Motorvehicle University of Emilia-Romagna

Master Degree in Advanced Automotive Electronic Engineering

MPC for Dynamic Obstacle Avoidance

Technical report on the group project activity

Course of Compliance Design of Automotive Systems

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

a.y. 2020-2021

Abstract

Every year 1.25 million people die and as many as 50 million are injured in road traffic accidents worldwide, according to United Nations statistics. Human error is involved in about 95% of all road traffic accidents in the EU, and in 2017 alone, 25,300 people died on the Union's roads [1]. Autonomous cars can improve road safety and through new technologies it is also possible to reduce traffic congestion and CO2 emissions.

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

Contents

1 Introduction **5**
1.1 Model-based Design 6

2 System requirements **8**

3 System Implementation **9**
3.1 Analysis of requirements 9
3.2 Partitioning 9

List of Figures

1 MPC Block Diagram 5
2 System Development V-model 6
3 System Partitioning 10

List of Tables

1 System requirements and possible validation methods 9

1 Introduction

The task of avoiding obstacles is one of the key issues when it comes to the vehicular scenario and it is more difficult to perform it here than it is in static environments. A vehicle with an obstacle avoidance system is equipped with sensors that measure the distance between the car itself and the obstacle in the same lane. If an autonomous car encounters an obstacle, it is expected to move temporarily to another lane and move back to the original lane once it has driven past the obstacle. This type of control can be implemented using a Model Predictive Control (MPC). An MPC is an advanced control method that works in discrete time and uses a model of the system to make predictions about the system's future behavior. MPC solves an online optimization algorithm to find the optimal control action that drives the predicted output to the reference [2]. In other words, from a set of state values, and with respect to a model, it optimizes a problem around an objective and gives a sequence of control signals as outputs. The first set of control values are then used as inputs to the system plant, and after a short period, set as the *system time step*, the new state values are measured and the process is repeated. The overall objectives of the MPC are:

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

Figure 1 shows a block diagram for a Model Predictive Control.

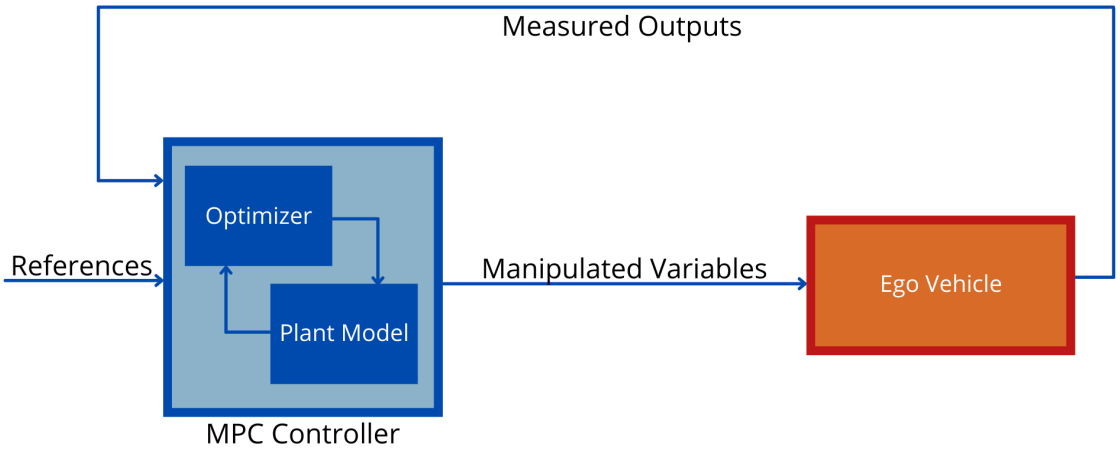

Figure 1: MPC Block Diagram

An MPC controller has two main functional blocks: the optimizer and the plant model. The dynamic optimizer allows to find the optimal input that gives the minimum value of the cost function taking into account all the constraints. Generally, a non linear model is used for the validation of the controller, while the plant model used for the MPC is a linearized version of the actual plant.

Our project is based on the use of an *adaptive* MPC which means that the plant state has to be measured again to be adopted as the linearization point for the next step of the predictive control. When the plant state is re-sampled, the whole process computes again the calculations starting from the new current state.

