TÉCNICO LISBOA

MPC Motion Control of an Autonomous Car

Tomás Fernandes dos Santos Cabrita Carneiro

Thesis to obtain the Master of Science Degree in

Aerospace Engineering

Supervisors: Prof. João Manuel Lage de Miranda Lemos
Prof. Rita Maria Mendes de Almeida Correia da Cunha

Examination Committee

Chairperson: Prof. Paulo Jorge Coelho Ramalho Oliveira
Supervisor: Prof. Rita Maria Mendes de Almeida Correia da Cunha
Member of the Committee: Prof. António Pedro Rodrigues de Aguiar

January 2021

Acknowledgments

First, I would like to thank my supervisors Professor João Miranda Lemos and Professor Rita Cunha for sharing their invaluable knowledge and advise throughout every step of this work. Without their feedback none of this would have been possible.

To all my friends, especially Fon, each in your own way, thank you for helping me throughout this journey. To my girlfriend Inês, thank you for always being there to comfort me and support me to achieve my goals.

Finally, to my Grandparents, Parents, and Brothers, I would like to express my deepest gratitude for the love and support that helped me to grow into the person I am today. Specially to my Avó Carmo, who kept motivating me to finish this hard, but amazing, journey.

This work was developed under project HARMONY, Distributed Optimal Control for Cyber-Physical Systems Applications, funded by FCT through POR Lisboa - LISBOA-01-0145-FEDER- 031411.

Resumo

Este trabalho aborda o projeto de um controlador usando Controlo Preditivo (MPC) para controlar um veículo de corrida autónomo. O objetivo do controlador é conduzir o veículo para seguir uma referência virtual que se move com uma velocidade imposta externamente sobre um caminho de referência. O controlador deve, portanto, minimizar o erro do veículo relativamente à referência, ao mesmo tempo que a tenta acompanhar em velocidade.

Este problema é formulado num referencial de Frenet-Serret que acompanha o movimento da referência virtual sobre uma curva, que, por sua vez, se encontra parametrizada relativamente ao seu comprimento de arco. Este trabalho contém uma derivação detalhada do modelo de qualquer veículo no referencial de Frenet-Serret a partir do seu modelo no referencial inercial. O modelo do veículo no referencial de Frenet-Serret é depois usado com MPC para controlar o veículo. São discutidas formas de evitar obstáculos e de garantir as restrições associadas aos limites laterais do caminho, bem como um método que varia a importância da orientação do veículo de acordo com o seu erro numa direção normal ao trajeto a seguir. Este trabalho apresenta, também, um método para controlar a velocidade da referência virtual baseado no erro da distância entre o veículo e a referência, o que permite ao veículo estar sempre próximo da referência ao mesmo tempo que tenta acompanhar a mesma.

Por fim, são apresentados e discutidos diversos resultados de experiências realizadas utilizando o modelo cinemático e dinâmico do uniciclo no referencial de Frenet-Serret.

Palavras-chave: Controlo Preditivo, Seguimento de Caminho, Veículo Autónomo de Corrida, Referência Virtual Móvel, Referencial de Frenet-Serret, Controlo da Velocidade da Referência

Abstract

This work addresses the design of a controller using Model Predictive Control (MPC) to drive an autonomous racing vehicle. The objective is to control a vehicle to follow a virtual reference that is moving with a certain velocity over a reference path. The controller must be capable of minimizing the error between the vehicle and the reference, while moving with the same velocity as the virtual target. The velocity of the reference is imposed externally, and thus corresponds to an extra control variable.

The problem is formulated in a Frenet-Serret reference frame that moves with the virtual target over the path, which, in turn, is parametrized according to its arc-length. This work also provides a detailed explanation on how to perform a conversion of an inertial vehicle model to its Frenet-Serret frame equivalent.

The Frenet-Serret vehicle model is then used with MPC to control the vehicle. The implementation of obstacle avoidance and path limits is discussed under the current MPC formulation. This work also proposes a method to control the speed of the virtual target based on the distance between the vehicle and the virtual target. Another method is also presented to dynamically change the focus the controller puts on the orientation of the vehicle based on its normal error to the path.

Finally, several results of different experiments performed using the unicycle kinematic and dynamic models in the Frenet-Serret frame are presented and discussed.

Keywords: Model Predictive Control, Path Following, Autonomous Racing Vehicle, Moving Virtual Reference, Frenet-Serret Frame, Virtual Reference Speed Control

Contents

List of Tables

List of Figures

Glossary

MPC	Model Predictive Control
NMPC	Nonlinear Model Predictive Control
ODE	Ordinary Differential Equation
RK4	Runge-Kutta 4^{th} Order Method
SQP	Sequential Quadratic Programming

Chapter 1

Introduction

Although with its roots in 1960s, Model Predictive Control (MPC) has been around since the 1980s, when it was first used to control chemical processes. The main advantage MPC with respect to the other forms of control is its ability to incorporate, in an explicit and computationally effective way, constraints on all the process variables. The control decision is obtained by solving an online finite horizon optimization problem, and thus the controller can take preventive actions regarding future events.

The predictive capabilities of MPC make it a very attractive option to be deployed as the controller of autonomous vehicles. That being said, the main disadvantage of MPC is the usually high associated computational loads, that require big amounts of computational power onboard. However, in the past decade, breakthroughs in battery technology and the development of more powerful and energy efficient computation units, as well as more efficient numerical algorithms, made it possible for MPC to be deployed in a practical manner onboard vehicles. For example, in 2012, the Ford Motor Company successfully studied the use of MPC in production vehicles to control different parts of the vehicle [1]. In this study, MPC was used, among other applications, to obtain better fuel efficiency when the motor is idling, and was also used to achieve front steering and differential breaking of the vehicle with positive results. More recently, MPC was successfully used in a Scania construction truck to achieve full control of the vehicle and follow a narrow path [2] [3].

The use of MPC in autonomous vehicles is by now a common practice and the research community is still very active around this topic as further levels of vehicle autonomy are being sough after. However, autonomous race cars, and thus the use of MPC in these vehicles, is still very new. To the author's knowledge, the first purpose-built autonomous race car, the *Robocar*, was only first shown to the public in 2017 [4].

At the time of writing, there is still no official competition for this type of vehicles, although it is the aim of *Roborace* to be that competition [5] [6]. *Roborace* uses the same race format of FIA Formula E Championship, with the teams using same vehicle chassis and powertrain, and having to develop the real-time algorithms [7]. There have already been several test events, including car on car racing [8], over the past seasons of Formula E and, more recently, the test vehicle, the *Robocar*, set the world record for the fastest autonomous race car at 282.42 km/h [9].

1.1 Motivation

The motivation for this work is provided by the aim of providing the first autonomous race car prototype, code-named FST10d, of the Formula Student team of Instituto Superior Técnico de Lisboa, named FST Lisboa, with a guidance and control system based on MPC.

Formula Student is an international student engineering competition where teams from different universities all over the world compete with each other in different challenges with a race car prototype designed, manufactured, and assembled by each team. Each year, several competitions are held across the globe and, in 2017, following the recent trends in technology, an autonomous driving category was added to the existing combustion and electric car categories.

Nevertheless, there is still no car on car racing for safety reasons and the autonomous driving challenges evaluate the performance of the cars via time scores (which are then converted to points) in different tracks, some of which are unknown *a priori*, so each car must complete each challenge as fast as possible.

The speeds of Formula Students cars are much slower then the well known cars of Formula 1 or 2, or even Formula E, and the cars are also much smaller and lighter, therefore, they present a good test-bench to evaluate the capabilities of using MPC to control a Formula Student autonomous car. The study of using MPC in a race car environment, in addition to the recent rise in interest of autonomous driving vehicles is the motivation of this work.

1.2 Contextualization and Problem Formulation

Nowadays, autonomous vehicles are very complex systems that involve many modules working together. A simplified high level view of the general architecture of an autonomous car includes a sensing and mapping module, a motion planning module, a vehicle control module, and an actuator control module [10]. The perception module gathers and processes sensor data to supply the other modules with the state of the car. With this information, the motion planning module generates the desired trajectory for the vehicle, which in turn is given to the vehicle control module to follow. With the state of the car and a reference to follow, the controller generates control decisions to be applied to the vehicle via the actuator control module.

Given a race track, the task of the vehicle is to minimize the time it takes to lap the track. The best times are going to be achieved by the vehicles that take the trajectory that maximizes the linear velocity of the vehicle while minimizing the length of track traveled. Although MPC can handle both optimal path generation and vehicle control, this work only focuses on the latter.

Given a path to follow, the idea is to use MPC to drive a vehicle towards a virtual reference that moves along the path. The velocity of the virtual target is imposed externally, and thus it presents an extra control input that can be explored to maximize the velocity of the vehicle while maintaining the vehicle on the path. Attached, and moving with the virtual reference, there is a Frenet-Serret frame in which a new model for the vehicle is defined and used with MPC. This new model under the Frenet-

Serret formulation provides a simpler and intuitive MPC formulation that reduces the tracking problem to a regulation problem and allows other functionalities to be added, such as obstacle avoidance and track limits.

1.3 State-of-the-Art

The focus of this work is on controlling an autonomous vehicle with MPC to follow a given path with a maximum velocity. Although the literature on path-following with autonomous vehicles is by now very extensive, formulating the problem in a Frenet-Serret frame that moves along the path has seen less attention.

To the author's knowledge, Samson [11] was the first to formulate a path-following problem in a Frenet-Serret frame. In his work, a path parametrization with respect to the arc-length is introduced, which allows the vehicle model to be defined relatively to a Frenet-Serret frame whose origin corresponds to the orthogonal projection of the vehicle in the path. This work was later complemented and extended in Micaelli and Samson [12] to two-steering-wheels robots. However, as the authors point out in their work, by projecting the vehicle in the path, a singularity is created in the vehicle model.

In Soetanto et al. [13], which serves as the foundation for this work, the authors propose considering a virtual target that moves independently along the path, which, in turn, detaches the origin of the Frenet-Serret frame from the vehicle. This step allows the velocity of the virtual target to be controlled independently from the vehicle and removes the singularity created in [12]. The controller design relies on Lyapunov function methods.

Following the same problem formulation of [13], in Lima [3] the author presents a Model Predictive Control formulation which prioritizes maximizing the progress of the reference over the path. Using the dynamic bicycle model, the velocity of the reference is maximized with MPC based on the velocity of the vehicle, its orientation and normal errors relatively to the reference on path. The MPC formulation presented also incorporates other constraints such as keeping the vehicle inside the road boundaries and enforcing handling limits.

1.4 Objectives

The goal of this work is to develop a MPC controller capable of controlling an autonomous racing vehicle under the Frenet-Serret problem formulation. This controller must be capable of following a virtual moving reference by matching its speed, while it minimizes the error to relative to the reference. The controller must also be capable of basic obstacle avoidance. It is also the objective of this work to explore methods that can increase the performance of the controller, such as exploring the extra degree of freedom provided by the control of the speed of the virtual reference.

1.5 Contributions

This work presents a detailed step-by-step explanation on how a vehicle model can be obtained in the Frenet-Serret frame from its inertial model.

Expanding on the work of others, this work uses MPC with non-linear vehicle models defined in the Frenet-Serret frame. The main contribution is a virtual target speed controller that slows down or speeds up the reference based on how far behind or in front the vehicle is relatively to the virtual target.

Another contribution of this work is a method to dynamically adjust the weight given to the orientation error in the MPC cost function based on the normal error between the vehicle and the path.

This work also features a simple and quick method to generate basic paths under the Frenet-Serret formulation, so that those unfamiliar with this topic can effortlessly start to test the controller.

With the framework described, a number of problems are studied, including obstacle avoidance and the cooperation of target and vehicle in maneuvers in which the last starts away from the lane.

1.6 Thesis Outline

Chapter 2 starts by introducing the path parametrization and the Frenet-Serret frame, that constitute the foundation of this work. A detailed deduction is then made to obtain the generic vehicle model in the Frenet-Serret frame, that can be used to convert any inertial vehicle model into the Frenet-Serret frame model. The unicycle kinematic and dynamic models are then deduced in the Frenet-Serret frame.

Chapter 3 starts with a brief introduction to Model Predictive Control theory, which is then followed by a discussion on how state constraints can be used to implement functionalities such as obstacle avoidance and path limits. An exponential function is then proposed as method to control the speed of the virtual target and a Gaussian function is presented as a solution to gradually place importance on the orientation error. A comparison with results is also made between the Euler and 4^{th} order Runge-Kutta integration methods, used to solve the vehicle model equations, to demonstrate the impact that each method has on the vehicle trajectory. In the end of this chapter, several code optimization that revealed to be useful during this work are also presented, as well as the search algorithm developed to retrieve path information with the arc-lenght is briefly explained.

Chapter 4 shows the results of several experiments using, first, the unicycle kinematic model, and then, the unicycle dynamic model. The experiments start of with a barebones MPC controller, to which the functionalities explained in Chapter 3 keep getting added and tested.

Chapter 2

Path-Following

Path-following covers the topic of controlling a vehicle to converge and follow a path where the reference is not subject to any temporal constraints. This formulation usually leads to less demanding control signals and smoother vehicle trajectories.

In this chapter the theory behind the problem formulation is explained. Section 2.1 presents several definitions while introducing the Frenet-Serret frame and the path parametrization. Afterwards, Section 2.2 provides a detailed explanation on how a general vehicle model can be deduced in the Frenet-Serret frame from its inertial model. Finally, the kinematic and dynamic models of the unicycle are deduced in the Frenet-Serret frame using the generic vehicle model.

2.1 The Frenet-Serret Frame and Path Parametrization

In 2D space, let $\{I\}$ designate an inertial reference frame, where the position of an arbitrary point P is given by vector $\boldsymbol{p}$. Furthermore, consider P as being part of an arbitrary smooth curve ζ defined in $\{I\}$, as shown in Figure 2.1. At point P, let $\{F\}$ designate a Frenet-Serret reference frame, which has one axis tangent and another normal to the curve. Additionally, let θ_c designate the angle that the positive tangent axis of $\{F\}$ makes with the horizontal axis of $\{I\}$.

Define now point Q as the center of mass of a vehicle that tracks point P on the curve. Since the position error between Q and P, designated by vector $\boldsymbol{r}$, also corresponds to the position of the vehicle in $\{F\}$, rotations aside, point Q can either be defined in $\{I\}$ by the vector

$$\boldsymbol{q} = \begin{bmatrix} X & Y & 0 \end{bmatrix}^T, \tag{2.1}$$

or in $\{F\}$ by the vector

$$\boldsymbol{r} = \begin{bmatrix} s_1 & y_1 & 0 \end{bmatrix}^T. \tag{2.2}$$

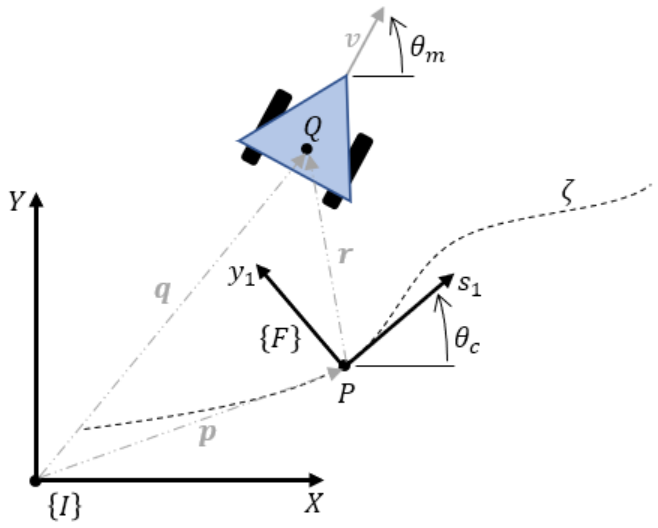


Figure 2.1: Vehicle geometry and frame definitions.

The position of Q in $\{I\}$ and $\{F\}$ can be related by

$$\boldsymbol{q} = \boldsymbol{p} + \boldsymbol{R}^{-1}\,\boldsymbol{r}, \tag{2.3}$$

where $\boldsymbol{R} = \boldsymbol{R}(\theta_c)$ denotes the axes rotation matrix from $\{I\}$ to $\{F\}$ and is equal to

$$\boldsymbol{R}(\theta_c) = \begin{bmatrix} cos\,\theta_c & sin\,\theta_c & 0 \\ -sin\,\theta_c & cos\,\theta_c & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.4}$$

Defining θ_m as the orientation of the vehicle in $\{I\}$, the orientation of the vehicle in $\{F\}$ is given by

$$\theta = \theta_m - \theta_c. \tag{2.5}$$

Since P is the target position of the vehicle, by making P move along the curve, or path, the vehicle is forced to follow P, and thus P acts like a virtual target or, in other terms, a moving reference, for the vehicle. By considering the tangent axis of $\{F\}$ positive in the direction of movement of P, by defining the vehicle model in $\{F\}$, one is effectively defining the error model of the problem. This formulation provides a natural approach to the problem and introduces simplicity, since the reference position and orientation is always zero for both.

The fact that P only moves along the path motivates the introduction of a path parametrization with respect to its arc-length s. The arc-length s between two points is defined by the length of the section of the path that unites those two points. By considering a starting point where $s = 0$, any point of the path can be identified by its arc-length with respect to the initial point, and thus each point has an unique s.

The velocity in $\{F\}$ at which P moves along the curve has no orthogonal components to the tangent

direction, and can thus be expressed as

$$\left(\frac{d\boldsymbol{p}}{dt}\right)_F = \begin{bmatrix} \dot{s} & 0 & 0 \end{bmatrix}^T, \tag{2.6}$$

where $\dot{s}$ denotes the tangent velocity of P along the path and encodes the progression of the virtual target along the path. The variable $\dot{s}$ is imposed externally, and, therefore, it is actually an extra controller design parameter.

By taking the time derivative of $\boldsymbol{q}$ and $\boldsymbol{r}$, the velocity of Q in $\{I\}$ is given by

$$\left(\frac{d\boldsymbol{q}}{dt}\right)_I = \begin{bmatrix} \dot{X} & \dot{Y} & 0 \end{bmatrix}^T, \tag{2.7}$$

or, in $\{F\}$, by

$$\left(\frac{d\boldsymbol{r}}{dt}\right)_F = \begin{bmatrix} \dot{s_1} & \dot{y_1} & 0 \end{bmatrix}^T. \tag{2.8}$$

It is remarked that, in some of the equations present in this section, the vectors are expressed in 3D space, even though this work only covers movement on a single plane. As detailed in the following sections, this representation is needed because some mathematical operations, like the cross-product, require the vectors to be defined in $\mathbb{R}^3$, and, therefore, the position and velocity vectors have their last entry equal to zero, whereas the angular velocity vectors only have a nonzero component in the last entry.

2.1.1 Relating θ_c with the Path Profile

Considering the case where the path is not just a straight line, θ_c varies as P moves along the path. It is, therefore, convenient to find an expression that relates the variation of θ_c with the path profile.

In 2D space, the Frenet-Serret equations [14] are given by

$$\frac{d\boldsymbol{T}}{ds} = \kappa\, \boldsymbol{N} \tag{2.9}$$

$$\frac{d\boldsymbol{N}}{ds} = -\kappa\, \boldsymbol{T} \tag{2.10}$$

where $\kappa = \kappa(s)$ denotes the signed curvature at point $P = P(s)$, and $\boldsymbol{N} = \boldsymbol{N}(s)$ and $\boldsymbol{T} = \boldsymbol{T}(s)$ are the normal and tangent vectors to the path at point P, respectively, that compose the axis of $\{F\}$.

Taking the derivative of $\boldsymbol{T}$ with respect to time and applying the chain rule for derivatives,

$$\dot{\boldsymbol{T}} = \frac{d\boldsymbol{T}}{ds}\frac{ds}{dt} \tag{2.11}$$

and using (2.9), yields

$$\dot{\boldsymbol{T}} = \kappa\, \boldsymbol{N}\, \dot{s}. \tag{2.12}$$

From Figure 2.2, and taking into account that $\boldsymbol{T}$ is unitary, this vector can be expressed in terms of

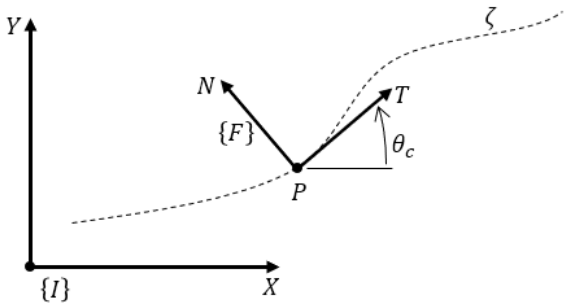


Figure 2.2: Representation of $\boldsymbol{T}$ and $\boldsymbol{N}$ in $\{I\}$.

