(i-1)} \tag{5.37}$$

In both cases the input is kept constant over the prediction horizon:

$$u(\tau) = c_1 = const, \quad \tau \in [t, t+T_p] \tag{5.38}$$

At this point the optimization problem becomes:

$$c^* = arg \quad \min_{c \in R^{n_u \times m}} J(c) \tag{5.39}$$

The optimal input is $u^*(\tau) = c^*\phi(\tau)$, which in an open-loop input since it depends on $x(t)$ but not on $x(\tau), \quad \tau > t$. So if we apply it for the entire time interval $[t, t + T_p]$, we don't have the desired feedback action. For this reason the receding horizon strategy is used, where:

1. At time $t = t_k$ an optimal input $u^*(t : t + T_p)$ is calculated. Only the first input value is applied and kept constant:

$$u(\tau) = u^*(t = t_k) \quad \forall \tau \in [t_k, t_{k+1}] \tag{5.40}$$

2. The step 1. is repeated for $t = t_{k+1}, t_{k+2}, ...$

The complete algorithm is reported.

1 At time $t = t_k$, for $\tau \in [t, t + T_p]$ solve the optimization problem

$$\begin{aligned} & c^* = arg \min_{c \in R^{n_u \times m}} J(c) \\ & subject \quad to: \\ & \dot{\tilde{x}}(\tau) = f(\tilde{x}(\tau), u(\tau)), \quad \tilde{x}(t) = x(t) \\ & \tilde{y}(\tau) = h(\tilde{x}(\tau), u(\tau)) \\ & u(\tau) = c\phi(\tau) \\ & \tilde{x}(\tau) \in X_c, \tilde{y}(\tau) \in Y_c, u(\tau) \in U_c \\ & u(\tau) = u(t + T_c), \tau \in [t + T_c, t + T_p] \end{aligned} \tag{5.41}$$

2 Solution: $u^*(\tau) = c^*\phi(\tau)$

3 Receding horizon strategy for closed-loop control law: $u(\tau) = u^*(t_k), \forall \tau \in [t_k, t_{k+1}]$

4 Repeat from step 1. for $t = t_{k+1}, t_{k+2}, ...$

5.4.4 Setup and simulation

In the design phase, the following parameters were set:

- Prediction horizon T_p
- Sampling time T_s
- Upper and lower bounds on the inputs
- Tolerance on the tracking error
- Weights of the matrices R,P and Q

The complete control scheme can be seen in figure 5.10, while the values of the aforementioned parameters can be seen in table 5.4.

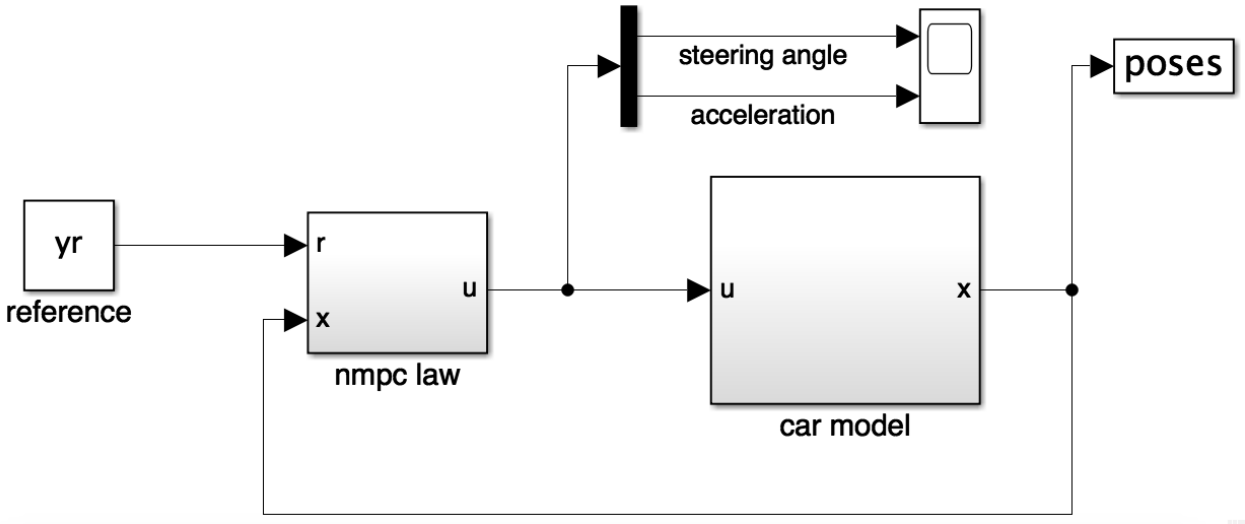


Figure 5.10: Control scheme of the kinematic MPC

MPC Parameters	
Prediction horizon T_p	10 s
Sampling time T_s	0.1 s
Upper bound for steering angle δ_f	30°
Lower bound for steering angle δ_f	$-30°$
Upper bound for acceleration	0.2 m/s^2
Lower bound for acceleration	-1.8 m/s^2
Tolerance on tracking error	0.2 m
R matrix weights r_{11}, r_{22}	1 1
P matrix weights p_{11}, p_{22}	2 6
Q matrix weights q_{11}, q_{22}	10 50

Table 5.4: Kinematic MPC parameters

In particular the nmpc law block includes the nmpc law itself and a reference generator (figure 5.11), which creates the predicted output $\tilde{y}(t)$. In fact, due to how the nmpc law was set, it simply solves the optimization problem, giving only the optimal input, so it's necessary to obtain the predicted output in order to compare the performance of the MPC as controller. The predicted output indeed can be seen as the optimal path to be followed and in the next application, they will be used as reference path for another type of controller. The model used for the predicted output is:

$$\begin{aligned} \beta &= arctan\left(\frac{1}{2}\tan(\delta)\right) \\ x_{next} &= x + V\cos(\psi+\beta)T_s \\ y_{next} &= y + V\sin(\psi+\beta)T_s \\ \psi_{next} &= arctan\left(\frac{y_{next}-y}{x_{next}-x}\right) \end{aligned} \tag{5.42}$$

The predicted positions are calculated in the reference generator block (figure 5.11), starting from the actual states x, y and ψ, the sampling time T_s and the optimal input δ, while the heading angle is calculate with the same formula as 5.8.

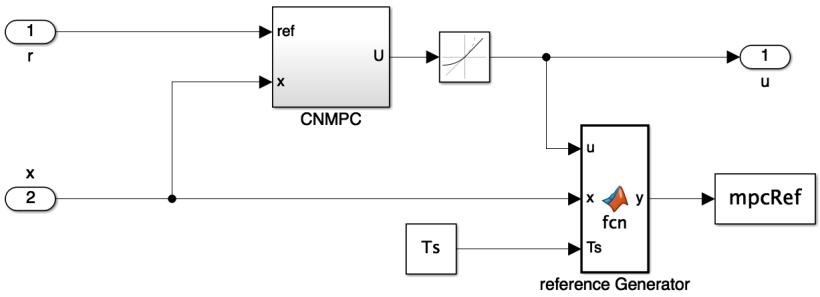


Figure 5.11: NMPC law block

5.5 MPC as path planner

Regarding the last application we decide to test the MPC only as path planner and to see its performance with an LQR controller. The employment of the MPC in fact, both as path planner and controller is still under research in the automotive field, even if some applications are already present as depicted in section 3.2.1. To have a more realistic and precise MPC in fact, a dynamic model as prediction model has to be employed, such as done in [36], [43] and [14]. Some attempts were done, but, with the instruments at our disposal, the simulations resulted too slow and thus non-applicable. For this reason, in order to have a more realistic approach, we use the kinematic MPC only as path planner together with the already designed LQR, which can properly manage a lateral dynamic model. Moreover, thanks to this action, the MPC can work at a lower frequencies, while the control action done by LQR can be done at higher frequencies. Since the two main instruments were already explained in section 5.3 and 5.4, more details about the complete control scheme are reported in this section.

5.5.1 Assumptions

For this last application we tried to set a more realistic scenario. The two main problems of the kinematic MPC were that the obstacle was fixed,i.e. a barrier, and that the vehicle spent too much time for re-entering in the initial lane. Thus some modifications were applied. In particular we set a trajectory for the obstacle, while regarding the timing of the maneuver, we apply an update of the goal position, which in this application is limited to the center of the lane.

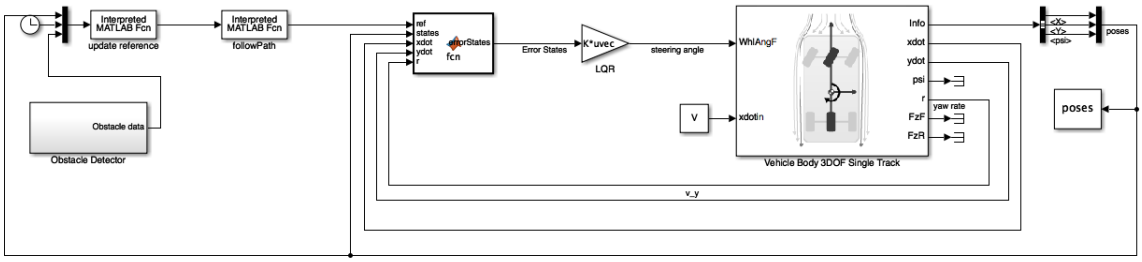


Figure 5.12: Control scheme of MPC as planner and LQR controller

5.5.2 Obstacle

The *Obstacle detector* block has the function of determining the obstacle's trajectory. The created scenario is very simple. The vehicle starts its maneuver with another vehicle at 200 meters ahead, which is moving at lower speed at the center of the initial lane always straight ahead as shown in figure 5.13. The setting of the block can be seen in figure 5.14, where the positions of the obstacle are sent to the *Update reference* block, so that the MPC can have the updated data about the obstacle.