1.1 Model-based Design

When developing a project, especially concerning embedded systems, it is crucial to follow a process model which illustrates the high-level activities and their phasing during development.
As stated in [3] *'Process models provide high-level perspective that helps team members understand what activities to do and what progress has been made on each of those development activities"*.
The development process of our system is based on the V-model shown in Figure 2. Starting from the general V-model we have tailored our own V-model in order to meet our needs; doing so we have managed to skip some unnecessary steps in the development process while still being compliant with the definition of the V-model itself.

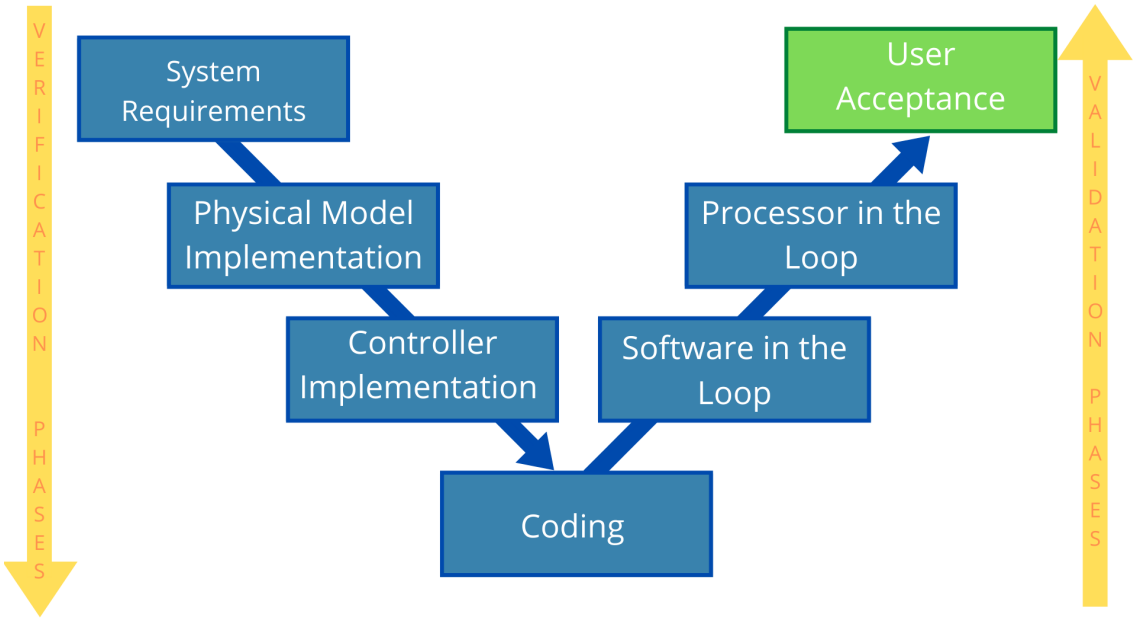

Figure 2: System Development V-model

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

- **System requirements**: this is the first step of each project, it consists in the requirements gathering from the customer;
- **Physical model implementation**: this phase contains the system design of our vehicle in terms of sensors required and dynamic and kinematic model;

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

2 System requirements

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

1. Maximum lateral error from reference of $0.5m$;
2. Detection of obstacles within $100m$ ahead of vehicle;
3. Move on left lane within a predetermined safe zone from the obstacle[1];
4. Once passed the obstacle, come back on right lane with no less than $10m$ but no more than $50m$ ahead of it;
5. Maximum lateral acceleration of $0.5m/s^2$;
6. All previous requirements satisfied in speed range from $10km/h$ to $100km/h$.

[1]Third to fifth requirements are requested when an obstacle is detected

3 System Implementation

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

3.1 Analysis of requirements

Starting from the detailed system requirements described in Section 2, we can obtain low level requirements suitable for the controller implementation; each of these is associated with a draft of the validation method needed to assert each of them. Table 3.1 shows the system requirements, low level requirements and the corresponding validation methods.