θ_c as

$$\boldsymbol{T} = \begin{bmatrix} cos\,\theta_c \\ sin\,\theta_c \end{bmatrix} \tag{2.13}$$

Again, using the chain rule for derivatives, the time derivative of the equation above is given by

$$\dot{\boldsymbol{T}} = \begin{bmatrix} -sin\,\theta_c \\ cos\,\theta_c \end{bmatrix} \dot{\theta}_c \tag{2.14}$$

Since the normal vector $\boldsymbol{N}$ can be obtained from $\boldsymbol{T}$ by

$$\boldsymbol{N} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \boldsymbol{T}, \tag{2.15}$$

and replacing (2.13) in the equation above yields

$$\boldsymbol{N} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} cos\,\theta_c \\ sin\,\theta_c \end{bmatrix} = \begin{bmatrix} -sin\,\theta_c \\ cos\,\theta_c \end{bmatrix} \tag{2.16}$$

Substituting $\boldsymbol{N}$ in equation (2.12) yields the following expression

$$\dot{\boldsymbol{T}} = \kappa \begin{bmatrix} -sin\,\theta_c \\ cos\,\theta_c \end{bmatrix} \dot{s}, \tag{2.17}$$

but, since $\dot{\boldsymbol{T}}$ can also be expressed as (2.14), then the equation above can rewritten as

$$\begin{bmatrix} -sin\,\theta_c \\ cos\,\theta_c \end{bmatrix} \dot{\theta}_c = \kappa \begin{bmatrix} -sin\,\theta_c \\ cos\,\theta_c \end{bmatrix} \dot{s} \tag{2.18}$$

It is thus concluded that the model for θ_c is given by

$$\dot{\theta}_c = \kappa\,\dot{s} = \omega_c, \tag{2.19}$$

where ω_c is the angular speed of $\{F\}$.

The above result is inline with what is to be expected, since the rate of change of the orientation of θ_c increases in absolute value as the velocity of point P increases, *i.e.*, as the virtual target goes through the curve faster, or with bigger curvatures values, or both, and vice-versa.

2.2 Generic Model of a Vehicle in the Frenet-Serret Frame

As the reference P moves along the path, the Frenet-Serret frame moves and rotates relatively to the inertial frame. Since most kinematic and dynamic models of common vehicles are defined in the inertial frame, these model have to be converted to be used in $\{F\}$.

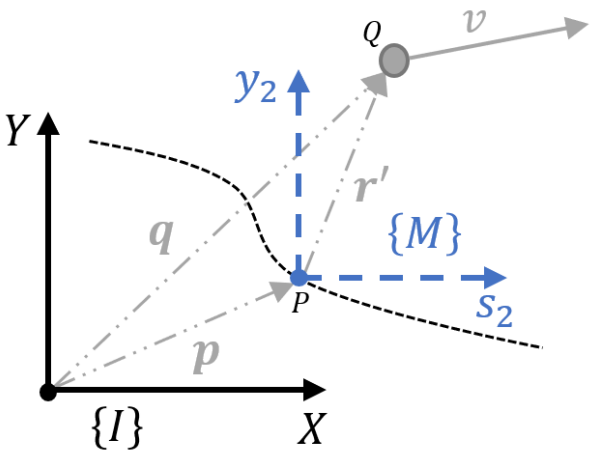


Figure 2.3: Velocity vector in a moving frame.

The process to obtain the new model in $\{F\}$ begins by establishing a velocity relationship between the two reference frames $\{I\}$ and $\{F\}$. Consider Figure 2.3 where $\{M\}$ is introduced as an intermediate reference frame that moves with P but does not rotate relatively to $\{I\}$, therefore its axis are always aligned with those of $\{I\}$. Also in the same figure, $\boldsymbol{r'}$ is the vehicle position vector in $\{M\}$. It is concluded that the velocity of the vehicle in $\{I\}$ is given by

$$\left(\frac{d\boldsymbol{q}}{dt}\right)_I = \left(\frac{d\boldsymbol{p}}{dt}\right)_I + \left(\frac{d\boldsymbol{r'}}{dt}\right)_M, \tag{2.20}$$

where $\left(\frac{d\boldsymbol{r'}}{dt}\right)_M$ denotes the velocity of the vehicle in $\{M\}$.

Consider now Figure 2.4 where $\{M\}$ and $\{F\}$ share the same origin, P. The reference frame $\{F\}$ rotates but does not move relatively to $\{M\}$. Viewed from $\{M\}$, the velocity of the vehicle has two components: its velocity in $\{F\}$ rotated to $\{M\}$, and another derived from the fact that $\{F\}$ is rotating, again rotated to $\{M\}$. Therefore, the velocity of the vehicle in $\{M\}$ can be expressed as

$$\left(\frac{d\boldsymbol{r'}}{dt}\right)_M = \boldsymbol{R}^{-1}\left(\frac{d\boldsymbol{r}}{dt}\right)_F + \boldsymbol{R}^{-1}\left(\boldsymbol{\Omega}\times\boldsymbol{r}\right), \tag{2.21}$$

where $\boldsymbol{\Omega} = \begin{bmatrix}0 & 0 & \omega_c\end{bmatrix}^T$ is the angular speed vector of $\{F\}$, since it only rotates around the axis

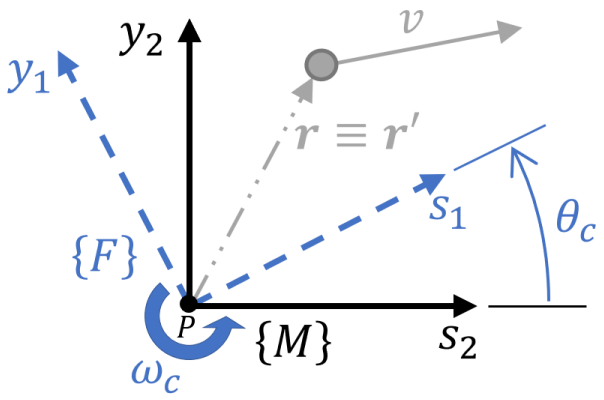


Figure 2.4: Velocity vector in a rotating frame.

perpendicular to the plane of the path, and $\boldsymbol{R} = \boldsymbol{R}(\theta_c)$ is the axis rotation matrix from $\{I\}$ (or $\{M\}$) to $\{F\}$.

Finally, substituting (2.21) in (2.20),

$$\left(\frac{d\boldsymbol{q}}{dt}\right)_I = \left(\frac{d\boldsymbol{p}}{dt}\right)_I + \boldsymbol{R}^{-1}\left(\frac{d\boldsymbol{r}}{dt}\right)_F + \boldsymbol{R}^{-1}\left(\boldsymbol{\Omega} \times \boldsymbol{r}\right), \tag{2.22}$$

and multiplying on the left by $\boldsymbol{R}$ yields

$$\boldsymbol{R}\left(\frac{d\boldsymbol{q}}{dt}\right)_I = \boldsymbol{R}\left(\frac{d\boldsymbol{p}}{dt}\right)_I + \left(\frac{d\boldsymbol{r}}{dt}\right)_F + \boldsymbol{\Omega} \times \boldsymbol{r}. \tag{2.23}$$

But, since

$$\boldsymbol{R}\left(\frac{d\boldsymbol{q}}{dt}\right)_I = \left(\frac{d\boldsymbol{q}}{dt}\right)_F \tag{2.24}$$

and

$$\boldsymbol{R}\left(\frac{d\boldsymbol{p}}{dt}\right)_I = \left(\frac{d\boldsymbol{p}}{dt}\right)_F, \tag{2.25}$$

the velocity of Q can be expressed in $\{F\}$ as

$$\boldsymbol{R}\left(\frac{d\boldsymbol{q}}{dt}\right)_I = \left(\frac{d\boldsymbol{p}}{dt}\right)_F + \left(\frac{d\boldsymbol{r}}{dt}\right)_F + \boldsymbol{\Omega} \times \boldsymbol{r}. \tag{2.26}$$

It is remarked that the leftmost side of the equation above was not replaced since $\boldsymbol{R}$ and $\left(\frac{d\boldsymbol{q}}{dt}\right)_I$ are terms that are known.

Replacing (2.6), (2.7), and (2.8) and using

$$\boldsymbol{\Omega} \times \boldsymbol{r} = \begin{bmatrix} 0 \\ 0 \\ \omega_c = \kappa\,\dot{s} \end{bmatrix} \times \begin{bmatrix} s_1 \\ y_1 \\ 0 \end{bmatrix} = \begin{bmatrix} -\kappa\,\dot{s}\,y_1 \\ \kappa\,\dot{s}\,s_1 \\ 0 \end{bmatrix} \tag{2.27}$$

in (2.26) yields

$$\boldsymbol{R}\begin{bmatrix}\dot{X}\\ \dot{Y}\\ 0\end{bmatrix} = \begin{bmatrix}\dot{s}(1-\kappa\, y_1)+\dot{s}_1\\ \dot{y}_1+\kappa\,\dot{s}\,s_1\\ 0\end{bmatrix}. \tag{2.28}$$

Solving for s_1 and y_1, the above equation can be rewritten as

$$\begin{cases}\dot{s}_1 = \begin{bmatrix} cos\,\theta_c & sin\,\theta_c\end{bmatrix}\begin{bmatrix}\dot{X}\\ \dot{Y}\end{bmatrix} - \dot{s}\,(1-\kappa\, y_1)\\ \dot{y}_1 = \begin{bmatrix}-sin\,\theta_c & cos\,\theta_c\end{bmatrix}\begin{bmatrix}\dot{X}\\ \dot{Y}\end{bmatrix} - \dot{s}\,\kappa\, s_1\end{cases}. \tag{2.29}$$

The model (2.29) only addresses the position of the vehicle in $\{F\}$ and, consequently, must be augmented to include the model of the orientation of the vehicle in this reference frame. Since the vehicle orientation in $\{F\}$ is given by (2.5), the model is obtained by applying the derivative with respect to time to (2.5)

$$\dot{\theta} = \dot{\theta}_m - \dot{\theta}_c. \tag{2.30}$$

Replacing (2.19) in the previous equation allows the model above to be rewritten as

$$\dot{\theta} = \dot{\theta}_m - \kappa\,\dot{s}. \tag{2.31}$$

Finally, augmenting the model (2.29) with the new state variable θ yields

$$\begin{cases}\dot{s}_1 = \begin{bmatrix} cos\,\theta_c & sin\,\theta_c\end{bmatrix}\begin{bmatrix}\dot{X}\\ \dot{Y}\end{bmatrix} - \dot{s}\,(1-\kappa\, y_1)\\ \dot{y}_1 = \begin{bmatrix}-sin\,\theta_c & cos\,\theta_c\end{bmatrix}\begin{bmatrix}\dot{X}\\ \dot{Y}\end{bmatrix} - \dot{s}\,\kappa\, s_1\\ \dot{\theta} = \dot{\theta}_m - \kappa\,\dot{s}\end{cases}. \tag{2.32}$$

The equations in (2.32) represent the generic model of any vehicle in $\{F\}$ following a moving reference P along a path. This model can now be used to generate other kinematic or dynamic models in $\{F\}$, specific to the type of vehicle being used. To do so, the placeholders $\dot{X}$, $\dot{Y}$, and $\dot{\theta}_m$ must be replaced by the correspondent inertial model equations of the vehicle. The system of equations can also be augmented to include any other equations relevant to the model.

The relevance of (2.32) to the control problem stems from the fact that these equations provide a relative motion model that allows to transform the tracking control problem into a regulation control problem, in which the variables s_1, y_1 and θ, that measure the deviation of the vehicle with respect to its desired position and orientation, are to be driven to zero.

2.2.1 Kinematic Model for the Unicycle

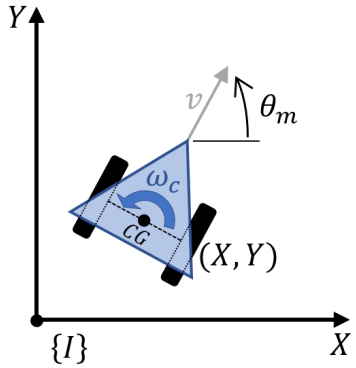


Figure 2.5: Unicycle parameters definition.

Consider that the motion of the vehicle traversing the path can be approximated by the kinematic model of the unicycle represented in Figure 2.5. This model is described by

$$\begin{cases} \dot{X} = v\,cos\,\theta_m \\ \dot{Y} = v\,sin\,\theta_m \\ \dot{\theta}_m = \omega_m \end{cases}, \tag{2.33}$$

where the state variables X and Y denote the position of the vehicle and θ_m its orientation. The inputs of the system are v and ω_m, that denote the linear and angular velocities of the vehicle, respectively.

Replacing the unicycle model (2.33) in the generic kinematic model for a vehicle in $\{F\}$ (2.32) yields

$$\begin{cases} \dot{s}_1 = v \begin{bmatrix} cos\,\theta_c & sin\,\theta_c \end{bmatrix} \begin{bmatrix} cos\,\theta_m \\ sin\,\theta_m \end{bmatrix} - \dot{s}\,(1 - \kappa\,y_1) \\ \dot{y}_1 = v \begin{bmatrix} -sin\,\theta_c & cos\,\theta_c \end{bmatrix} \begin{bmatrix} cos\,\theta_m \\ sin\,\theta_m \end{bmatrix} - \dot{s}\,\kappa\,s_1 \\ \dot{\theta} = \omega_m - \kappa\,\dot{s} \end{cases}. \tag{2.34}$$

Finally, by simplifying the equations above

$$\begin{cases} \dot{s}_1 = v\,(cos\,\theta_c\,cos\,\theta_m + sin\,\theta_c\,sin\,\theta_m) - \dot{s}\,(1 - \kappa\,y_1) \\ \dot{y}_1 = v\,(cos\,\theta_c\,sin\,\theta_m - sin\,\theta_c\,cos\,\theta_m) - \dot{s}\,\kappa\,s_1 \\ \dot{\theta} = \omega_m - \kappa\,\dot{s} \end{cases} \tag{2.35}$$

and applying known trigonometric identities yields

$$\begin{cases} \dot{s}_1 = v\,cos(\theta_m - \theta_c) - \dot{s}\,(1 - \kappa\,y_1) \\ \dot{y}_1 = v\,sin(\theta_m - \theta_c) - \kappa\,\dot{s}\,s_1 \\ \dot{\theta} = \omega_m - \kappa\,\dot{s} \end{cases} . \tag{2.36}$$

The kinematic model for the unicycle in $\{F\}$ is given by

$$\begin{cases} \dot{s}_1 = -\dot{s}(1 - \kappa\,y_1) + v\,cos\,\theta \\ \dot{y}_1 = -\kappa\,\dot{s}\,s_1 + v\,sin\,\theta \\ \dot{\theta} = \omega_m - \kappa\,\dot{s} \end{cases} . \tag{2.37}$$

The set of equations (2.37) describe the kinematic motion of the unicycle in $\{F\}$ when following a path. The state variables are $\begin{bmatrix} s_1 & y_1 & \theta \end{bmatrix}^T$ and the control inputs are $\begin{bmatrix} v & \omega_m & \dot{s} \end{bmatrix}^T$.

2.2.2 Dynamic Model for the Unicycle

It is assumed that the unicycle depicted in Figure 2.5 has two parallel, nondeformable wheels, which are controlled by two independent motors (plus powertrain) that generate the control torques τ_1 and τ_2. Furthermore, the plane of each wheel is perpendicular to the ground and the contact between the wheels and the ground is assumed to be pure rolling and nonslipping, meaning that the velocity of the center of mass of the unicycle is normal to wheels axis. The masses and inertias of the wheels are considered to be negligible and the center of mass of the unicycle is located at the center point on the axis connecting the wheels.

In addition to (2.33), the dynamics model of the unicycle introduces two new equations

$$\begin{cases} \dot{v} = \dfrac{F}{m} \\ \dot{\omega}_m = \dfrac{N}{I} \end{cases} , \tag{2.38}$$

that correspond to two additional states v and ω_m, and F and N are given by

$$\begin{cases} F = \dfrac{\tau_1 + \tau_2}{R} \\ N = \dfrac{\tau_1 - \tau_2}{R}\,L \end{cases} , \tag{2.39}$$

where m is the mass of the unicycle, I its moment of inertia, R the radius of the wheels, and L is half the length of the axis that connects the centers of the two wheels, *i.e.*, the shortest length between the center of mass of the unicycle and the center of the each wheel. The new manupilated variables are now the torques applied to each wheel τ_1 and τ_2.

Augmenting the kinematic model of the unicycle in $\{F\}$ (2.37) with the equations above, the state

model in $\{F\}$ becomes

$$\begin{cases} \dot{s}_1 = -\dot{s}\,(1 - \kappa\, y_1) + v\, cos\,\theta \\ \dot{y}_1 = -\kappa\,\dot{s}\,s_1 + v\, sin\,\theta \\ \dot{\theta} = \omega_m - \kappa\,\dot{s} \\ \dot{v} = \dfrac{1}{m}\dfrac{\tau_1 + \tau_2}{R} \\ \dot{\omega}_m = \dfrac{L}{I}\dfrac{\tau_1 - \tau_2}{R} \end{cases} . \tag{2.40}$$

The state variables are now $\begin{bmatrix} s_1 & y_1 & \theta & v & \omega_m \end{bmatrix}^T$ and the control inputs are $\begin{bmatrix} \tau_1 & \tau_2 & \dot{s} \end{bmatrix}^T$

Chapter 3

Motion Control With Model Predictive Control

This chapter addresses the topics related to Model Predictive Control (MPC) applied to vehicle motion. This chapter starts in Section 3.1 by making a brief introduction to Model Predictive Control theory. Section 3.2 shows how to enforce path boundaries and obstacle avoidance with MPC and discusses its limitations. Section 3.3 proposes a method to profile the speed of the virtual target using an exponential function, and Section 3.4 proposes a method to prioritize the convergence with the path by dynamically adjusting the MPC cost function.

Later on in the chapter, more implementation specific topics are discussed. Section 3.6 discusses how the paths are defined and presents a quick method to generate test paths. Afterwards, Section 3.7 discusses the impact that the integration method used has on the trajectory of the vehicle. Finally, Section 3.8 lists some code optimizations and presents a search algorithm that uses the arc-length to retrieve path information stored in a table.

3.1 Model Predictive Control

Model Predictive Control (MPC) is a control method that makes explicit use of a model of the system to predict its future outputs and states over a finite horizon, and, based upon that knowledge, solve an optimization problem to obtain a control decision.

Figure 3.1 illustrates the typical MPC control strategy. At each sampling instant, the optimization problem is solved by minimizing a cost function within a finite horizon, also called the prediction horizon, while considering the latest available state measurement or estimate as the initial condition. The solution of the problem is a sequence of control actions for future sampling instants within the prediction horizon, although only the first of this sequence is actually executed. Subsequently, a new system state is available and the problem is repeated with the prediction horizon displaced towards the next sampling instant. The shifting forward of the prediction horizon is also called receding or moving horizon strategy.

As a result of explicitly utilizing the model of the system to solve the optimization problem, MPC can

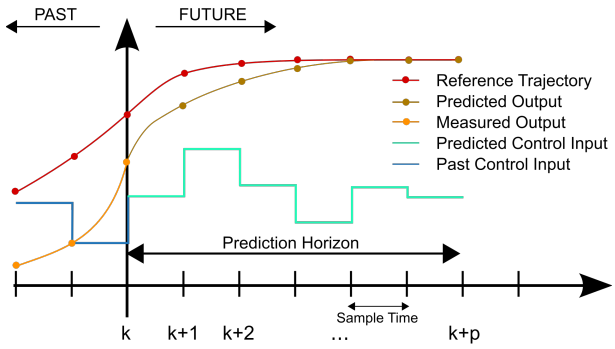


Figure 3.1: Model Predictive Control strategy. Available at [15].

handle tightly coupled multivariate systems, even if those systems are non-linear (Non-Linear MPC or NMPC). The formulation as an optimization problem also allows the MPC to directly impose constraints on both the state and control variables, this feature being a major advantage over other control design methods.

The major drawback of MPC has to do with having to solve an online open-loop iterative optimization problem every sampling step, and for this reason MPC has been associated with big computational loads. Because each optimization problem must to be solved before the next sampling step, MPC applications have favored in the past systems with slow dynamics, but recent breakthroughs in both computer hardware and software, as well as theoretical advances, have brought computation times down significantly, allowing MPC to be deployed in applications with increasingly faster systems. At least as prototypes, there exist application examples to engine control, such as the VW Polo, aircraft motion control and robot motion control.