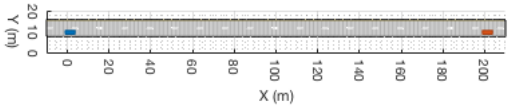


Figure 5.13: Vehicle and obstacle at start positions

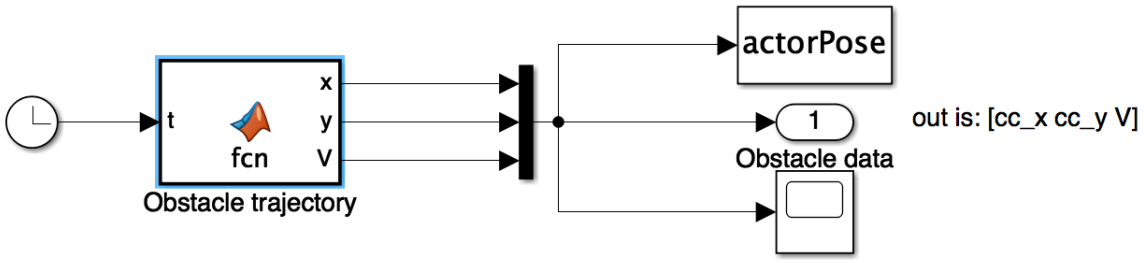


Figure 5.14: Obstacle detector block in Simulink

5.5.3 MPC and LQR interaction

As already explained the MPC has to provide a reference path directly to the LQR controller. The path can be evaluated once, like as with RRT* planner, or updated in real-time, in order to manage sudden changes of the environment. For this reason we decide to update the path evaluated by the MPC with a certain time interval. This is done in the *Update Reference* block, where a kinematic MPC is set. With a prefixed time interval, called delta time $\triangle t$, the MPC works for a predetermined simulation time T_{sim} and provides a reference path covering a certain distance. The elements of this MPC are the same already explained in section 5.4, but being only a path planner, some modifications were done in order to make it even more fast and performing. The prediction horizon T_p was decreased since it's necessary to define a trajectory on a lower distance, as well as the simulation time in order to keep a

good timing performance. The numerical values of the described parameters can be seen in table 5.5. The detailed Matlab function we wrote, that works in the *Update Reference* block, is reported in the Appendix. The previous design of the kinematic MPC worked on the 3DOF Single Track block, while in this application we simplify it in a kinematic model reported below:

$$\begin{aligned} \beta &= arctan\left(\frac{1}{2}tan(\delta)\right) \\ \dot{x} &= V\cos(\psi+\beta) \\ \dot{y} &= V\sin(\psi+\beta) \\ \dot{\psi} &= \frac{V}{l/2}\sin(\beta) \\ \dot{V} &= 0 \end{aligned} \tag{5.43}$$

Some simplifications were done also in the model. The fourth state in fact, i.e. the speed, is equal to zero due to the fact that we are considering the speed constant. The related block can be seen in figure 5.15.

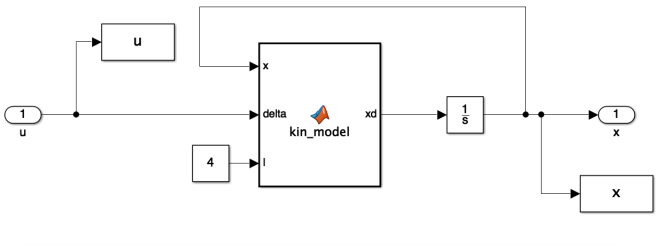


Figure 5.15: Car model block for MPC planner

Moreover an update of reference is present. It's a function working within the *Update Reference* block which, when a certain distance from the obstacle is reached, changes the reference from the actual lane to the fast one, and with the same principle does the opposite when the obstacle has been overtaken. More details about this function can be seen in the Appendix, in the GoalUpdate section. The aim of this function is re-creating a possible sensor system, that detects external obstacles and sends data to the path planner. The block *Follow Path*, like in the first application, gives the exact point coordinates in the reference path to the LQR controller. Also in this case, details of the function can be found in the Appendix.

MPC Parameters	
Prediction horizon T_p	4 s
Sampling time T_s	0.1 s
Delta time $\triangle t$	2 s
Simulation time T_{sim}	5 s
Upper bound for steering angle δ_f	30°
Lower bound for steering angle δ_f	$-30°$
Tolerance on tracking error	0.1 m
R matrix weight	0.1
P matrix weights p_{11}, p_{22}	0.1 0.05
Q matrix weights q_{11}, q_{22}	6 1.5

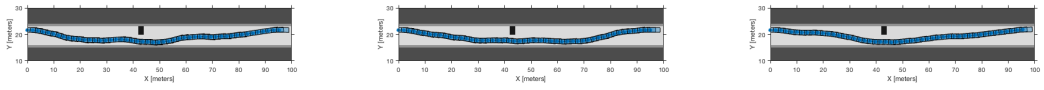
Table 5.5: MPC as path planner parameters

Chapter 6

Results and conclusions

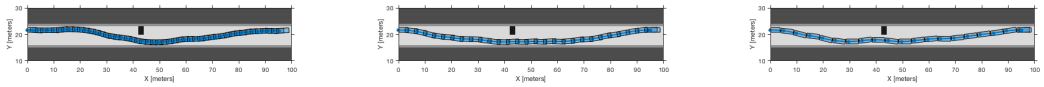
6.1 RRT* path planner as first attempt

Among the possible local path planners enumerated in section 4.1, the cell decomposition is one of them, in particular the RRT* planner can be employed for our purposes.
As described in section 5.3, in the first place we set the planner based on a costmap, that describes our chosen scenario. The chosen planner in Matlab is quite easy to set and since it works on a small map (i.e 100x100 m), also the computational time is pretty low. The path is evaluated once, as result of the probabilistic algorithm working on the entire map. In order to calculate the best trajectory we worked on two parameters: the minimum turning radius and the connection distance between two consecutive points. In particular, in 6.1 the tests with different turning radius are shown, while in figure 6.2 the connection distance parameter is analyzed. The optimal combination is given by the highest turning radius,i.e. 40 with the lowest connection distance,i.e. 2, shown in 6.1c and 6.2a, since they provide a smoother trajectory with more points to be followed.



(a) Test 1 (b) Test 2 (c) Test 3

Figure 6.1: Optimal paths by RRT* planner with turning radius equal to (a) 25 (b) 30 (c) 40



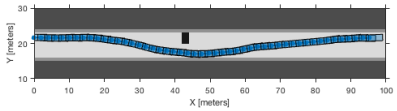
(a) Test 5 (b) Test 6 (c) Test 7

Figure 6.2: Optimal paths by RRT* planner with connection distance equal to (a) 2 (b) 3.5 (c) 5

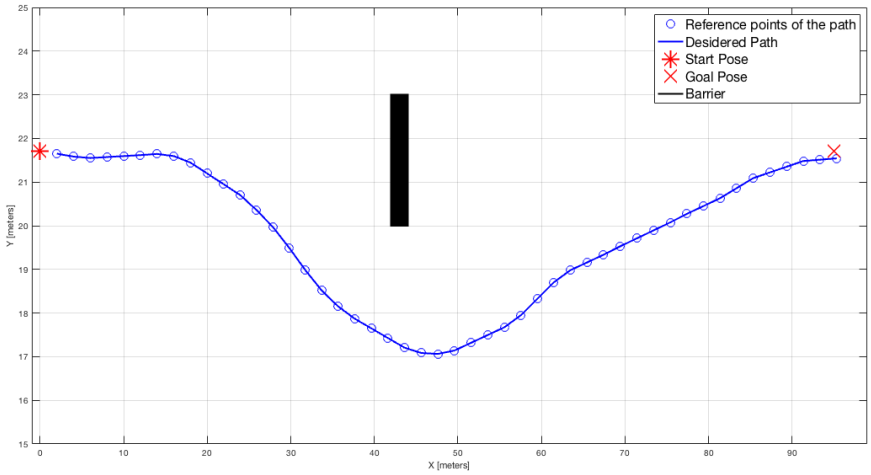
Once set the parameters of the planner, we carried three tests at different velocities. The values of the LQR parameters are reported in the table 6.1. The map with reference paths are shown in figures 6.3a, 6.4a and 6.5a. From figures 6.3c,6.4c and 6.5c instead, the tracking performance can be evaluated and it's clear that this task is harder to be achieved at higher speeds. Moreover, even if the RRT* planner considers the vehicle's kinematic in creating the trajectory, this one isn't so smooth. From the graphs reported in figures 6.3b,6.4b and 6.5b, where all the reference points coordinates are shown and then connected to form the complete trajectory, small changes of direction can be noticed in all three trajectories and this influences the control performance, creating a small tracking error at high velocities. Lastly, it can be noticed that even if these three trajectories are created with the same setting of the RRT* planner, they are slightly different in shape underling the fact that it's a randomized algorithm.

LQR matrices' weights					
V=5 m/s	Q	0.12	25	80	0.001
	R	5			
V=10 m/s	Q	0.12	30	80	0.001
	R	8			
V=15 m/s	Q	0.12	50	80	0.001
	R	20			

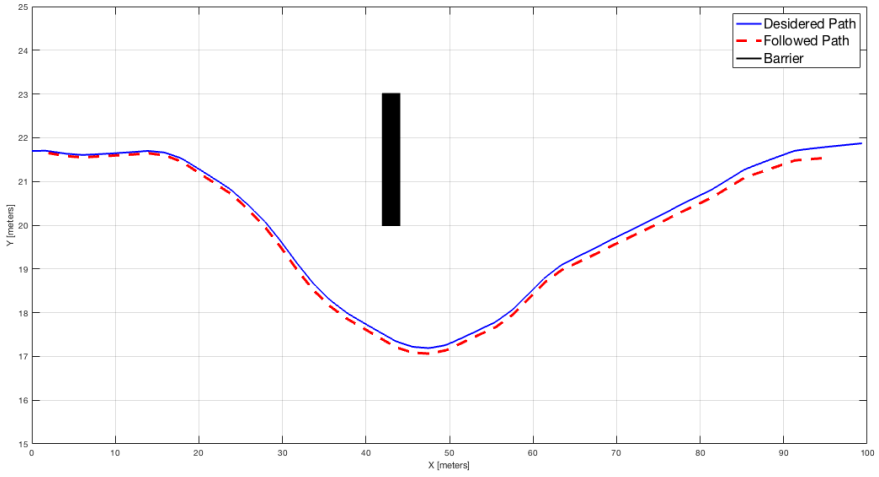
Table 6.1: LQR controller: values of Q and R matrices' weights for RRT* planner