System requirements	Low level requirements	Validation methods
Maximum lateral error from reference of $0.5m$	Inequality constraint: $e_y = abs(Y_{reference} - Y_{vehicle}) <= 0.5m$	Check in each point that the difference between the actual vehicle position and the reference path is not greater than $0.5m$
Detection of obstacles within 100 m ahead of vehicle	Sensor fusion able to provide useful data in the range $0 - 100m$ from the vehicle	Drive toward a static obstacle and confirm that is detected when is $100m$ far
Move on left lane within a predetermined safe zone from the obstacle	Inequality constraint when obstacle detected: $dist = position_{vehicle} - position_{obstacle} \in SafeZone$	Confirm that when changing lane the distance from the obstacle measured by the sensors is greater than the predetermined safe zone
Once passed the obstacle, come back on right lane with no less than $10m$ but no more than $50m$ ahead of it	Inequality constraint when obstacle passed: $10m <= dist <= 50m$	Confirm that when changing lane the distance from the obstacle measured by the sensors is greater than the predetermined safe zone
Maximum lateral acceleration of $0.5m/s^2$	Constraint on Model Predictive Control input: $a.max = 0.5$	Check that the maximum value registered by the IMU−accelerometer is not greater than $0.5m/s^2$
All previous requirements satisfied in speed range from $10km/h$ to $100km/h$	Verify all previous constraint when simulating at speed range $10 - 100km/h$	Verify all the above methods with vehicle travelling from $10km/h$ to $100km/h$

Table 1: System requirements and possible validation methods

3.2 Partitioning

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

project on the tasks shown in Figure 3.

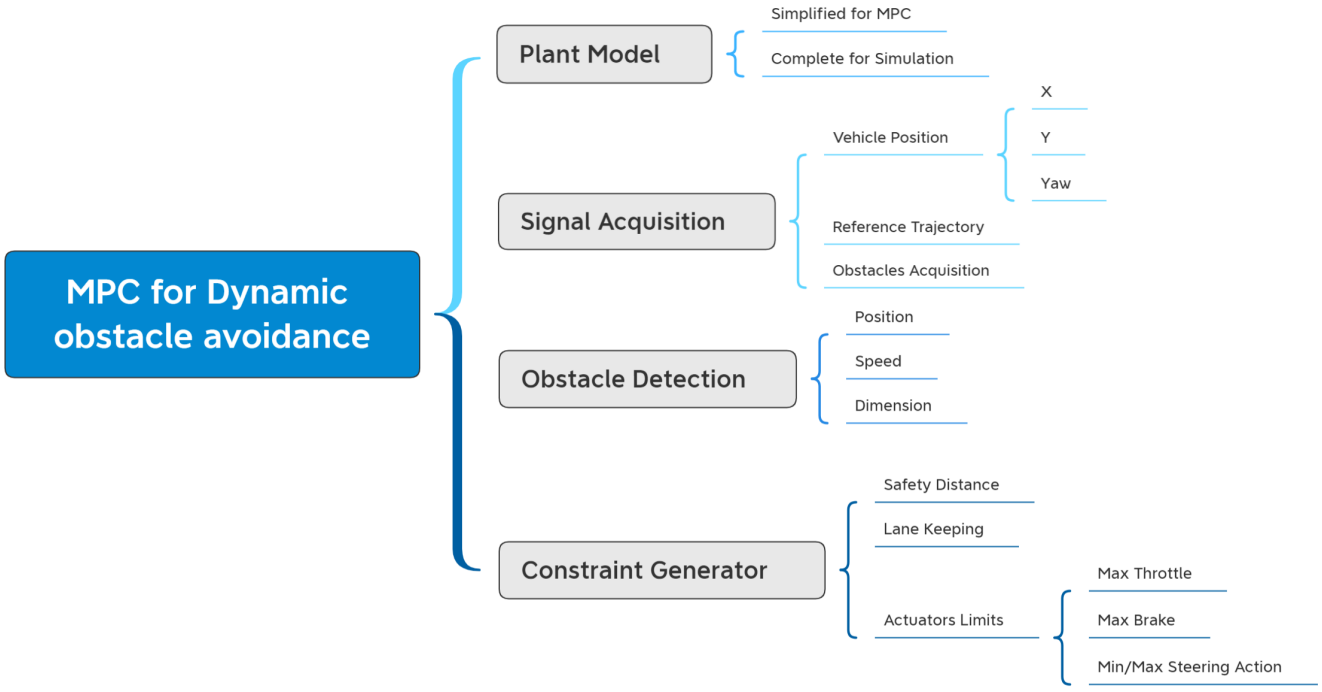

Figure 3: System Partitioning

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

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

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

References

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

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

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