3.1.1 Model Predictive Control Formulation

Consider a system where the state variables, control inputs, and output variables are represented by $\boldsymbol{x}$, $\boldsymbol{u}$, and $\boldsymbol{y}$, respectively. The system can be described by its state-space representation as follows

$$\frac{d\boldsymbol{x}}{dt} = f(\boldsymbol{x}, \boldsymbol{u}) \tag{3.1a}$$

$$\boldsymbol{y} = g(\boldsymbol{x}, \boldsymbol{u}), \tag{3.1b}$$

where $f(.)$ denotes the state dynamics of the system, which can be linear or not , and $g(.)$ its sensor or observation model.

For discrete-time invariant systems, which are the case of the systems used in this work, the Model Predict Control optimization problem can be formulated as follows

$$\underset{\overline{\boldsymbol{x}},\,\overline{\boldsymbol{u}}}{\text{minimize}} \quad J(\overline{\boldsymbol{x}},\overline{\boldsymbol{u}}) = \sum_{i=1}^{N} \ell(\overline{\boldsymbol{x}}_i,\overline{\boldsymbol{u}}_i) \tag{3.2a}$$

$$\text{subject to} \quad \overline{\boldsymbol{x}}_0 = \boldsymbol{x}_t \ , \tag{3.2b}$$

$$\overline{\boldsymbol{x}}_i = f(\overline{\boldsymbol{x}}_{i-1},\overline{\boldsymbol{u}}_{i-1}), \tag{3.2c}$$

$$\overline{\boldsymbol{x}}_i \in \boldsymbol{\mathcal{X}}_i,\, i = 1,\ldots,N, \tag{3.2d}$$

$$\overline{\boldsymbol{u}}_i \in \boldsymbol{\mathcal{U}}_i,\ i = 1,\ldots,N-1, \tag{3.2e}$$

where $\overline{\boldsymbol{x}}$ and $\overline{\boldsymbol{u}}$ represent the predictions of $\boldsymbol{x}$ and $\boldsymbol{u}$ over the prediction horizon N, and $\ell(.)$ is the instantaneous cost function to be minimized. The optimization problem is subject to the constraints (3.2b) to (3.2e), where constraint (3.2b) imposes the initial condition on the state ($\boldsymbol{x}_t$ represents the state at time t), and constraint (3.2c) imposes the system model. The state is also subject to a set of constraints represented by $\boldsymbol{\mathcal{X}}$ in (3.2d), and, likewise, the control inputs are subject to the constraints represented by $\boldsymbol{\mathcal{U}}$ in (3.2e).

3.1.2 Quadratic Cost Function

The cost function selected for optimization problem dictates the control strategy used by the MPC. A common and effective approach is to use the quadratic cost function

$$\ell(\boldsymbol{x},\boldsymbol{u}) = \boldsymbol{x}^\top \boldsymbol{Q}\boldsymbol{x} + \boldsymbol{u}^\top \boldsymbol{R}\boldsymbol{u}, \tag{3.3}$$

where $\boldsymbol{Q}$ and $\boldsymbol{R}$ are positive definite weight matrices used to tune the controller. In this work, both $\boldsymbol{Q}$ and $\boldsymbol{R}$ are diagonal matrices

$$\boldsymbol{Q} = \begin{bmatrix} \rho_{x_1} & 0 & \ldots & 0 \\ 0 & \rho_{x_2} & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \rho_{x_m} \end{bmatrix} \quad \boldsymbol{R} = \begin{bmatrix} \rho_{u_1} & 0 & \ldots & 0 \\ 0 & \rho_{u_2} & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \rho_{u_n} \end{bmatrix}, \tag{3.4}$$

where each diagonal entry is the weight attributed to the corresponding state or input variable. Increasing/decreasing the value of a weight in $\boldsymbol{Q}$ relatively to the other weights increases/decreases the importance that the controller will place on that state. Furthermore, increasing a value of $\boldsymbol{R}$ relatively to $\boldsymbol{Q}$ penalizes the control action of the corresponding control variable.

Notice that the cost function (3.3) directly penalizes the non-zero states and inputs (if the respective weight is not zero), meaning that in this configuration the controller will try to control the system to the origin. This formulation is also known as the ”quadratic regulator”.

3.1.3 Reference Tracking

To control the system towards a reference state $\boldsymbol{x}_{ref}$ other than zero, the cost function (3.3) has to be modified to penalize the tracking error between the state $\boldsymbol{x}$ and the reference state $\boldsymbol{x}_{ref}$, that can be achieved by the cost function

$$\ell(\boldsymbol{x}, \boldsymbol{u}) = (\boldsymbol{x} - \boldsymbol{x}_{ref})^\top \boldsymbol{Q} (\boldsymbol{x} - \boldsymbol{x}_{ref}) + \boldsymbol{u}^\top \boldsymbol{R} \boldsymbol{u}, \tag{3.5}$$

where $\boldsymbol{x} - \boldsymbol{x}_{ref}$ represents the state tracking error. For systems with integral action, $\boldsymbol{u}$ tends to zero as the system approaches $\boldsymbol{x}_{ref}$, and thus the reference tracking is achieved without static errors. However, for systems without integral action, if $\boldsymbol{x}_{ref}$ is not the origin, then $\boldsymbol{u}$ must be different than zero in steady state, and, therefore, will weight the cost function (3.5), causing the reference to be tracked with a static error (because the optimal solution is no longer the one that makes $\boldsymbol{x} = \boldsymbol{x}_{ref}$). To achieve error-free reference tracking in systems without integral action, a feed-forward term must be included in (3.5) that allows $\boldsymbol{u}$ to be non-zero in steady state. The resulting cost function is

$$\ell(\boldsymbol{x}, \boldsymbol{u}) = (\boldsymbol{x} - \boldsymbol{x}_{ref})^\top \boldsymbol{Q} (\boldsymbol{x} - \boldsymbol{x}_{ref}) + (\boldsymbol{u} - \boldsymbol{u}_{ref})^\top \boldsymbol{R} (\boldsymbol{u} - \boldsymbol{u}_{ref}), \tag{3.6}$$

where $\boldsymbol{u}_{ref}$ represents the control reference.

For more on Model Predictive Control theory see Rawlings et al. [16].

3.1.4 Implementing MPC with MATLAB

In MATLAB, the cost function of the optimization problem is solved using the *fmincon* function. This function finds a minimum of constrained nonlinear multivariable function, and allows the linear and non-linears constraints to be implemented. The algorithm used is the Sequential Quadratic Programming algorithm (SQP), since it is the one that provides the best performance in terms of computational speed for this problem. All other *fmincon* options are set to default.

3.2 State Constraints

Model Predictive Control provides the ability to set constraints on the the state and control variables. In the context of motorsport, but also regarding autonomous vehicles in general, this ability can be explored to enhance the controller with extra functionalities, such as path boundaries awareness nd obstacle avoidance.

3.2.1 Path Boundaries

For every point of the path, a boundary is defined as the normal distance beyond which the vehicle is not allowed to go. For a race track there are two boundaries, one for each side of the path, therefore, there are two boundary conditions for every point of the path.

For the vehicle models defined in the Frenet-Serret frame, the normal distance between the vehicle and the reference point on the path is given by the state variable y_1, therefore, the path boundaries can be defined via inequality constraints on the state y_1 as

$$l_{lower} \leq y_1 \leq l_{upper} \ , \tag{3.7}$$

where l_{lower} and l_{upper} denote the lower and upper limits of the path, with $l_{lower} < l_{upper}$. It is to expect the lower limit to be a negative number and the upper a positive one, even though this is not a requirement.

Although the use of y_1 to define the path boundaries is the most intuitive and simple solution, it can also become a problem when the state variable s_1 is not close zero. The state y_1 represents the normal position of the vehicle relatively to the reference point, which, in turn, is not the closest point of the path to the vehicle if $s_1 \neq 0$. When $s_1 \neq 0$ and the virtual target is going through a curve ($\kappa \neq 0$), the situation depicted in Figure 3.2 occurs.

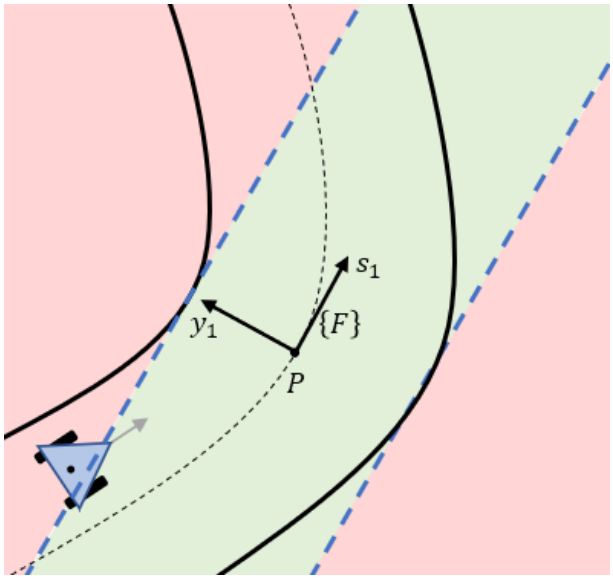


Figure 3.2: Path boundaries error.

Notice how, in Figure 3.2, the path boundaries represented by the two blue lines do not accurately portrait the limits of the path in the vicinity of the vehicle if $s_1 \neq 0$. Furthermore, from the controller point of view, some positions beyond the path limits become valid ones and, on the flip-side, some positions that are within the path boundaries are considered invalid. This situation also becomes worse as the radius of the curvature decreases.

The situation shown in Figure 3.2, which is exclusive to curved sections of the path, happens because the path boundaries are defined around the virtual target position and not the vehicle position. Only when $s_1 = 0$ does the virtual target become the closest point of the path to the vehicle, and thus the path boundaries become correctly defined around the vehicle.

One possible solution to this problem is to compute the path boundaries every iteration based on the position of the vehicle. This method, which is not explored in this work, adds complexity to the problem

because it involves finding the path limits around the closest point of the path to the vehicle, and then compute new boundary constraints in $\{F\}$. Although this solution produces the most accurate limits, the boundary constraints have to be computed online every iteration, and thus this method increases the computational load and degrades the temporal performance of the controller.

The proposed solution is to increase the focus of the controller on making $s_1 \approx 0$, because, as s_1 tends to zero, the errors in the limit constraints also decrease, as seen in Figure 3.2. So, when $s_1 \approx 0$, the boundary conditions defined by (3.7) approximately match the path limits when viewed from the vehicle position. This solution keeps the simplicity of (3.7), while also allowing the path boundaries to be computed offline for each point of the path. If not constant throughout the path, the path boundaries can be updated each iteration via table lookup, with no further computations needed, although that requires the limits to be implement via non-linear constraints for the same reasons explained in Section 3.6.2.

To increase the pressure on making $s_1 = 0$, the weight coefficient of s_1 in the cost function can be increased or the virtual target can be controller to stay close to the vehicle, like it is explained in Section 3.3. The weight increase may lead to an increase in the absolute error of y_1, but since the path boundaries provide tolerance for deviations in y_1 the increase in error does not present a problem, if the controller is configured properly.

3.2.2 Obstacle Avoidance

State constraints can also be used to implement obstacle avoidance with a MPC controller. Similarly to how path boundaries are implemented in Section 3.2.1, obstacle avoidance can be achieved by restricting the region of positions where the vehicle is allowed to be.

Knowing the coordinates of the center of the obstacle in $\{F\}$, an exclusion zone representing the obstacle and where the vehicle is not allowed to be can be defined by a circle centered on the obstacle and with a radius large enough to encompass the entire obstacle or the vehicle, whichever is bigger. Then, a nonlinear inequality constraint can be created using the equation of the circle

$$(s_1 - C_{s_1})^2 + (y_1 - C_{y_1})^2 \geq r^2, \tag{3.8}$$

where (C_{s_1}, C_{y_1}) represents the center of the obstacle in $\{F\}$ at certain time instant, and r is the radius of the exclusion zone. Note that r must be bigger than the biggest radius between the circles the represent the vehicle and the obstacle, to avoid a collision, geometrically speaking.

Note also that, as a result of the movement of the virtual target, $\{F\}$ also moves, which, in turn, causes the position of the obstacle to change between time instants. If existent, the movement of the obstacle itself also contributes to the change of position between time steps. Because MPC predicts future states when solving the optimization problem, the non-linear inequality constraint that defines the obstacle must also be updated each time step to include the new positions of the reference, vehicle and obstacle.

So far it was assumed that the position of the obstacle in $\{F\}$ is known at all times, which is not always true. Every sampling instant, when the MPC controller begins to solve a new optimization problem it only

has access to the latest position of the obstacle (and eventually other data), and thus it has no way to precisely know the future positions of the obstacle if this one is moving. The position can eventually be predicted with some degree of confidence knowing its velocity and direction, but, at that point, the circular exclusion zone can no longer guarantee an obstacle free trajectory for the vehicle, since the position of the obstacle is only a probability. For this and other reasons, obstacle avoidance with moving obstacles is a topic that requires more research, and thus is out of the scope of this work.

When the obstacle is not moving in $\{I\}$, then its position in $\{F\}$ can be computed. Assuming that the inertial coordinates of the path are known (refer to Section 3.6.1 for more details on this subject), then the coordinates of the virtual target in $\{I\}$ can be known using its arc-length position, and in the same way the curvature is retrieved. Given the coordinates of an obstacle in $\{I\}$, by changing the reference frame origin to the position virtual target and then rotating those coordinates by the orientation angle of $\{F\}$, the coordinates of the obstacle in $\{F\}$ are obtained. In mathematical terms, this conversion is given by

$$\begin{bmatrix} C_{s_1} \\ C_{y_1} \end{bmatrix} = \begin{bmatrix} cos\,\theta_c & -sin\,\theta_c \\ sin\,\theta_c & cos\,\theta_c \end{bmatrix} \begin{bmatrix} X_{obs} - X_{ref} \\ Y_{obs} - Y_{ref} \end{bmatrix} \tag{3.9}$$

where $[X_{obs} \;\; Y_{obs}]^\top$ and $[X_{ref} \;\; Y_{ref}]^\top$ are the obstacle and virtual target coordinates in $\{I\}$, respectively, and θ_c is the orientation angle of $\{F\}$.

Notice in equation (3.9) that the state θ_c is needed to compute the position of the obstacle in $\{F\}$, even though this state is not explicitly tracked in the vehicle models defined in $\{F\}$. An alternative to θ_c is to use the vehicle orientation θ_m to calculate θ_c via the equation $\theta = \theta_m - \theta_c$, where θ is already a state variable. The tangent angle θ_c can be computed offline for every point of the path (refer to Section 3.6.3 for more details on this topic), or can be computed with equation (2.19) every sampling step. To track either θ_c or θ_m, the variable must be added to the vehicle model at the cost of added computational load. The alternative is track one of these variables outside the controller and then to use a method similar to that explained in Section 3.6.2 to track that variable inside the MPC without having to add it as a state variable.

Due to the discrete nature of the controller, it can happen that the obstacle constraints are satisfied for each time step but vehicle actually collides with the obstacle between sampling instants. Figure 3.3 depicts one of those situations.

Notice in Figure 3.3 that, even though the vehicle is controller outside the exclusion zone, the vehicle violates the obstacle constraints between sampling instants. The problem gets worse with higher vehicle velocities or increased sampling periods, since the length of path traveled between time steps also increases, meaning that, in an extreme situation, the vehicle could go right through the obstacle and still satisfy all obstacle constraints.

One solution is to incorporate a margin of safety into the radius of the exclusion zone, such that, even if the vehicle violates the exclusion zone with the margin of safety included, it does not enter the physical area of the obstacle. The margin of safety must be dimensioned by taking into account the maximum velocity expected from the vehicle, as well as the sampling time, and can either be a fixed value or a variable one that changes with the speed of the vehicle (smaller velocity, smaller margin of

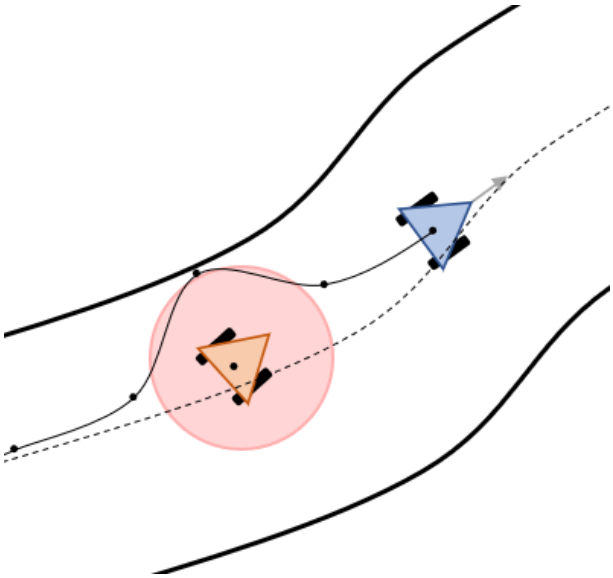


Figure 3.3: Example of an obstacle avoidance violation.

safety). The limitation of this solution lies in the possible total size of the exclusion zone that avoids a collision, because, if the path has boundaries, for example, there might not be enough space left in the path for overtakes.

The second solution is to decrease the sampling period of the controller, therefore, shortening the length of the distance traveled by the vehicle between time instants. This solution ties into the first one as it allows the margin of safety to be smaller. However decreasing the sampling period increases the computational requirements of the controller as more computations have to be performed for the same amount of time, therefore, it might not be always feasible. The best simple solution lies in a balance of these two methods, with a sampling period suitable to the available computational power and a margin of safety that fills in the remaining area needed to avoid a collision.

Other complementary solutions not explored involve using other geometric shapes for the exclusion zone of the obstacle, such as two overlapping circles instead of one, an ellipse with the major axis aligned with the direction of the movement of the vehicle, or a rectangle. These geometric shapes leave more lateral space for overtakes while maintaining the safety aspect.

Another, more complex, solution involves detecting if the line unites two consecutive vehicle positions intersects the exclusion zone. This solution requires the detection to be made in the inertial frame and also requires solving a systems of equations every time step for every obstacle, therefore, it is to expect a big increase in the computational load. For this reason, this solution is not explored in this work.

This section explains how basic obstacle avoidance can be accomplished with a MPC controller using a vehicle model in $\{F\}$. However, it also evidences the limitations and obstacles in achieving a feasible solution that guarantees a safe trajectory for the vehicle. Furthermore, because the controller is tuned to follow the path, any forced deviation from it results in the controller taking longer to solve the optimization problem, which results in a degradation of the temporal performance of the controller.

For all these reasons, it is recommended that the avoidance of obstacles be built into the path the controller is tasked to follow, since it allows the safety features to be built into the path and also allows for

a better tuning of the trajectory the vehicle is suppose to follow. For added safety, the obstacle can still be represented by constraints in the controller, but, because the path does not go through the obstacle, it leads to lower computations times overall.

3.3 Virtual Target Speed Control

The path parametrization explained in Section 2.1 introduced the ability to control the speed at which the virtual target moves along the path via the control variable $\dot{s}$.

Consider the situation where the vehicle gets off track and thus s_1 and y_1 are no longer zero. If $\dot{s}$ is constant, the controller will try to converge with the reference again by reducing y_1 and s_1, except that the reference is kept on moving and thus the position error is also kept on increasing. The situation becomes is even more complicated in curves because the vehicle might be forced to take a shortcut and not go anywhere near the path to catch up with the virtual target again.

In addition, the path boundaries presented in Section 3.2.1 require s_1 to be as small as possible for the path boundaries to be valid in curved sections of the path, therefore, there is an added incentive to make the vehicle move as close to the virtual target as possible.

To make s_1, y_1, and θ zero, either the vehicle has to be controlled to converge with the virtual target, or the latter can also be controlled via $\dot{s}$ to converge with the vehicle when this one is not able to keep up, and thus eliminating the situation where they get too far apart. One advantage of controlling the speed of the virtual target is that it is a virtual reference, *i.e.*, it has no dynamics, therefore it can move has fast as possible from one instant to the other, which is not true for the vehicle.

The virtual target must behave in such a manner that when the vehicle is close behind (or on top) it moves at a predefined reference speed, and, when the vehicle starts to lag behind, it starts slowing down (and stop if necessary) until the vehicle starts to catch up again, by which point it starts to speed up again. The reverse must also happen if the vehicle is to overtake the virtual target for some reason, *i.e.*, the virtual target must speed up to catch up with the vehicle. If the vehicle ends up in front of the virtual target it can cause the vehicle to slow down or even to reverse the direction of movement, which is highly undesirable when the objective is to move as fast as possible. The objective is to make the vehicle always be behind or on top of the virtual target, because the latter one sets the speed at which the vehicle will move through the path, effectively pulling the vehicle towards it.