(a) Map with reference path

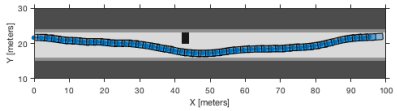


(b) Reference points

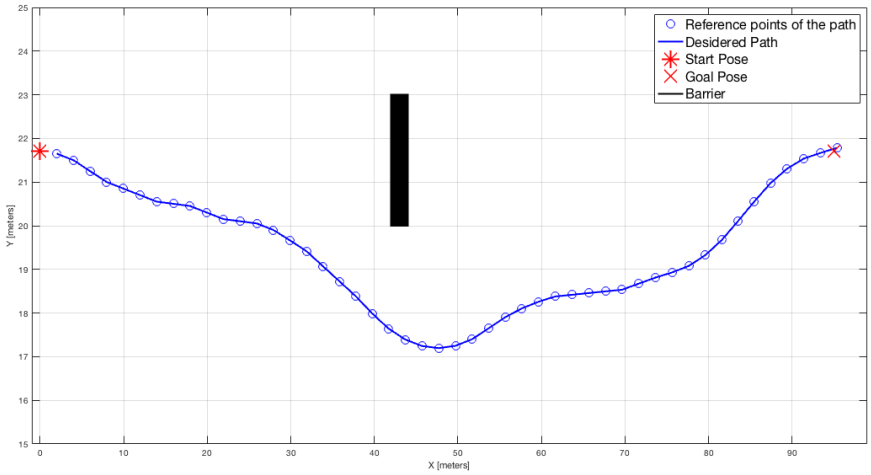


(c) Desidered and followed paths

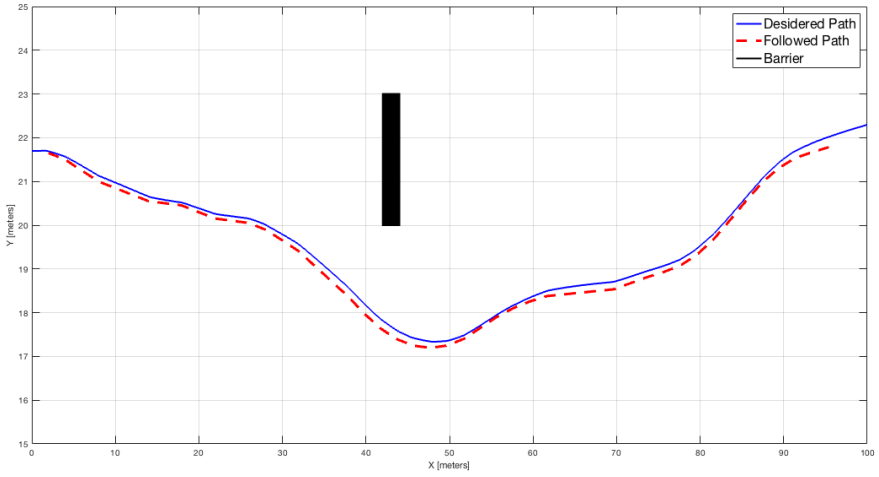
Figure 6.3: Reference trajectory at 5 m/s



(a) Map with reference path

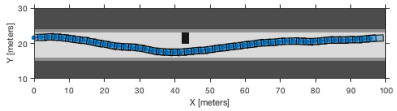


(b) Reference points

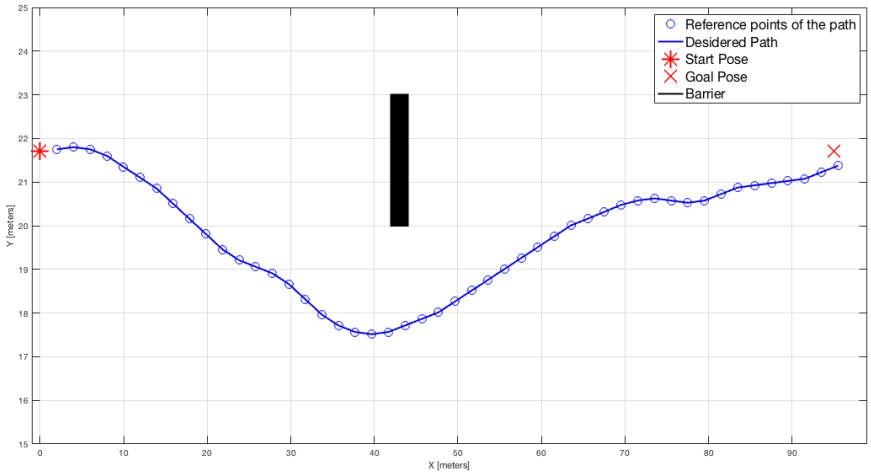


(c) Desidered and followed paths

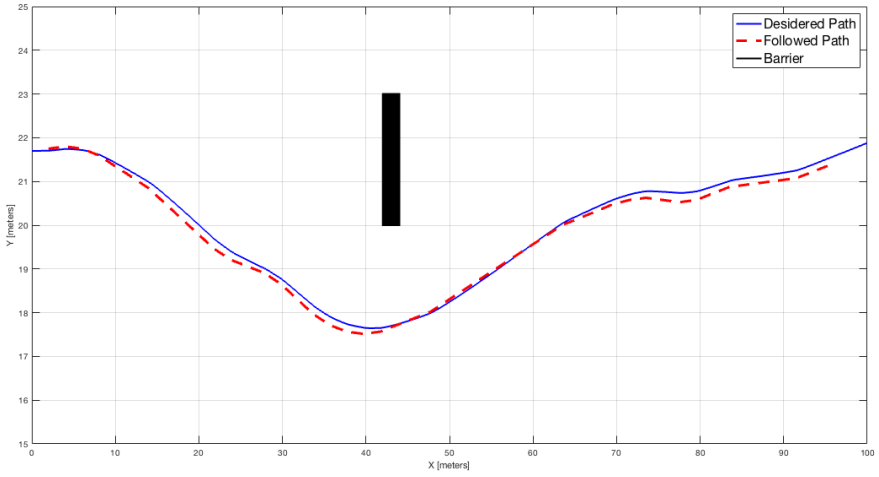
Figure 6.4: Reference trajectory at 10 m/s



(a) Map with reference path



(b) Reference points



(c) Desidered and followed paths

Figure 6.5: Reference trajectory at 15 m/s

6.2 Kinematic MPC as path planner and controller

In this section results related to the kinematic MPC are reported. As described in section 5.4, the MPC is both the path planner and the controller, thus these two properties will be investigated. The setup of this control scheme is described in the same aforementioned section, together with the values of the constraints and parameters in table 5.3 and table 5.4. In the first test the acceleration is kept variable, and so the speed, while in the following tests the speed is kept constant respectively at 5 m/s, 10 m/s and 15 m/s, meaning a zero acceleration value obtained by a change in the bounds' values.

6.2.1 Variable speed

For this test we set as initial speed a value of 10 m/s and as limit 20 m/s. The obstacle is 400 meters ahead of the vehicle at standstill. The vehicle's behavior results are shown below in figure 6.6, together with the two inputs, i.e. steering angle and acceleration. In particular from the speed graph it's clear how the vehicle accelerates in order to get to the goal point, decelerates encountering the obstacle and then accelerates again when the obstacle is overtaken. In figure 6.7 both the predicted trajectory and the followed one are shown. It's clear that the error between the two is so small to be considered irrelevant, in fact in the last graph of the figure where the two lines are superimposed, a difference cannot be noticed.