The most immediate solution is to make the speed of the virtual target a function of the distance between the vehicle and the origin of $\{F\}$. However, the distance does not provide any information on whether the vehicle is in front or behind the reference.

In this work, it is proposed to use the state s_1 as variable that controls the speed of the virtual target. The variable s_1 provides some information regarding the distance between the vehicle and the virtual target, as well as if the vehicle is behind or in front of the virtual target. Information regarding the normal distance of the vehicle is lost when using s_1, but that does not present a problem because the controller will always work to minimize both s_1 and y_1 at the same time, if configured properly. Furthermore, as s_1 becomes smaller, the speed of the virtual target starts increasing as well, ensuring that the reference is

always a small distance in front of the vehicle to allow a smooth convergence with the path, after which they move as one at the predefined reference speed (which can be the same as the vehicle maximum achievable velocity).

It is also recommend that the virtual target speed follows a continuous function to prevent feasibility issues while optimizing the functional and unexpected behavior from the controller. The exponential function fulfills all the criteria mentioned up until this point, therefore, the virtual target speed can be given as a function of s_1 by

$$\dot{s}(s_1) = \dot{s}_{ref} \exp\left(\frac{s_1}{\lambda}\right), \tag{3.10}$$

where $\dot{s}_{ref}$ is the reference speed when $s_1 = 0$ and λ is a tuning parameter. Figure 3.4 shows the virtual target speed profile based on s_1, as well as the impact of the tuning parameter λ.

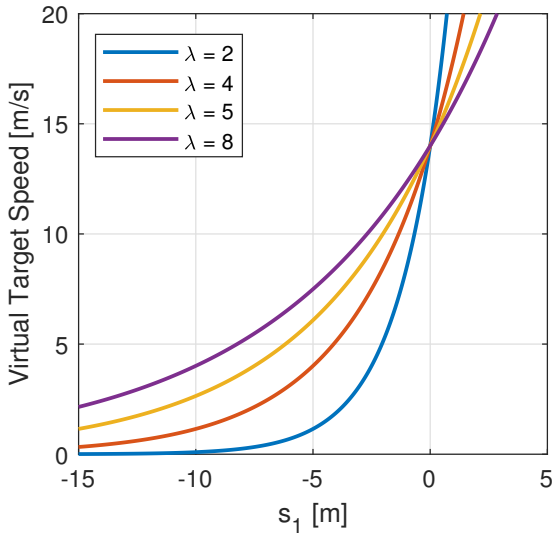


Figure 3.4: Virtual target speed profiles for a reference speed of 14 m/s.

Notice in Figure 3.4, how, for $\lambda = 2$, the virtual target stops when the vehicle is just 10 meters behind the reference. Increasing λ also increases this distance, but the increase in velocity is also less aggressive when compared to lower values of λ, so it is a matter of balancing these two properties.

3.3.1 Incorporating Virtual Target Speed Control with MPC

The recommended way and the one that produces the best results is to implement the speed controller directly with the function that generates the nonlinear constraints, such as the vehicle model, used by the MPC controller. Since this function is called every step of the iteration cycle, it has access to the state and control variables in every step of the prediction window, and thus it can update the velocity of the reference accordingly, resulting in more accurate predictions. This solution is effectively the same as replacing $\dot{s}$ in the vehicle models by equation (3.10).

The alternative is to only compute the velocity with the latest available state after each call to *fmincon*, *i.e.*, each sampling instant, and then pass that velocity as an extra parameter to the function that generates the nonlinear vehicle constraints. The advantage is that this method can also be used with

linearized vehicle models since the only extra step is to update the matrices that define the vehicle model linear constraints. The downside is that the velocity remains constant throughout the prediction window, which leads to inaccurate predictions beyond the first prediction step.

At an earlier stage of this work, before implementing the proposed exponential function, it was also tested using MPC to maximize the velocity of virtual target speed that made the state error zero. This method quickly proved unreliable and caused instability and feasibility issues with the controller.

The alternative is to make the velocity of the reference an optimization variable of the vehicle model, but with the exponential function as the reference to the virtual target speed variable in the cost function, like is shown in (3.6). This method produces very similar results to the recommended one, although at the cost of added computational load, since an extra optimization variable is added to the model.

3.4 Variable Orientation Error Weight

Consider again the situation where the vehicle gets off track, although this time the virtual target speed controller introduced in Section 3.3 is already in place. In this scenario, there are two different forces opposing each other: the controller wants to decrease the error in both y_1 and θ (and s_1), although the action to decrease one increases the error of the other. When turning the vehicle towards the path to decrease y_1, the controller is, at the same time, increasing the orientation error, θ, since the error θ requires the vehicle to be aligned with the path to be zero. The solution is eventually a balance of both actions that minimizes the cost function. This duality can make it harder for the controller to find a solution to the optimization problem, and thus increase the computation time.

A fixed weight for the orientation error θ means that the controller is always trying to correct the orientation of the vehicle, regardless of the vehicle position relatively to the path. Since the orientation error between the vehicle and the path does not matter as much as the normal distance when the vehicle is not on the track, by dynamically decreasing the weight put on θ based on increasing values of $|y_1|$, the controller is going to prioritize decreasing y_1. The same way, once y_1 gets within a certain range of the path, by increasing the weight put on θ, the controller is going to start adjusting the orientation of the vehicle again.

The proposed method effectively splits the problem into two. At an initial stage, the focus of the controller is to steer the vehicle back to the track without having to worry about the orientation error (since the weight of θ is very small compared to the weight of y_1). At a final stage when the vehicle is close to the path again and y_1 is small enough, gradually increase the focus put into the orientation error to align the vehicle with the path, and thus allow for a smooth approach. During the beginner stage the optimization problem gets relaxed because the controller does not have to worry about minimizing one state variable, which can help to decrease the computation time of the controller.

In this work it is proposed to implement the above described functionality by having the weight of θ follow the Gaussian function of y_1

$$\rho_\theta(y_1) = \alpha \, \exp\left(\left[\frac{y_1}{\beta}\right]^2\right), \tag{3.11}$$

where α and β are a tuning parameters. Figure 3.5 displays the variation of the weight of θ as a function of y_1, as well as the effect of α and β tuning parameters.

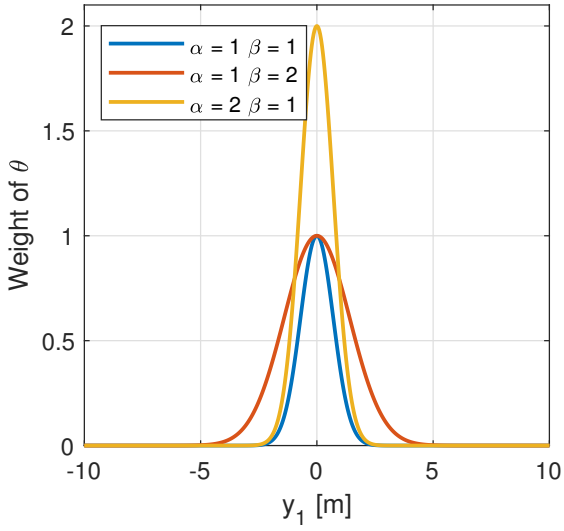


Figure 3.5: Variation of the θ weight coefficient *vs* y_1.

The Gaussian function is a suitable candidate function because it presents symmetry relatively to the vertical axis and is a bounded continuous function. Like shown in Figure 3.5, the weight has its maximum α when y_1 is zero, *i.e.*, when the vehicle is moving on top of the reference (assuming $s_1 \approx 0$). When $|y_1|$ increases, the weight of θ decays within an interval of y_1 controlled by β, outside which it can be considered to be zero. The tuning parameter β determines how far from the path the controller must start to correct the orientation as well. Each unitary increase on the value of β approximately increases the interval range 2.5 meters in each side.

To implement this functionality with the MPC controller described in Section 3.1, the entry corresponding to θ in the matrix of fixed weights $\boldsymbol{Q}$ in (3.4) must be changed to zero. Furthermore, to the cost function in (3.2), the following cost must be added

$$\ell_\theta(y_1, \theta) = \rho_\theta(y_1)\,\theta^2, \tag{3.12}$$

resulting in a new cost function

$$\ell'(\boldsymbol{x}, \boldsymbol{u}) = \ell(\boldsymbol{x}, \boldsymbol{u}) + \ell_\theta(y_1, \theta), \tag{3.13}$$

where y_1 and θ are part of $\boldsymbol{x}$.

3.5 Summarizing the MPC Problem Formulation

The functionalities introduced in the previous sections require changes to the MPC problem formulation, and thus this section briefly summarizes that information.

The new MPC formulation is given by

$$\underset{\overline{\boldsymbol{x}},\overline{\boldsymbol{u}}}{\text{minimize}} \quad J(\overline{\boldsymbol{x}},\overline{\boldsymbol{u}}) = \sum_{i=1}^{N} \ell(\overline{\boldsymbol{x}}_i,\overline{\boldsymbol{u}}_i) + \ell_\theta(\overline{y_1}_i,\overline{\theta}_i) \tag{3.14a}$$

$$\text{subject to} \quad \overline{\boldsymbol{x}}_0 = \boldsymbol{x}_t \ , \tag{3.14b}$$

$$\overline{\boldsymbol{x}}_i = f(\overline{\boldsymbol{x}}_{i-1},\overline{\boldsymbol{u}}_{i-1}), \tag{3.14c}$$

$$\overline{\boldsymbol{x}}_i \in \boldsymbol{\mathcal{X}}_i,\ i = 1,\ldots,N, \tag{3.14d}$$

$$\overline{\boldsymbol{u}}_i \in \boldsymbol{\mathcal{U}}_i,\ i = 1,\ldots,N-1, \tag{3.14e}$$

where $\overline{x}$ and $\overline{u}$ represent the predictions of x and u, and N denotes the prediction horizon. The function

$$\ell(\boldsymbol{x},\boldsymbol{u}) = \boldsymbol{x}^\top \boldsymbol{Q}\boldsymbol{x} + \boldsymbol{u}^\top \boldsymbol{R}\boldsymbol{u} \tag{3.15}$$

denotes the instantaneous linear quadratic fixed cost function to be minimized and introduced in Section 3.1.2, and

$$\ell_\theta(y_1,\theta) = \alpha\ \exp\left(\left[\frac{y_1}{\beta}\right]^2\right)\theta^2 \tag{3.16}$$

is the instantaneous variable weight quadratic cost function of the orientation error θ, introduced in Section 3.4, also to be minimized. It is remarked that, the entry corresponding to θ in $\boldsymbol{Q}$ must be equal to zero.

The optimization problem is subject to the constraints (3.14b) to (3.14e), where constraint (3.14b) imposes the initial condition on the state ($\boldsymbol{x}_t$ represents the state at time t), and constraint (3.14c) imposes the system model. It is remarked that, the control of the velocity of the virtual target presented in Section 3.3 is incorporated directly into the vehicle model $f(.)$, through the substitution of $\dot{s}$ in the generic vehicle model (2.32) by the function

$$\dot{s}(s_1) = \dot{s}_{ref}\ \exp\left(\frac{s_1}{\lambda}\right), \tag{3.17}$$

The state is also subject to a set of constraints represented by $\boldsymbol{\mathcal{X}}$ in (3.14d), and, likewise, the control inputs are subject to the constraints represented by $\boldsymbol{\mathcal{U}}$ in (3.14e). The path boundaries constraints, introduced in Section 3.2.1 and given by

$$l_{lower} \leq y_1 \leq l_{upper}\ , \tag{3.18}$$

and obstacle constraints, introduced in Section 3.2.2 and given by

$$(s_1 - C_{s_1})^2 + (y_1 - C_{y_1})^2 \geq r^2 \tag{3.19}$$

for a single obstacle, are included in $\boldsymbol{\mathcal{X}}$.

3.6 Path Definition

This section explains the composition of the path under the path parametrization of Section 2.1, as well as a method to quickly generate test paths to allow testing of the MPC controller.

3.6.1 Path Data Points

In the generic vehicle model in $\{F\}$ (2.32), it can be seen that the path influences the model via its curvature, $\kappa = \kappa(s)$. Due to the path parametrization, introduced in Section 2.1, each point of the path is identified by its unique arc-length s, and thus the path is defined by data points composed of pairs (s, κ).

Although, the arc-length and curvature are the only requirements of a vehicle model in $\{F\}$, each path data point must be augmented with the corresponding inertial coordinates and tangent angle θ_c to allow the computation of the new state after a control decision is applied to the vehicle. The resulting data points are comprised of $(s, \kappa, X, Y, \theta_c)$, although θ_c can also be computed online via equation (2.19).

To access curvature value (and other information) at the reference position, the controller must keep track of the arc-length position of the reference. The value can then be retrieved, using table lookup for example, with s as the query variable.

3.6.2 Tracking the Arc-length and Limitations

The arc-length localizes the controller in the path and, therefore, it must be known by the controller at all times to obtain information, such as the curvature. Tracking the arc-length can be achieved by either augmenting the vehicle model to track this variable or by exploiting non-linear constraints. Non-linear constraints in *fmincon* are implemented using a function that is called every iteration, therefore, by knowing the velocity of the reference, the arc-length can be computed for every step of the prediction window without having to be added as an optimization variable, and thus lower the added computational load. However, the arc-length must be tracked outside the *fmincon* function and can be passed as an external argument to this function.

Using the arc-length to retrieve information of the path can also be a limiting factor of the controller. First, it prevents linear models from being implemented with linear constraints, at least using the *fmincon* function of MATLAB. Since the linear constraints are set via matrices before the call of *fmincon*, they cannot be updated during the optimization cycle to account for the movement of the reference. However, at the cost of more inaccurate predictions, there is the option of leaving the curvature constant throughout the optimization cycle and then, in the next sampling instant, update the model with the new curvature value.

Furthermore, if a table is used to store the data points of the path, then searching through and accessing the information in this table implies a noticeable computational cost. Since the reference can change positions from iteration to iteration, the curvature value must be retrieved at every iteration, which accounts for hundreds or thousands of times per call to *fmincon*, depending how the latter one is configured. In Section 3.8 some code optimizations are presented to help mitigate this problem.

3.6.3 Test Path Generation

As part of a bigger system, in a real world application of an autonomous race car, the controller would be given by other modules an updated path to follow every sampling instant. However, since those other modules are absent in this work, this section explains a quick method to offline generate paths (or tracks) for the vehicle to follow, and thus allow the controller to be tested.

In motorsport, the majority of the circuits are closed tracks, meaning the start and end points are the same. A simple way to generate closed track is to start by using the parametric equations of the ellipse

$$\begin{cases} X(\phi) = W\, cos(\phi) + C_x \\ Y(\phi) = H\, sin(\phi) + C_y \end{cases} , \phi \in [0,\ 2\pi[\tag{3.20}$$

where W and H control the width and height of the footprint of the ellipse, in meters, respectively, (C_x, C_y) is center of the ellipse, and ϕ is a parametrization variable. If $W = H = r$, then the track obtained is a circle with radius r.

A slightly more complex track can be created by modifying the equations above to produce a "∞" shaped track

$$\begin{cases} X(\phi) = W\, cos(\phi) + C_x \\ Y(\phi) = H\, sin(\phi)\, cos(\phi) + C_y \end{cases} , \phi \in [0,\ 2\pi[\ . \tag{3.21}$$

The signed curvature κ and the the tangent angle θ_c of each point of the path can be obtained, respectively, from the parametric equations by

$$\kappa = \frac{X'Y'' - Y'X''}{(X'^2 + Y'^2)^{\frac{3}{2}}} \tag{3.22}$$

$$\theta_c = tan^{-1}\left(\frac{Y'}{X'}\right), \tag{3.23}$$

where the primes represent the derivatives with respect to the parametrization variable, in this case ϕ.

To obtain the arc-length, the path has to be discretized into points by discretizing the interval where ϕ is defined. The arc-length s_i of point P_i corresponds to the length of the section of the path that beginnings in the first point of the path, where $s_0 = 0$, and ends at point P_i. The arc-length of a point can be approximated by the sum of the arc-lenght of previous point and the Euclidean distance between the two, *i.e.*, is the cumulative Euclidean distance between every point of the path until P_i. In mathematical terms, the arc-length at P_i is given by

$$s_i = \sum_{j=1}^{i} ||P_j - P_{j-1}|| \ , i > 0, \tag{3.24}$$

where P represents the coordinates in $\{I\}$.

The discretization of the interval where ϕ is defined sets the number of data points of the path. The

controller computes the path by integrating the curvature and the velocity of the reference, therefore, there will always be some numerical errors associated with this operation. One potential source of numerical errors comes from discretization of the path, therefore, to mitigate this source an appropriate number of points must be used to define the path.

There must be enough points to smoothly capture changes of the curvature. Furthermore, notice that in (3.24) the arc-length between two points is approximated by the Euclidean distance between the two, and, as a consequence, there are numerical errors that accumulate from point to point, especially in paths with curved sections. Having more points to define the path mitigates this problem to a point where these errors can be considered negligible. However, attention must also be paid not to have an excessively large number of points per unit of length of path, since there are diminishing returns associated. Beyond a certain limit, having more points does not have a noticeable effect in the numerical error but makes searching through the points slower, therefore, the temporal performance of the controller is degraded.

The numerical errors associated with the path discretization are only a noticeable problem when testing the controller in standalone mode, because the hole path is computed offline beforehand, and thus the numerical errors accumulate from point to point over the entire length of the path. In a real world application, the path is regenerated every sampling instant based on the current position of the vehicle, therefore, any eventual errors in the vehicle position accumulated from the discretization of the path get compensated by the feedback loop of the path generation.

3.7 Runge Kutta 4$^{\text{th}}$ Order *vs* Euler Integration Method

The purpose of the experiments shown in this section is to demonstrate the impact that different integration methods of the state equation have on the trajectory of the vehicle under the Frenet-Serret formulation, and thus it is not the intent to evaluate the performance of the controllers, since that is done in more detail in Chapter 4.

The vehicle models derived from (2.32) are composed of first-order ordinary differential equations (ODEs), that can be solved using an integration method. The most common method used to solve ODEs is the Euler method. Given the following initial value problem

$$\dot{y}(t) = f(t, y(t)) \tag{3.25}$$

$$y(t_0) = y_0, \tag{3.26}$$

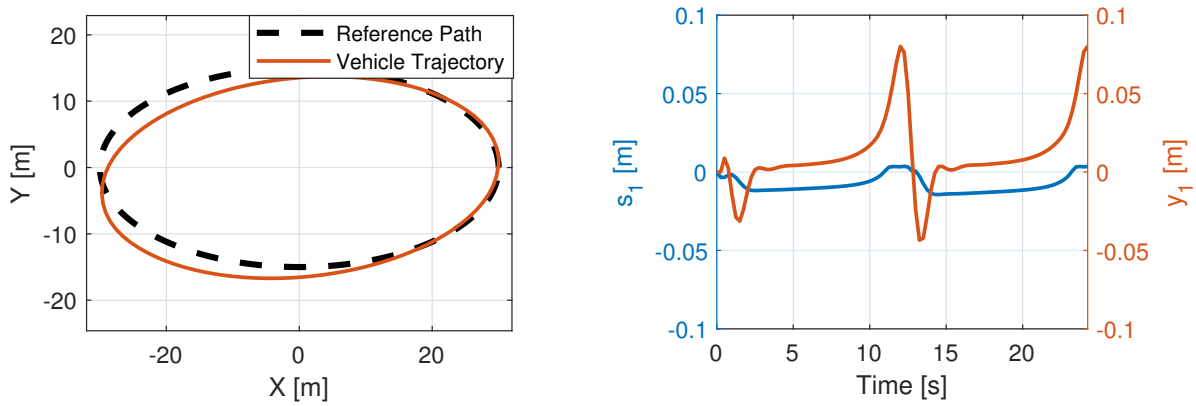
for a certain integration step h, the Euler method gives an approximate solution at time $t_{n+1} = t_n + h$ equal to

$$y_{n+1} = y_n + hf(t_n, y_n). \tag{3.27}$$

Because the solution obtained is only an approximation, there are numerical errors associated with the Euler Method, which are directly proportional to the integration step h. In open-loop the numerical errors are free to accumulate over time, but in a closed-loop environment these errors are compensated

by the controller, and thus are usually bounded to a range of value.

An example of the accumulation of numerical errors can be seen in Figure 3.6, where a simulation performed using the Euler method with an integration step of 0.25 seconds is shown. The model used was the unicycle kinematic model (2.37).



(a) Trajectory of the vehicle *vs* the reference path

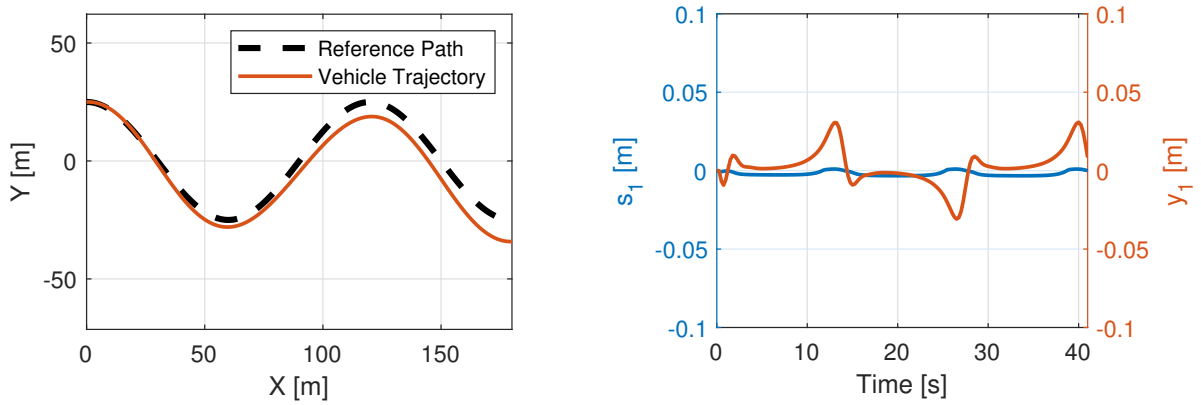
(b) s_1 and y_1 state variables over the course of the lap

Figure 3.6: Simulation of the unicycle kinematic model using the Euler method with $h = 0.25\,s$ in a elliptical track.

Notice how in, Figure 3.6, the position errors in the Frenet-Serret frame are bounded to a 10 centimeter range around the reference, and yet the trajectory of the vehicle in the inertial frame shows errors of several meters. The discrepancy seen between the actual trajectory of the vehicle and the evolution of the position error can be explained by where the controller thinks the reference is. The position of the reference in $\{I\}$ is given by a model that is built into the vehicle model under the Frenet-Serret formulation explained in Chapter 2. Consequently, the position of the reference is also computed using the Euler method (from the path curvature and virtual target velocity), and thus is also bound to accumulate numerical errors. However, unlike the vehicle position, the error between the computed reference position and its actual location in $\{I\}$ is not corrected by a feedback loop, and thus the position of the reference accumulates numerical errors over time. This reason explains why the controller shows the vehicle as being nearly on top of the reference, but the actual trajectory is displaced relatively to the reference path.

Even though Figure 3.6 shows errors in the trajectory, the overall shape of the vehicle trajectory is similar to the true reference path. Futhermore, the vehicle is able to return to its initial position at (30,0), even though the computed reference position is free to accumulate errors. This phenomenon is explained by the closed nature of the path, since the errors accumulated in one half of the trajectory get canceled by the errors of the other half when the vehicle inverts its direction of movement. That is why the halfway point of the trajectory is the point that exhibits the biggest displacement from its intended location. The simulation shown in Figure 3.7 shows what happens when the reference path is not closed.

Notice, in Figure 3.7, how the trajectory of the vehicle keeps getting more and more displaced from the true reference path. Once again, the position errors are bounded to a few centimeters, but the trajectory of the vehicle gradually accumulates errors as the simulation progresses.



(a) Trajectory of the vehicle *vs* the reference path

(b) s_1 and y_1 state variables over the course of the path

Figure 3.7: Simulation of the unicycle kinematic model using the Euler method with $h = 0.25\,s$ in a sinusoidal track.

The numerical errors accrued with the Euler method can be made smaller over time by decreasing the size of the integration step, although at the cost of increased computational load, which might not be a feasible solution every time.

Another option, to mitigate the numerical errors accumulated with the Euler method, is to use another integration method to solve the model. In this work, the Runge-Kutta 4$^{\text{th}}$ Order method (RK4) is proposed as an alternative to the Euler method. Given the same initial value problem and integration step, the RK4 method provides the following approximate solution

$$y_{n+1} = y_n + \frac{1}{6}h(k_1 + 2k_2 + 2k_3 + k_4), \tag{3.28}$$

where

$$k_1 = f(t_n, y_n), \tag{3.29}$$

$$k_2 = f(t_n + h/2, y_n + hk_1/2), \tag{3.30}$$

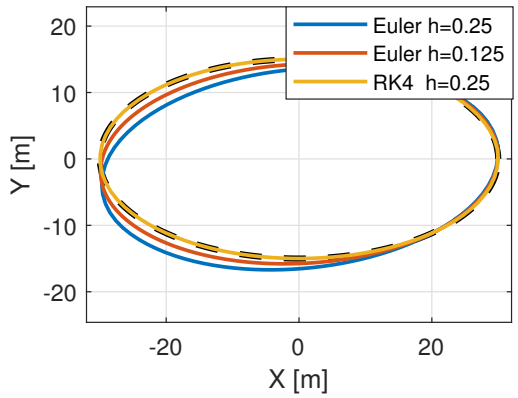
$$k_3 = f(t_n + h/2, y_n + hk_2/2), \tag{3.31}$$

$$k_4 = f(t_n + h, y_n + hk_3). \tag{3.32}$$

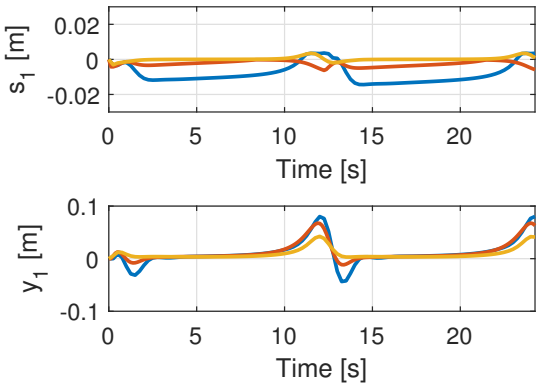
Figures 3.8 and 3.9 show the comparison between the resulting vehicle trajectories using the Runge-Kutta 4$^{\text{th}}$ order method with $h = 0.25$ seconds and the Euler method using $h = 0.125$ and $h = 0.25$ seconds. The other simulation parameters are kept the same between simulations.

Notice, in Figures 3.8 and 3.9, how the use of the RK4 method allows the controller to perform better in every aspect when compared to the Euler method. The position errors are bounded to a smaller range of values, while the resulting trajectories of the vehicle are both on top of the true reference path in the two simulations. Furthermore, the RK4 method performs noticeably better using $h = 0.25$ seconds when compared to Euler method using $h = 0.125$ seconds.

However, it is noted that, because the RK4 method also provides an approximate solution, the numerical errors still exist when computing the reference position, although much smaller in magnitude. Even

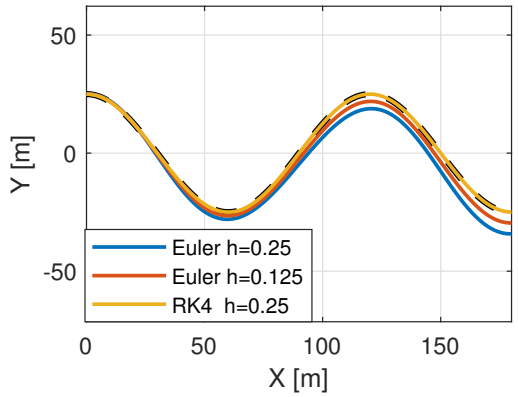


(a) Trajectories of the vehicle using different integration methods

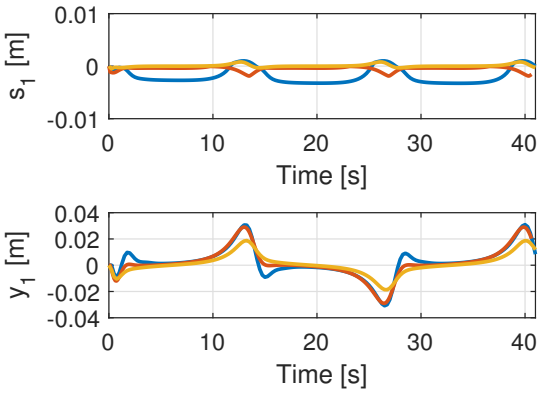


(b) s_1 and y_1 state variables for the different integration methods

Figure 3.8: Comparison between the Euler and RK4 methods using the unicycle kinematic model in an elliptical track.



(a) Trajectories of the vehicle using different integration methods



(b) s_1 and y_1 state variables for the different integration methods

Figure 3.9: Comparison between the Euler and RK4 methods using the unicycle kinematic model in a sinusoidal track.

tough the Runge-Kutta 4th Order method requires more computations per time step, it compensates by obtaining better results with bigger integration steps when compared with the Euler method.

It is remarked that, in the simulations shown in this section, the state used as the initial condition every controller sampling instant is the one computed (*i.e.*, predicted) by the MPC in the previous time step, and thus the control decisions are not fed into a model outside the MPC. If the model is integrated outside the MPC, the integration period for the model equations in the simulation must be smaller than the controller sampling period, *i.e.*, the model equations must be integrated over smaller periods between controller sampling instants.

Since the initial condition is computed by the MPC in the previous sampling instant, these simulations mainly highlight the accumulation of numerical errors during the computation cycles, *i.e.*, between prediction steps. As shown in this section, the accumulated numerical errors are quite significant over the prediction window, especially when using the Euler method, and thus the Runge-Kutta 4th Order method is used to obtain more accurate prediction during the prediction window.

For the reasons explained in this section, the remaining simulations in this work are performed using Runge-Kutta 4th Order method, unless otherwise stated. Furthermore, the initial state of the vehicle in $\{F\}$ is computed every controller sampling instant with the measured (or simulated) inertial state of the vehicle and virtual target, since these are states that can be directly measured in a real world scenario.

In a real world application, any remaining accumulated numerical errors are corrected by the feedback loop created by the motion planning module by generating a new path, every controller sampling instant, based on the latest position of the vehicle.

3.8 Code Optimizations and Table Lookup Algorithm

As discussed in Section 3.6, the path is discretized into data points in this work. To retrieve information from the set of path data points, the arc-length s is used as a query variable because it has a unique value for every point of the path. Since the position of the reference can change between time steps, path information, like the curvature, has to be retrieved every time step. This information gathering operation is very quick by itself, especially if the data points are stored in memory. However, it quickly starts to add up and become noticeable when this same operation is performed hundreds or thousands of times during the resolution of the optimization problem.

This work was developed in MATLAB and these are some of the followed coding best practices that were found to produce a significant increase in computation speed:

- Using a column vectors to store the same kind of path information,*i.e.*, each data point corresponds to a row in a matrix. In MATLAB, columns are stored in series in memory, and thus it is faster to search through the same type of information, like the arc-length.
- Avoiding global variables and structures, especially the later, in anything that is called by fmincon, since accessing these types of data takes significantly longer. It is also recommended that any function called by fmincon be optimized to run as fast as possible.
- Use a search algorithm to find the closest point that matches the given arc-length and thus avoiding having to search every element of the vector.

For this work, a simple search algorithm was developed which improved the computations speeds by an average factor of 10, when compared to searching through every element of the vector. There are other algorithms that probably decrease the computation time even further, but, since the computation time is not the focus of this work, these were not tested.

Assuming that the path data points are stored sequentially, to find a point with a given s, the algorithm divides the arc-length vector into smaller vectors, and then searches the last element of each sub-vector sequentially until it finds one that is bigger than the given s. At that point it, knows that the given point lies withing the sub-vector whose last element it just checked. Finally, it goes through every element of the sub-vector until it finds the point which has the closest arc-length.

Some modifications can be made to the algorithm, which include the interpolation of the data between the two closest points, and so on. However, interpolation increases the execution time of this

algorithm and thus is not implemented in this work. The error caused by not using the exact reference exists, but can be considered negligible, given the order of magnitude of the problem, if the set of data points is properly sized. In this work, the path points are spaced by just 3-4 millimeters.

Chapter 4

Results

In this chapter several experiments are performed to test the controller using different configurations. The methodology is to gradually implement new features and test the influence of controller parameters to provide a basis of comparison between controllers. The initial conditions are also kept the same for the most part to allow for an easier comparison between experiments, and the vehicle used is always the unicycle.

The test path is the "∞" shaped track with 335 meters in length and a maximum radius of curvature of about 9 meters. This track contains different sections simulating different scenarios that can be found in real world race tracks, such as straights, corners, and chicanes, an thus provides a suitable test environment.

The results shown in this chapter were obtained using a computer with a stock Intel(R) Core(TM) i5-7600K CPU @ 3.8GHz and 16.0 GB of RAM.

4.1 Unicycle Kinematic Model

In this experiment, the MPC controller uses the kinematic model of the unicycle, defined by the equations in (2.37), with no constraints on either the state or control variables, and thus the problem can be formulated as follows

$$\underset{\boldsymbol{x},\,\boldsymbol{u}}{\text{minimize}} \quad \sum_{i=1}^{N} \overline{\boldsymbol{x}}_i^{\top} \boldsymbol{Q} \overline{\boldsymbol{x}}_i + \overline{\boldsymbol{u}}_i^{\top} \boldsymbol{R} \overline{\boldsymbol{u}}_i \tag{4.1a}$$

$$\text{subject to} \quad \overline{\boldsymbol{x}}_0 = \boldsymbol{x}_t, \tag{4.1b}$$

$$(2.37), \tag{4.1c}$$

where $\boldsymbol{Q}$ and $\boldsymbol{R}$ are the fixed cost diagonal matrices of the state and input variables, respectively, defined in Section 3.1.2.

The purpose of this experiment is to demonstrate the core path following capabilities of using a vehicle model in $\{F\}$ with a MPC controller. The controller is tuned with the parameters shown in Table 4.1, where T_s and N denote the sampling period and prediction horizon, respectively. In the same table,

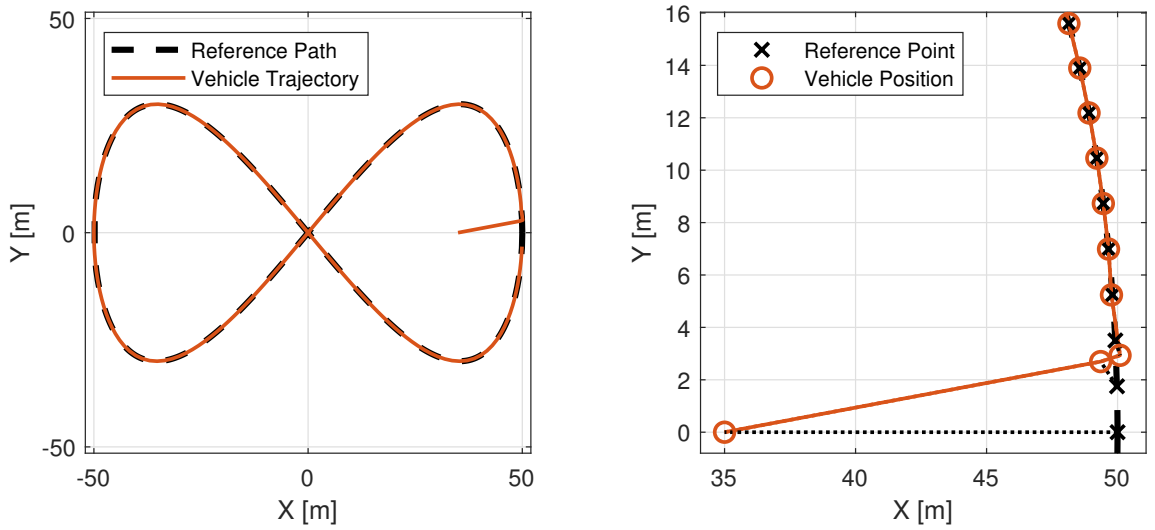
the weights correspond to the cost function weights given to each state and control variables.

			Weights				
			States			Inputs	
Figures	T_s	N	s_1	y_1	θ	v	ω_m
4.1, 4.2	0.125 s	5	1	1	1	0	0

Table 4.1: Controller parameters using the unicycle kinematic model.

This experiment uses a constant speed for the virtual target equal to 14 m/s (or 50.4 km/h) and it moves anti-clockwise starting from the rightmost point of the reference path shown in Figure 4.1, or point (50, 0) in $\{I\}$.

As an introductory reminder, at the virtual target start position of (50, 0) and with the chosen movement orientation, the tangential axis of the $\{F\}$ is pointing upwards and the normal axis is pointing towards the origin, which means the orientation of the Frenet-Serret frame is θ_c = 90° . With this description in mind, vehicle initial conditions are equal to $x_0 = \begin{bmatrix} 0 & 15 & 0 \end{bmatrix}^\top$, meaning that the vehicle starts with a tangential and normal offset of 0 and 15 meters, respectively, relatively to the initial position of the virtual target, which equates to a vehicle the start position (35, 0) in $\{I\}$. Also, θ = 0° means that the vehicle starts aligned with the tangential axis of $\{F\}$, and thus the initial orientation of the vehicle is θ_m = 90° .

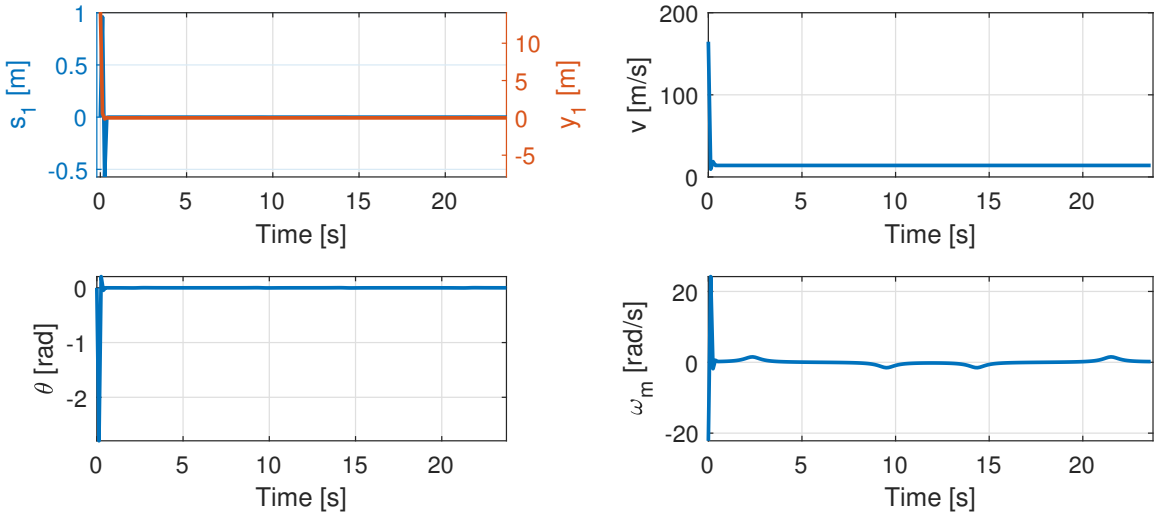


(a) Trajectory of the vehicle

(b) Zoomed-in details of the trajectory of the vehicle

Figure 4.1: Path following using the kinematic model of the unicycle and a constant virtual target speed.

Consider Figure 4.1 where the resulting vehicle trajectory is shown. It can be seen that the controller is successful in controlling the vehicle to stay with the virtual target over the entire length of the path. The zoomed-in details of the initial part of the trajectory show the position of the vehicle as well as the position of the virtual target that the controller is tracking at that time instant. In the same figure, it can be seen that after three sampling periods the controller is able to converge to the position of the virtual target, after which they move along together.



(a) State variables

(b) Control variables

Figure 4.2: Evolution of the state and control variables over time using the unicycle kinematic model.

Figure 4.2 proves the controller is successful in following the path, since it shows both the position and orientation error of the vehicle quickly converging to zero, although at the cost of very strong control signals. This behavior is to be expected since the vehicle model is purely kinematic and the control variables are not constrained or weighted.