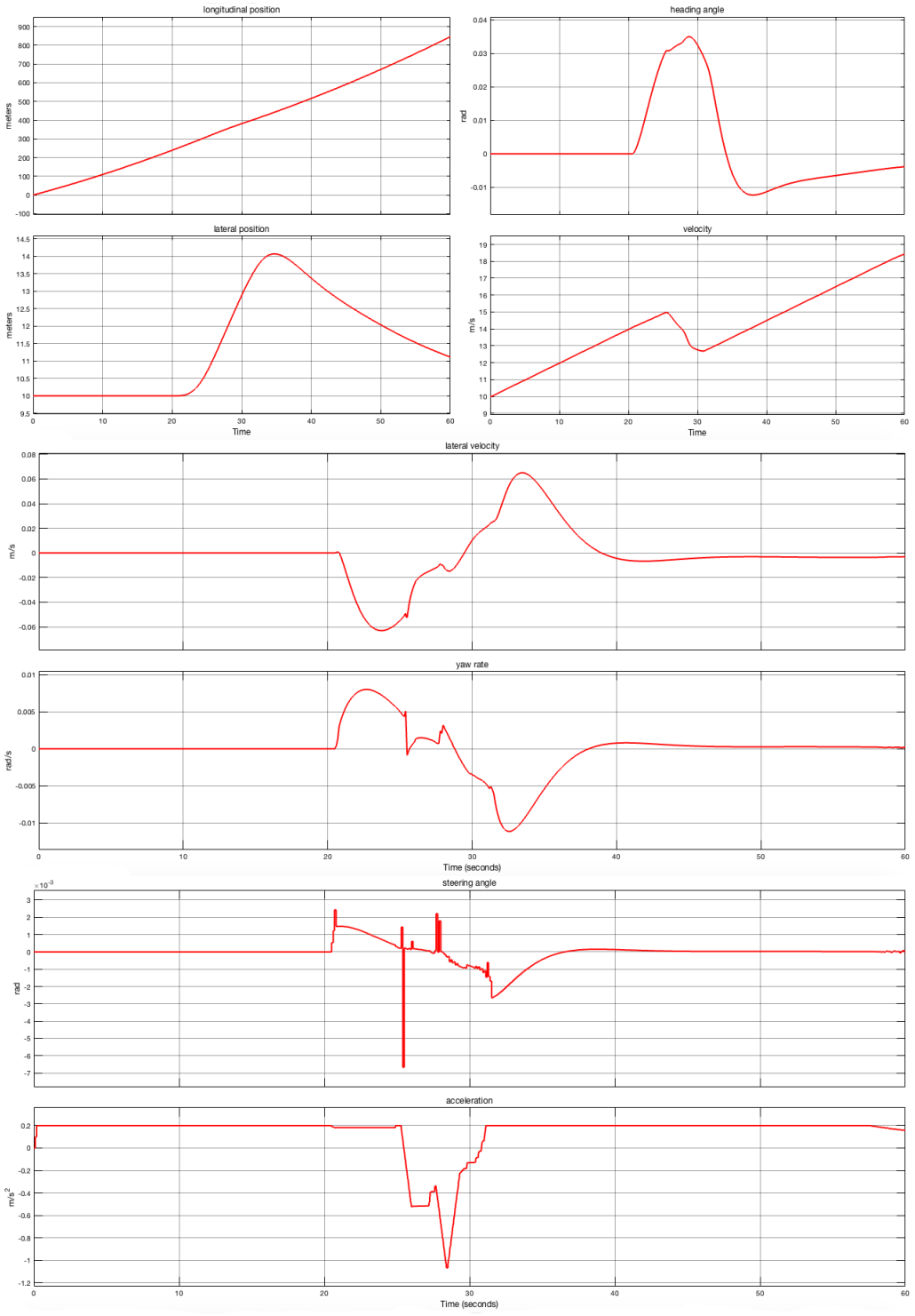


Figure 6.6: Vehicle's parameters and commands at variable speed

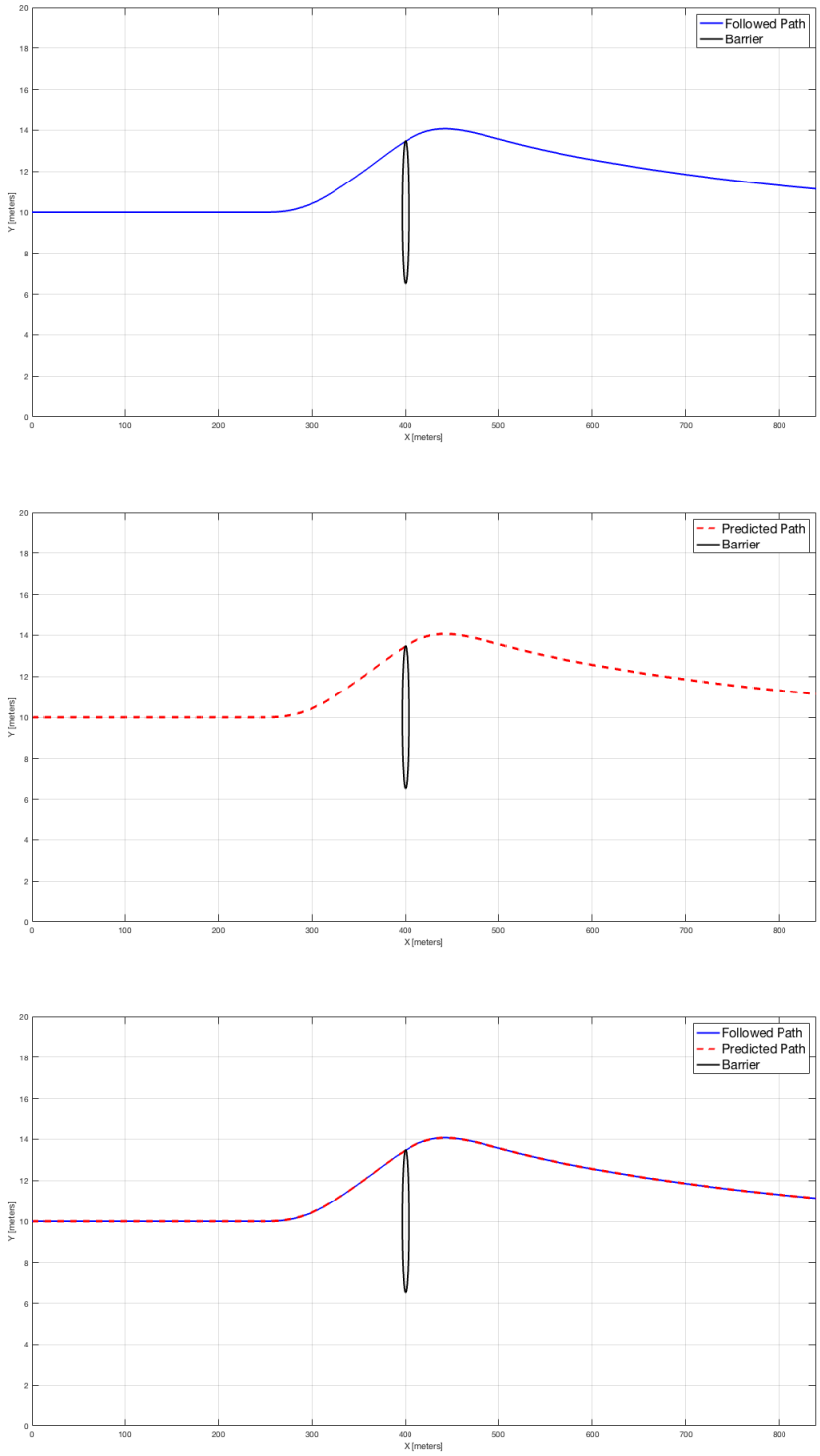


Figure 6.7: Followed trajectory, predicted trajectory and the two superimposed at variable speed;the circle stands for the obstacle at the center of the lane

6.2.2 Constant speed

In these tests the speed was kept constant at different values. To this purpose the acceleration input was locked to zero, as shown in the following figures.
From figures 6.10,6.12 and 6.14 it can be seen how the obstacle avoidance starts at different distances from the obstacle itself depending on the speed. Of course lower is the speed, later the maneuver starts. Moreover it can be observed that in all three tests, the vehicle returns very slowly in the initial lane, resulting in an unrealistic maneuver. In addition, this configuration underlines a speed limit. In fact, as shown in figure 6.8, at 21 m/s big oscillations are present after the overtaking.

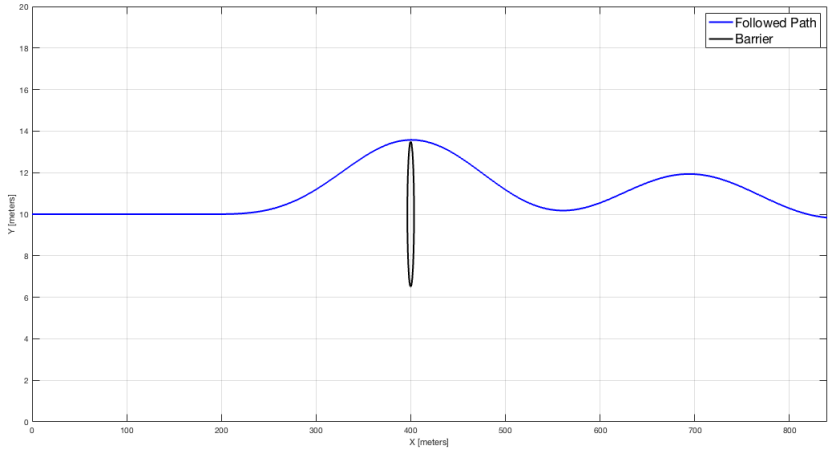


Figure 6.8: Limit trajectory at 21 m/s

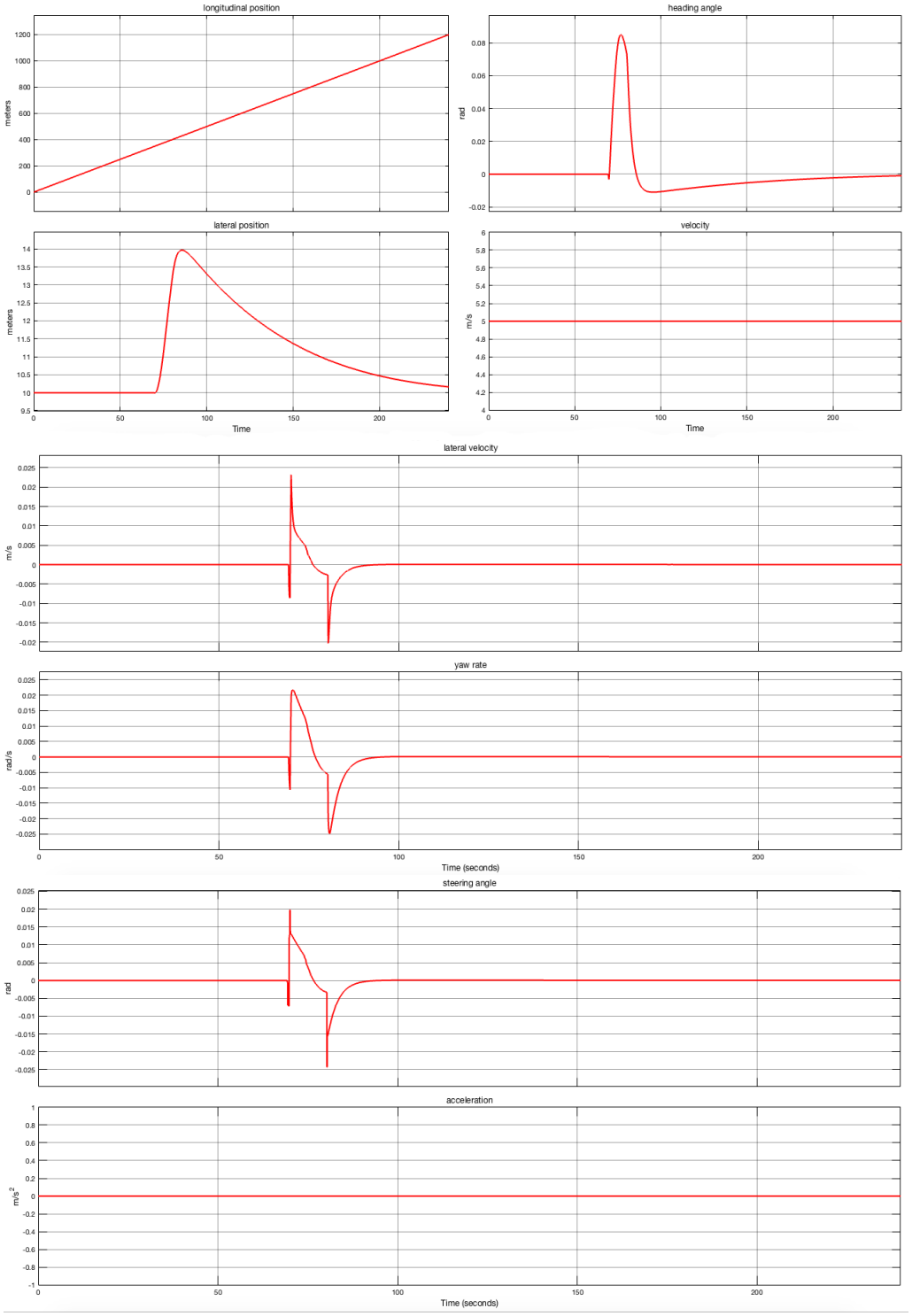


Figure 6.9: Vehicle's parameters and commands at 5 m/s

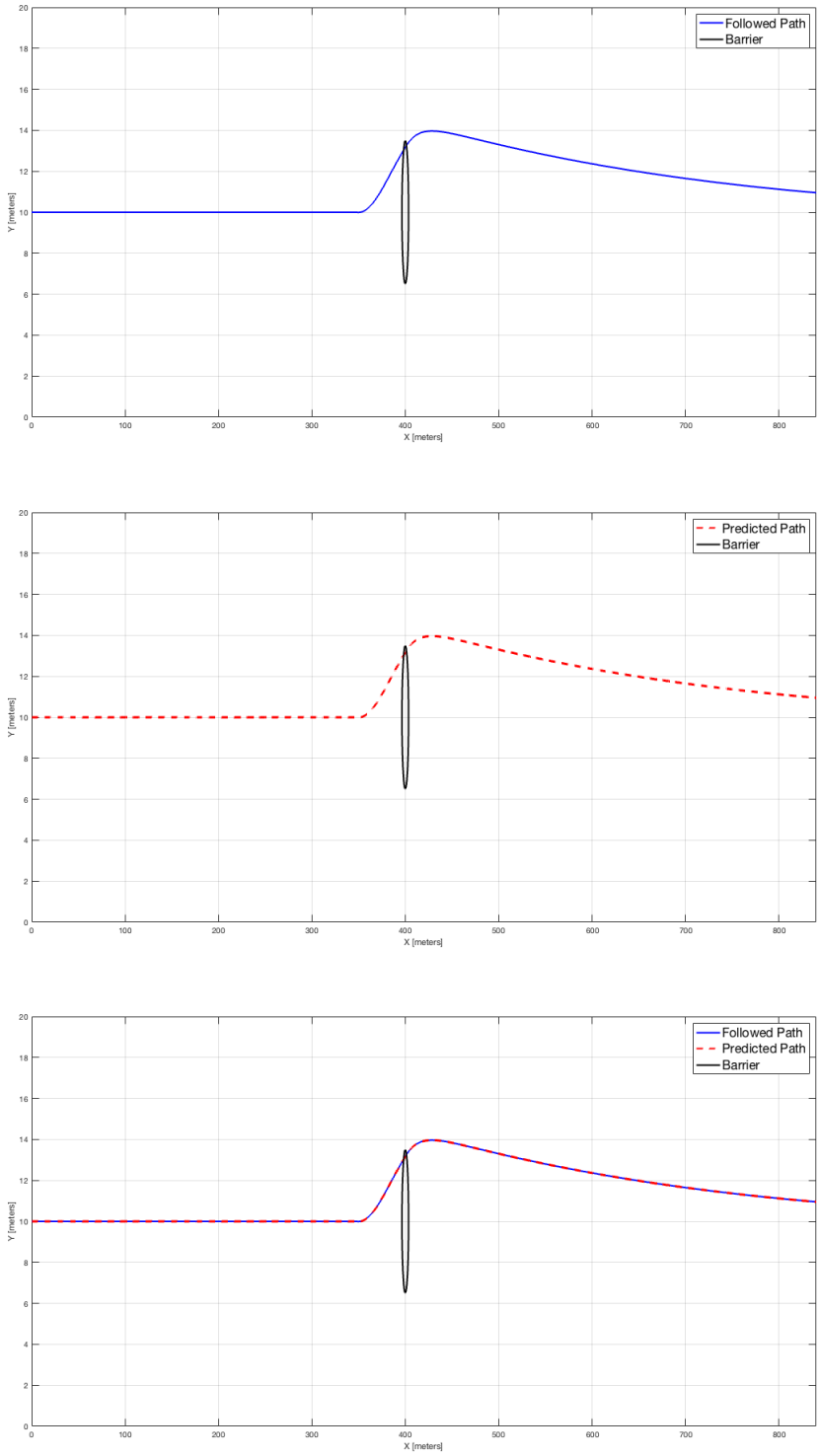


Figure 6.10: Followed trajectory, predicted trajectory and the two superimposed at $5\,m/s$; the circle stands for the obstacle at the center of the lane

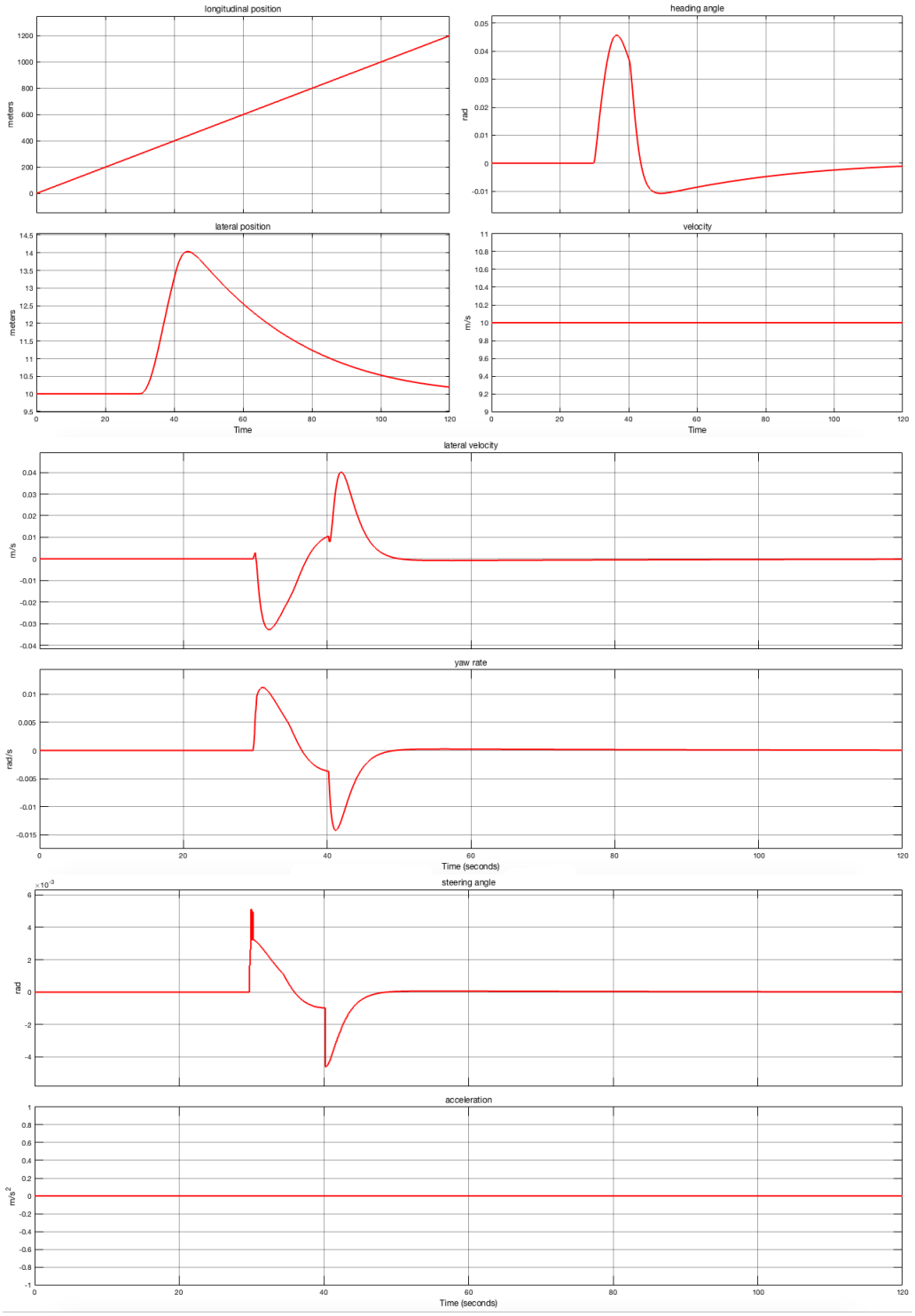


Figure 6.11: Vehicle's parameters and commands at 10 m/s

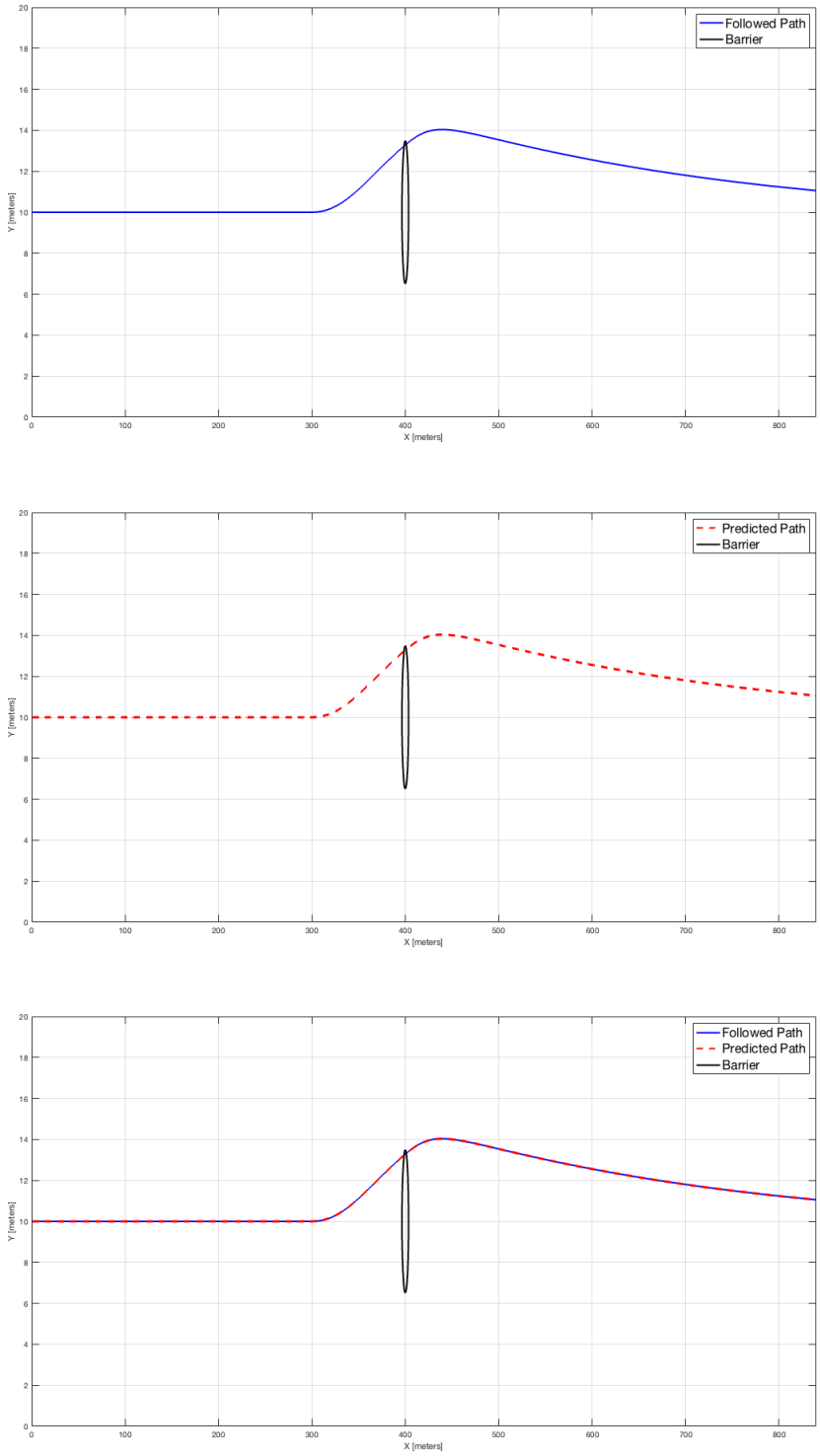


Figure 6.12: Followed trajectory, predicted trajectory and the two superimposed at 10 m/s; the circle stands for the obstacle at the center of the lane