4.2 Unicycle Dynamic Model

The unicycle kinematic model experiment provides the proof of concept that path following can be achieved with a MPC controller using a vehicle model in $\{F\}$. However, this experiment does not consider any vehicle dynamics, therefore, to expand on the previous results, a new similar experiment is performed using the unicycle dynamic model defined by the equations in (2.40). Although limited in the dynamics it accounts for, the unicycle dynamics model provides a solid test case to showcase and analyze the path following performance of the controller under different conditions, as well as the influence that different tuning parameters have on the behavior of the vehicle. The problem formulation is very similar to the previous experiment and is given by

$$\underset{\boldsymbol{x},\,\boldsymbol{u}}{\text{minimize}} \quad \sum_{i=1}^{N} \overline{\boldsymbol{x}}_i^\top \boldsymbol{Q} \overline{\boldsymbol{x}}_i + \overline{\boldsymbol{u}}_i^\top \boldsymbol{R} \overline{\boldsymbol{u}}_i \tag{4.2a}$$

$$\text{subject to} \quad \overline{\boldsymbol{x}}_0 = \boldsymbol{x}_t, \tag{4.2b}$$

$$(2.40), \tag{4.2c}$$

$$\overline{\tau_1}, \overline{\tau_2} \in [-100, 100], \tag{4.2d}$$

where $\boldsymbol{Q}$ and $\boldsymbol{R}$ are now defined considering to the new extended state and new input variables.

The unicycle considered in this experiment has a rectangular shape measuring 2.8 by 1.3 meters

in length and width, respectively, and the other parameters of the vehicle are shown in Table 4.2. The unicycle is actuated by two independent motors, whose individual torque output, including the drivetrain, is limited to the interval $[-100, 100]$ N.m via constraints on the input variables τ_1 and τ_2. The physical properties of the model were chosen to bring the unicycle in line with the typical characteristics of a Formula Student electric car.

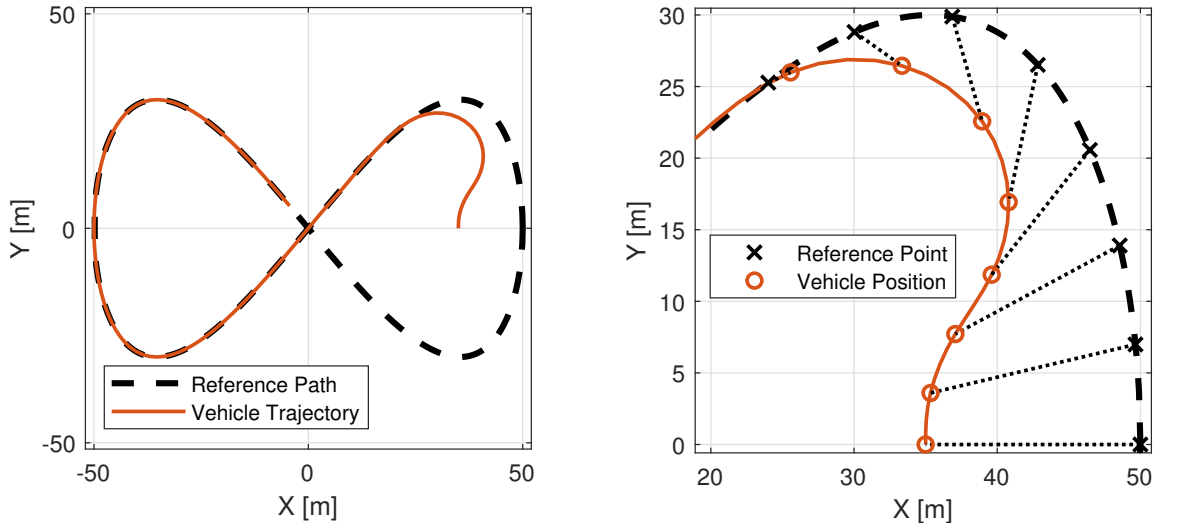
m [kg]	L [m]	R [m]	I [kg.m^2]
200	0.5	0.25	158.8(3)

Table 4.2: Unicycle physical parameters.

The track, virtual target and vehicle initial conditions are the same as the previous experiment, except for the addition of a vehicle initial linear velocity, v, which is set to 7 m/s (or 25.2 km/h), and an angular velocity, ω_m, of 0 rad/s, resulting in the initial state $x_0 = \begin{bmatrix} 0 & 15 & 0 & 7 & 0 \end{bmatrix}^\top$. As a reminded, the virtual target moves at a constant speed of 14 m/s (or 50.4 km/h). The controller parameters for this experiment are presented in Table 4.3.

		Weights							Constraints	
		States					Inputs		States	Inputs
T_s	N	s_1	y_1	θ	v	ω_m	τ_1	τ_2	-	τ_1, τ_2
0.125 s	10	1	1	1	0	0	0	0	-	$[-100, 100]$

Table 4.3: Controller parameters using the unicycle dynamic model.



(a) Trajectories

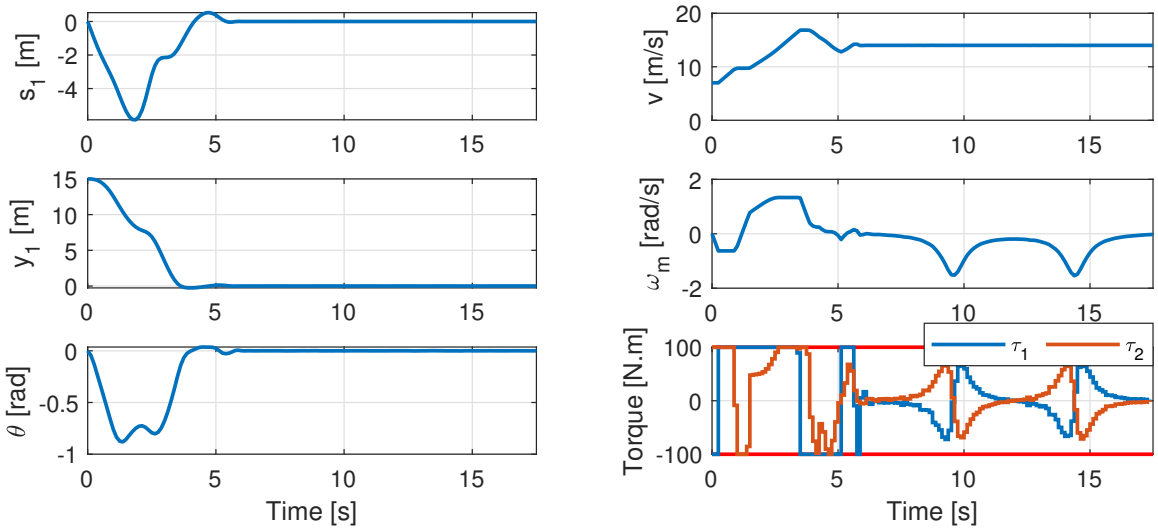
(b) Zoomed-in details of the trajectory spaced every 0.5 s

Figure 4.3: Resulting trajectory of the unicycle using a constant virtual target speed of 14 m/s.

Consider Figure 4.3 where the trajectory of the vehicle as well as the zoomed-in details of the initial part of the trajectory are shown. In this experiment the vehicle takes longer to converge with the moving reference, since it starts with half the speed of the virtual target and the model now accounts for some dynamics, preventing the vehicle from almost instantaneously converging to the reference, as shown

in the previous experiment with the kinematic model. Eventually the vehicle is able to successfully converge with the reference and move along with it for the rest of the path.

In the zoomed-in details of the initial part of trajectory shown in Figure 4.3, notice that the vehicle is not able to converge to the reference point immediately, because the latter is moving faster than the vehicle, and so the vehicle takes a shorter path that allows it to simultaneous get to speed and catch up with the virtual target later on. To note is that the points shown are spaced 0.5 seconds apart to avoid cluttering the image, and thus there are more points in between not shown.



(a) States s_1, y_1, and θ

(b) States v, w_m, and inputs τ_1 and τ_2

Figure 4.4: Evolution of the state and control variables over time.

Figure 4.4 shows the evolution over time of the state and control variables. Notice the red horizontal lines in the motor torques indicating the MPC constraints imposed to these variables. This figure shows the position and orientation errors converging to zero, and thus the controller is successful in controlling the vehicle to follow the path with dynamics accounted for. Notice also the vehicle and reference velocities converging to the same value of 14 m/s after a successful convergence.

4.3 Sampling Period and Prediction Horizon Influence

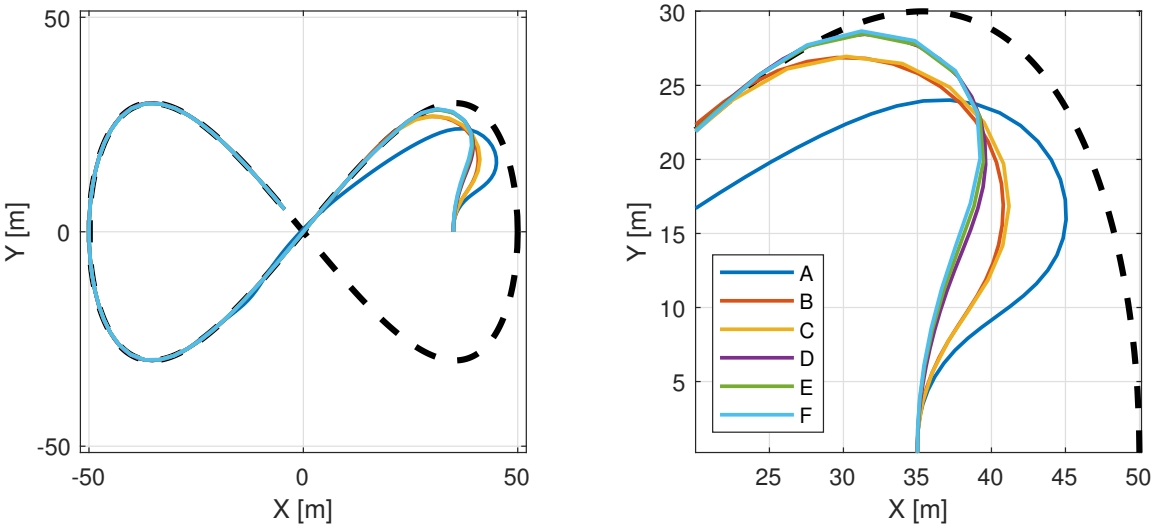
The purpose of this experiment is to study the influence that the sampling period, T_s, and prediction horizon, N, have on the performance of the controller. This experiment expands on the work of the previous experiment, with the vehicle parameters, initial conditions, and controller parameters kept the same, except for T_s and N, so refer to Section 4.2 to obtain more details on the conditions of this experiment.

The parameters of the different controllers are shown in Table 4.4, where controller B is the same used in the previous experiment of Section 4.2. Table 4.5 contains the average computations times per time step and their variance.

Figure 4.5 shows the resulting vehicle trajectories of the differently tuned controllers. Looking at the

			Weights							Constraints	
			States					Inputs		States	Inputs
Controller	T_s	N	s_1	y_1	θ	v	ω_m	τ_1	τ_2	-	τ_1, τ_2
A	0.125 s	5	1	1	1	0	0	0	0	-	$[-100, 100]$
B	0.125 s	10	1	1	1	0	0	0	0	-	$[-100, 100]$
C	0.25 s	5	1	1	1	0	0	0	0	-	$[-100, 100]$
D	0.125 s	20	1	1	1	0	0	0	0	-	$[-100, 100]$
E	0.25 s	10	1	1	1	0	0	0	0	-	$[-100, 100]$
F	0.25 s	15	1	1	1	0	0	0	0	-	$[-100, 100]$

Table 4.4: Parameters of the controllers using different prediction horizons and sampling periods.



(a) Resulting trajectories

(b) Zoomed-in initial part of the different trajectories

Figure 4.5: Trajectories using different prediction horizons and sampling periods.

zoomed-in initial section, the trajectories can be aggregated in three different groups. The first group is composed only by the trajectory of controller A, which in turn uses the smallest combination of T_s and N of all controllers.

	A	B	C	D	E	F
Mean [s]	0.19	1.10	0.29	6.59	1.60	4.00
Variance [s]	0.01	0.22	0.01	0.94	0.09	0.27

Table 4.5: Average computation times, per time step, and their variance using different T_s and N.

The second group is composed by controllers B and C, which present very similar trajectories. Controller B uses a half the T_s but double the N when compared to controller C, meaning that both controllers have effectively the same prediction window of 1.25 seconds into the future. Controller B has more prediction steps in this time frame, at the cost of extra +280% in the computation time, although it does not benefit from that extra knowledge, since both trajectories are almost the same. When comparing controller A with B or C, a conclusion can be made that controller A does not have a large enough prediction window to allow it to effectively converge with the path as soon as the other controllers are able to.

The third group is composed by the remaining controllers D, E, and F. As it is the case for the

controllers in the second group, controller D and E have the same prediction window, although two times bigger, at 2.5 seconds into the future, when compared to B and C. The trajectories taken by the vehicles are, again, almost identical between the D and E. However, when compared with B and C, a larger prediction window allows the vehicle to converge with the path a little sooner, meaning the controller benefits from the extend prediction window, although not as much as the controller B and C benefited when compared to A. Controller F has the largest prediction window of all at 3.75 seconds, but it does not benefit from it since the trajectory is the same has the controllers D and E.

Overall, all controllers are able to converge and follow the path for the remaining part of the experiment, meaning that the sampling period and prediction horizon only seem to have a noticeable role when the vehicle gets off track under these conditions. It is also to be expected that an increase of the reference velocity of the virtual target (which is set to 14 m/s in this experiment) must be accompanied with an increase of the prediction window, since the vehicle travels a larger distance in a the same amount of time.

Regarding computation times, notice in Table 4.5 how doubling only the sampling period seams to increases the computation time linearly by a factor of approximately 1.5. On the other hand, increasing the prediction horizon, while keeping the sampling period, increases the computation time exponentially. Although this work does not try to prove computational feasibility, none of the controllers in this section are able to consistently finish their computations before the next sampling step.

In conclusion, the sampling period and prediction horizon must sized to allow a balance between computational load and prediction window into the future. For the same prediction window, a smaller T_s and bigger N increases the computational load, but, at the same time, increases the amount of future states the controller has to play with, so a balance must be achieved here.

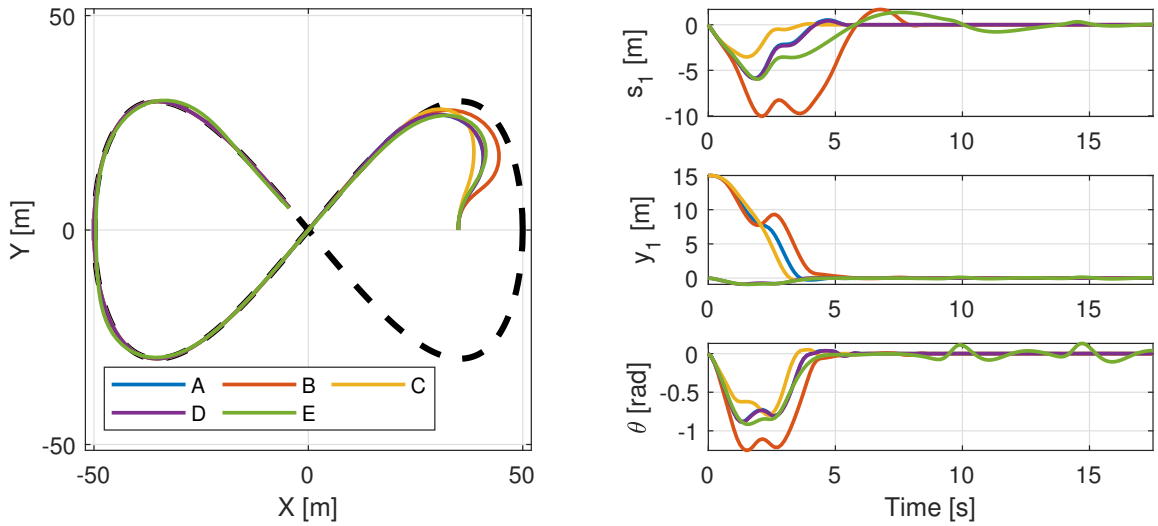
4.4 Cost Function Weights Influence

The cost function weight given to each state or control variables influences the control decision obtained with an MPC controller, and thus this experiment aims to showcase the influence that some weights have on the behavior of the vehicle. Once again, the template for this experiment is Section 4.2, where only the weights have been changed between controllers. The controller parameters are shown in Table 4.6, with controller A being the same as the one in experiment of Section 4.2.

			Weights							Constraints	
			States					Inputs		States	Inputs
Controller	T_s	N	s_1	y_1	θ	v	ω_m	τ_1	τ_2	-	τ_1, τ_2
A	0.125 s	10	1	1	1	0	0	0	0	-	$[-100, 100]$
B	0.125 s	10	1	3	1	0	0	0	0	-	$[-100, 100]$
C	0.125 s	10	3	1	1	0	0	0	0	-	$[-100, 100]$
D	0.125 s	10	1	1	0	0	0	0	0	-	$[-100, 100]$
E	0.125 s	10	1	1	1	0	0	0.001	0.001	-	$[-100, 100]$

Table 4.6: Parameters of the different controllers.

Under the current MPC formulation, any optimization variable whose weight is not equal to zero is penalized in the cost function if its value is not equal to zero, and thus the controller will try to make that variable go to zero. This behavior is desirable in the error states such as s_1, y_1 and θ since these states must be zero to make path following possible, but variables such as v, ω_m, τ_1 and τ_2 take non-zero values most of the time, therefore any non-zero weight given to these variables negatively impacts the ability of the controller to make the error states go to zero. For this reason v and ω_m are not weighted in any of the controllers of Table 4.6, and τ_1 and τ_2 are only weighted in controller E for demonstration purposes.



(a) Trajectories

(b) Evolution over time of the error states

Figure 4.6: Trajectories and evolution of the error states of the differently tuned controllers.

Consider Figure 4.6 where the trajectories of the different controllers are shown alongside the evolution of the error states. Notice that, even though controller D has the orientation error weight equal to zero, θ still converges to zero. Furthermore, the trajectories and state evolution of controller D are the same as controller A (they overlap), therefore, a conclusion can be made that the weight of θ does not have a big influence on the controller. This behavior can be explained by the fact that the virtual target is always moving, therefore, for the vehicle to move along with it, and thus make s_1 and y_1 zero, it has to be moving in the same direction as the virtual target, making θ equal to zero as well. Although it has reduced influence, it is recommended that the weight of θ be a smaller value when compared to the other error states to force the controller to track this variable and effectively make it zero. That being said, in experiment 4.6, the weight of θ is seen to play a bigger role once the speed of the virtual target starts being controlled.

In controller C, the weight of s_1 is increased relatively to the other variables, and thus the corresponding state variable is seen within the smallest interval of values of all controllers, due to the added importance. The controller prioritizes minimizing s_1, and thus is the first controller to converge to the reference by taking a more straight trajectory to the virtual target. On the flip side, controller B puts added pressure on the state variable y_1, and thus, in the beginning, it attempts to decrease the normal distance

to the path more than the other controllers. However, because it never reached the virtual target, the latter eventually enters the curve, which, consequently, starts changing the orientation of $\{F\}$, which leads to an increase of y_1 and θ. The controller eventually catches up with the moving reference and makes the errors go to zero.

Controller E has the motor torques weighted by a small value, but its enough to see a noticeable degradation in the performance of the controller, specially in the curves. As is to be expected, the controller is reluctant in using higher values for τ_1 and τ_2 as it increases their weight in the cost function, thus the vehicle is inevitably slower in reacting to the changes of the path. In certain occasions, weighting the control variables produces smother control signals, which is the case even though these signals are not shown, but it always comes with some degree of degradation in performance, thus a balance must be achieved if these variables are to be weighted. It is also to be expected that weighting v or w_m result in the same kind of behavior, as these two variables are directly tied with the motor torques.

4.5 Virtual Target Speed Control

In Section 2.1, when introducing a moving reference that moves with velocity $\dot{s}$ over the path, it is said that this variable is effectively an extra control variable besides the regular inputs of the vehicle, although, up until this point, the speed of the virtual target has been kept constant throughout all experiments.

The previous experiments show some of the capabilities of the MPC controller, but also demonstrate that the controller is limited in what it can do when the reference keeps on moving and the motor torques are physically limited. The result is the controller taking a shortcut in other catch up with the virtual target, beyond which point has no problem keeping up with it.

In Section 3.3 it is proposed to use an exponential function of s_1 to control the speed of the virtual target. The purpose of this experiment is to demonstrate the added benefits of controlling the speed of the virtual target with the proposed method.

To provide a base of comparison, the vehicle properties and initial conditions are the same as the previous experiments. The controller parameters, which are the same as experiment of Section 4.2, are shown in Table 4.7. Regarding the virtual target, the reference speed is set to be 14 m/s when $s_1 = 0$ and the tuning parameter λ, which controls the aggressiveness of the virtual target speed profile, is shown in Table 4.8 for the different controllers. The problem formulation is the same as Section 4.2, with the exception of the inclusion (by substitution) of the exponential virtual target speed controller, given by equation (3.10) in the vehicle model (2.40).