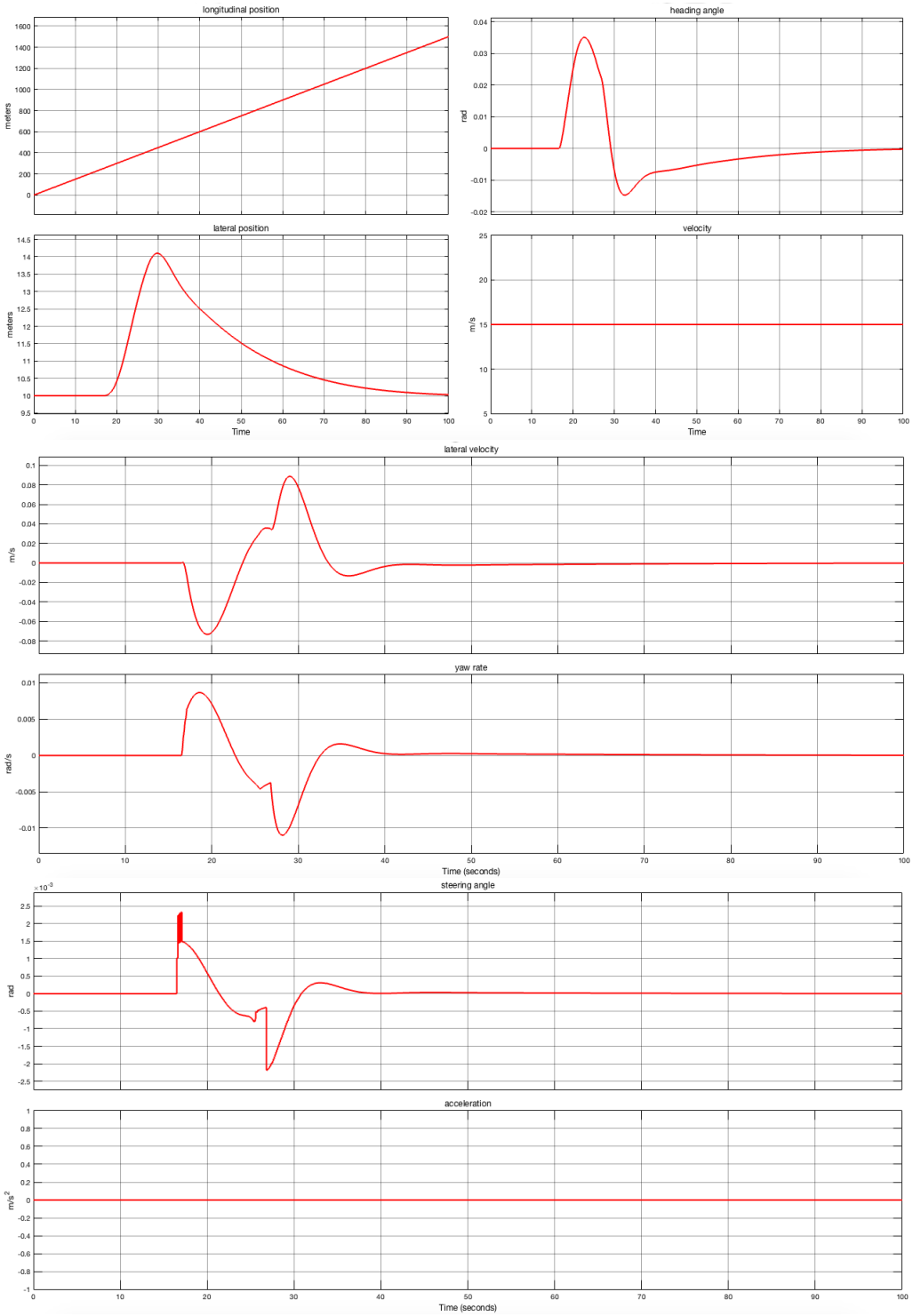


Figure 6.13: Vehicle's parameters and commands at 15 m/s

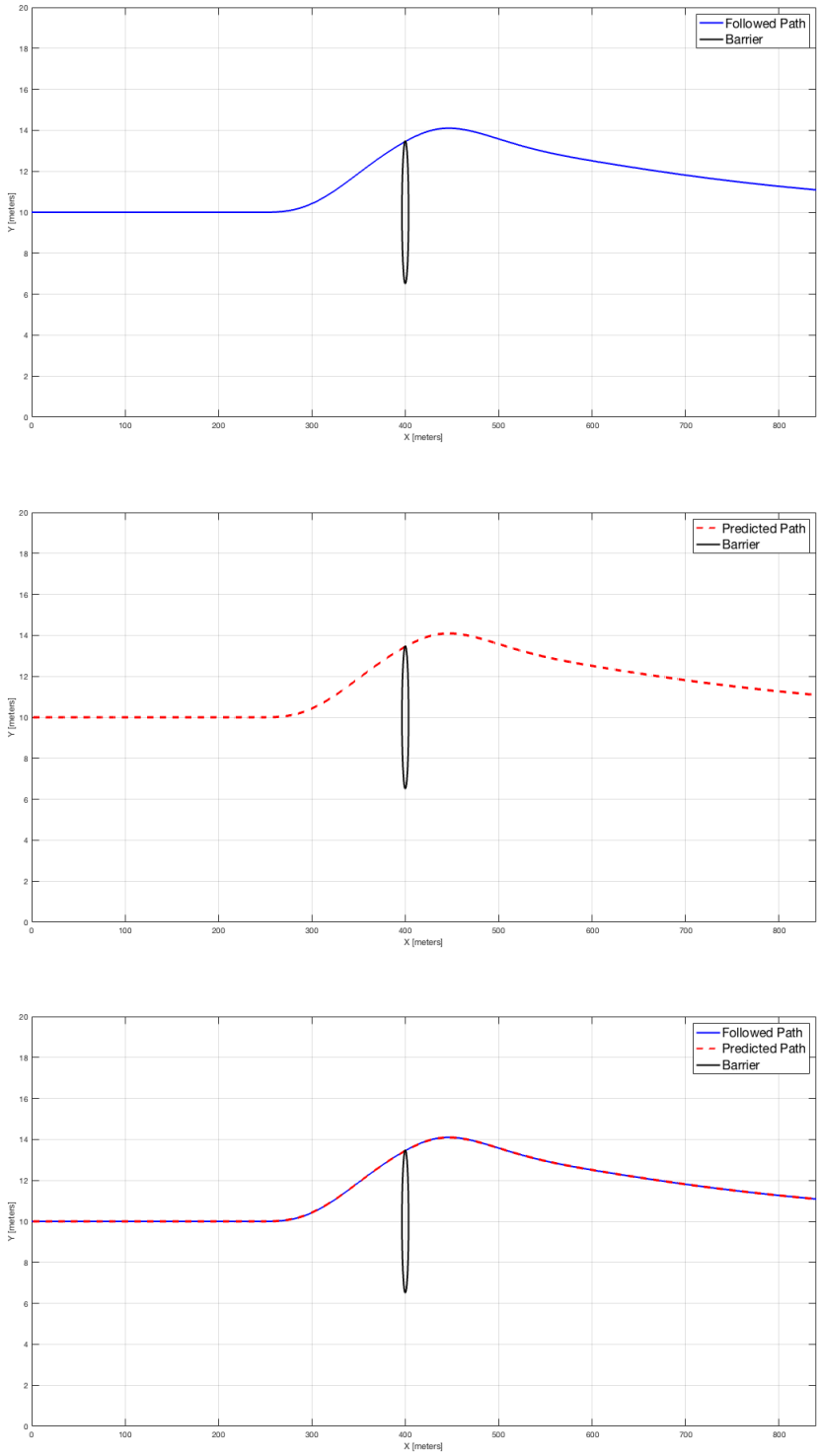


Figure 6.14: Followed trajectory, predicted trajectory and the two superimposed at 15 m/s; the circle stands for the obstacle at the center of the lane

6.3 MPC as path planner

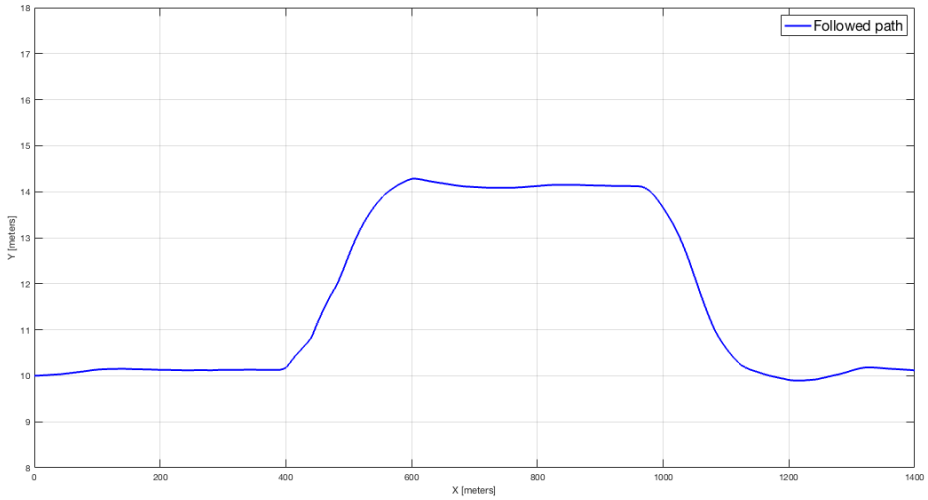
The main disadvantages of the previous design were already underlined. So, in this last phase we focus on improving the following features:

- Speed's limit
- Trajectory shape

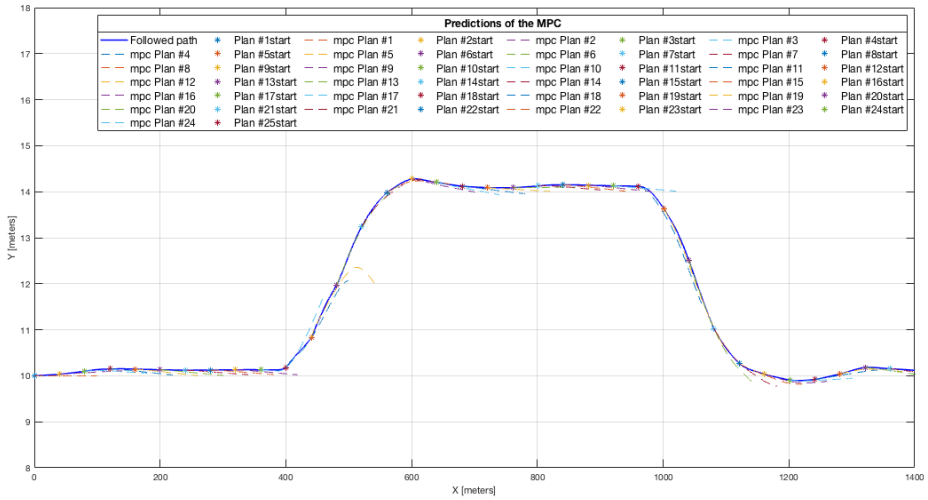
For this reason we limited the MPC only to the path planner function, while a standard LQR was used as controller. Furthermore, in order to solve the aforementioned features, we added an update of the reference of the MPC, as explained in section 5.5.3 and we designed the entire control scheme at high speeds. In table 6.2 the values of the LQR matrices are shown. Two tests are reported at 20 m/s and 25 m/s. The conditions are to the limits. The models we used in fact, do not take into account the non-linearities of the tires, which are present at these velocities. Really small oscillations can be seen in the heading angle graph reported in figures 6.17a and 6.17b. Nonetheless the created trajectory is satisfactory. In figures 6.15a and 6.16a the followed paths are shown. The shape of the trajectory is smooth and realistic, with the right distance before and after the obstacle. In figures 6.15b and 6.16b we can understand how the MPC works. In fact, all the predicted trajectories of the MPC during the simulation are displayed. As already explained, every $\triangle t$, which we set at 2 seconds, a prediction is made, so in a complete simulation we have 40 predictions.

LQR matrices' weights				
Q	0.00005	18	25	0
R	15			

Table 6.2: LQR controller: values of Q and R matrices' weights for MPC as path planner

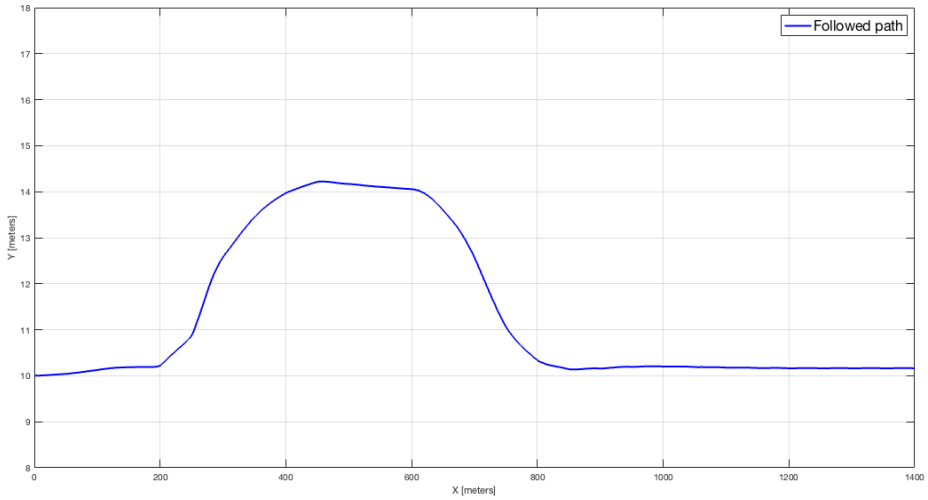


(a) Followed path: the vehicle to be overtaken is 200 meters ahead at the start position and it moves at 15 m/s

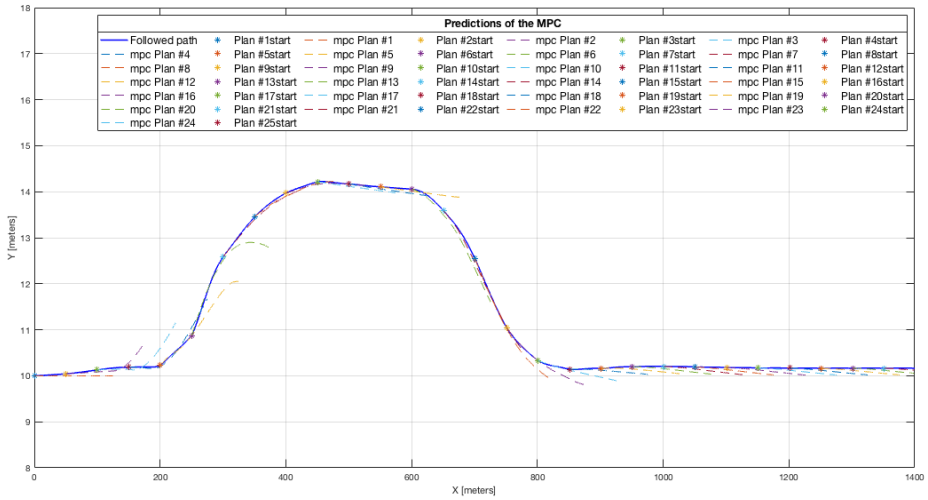


(b) MPC predicted trajectories

Figure 6.15: MPC as path planner: followed path and predicted trajectories at 20 m/s

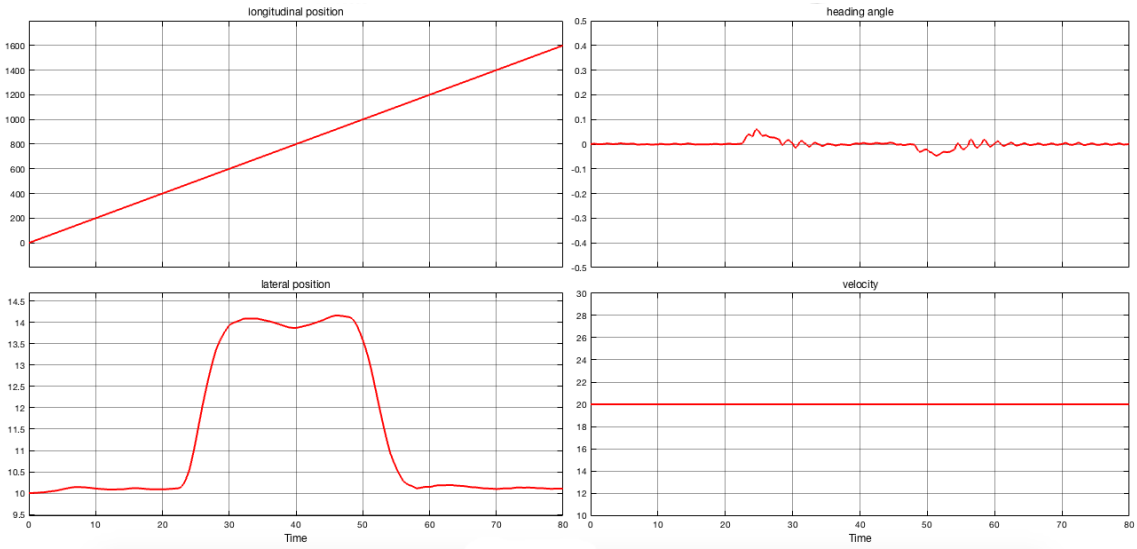


(a) Followed path: the vehicle to be overtaken is 200 meters ahead at the start position and it moves at 15 m/s

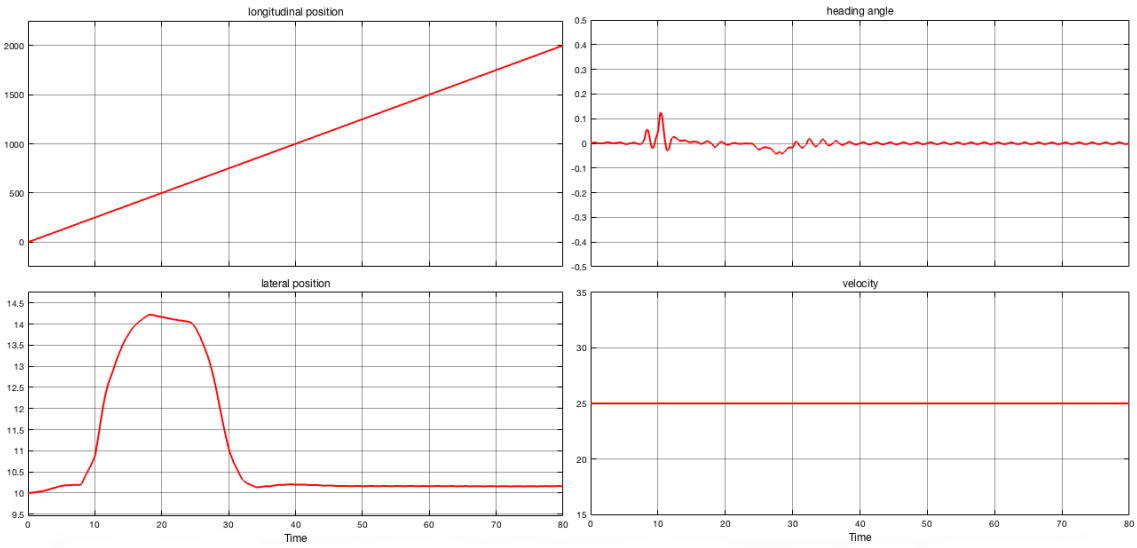


(b) MPC predicted trajectories

Figure 6.16: MPC as path planner: followed path and predicted trajectories at 25 m/s



(a) Trajectory parameters on Simulink at 20 m/s



(b) Trajectory parameters on Simulink at 25 m/s

Figure 6.17: Trajectory parameters on Simulink at 20 m/s and 25 m/s

6.4 Conclusions and future works

In this chapter the results of the three different phases of the project were reported. In a preliminary phase, we designed a RRT* planner which has to calculate an optimal trajectory based on a map. The scenario was very simple,i.e. a road and a barrier to be overtaken. Several tests were done in order to obtain a satisfactory lane change trajectory, but as shown in section 6.1 small turns were still present, resulting in difficult path to track at high speeds. Also the setting of the controller was not so trivial, since the RRT* planner gives a slightly different trajectory every time the algorithm is run due to the randomly building of the space-filling tree. Secondly we tested a kinematic MPC, with a similar scenario of the previous case,i.e. road with a barrier, and the advantages of this technique have been confirmed. The possibility of writing the constraints of the optimization problem in a so efficient way has been a plus for this design. The resulting trajectory is for sure smoother and the control action, done by the same MPC, is definitely better. Thus, this step brought to a better result with a less complex system. Lastly we isolated the MPC path planning function and brought some improvements in the trajectory's creation. Indeed, higher velocities have been achieved together with a proper timing in the maneuver, resulting a more realistic scenario, where the obstacle was not at standstill but moving along a straight line.
The obtained results can be a good base for further improvements in this field. For sure, the car simulation can be improved, avoiding the simulation of the car in Simulink and using more precise software such as CarSim. Moreover also the environmental data can be collected by a proper sensor data system, which is a key feature for the autonomous driving. Nonetheless this project highlighted the benefits of the Model Predictive Control technique, which for sure will be further tested in the Automatic Driving Assistance Systems field and more in general in the autonomous driving branch.