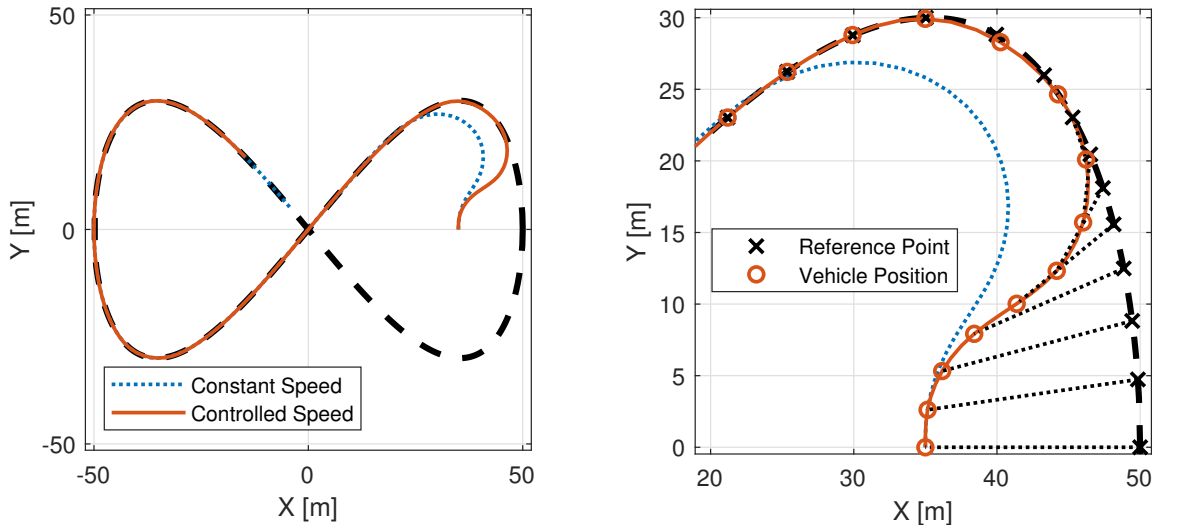
			Weights							Constraints	
			States					Inputs		States	Inputs
Controller	T_s	N	s_1	y_1	θ	v	ω_m	τ_1	τ_2	-	τ_1, τ_2
A, B, C, D	0.125 s	10	1	1	1	0	0	0	0	-	$[-100, 100]$

Table 4.7: Base parameters of the controllers used in the virtual target speed control experiments.

Consider Figure 4.7, where the resulting vehicle trajectory using controller C is shown. In the same

Controller	λ
A	2
B	4
C	6
D	8

Table 4.8: Configuration of the virtual target speed controller for the different controllers.



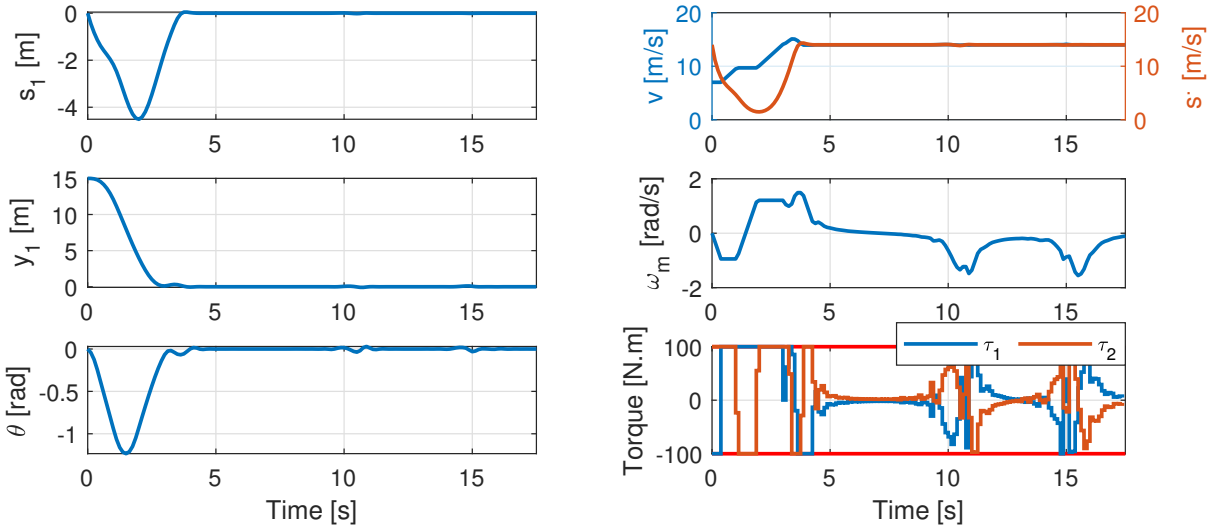
(a) Trajectory comparision

(b) Zoomed-in details of the trajectory spaced every 0.375 s

Figure 4.7: Comparison of two trajectories using and not using a controlled virtual target speed.

picture, the vehicle trajectory of the experiment of Section 4.2, which uses a constant virtual target speed is also shown. Notice how the vehicle is able to converge with the path much sooner when compared to the controller that uses a constant speed for the virtual target. Notice also how the virtual target first speeds up, then slows down for the vehicle to be able to get close, after which it picks up speed again, resulting in a smooth convergence with the path. This behavior allows the controller to focus on decreasing the normal distance to the path without having to worry about the virtual target getting way. Notice that, at first, the vehicle starts with $s_1 = 0$ meaning the virtual target is moving at 14 m/s. However, since the vehicle starts with a lower velocity of 7 m/s it quickly starts to lag behind the virtual target, increasing s_1. Eventually the virtual target slows down until the vehicle is able to come close again. This behavior can also be observed in Figure 4.8. As long as the controller is focused on making s_1 zero, the vehicle is moving as fast as it can while getting back to the path.

Figure 4.9 shows the impact that the tuning parameter λ has on the resulting vehicle trajectory. Decreasing λ makes the virtual target move at slower speeds for smaller values of s_1, but also means that, once s_1 starts approaching zero, the velocity of the virtual target will pick up much quicker, resulting in some overshoot for smaller values of λ (for this controller configuration), since the vehicle has to suddenly accelerate to catch the virtual target that picked up speed very quickly. Higher values of λ mean that the virtual target is slower slowing down, resulting in higher velocities for bigger negative values of s_1. On the flip-side, the virtual target is also slower picking up speed when the vehicle stars to



(a) Error states s_1, y_1, and θ

(b) States v, w_m, and inputs $\dot{s}$, τ_1, and τ_2

Figure 4.8: Evolution over time of the state and control variables of controller C (λ=6) .

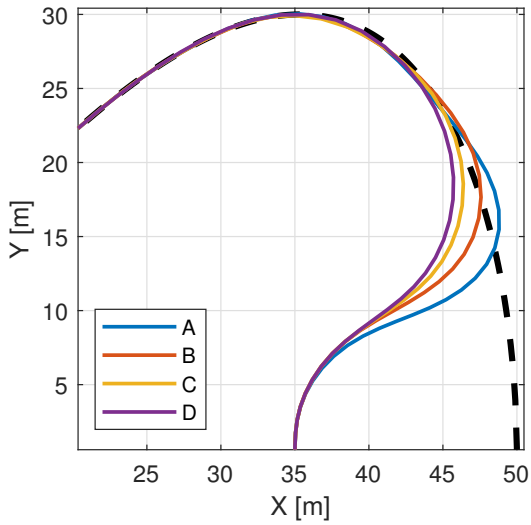


Figure 4.9: Trajectories of differently tuned virtual target speed controllers.

get close, which results in smoother trajectories with less abrupt turns towards the path.

	A	B	C	D
Mean [s]	0.72	0.90	0.98	0.99
Variance [s]	0.15	0.22	0.25	0.24

Table 4.9: Average computation times, per time step, and their variance using different virtual target speed controllers.

Regarding computation times, Table 4.9 shows that, under the same conditions, controlling the speed of the virtual target seems to reduce the computation time by at least 10% when compared to controller B of Section 4.3, which does not control the velocity of the reference.

4.6 Variable Orientation Error Weight

In this experiment, the method presented in Section 3.4 to vary the orientation weight based on y_1 is tested. This method must be seen as a complementary feature to the virtual target speed controller presented in Section 3.3 and tested in Section 4.5.

The experiment of Section 4.5 shows that decreasing the value of λ in the controller makes the virtual target "stop" closer to the vehicle and thus allows the vehicle to turn more abruptly towards the path. However, it is also shown that decreasing λ eventually leads to overshoots in the vehicle trajectory. In Section 3.4, it is suggested to vary the weight of the orientation error based on the normal distance of the vehicle to the path, in other to shift the focus of the controller to first to reducing y_1 and then, when the vehicle gets close to the path, start orienting the vehicle with the path.

The controller parameters for this experiment are very similar to controller A of Section 4.5 and are presented in Tables 4.10 and 4.11. The initial conditions and vehicle properties are the same as Section 4.2. While controllers A and B follow the same problem formulation of the experiment of Section 4.5, controller C is formulated as follows

$$\underset{\overline{\boldsymbol{x}},\,\overline{\boldsymbol{u}}}{\text{minimize}} \quad \sum_{i=1}^{N} \overline{\boldsymbol{x}}_i^\top \boldsymbol{Q} \overline{\boldsymbol{x}}_i + \overline{\boldsymbol{u}}_i^\top \boldsymbol{R} \overline{\boldsymbol{u}}_i + \alpha \exp\left(\left[\frac{\overline{y_1}}{\beta}\right]^2\right) \overline{\theta}^2 \tag{4.3a}$$

$$\text{subject to} \quad \overline{\boldsymbol{x}}_0 = \boldsymbol{x}_t, \tag{4.3b}$$

$$(2.40) \text{ with } (3.10), \tag{4.3c}$$

$$\overline{\tau_1}, \overline{\tau_2} \in [-100, 100], \tag{4.3d}$$

where the entry corresponding to θ in $\boldsymbol{Q}$ is equal to zero.

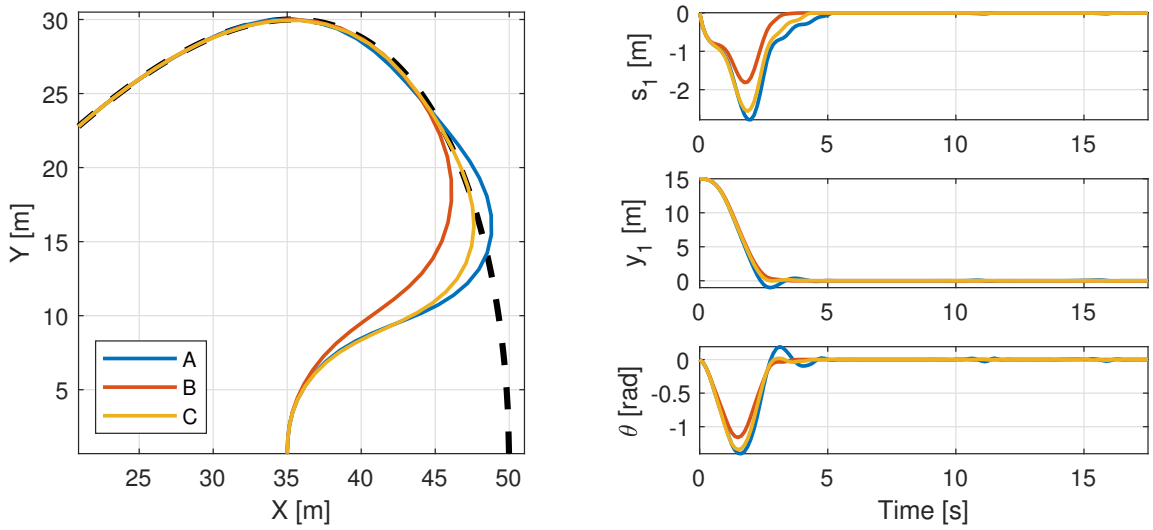
			Weights							Constraints	
			States					Inputs		States	Inputs
Controller	T_s	N	s_1	y_1	θ	v	ω_m	τ_1	τ_2	-	τ_1, τ_2
A	0.125 s	10	1	1	1	0	0	0	0	-	$[-100, 100]$
B	0.125 s	10	1	1	15	0	0	0	0	-	$[-100, 100]$
C	0.125 s	10	1	1	0	0	0	0	0	-	$[-100, 100]$

Table 4.10: Parameters of the different controllers.

Controller	λ	α	β
A	2	-	-
B	2	-	-
C	2	15	3

Table 4.11: Virtual target speed controller and variable θ weight tuning parameters.

Consider Figure 4.10 where the trajectories of the different controllers and the corresponding evolution of the error states is shown. Notice how increasing the weight of the orientation from controller A to B eliminates the overshoot. However, at the same time, the vehicle does not turn so aggressively to the path anymore, therefore, the benefit of using a higher value of λ is lost.



(a) Trajectories

(b) Evolution of the error states over time

Figure 4.10: Resulting trajectories and evolution of error states of the differently tuned controllers.

On the other hand, controller C uses the variable orientation weight method, introduced in Section 3.4, configured according to Table 4.11, where α and β are tuning parameters. The parameter α sets the maximum weight of θ when $y_1 = 0$ and β sets how far from the path the weight of θ starts to grow. The resulting trajectory of controller C shows that the vehicle turns to the path as aggressively as controller A, but as it starts to get close, the weight of θ quickly starts to grow, and thus the controller starts to align the vehicle with the path, preventing the overshoot from happening. The result is a smooth convergence with the path at an earlier point when compared with the other controllers. For the rest of the path the evolution of the error states shows that path following is achieved with slightly better results. Table 4.12 shows that adding a variable orientation weight leads to slightly higher computation times, under the same conditions.

	B	C
Mean [s]	0.81	0.87
Variance [s]	0.14	0.15

Table 4.12: Average computation times, per time step, and their variance using and not using a variable θ weight.

This experiment shows that, if properly configured, dynamically adjusting the weight of θ produces advantageous results, specially when the vehicle gets off track and must converge with the path again, although at the cost of a slight increase in computational load.

4.7 Path Boundaries and Obstacle Avoidance

In this experiment, the implementation of path boundaries and obstacle avoidance with MPC and discussed in Sections 3.2.1 and 3.2.2, respectively, is tested. The controller used is controller C of Section

4.6, whose parameters can be consulted in Tables 4.10 and 4.11. Unlike previous experiments, the vehicle starts on the path with a linear velocity equal to the virtual target reference velocity of 14 m/s.

To test obstacle avoidance, a single static obstacle is placed in $\{I\}$ at coordinates (35,30) with a circular exclusion zone of 2 meters in radius. The results of this experiment can be seen in Figure 4.11(a), where the physical obstacle is represented by the solid red line and the exclusion zone is represented by the dotted red line. The exclusion zone already includes the margin of safety and is the only obstacle constraint implemented in the controller. The problem formulation is the same as that of Section 4.6, with the addition of the obstacle and path constraint, resulting in

$$\underset{\overline{\boldsymbol{x}},\,\overline{\boldsymbol{u}}}{\text{minimize}} \quad \sum_{i=1}^{N} \overline{\boldsymbol{x}}_i^{\top} \boldsymbol{Q} \overline{\boldsymbol{x}}_i + \overline{\boldsymbol{u}}_i^{\top} \boldsymbol{R} \overline{\boldsymbol{u}}_i + \alpha \exp\left(\left[\frac{\overline{y_1}}{\beta}\right]^2\right) \overline{\theta}^2 \tag{4.4a}$$

$$\text{subject to} \quad \overline{\boldsymbol{x}}_0 = \boldsymbol{x}_t, \tag{4.4b}$$

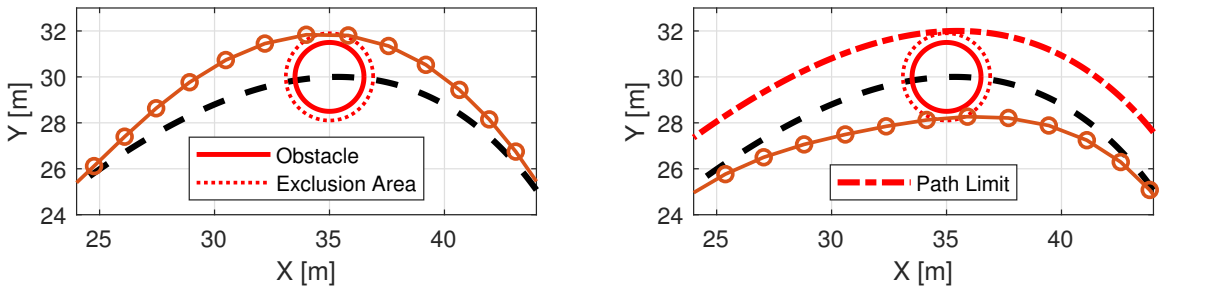
$$(2.40) \text{ with } (3.10), \tag{4.4c}$$

$$\overline{\tau_1}, \overline{\tau_2} \in [-100, 100], \tag{4.4d}$$

$$(\overline{s_1} - C_{s_1})^2 + (\overline{y_1} - C_{y_1})^2 \geq r^2, \tag{4.4e}$$

$$\overline{y_1} \in [l_{lower},\, l_{upper}], \tag{4.4f}$$

where the path constraint 4.4f only applies to one of the controllers.



(a) Obstacle avoidance with no path boundaries

(b) Obstacle avoidance with one path boundary place 2 m from the path

Figure 4.11: Obstacle avoidance with virtual target speed control and variable θ weight.

As seen by the resulting vehicle trajectory in Figure 4.11(a), the controller is successful in avoiding a collision with the obstacle and returning to the path afterwards. Notice also that, although the position of the vehicle is kept outside the exclusion zone every sampling instant, the vehicle actually enters the exclusion zone between sampling instants. This situation is discussed in Section 3.2.2 and demonstrates the importance of having a properly sized margin of safety around the obstacle.

To demonstrate the effect of path boundaries, a path limit is placed 2 meters on the right side of the path, when viewed from a vehicle moving from left to right, like is shown in Figure 4.11(b). The distance between the path and the limit is made equal to the exclusion zone radius to force the vehicle to take an alternate path around the obstacle, when compared to the trajectory of Figure 4.11(a). As expected, the vehicle is shown to take another path around the obstacle in Figure 4.11(b), and thus the

path boundaries are demonstrated to work successfully. On the other hand, because extra constraints are added to the optimization problem, Table 4.13 shows an increase in computation time in obstacle avoidance situations, with or without path limits.

	Obstacle	Obstacle with Path Limits
Mean [s]	0.92	0.93
Variance [s]	0.20	0.20

Table 4.13: Average computation times, per time step, and their variance using obstacle avoidance and path limits.

However, it must be said that, the several experiments performed using path limits and obstacle avoidance together, under similar conditions to those shown in Figure 4.11(b), did not produce reliable and feasible results every time. Furthermore, the feasibility of the problem is very depended on the prediction window of the controller, as expected, and on the size of the obstacle. These are the reasons why, in Section 3.2.2, it is recommended that obstacle avoidance be built into the reference path, and thus not be the job of the controller to deviate from the obstacle.

4.8 Controller Limits

The controller being built over the previous sections has some limitations regarding the initial position and orientation of the vehicle relative to reference. The position of the reference is decoupled from the position of the vehicle, and thus there are initial positions where the controller cannot guarantee convergence with the path. Figure 4.12 shows some of the vehicle trajectories resulting from different initial vehicle positions and orientations, where the reference always starts at position (50, 0). The controller used is controller C of Section 4.6, with all conditions not mentioned remaining the same.

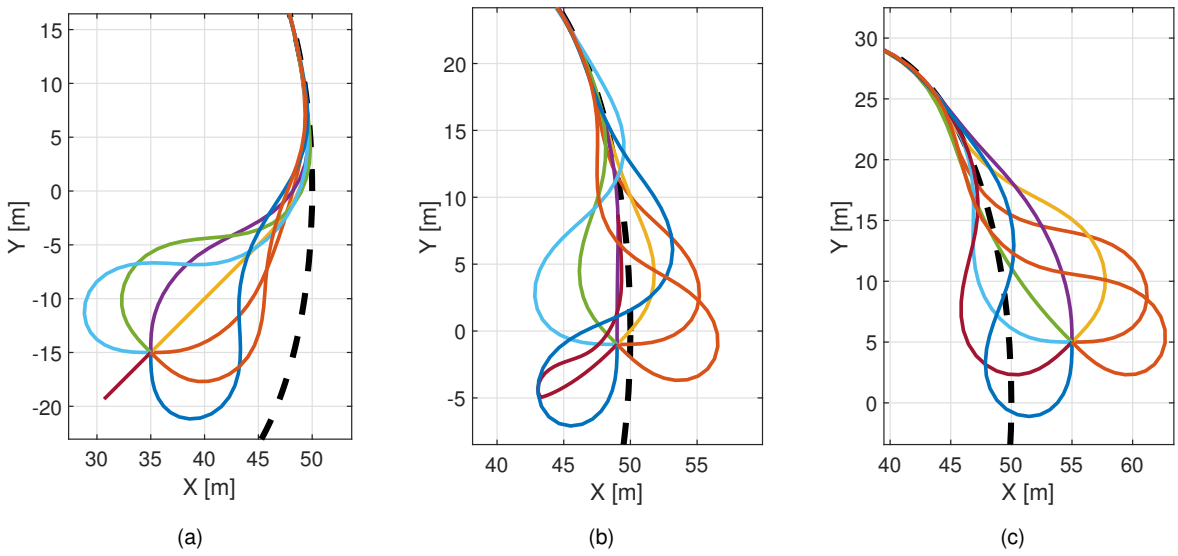


Figure 4.12: Trajectories of the vehicle starting with different positions and orientations relatively to the reference.

Notice in Figure 4.12(a) how the controller is not capable to steer the vehicle towards the path, and actually stops it, for θ_m equal to -135° in $\{I\}$. For the same orientation, Figure 4.12(b) shows the vehicle actually converging with the path, although the trajectory still shows difficulties in doing so.

Another limitation lies in the speed profile used for the virtual target. The exponential function works well if the reference starts behind or close in front the reference, like in shown in Figure Figure 4.12(c), because the controller can always keep the reference close in front of the vehicle. But, if the vehicle starts too far in front of the reference, the velocity of the reference can reach very high values, which can lead to the reference not being anywhere near the vehicle in the following time step. This problem can be solved by limiting the velocity of the reference to a maximum value, or by using a lower and upper bounded function to profile the speed of the velocity based on s_1. That being said, the situation just described is avoided if the initial tangent position s_1 of the vehicle is behind or on top of the reference.

So far, the reference is always set to target a 14 m/s velocity over the path. Figure 4.13 shows the trajectory of the vehicle when the vehicle and the reference are traveling at a speed of 28 m/s (100.8 km/h). Despite doubling the velocity, the controller is capable of controlling the vehicle to follow the path, while keeping the normal error below 0.5 meters. That being said, this simulation does not portray the reality very well. The tight curves have an average radius of 10 meters and Formula 1 cars, which are optimized to maximize the velocity in corners, can only take similar curves, like the Fairmount Hairpin in the Mónaco Grand Prix, at about 50 km/h. Thus, at higher velocities, this work starts to be limited by the unicycle dynamic model used, and not so much by the controller.

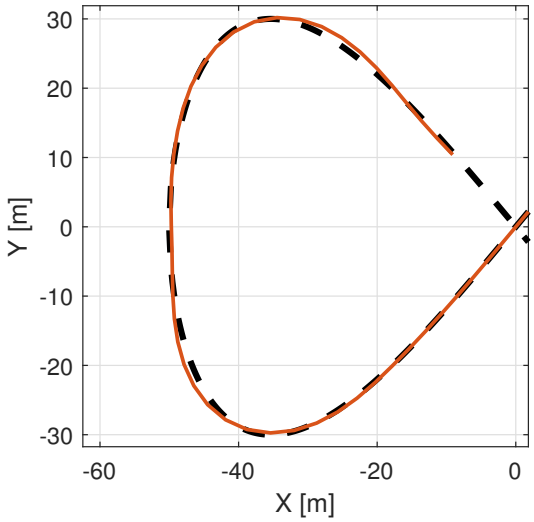


Figure 4.13: Trajectory of the vehicle at a reference velocity of 28 m/s.

Similarly, when studying the effect that the controller sampling period has on the performance of the controller in tight curves, Figure 4.14 shows that doubling the sampling period (from 0.125 seconds) does not produce worse overall results, despite a noticeable small increase in difficulty to keep the vehicle on the path for 0.25 seconds when taking smaller radii curves. The curvature radius of the semicircles present in Figure 4.14 are 12.5, 10, 7.5 meters and the vehicle travels at 14 m/s when entering each curve (from the bottom right of the figure). The results shown in Figure 4.14 further highlight that the

vehicle model used (2.40) is the limiting factor at this point, since curves with such small radii can hardly be taken at 50 km/h, and thus the results shown might not portray the reality closely. For this reason, a future step of this work is to use a more realistic vehicle model that considers more dynamics not accounted for in this work.

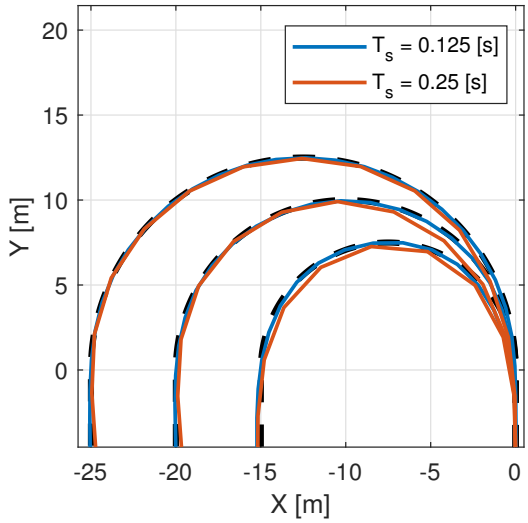


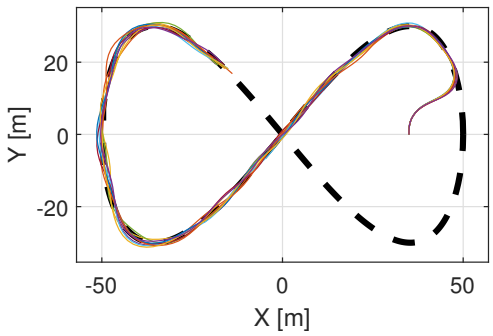
Figure 4.14: Trajectory of the vehicle at a reference velocity of 14 m/s for different curvature radii and controller sampling periods.

4.9 Robustness Experiments

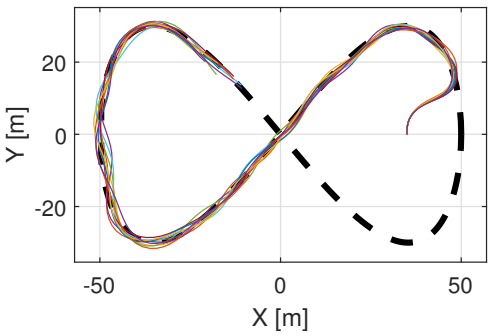
The previous experiments are performed under perfect conditions, with no perturbations or uncertainty regarding the state of the vehicle and its parameters. In this section, some experiments are performed to test the robustness of different controllers. Testing the robustness of a controller covers a vast amount of scenarios and configurations, therefore this section only covers a small amount of tests. The experiments are performed in the same conditions as Section 4.2 with the only variability being the controllers used and the uncertainties.

The first controller tested is controller C of section 4.6, which has virtual target speed control and a variable θ weight, and its parameters can be consulted in Tables 4.10 and 4.11. Because the problem is now stochastic, each experiment is performed ten times. Furthermore, the initial state of the vehicle is always assumed to be known.

Figure 4.15(a) and Figure 4.15(b) show the results of two experiments where zero mean Gaussian noise is added, every sampling step, to the inertial coordinates of the vehicle with a variance of 0.5 m and 1 meters, respectively. Table 4.14 shows the mean and variance of the absolute of the error states s_1, y_1, and θ, after the vehicle converges with the path. Despite the uncertainty on the position of the vehicle, the controller is able to follow the path. Furthermore, as to be expected, the error between the vehicle and the reference also decreases when the uncertainty decreases, and thus the tube of positions around the path is also smaller.



(a) Zero mean noise with variance of 0.5 meters



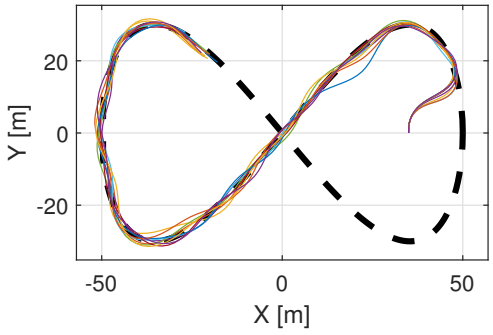
(b) Zero mean noise with variance of 1 meter

Figure 4.15: Trajectories of the dynamic unicycle under uncertainty of the position of the vehicle using a fully featured controller.

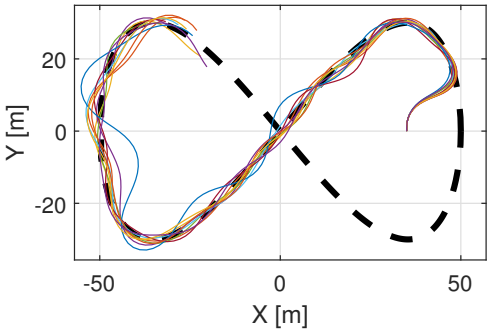
	$\lvert s_1 \rvert$ [m]		$\lvert y_1 \rvert$ [m]		$\lvert \theta \rvert$ [°]	
Figure	Mean	Variance	Mean	Variance	Mean	Variance
4.15(a)	0.81	0.42	0.75	0.31	8.14	15.75
4.15(b)	1.23	1.43	1.06	0.68	10.73	19.40
4.17(a)	1.04	0.62	1.07	0.57	9.69	17.96
4.16(a)	0.08	0.01	0.72	0.43	12.74	23.60
4.16(b)	0.15	0.02	1.20	1.56	16.50	22.19
4.17(b)	0.87	1.34	1.14	0.98	16.90	26.85

Table 4.14: Mean and variance of s_1, y_1 and θ for different simulations.

Using the same controller, Figure 4.16(a) and Figure 4.16(b) show the results of two experiments where zero mean Gaussian noise is added, every sampling step, to the orientation of vehicle θ_m with a variance of 0.5° and 1° , respectively. The state errors are characterized in Table 4.14. Notice how a uncertainty of 1° in the orientation does not produce significant errors in s_1, but actually causes worse normal tracking than an 1 m uncertainty (in variance) on the position of the vehicle.



(a) Zero mean noise with variance of 0.5°

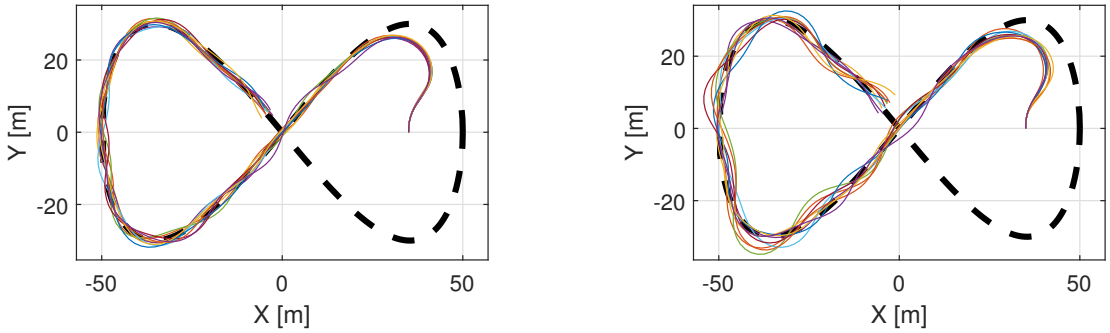


(b) Zero mean noise with variance of 1°

Figure 4.16: Trajectories of the dynamic unicycle under uncertainty of the orientation of the vehicle using a fully featured controller.

To test the impact that controlling the speed of the virtual target and varying the weight of θ have on the robustness of the controller, two experiments are performed using the controller of Section 4.2,

where these features are not present. In the experiment shown in Figure 4.17(a), zero mean Gaussian noise is added, every sampling step, to the inertial coordinates of the vehicle with a variance of 1 m, and, in the experiment of Figure 4.17(b), zero mean Gaussian noise is added with a variance of 1° to the orientation of the vehicle in $\{I\}$. The error states characterized in Table 4.14. It can be seen that the new features do not seam to negatively impact the robustness of the controller, which is desirable, although more simulations must be performed to take any definitive conclusion. The main noticeable difference is that the fully featured controller becomes more robust in s_1 when the velocity of the reference is controlled.



(a) Zero mean noise with variance of 1 meter

(b) Zero mean noise with variance of 1°

Figure 4.17: Trajectories of the dynamic unicycle under uncertainty of the position or orientation of the vehicle using a featureless controller.

	With Features	Without Features
Mean [s]	2.09	1.98
Variance [s]	0.37	0.29

Table 4.15: Average computation times, per time step, and their variance under uncertainty.

Table 4.15 shows the computation times of two controllers, one using reference velocity control and variable θ weight, and another not using these features. For the first controller, the simulations averaged correspond to those of Figure 4.15(b) and Figure 4.16(b), and, for the second controller, the simulations of Figure 4.17 are used. The average computation time is very similar between the two, but is much higher when compared to those seen under perfect conditions, which is something to take into consideration in future works related to this work.

Overall, the controller, with or without features, proves to be robust to big uncertainties in key states, such as the position of the vehicle and its orientation in $\{I\}$. It is to be expected the vehicle to perform better as these uncertainty values become more inline with those of the real world. For example, nowadays GPS can be augmented to provide real-time positioning with centimeter level accuracy. That being said, the tests performed in this section are still very limited by the small amount of simulations performed in too few scenarios, and thus more work still has to be done in this area.

Chapter 5

Conclusions

The main goal of developing a controller, using Model Predictive Control, to control an autonomous racing vehicle to follow a reference path is achieved in this work. Using as the foundation the work developed in [12], by parametrizing the path with respect to the arc-length, the path-following problem can be formulated in a Frenet-Serret frame that follows a virtual moving reference over the path. Consequently, the problem formulation becomes simpler and more intuitive, since the vehicle model in the Frenet-Serret frame also corresponds to the error model of the problem. It is remarked that, while [12] proposes an approach to design based on control Lyapunov functions, here, instead, model predictive control is used. The inspiration from [12] stems mainly from the use of a Frenet-Serret referential to reduce the tracking problem to a regulation problem, in which the vehicle state is driven to zero.

The path parametrization also introduces the ability to control the velocity at which the virtual reference moves along the path. This ability is explored in this work to make the virtual target stay relatively close to vehicle while it tries to move at a certain reference velocity. The proposed method is to profile the virtual target speed as an exponential function of the tangential position of the vehicle in the Frenet-Serret frame. This formulation allows the reference to speed up or slow down if the vehicle is in front or lagging behind the reference, respectively.

Developing the controller using Model Predictive Control allows constraints to be placed on both the state and control variables. This feature provides a big advantage relatively to other more traditional control methods and is successfully explored in this work to implement path boundaries and basic obstacle avoidance.

MPC also allows the tuning of the importance of each optimization variable, which is explored to implement a method that dynamically changes the weight of the orientation error of the vehicle based on its normal distance to the reference. This method increases the weight on the orientation error as the vehicle gets closer to the path, and thus allows the vehicle to first reduce the normal distance to the path and then, in a later stage, align the vehicle with the path for a smoother convergence.

In this work, several tests are also performed to demonstrate the impact that different tuning parameters of the controller have on the resulting trajectory of the vehicle. Furthermore, the robustness of the controller is also assessed by introducing uncertainty in the position of the vehicle and its orientation.

Although the scenarios tested are few, the controller proved to be robust for big uncertainty values in key state variables.

Although is not the intent of this work to prove computational feasibility, a simple computation time analysis is performed to demonstrate the impact that different parameters and addeds feature have on the temporal performance of the controller. Although the test environment is MATLAB, the computation times obtained showcase the biggest disadvantage of MPC, which is the high associated computational load, since all but one of the controllers tested proved to be unfeasible in real-time.

5.1 Future Work

This work highlights the potential of using Model Predictive Control in a path-following problem under the Frenet-Serret formulation, and thus here are some of the possible future steps to achieve the full potential of this idea:

- Perform a more comprehensive controller robustness test that includes actuator errors, perturbations, and more uncertainties in other state variables.
- Improve the obstacle avoidance formulation to include moving obstacle, as well as other methods to define the exclusion area of the obstacle.
- Utilize other more complete and realistic vehicle models, such as the dynamic model of the bicycle, that include tire grip models, motor dynamics, and other vehicle dynamics not accounted for in this work.
- Convert MATLAB to compiled code, which usually equates to 10-100x improvement factor in computation times, and implement numerical acceleration techniques to allow the algorithm to be run in real time.
- Use MPC, or other method, to generate the reference path given the path boundaries, or equivalent, and the obstacles on the track.
- Implement the MPC controller with real hardware in an autonomous vehicle.

Bibliography

[1] D. Hrovat, S. Di Cairano, H. E. Tseng, and I. V. Kolmanovsky. The development of model predictive control in automotive industry: A survey. In *2012 IEEE International Conference on Control Applications*, pages 295–302, 2012. doi: 10.1109/CCA.2012.6402735.

[2] P. F. Lima, M. Trincavelli, M. Nilsson, J. Mårtensson, and B. Wahlberg. Experimental evaluation of economic model predictive control for an autonomous truck. In *2016 IEEE Intelligent Vehicles Symposium (IV)*, pages 710–715, 2016. doi: 10.1109/IVS.2016.7535465.

[3] P. Lima. *Optimization-Based Motion Planning and Model Predictive Control for Autonomous Driving: With Experimental Evaluation on a Heavy-Duty Construction Truck*. PhD thesis, KTH Royal Institute of Technology, September 2018.

[4] Dow, Jameson. Roborace debuts their driverless "Robocar" on track at the Paris ePrix, May 2017. URL `https://electrek.co/2017/05/20/roborace-debuts-their-driverless-robocar-on-track-at-the-paris-eprix/`. [Online; accessed 30-October-2020].

[5] Wollman, Dana. Formula E is planning the first racing series for driverless cars, November 2020. URL `https://en.wikipedia.org/w/index.php?title=Roborace&oldid=969242126`. [Online; accessed 30-October-2020].

[6] Wikipedia contributors. Roborace — Wikipedia, the free encyclopedia, 2020. URL `https://en.wikipedia.org/w/index.php?title=Roborace&oldid=969242126`. [Online; accessed 30-October-2020].

[7] FIA Formula E. Formula E and Kinetik announce driverless support series, November 2015. URL `https://www.fiaformulae.com/en/news/2015/november/formula-e-kinetik-announce-roborace-a-global-driverless-championship.aspx`. [Online; accessed 30-October-2020].

[8] Kelion, Leo. Driverless Roborace car crashes at speed in Buenos Aires, February 2017. URL `https://www.bbc.com/news/technology-39027477`. [Online; accessed 30-October-2020].

[9] Suggitt, Connie. Robocar: Watch the world's fastest autonomous car reach its record-breaking 282 km/h, October 2019. URL `https://www.guinnessworldrecords.com/news/2019/10/`

robocar-watch-the-worlds-fastest-autonomous-car-reach-its-record-breaking-282-k. [Online; accessed 30-October-2020].

[10] K. Berntorp, T. Hoang, R. Quirynen, and S. Di Cairano. Control architecture design for autonomous vehicles. In *2018 IEEE Conference on Control Technology and Applications (CCTA)*, pages 404–411, 2018.

[11] C. Samson. Path following and time-varying feedback stabilization of a wheeled mobile robot. *Second International Conference on Automation, Robotics and Computer Vision*, 3, 01 1992.

[12] A. Micaelli and C. Samson. Trajectory tracking for two-steering-wheels mobile robots. *IFAC Proceedings Volumes*, 27(14):249 – 256, 1994. ISSN 1474-6670. doi: https://doi.org/10.1016/S1474-6670(17)47322-8. Fourth IFAC Symposium on Robot Control, Capri, Italy, September 19-21, 1994.

[13] D. Soetanto, L. Lapierre, and A. Pascoal. Adaptive, non-singular path-following control of dynamic wheeled robots. In *42nd IEEE International Conference on Decision and Control (IEEE Cat. No.03CH37475)*, volume 2, pages 1765–1770 Vol.2, 2003.

[14] K. Tapp. *Differential Geometry of Curves and Surfaces*. Springer International Publishing, 1st edition edition, 2016.

[15] M. Behrendt. File:mpc scheme basic.svg — wikimedia commons, the free media repository, 2020. URL https://commons.wikimedia.org/w/index.php?title=File:MPC_scheme_basic.svg&oldid=465567294. [Online; accessed 3-November-2020].

[16] J. Rawlings, D. Mayne, and M. Diehl. *Model Predictive Control: Theory, Computation, and Design*. Nob Hill Publishing, 2nd edition edition, 01 2017.