Appendix

FollowPath

```
function reference = fcn(x)
persistent i;
global newpath;
if isempty(i)
        i = 1;
end

actualPose = [x(1) ; x(2) ; x(3)];
actualGoal = [newpath(i,:)];
%longitudinal distance to evaluate next point
longitudinal_distance = sin(actualGoal(3))*((actualGoal(2)-x(2))+
(cos(actualGoal(3))*((actualGoal(1)-x(1))));
if longitudinal_distance ≤ 0
    if i < length(newpath)
        i = i+1;
    end
    actualGoal = [newpath(i,:)];
end
reference = [actualGoal(1:2), actualGoal(3)];
```

UpdateReference

```
function [out] = UpdateReferencePath(states,t,obs)
%% Init
persistent refPath;
persistent pp_iterations;
persistent ctrl;
persistent last_time;
persistent mpcPlans
global cc cr l changed Ts Tfin Tp V

cc = obs(1:2);   %obstacle position

changed =false;
if isempty(ctrl)
    ctrl = 1;
end
if isempty(last_time)
    last_time = 0;
end
if isempty(pp_iterations)
    pp_iterations = 0;
end

%% Parameters for MPC
Δ_time = 2;
Tfin=5; %MPC simulation time

yr = updateGoal(states,cc); %goal

%% Execution

if ctrl == 1 || (t-last_time ≥ Δ_time)
    ctrl = 2;
end


if ctrl==2 || pp_iterations == 0
    if pp_iterations == 0
        x0 = [0;10;0,;V];
    else
        x0 = [states;V];
    end
    l=4;
    cr=2; %safety radius

    par.model=@kinematic_model;
    par.nlc=@rover_con_1;
```

```
47      par.n=4;
48      par.Tp=Tp;
49      par.Ts=Ts;
50      par.ub=[30/180*pi];
51      par.lb=[-30/180*pi];
52      par.tol=[.1;.1];
53
54      par.R=0.1; %weight on MV
55      par.P=0.1*diag([1;0.5]); % final tracking error
56      par.Q=3*diag([2;0.5]); %error on tracking output
57
58      K=nmpc_design(par);
59
60      size_out=[Tfin/Ts+2 3];
61
62      simOut = sim('cinC2_cinobs','SrcWorkspace', 'current');
63
64      refPath = simOut.x.Data(:,1:3);
65      mpcPlans(:,:,pp_iterations+1) = refPath;
66      pp_iterations = pp_iterations+1;
67      if ctrl == 2
68          ctrl = 3;
69      end
70      changed = true;
71      last_time = t;
72
73      assignin('base','pp_iterations',pp_iterations);
74      assignin('base','mpcPlans',mpcPlans);
75  end
76
77  l_refPath = length(refPath);
78  out = zeros(Tfin/Ts+2,3);
79  out(1,:) = states';
80  out(2:end,:) = refPath(:,1:3);
81  end
```

UpdateGoal

```
function [goal] = updateGoal(x,cc)
%Change of reference based on distance from the obstacle
ΔBeyond = 40; %meters

ΔBefore = 80;  %meters

if ( x(1) ≥  cc(1) - ΔBefore ) && ( x(1) ≤ cc(1) + ΔBeyond )
    goal = [14 0];  %fast lane coordinates
else
    goal = [10 0];
end

end
```

References

[1] NHTSA-National Highway Traffic Safety Administration. Automated vehicles for safety, 2018.

[2] Nagy Zoltan K Allgower Frank, Findeisen Rolf et al. Nonlinear model predictive control: From theory to application. *Journal-Chinese Institute Of Chemical Engineers*, 35(3):299–316, 2004.

[3] Brian DO Anderson and John B Moore. *Optimal control: linear quadratic methods*. Courier Corporation, 2007.

[4] Sterling J Anderson, Steven C Peters, Tom E Pilutti, and Karl Iagnemma. An optimal-control-based framework for trajectory planning, threat assessment, and semi-autonomous control of passenger vehicles in hazard avoidance scenarios. *International Journal of Vehicle Autonomous Systems*, 8(2-4):190–216, 2010.

[5] Francesco Borrelli, Paolo Falcone, Tamas Keviczky, Jahan Asgari, and Davor Hrovat. Mpc-based approach to active steering for autonomous vehicle systems. *International Journal of Vehicle Autonomous Systems*, 3(2):265–291, 2005.

[6] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[7] Eduardo F Camacho and Carlos Bordons Alba. *Model predictive control*. Springer Science & Business Media, 2013.

[8] Novara Carlo. Non linear model predictive control. Lecture notes.

[9] Carvalho Ashwin Cesari Gianluca, Schildbach Georg and Borrelli Francesco. Scenario model predictive control for lane change assistance and autonomous driving on highways. *IEEE Intelligent Transportation Systems Magazine*, 9(3):23–35, 2017.

[10] Howie Choset. Robotic motion planning: A* and D* search, 2010.

[11] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

[12] Shilp Dixit, Saber Fallah, Umberto Montanaro, Mehrdad Dianati, Alan Stevens, Francis Mccullough, and Alexandros Mouzakitis. Trajectory planning and tracking for autonomous overtaking State-of-the-art and future prospects. *Annual Reviews in Control*, 45:76–86, 2018.

[13] Paolo Falcone, H Eric Tseng, Francesco Borrelli, Jahan Asgari, and Davor

Hrovat. Mpc-based yaw and lateral stabilisation via active front steering and braking. *Vehicle System Dynamics*, 46(S1):611–628, 2008.

[14] Huckleberry Febbo, Jiechao Liu, Paramsothy Jayakumar, Jeffrey L Stein, and Tulga Ersal. Moving obstacle avoidance for large, high-speed autonomous ground vehicles. In *American Control Conference (ACC), 2017*, pages 5568–5573. IEEE, 2017.

[15] Janick V Frasch, Andrew Gray, Mario Zanon, Hans Joachim Ferreau, Sebastian Sager, Francesco Borrelli, and Moritz Diehl. An auto-generated nonlinear mpc algorithm for real-time obstacle avoidance of ground vehicles. In *2013 European Control Conference (ECC)*, pages 4136–4141. IEEE, 2013.

[16] Lex Fridman. MIT Deep Learning, 2018.

[17] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[18] SAE international. Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles, 2018.

[19] Dongchul Kim Chulhoon Jang Kichun Jo, Junsoo Kim and Myoungho Sunwoo. Development of autonomous car - part i: Distributed system architecture and development process. *IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS*, 61(12):7131–7140, 2014.

[20] Jason Kong, Mark Pfeiffer, Georg Schildbach, and Francesco Borrelli. Kinematic and dynamic vehicle models for autonomous driving control design. In *Intelligent Vehicles Symposium*, pages 1094–1099, 2015.

[21] Steven M LaValle. Rapidly-exploring random trees: A new tool for path planning. *CiteSeerX*, 1998.

[22] Steven M. LaValle. *Planning Algorithms*. Cambridge Univeristy Press, 2006.

[23] MathWorks. Model predictive control toobox-getting started guide, 2018.

[24] MathWorks. pathplannerrrt function, 2018.

[25] Mathworks. Vehicle body 3dof, 2018.

[26] MathWorks. vehiclecostmap function, 2018.

[27] Piero Micelli. *Path planning for road vehicles by dynamic programming*. PhD thesis, Università di Parma, Dipartimento di Ingegneria ed Architettura, 2018.

[28] Zdzislaw Kowalczuk Michal Czubenko and Andrew Ordys. Autonomous driver based on an intelligent system of decision–making. *Springerlink.com*, 2015.

[29] neos Guide. Optimization guide, 2018.

[30] Iram Noreen, Amna Khan, and Zulfiqar Habib. A comparison of rrt, rrt* and rrt*-smart path planning algorithms. *International Journal of Computer Science and Network Security (IJCSNS)*, 16(10):20, 2016.

[31] Teddy Ort, Liam Paull, and Daniela Rus. Autonomous vehicle navigation in rural environments without detailed prior maps. In *International Conference on Robotics and Automation*, 2018.

[32] Michael Otte and Emilio Frazzoli. Rrtx: Real-time motion planning/replanning for environments with unpredictable obstacles. In *Algorithmic Foundations of Robotics XI*, pages 461–478. Springer, 2015.

[33] Brian Paden, Michal Čáp, Sze Zheng Yong, Dmitry Yershov, and Emilio Frazzoli. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on intelligent vehicles*, 1(1):33–55, 2016.

[34] Rajesh Rajamani. *Vehicle dynamics and control*. Springer Science & Business Media, 2011.

[35] Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, and Giuseppe Oriolo. *Robotics: modelling, planning and control*. Springer Science & Business Media, 2010.

[36] Eloy Snapper. Model-based path planning and control for autonomous vehicles using artificial potential fields. Master's thesis, Delft University of Technology, 2018.

[37] P Svestka, JC Latombe, and LE Overmars Kavraki. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[38] Charles W Warren. Global path planning using artificial potential fields. In *Robotics and Automation, 1989. Proceedings., 1989 IEEE International Conference on*, pages 316–321. IEEE, 1989.

[39] Wikipedia. Quadratic prgramming — wikipedia, the free encyclopedia, 2018.

[40] Wikipedia. Motion planning — wikipedia, the free encyclopedia, 2019.

[41] Michael T Wolf and Joel W Burdick. Artificial potential functions for highway driving with collision avoidance. In *2008 IEEE International Conference on Robotics and Automation*, pages 3731–3736. IEEE, 2008.

[42] Fitri Yakub and Yasuchika Mori. Comparative study of autonomous path-following vehicle control via model predictive control and linear quadratic control. *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of automobile engineering*, 229(12):1695–1714, 2015.

[43] Boliang Yi, Stefan Gottschling, Jens Ferdinand, Norbert Simm, Frank Bonarens, and Christoph Stiller. Real time integrated vehicle dynamics control and trajectory planning with mpc for critical maneuvers. In *Intelligent Vehicles Symposium (IV), 2016 IEEE*, pages 584–589. IEEE, 2016.