MPC Motion Control of an Autonomous Car

Tomás Carneiro
tomasccarneiro@tecnico.ulisboa.pt
Instituto Superior Técnico, Universidade de Lisboa, Portugal

January 2021

Abstract

This work addresses the design of a controller using Model Predictive Control (MPC) to steer an autonomous vehicle along a racing track. The objective is to control the vehicle to follow a virtual reference that is moving with a certain velocity over a reference path. 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. The velocity of the reference is imposed externally, and thus corresponds to an extra controller design parameter.

The vehicle model in the Frenet-Serret frame is then used with MPC to control the vehicle. This work discusses the implementation of obstacle avoidance and path boundaries under the current MPC formulation. In addition, it presents one method to control the velocity of the virtual target based on the distance between the vehicle and the virtual target and another method to dynamically change the focus the controller puts on correcting the orientation of the vehicle based on its orthogonal distance to the path.

1. Introduction

This work focuses on controlling an autonomous racing vehicle with MPC to follow a given path with a maximum reference velocity. The main advantage MPC has 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. However, the main disadvantage of MPC is the usually high associated computational loads.

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 reference path has received less attention. To the author's knowledge, [5] 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 [3] 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 [6], 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 [3]. The controller design relies on Lyapunov function methods.

Following the same problem formulation of [6], in [2] the author presents a MPC 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 the path. The MPC formulation presented also incorporates other constraints such as keeping the vehicle inside the road boundaries and enforcing handling limits.

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.

This work follows the following structure. In Section 2, a theoretical overview regarding the path parametrization and Frenet-Serret frame is provided. A brief derivation of the generic vehicle model in the Frenet-Serret frame is also present, along with an example with the unicycle dynamic model. Section 3 provides an introduction to MPC theory and discusses the implementation of different controller features under the current problem and MPC formulation. Finally, Section 4 includes several experiments where the incremental incorporation of the different features to the MPC controller is gradually tested.

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.

2.1. *The Frenet-Serret Frame and Path Parametrization*

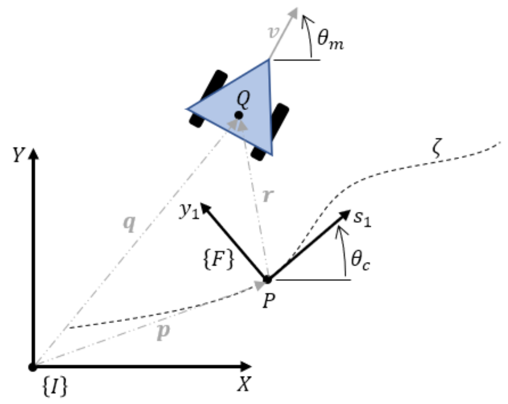


Figure 1: Vehicle geometry and frame definitions

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 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_m and θ_c designate the orientation of the vehicle in $\{I\}$ and the angle that the positive tangent axis of $\{F\}$ makes with the horizontal axis of $\{I\}$, respectively.

Define now point Q as the center of mass of a vehicle that tracks point P on the curve. Point Q can either be defined in $\{I\}$ by vector $\boldsymbol{q} = [X\ Y\ 0]^\top$ or in $\{F\}$ by vector $\boldsymbol{r} = [s_1\ y_1\ 0]^\top$. It is remarked that, the vehicle position error in $\{I\}$ corresponds to the position of the vehicle in $\{F\}$.

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

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

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

Similarly, the orientation of the vehicle in $\{F\}$ is also the orientation error in $\{I\}$, and is given by

$$\theta = \theta_m - \theta_c. \tag{2}$$

Since P is the target position of the vehicle, by making P move along the path, the vehicle is forced to follow P, and thus P acts like a virtual target 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.

The fact that P only moves along the path motivates the introduction of a path parametrization with respect to its arc-length, which here is denoted by 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

$$\left(\frac{d\boldsymbol{p}}{dt}\right)_F = \begin{bmatrix}\dot{s} & 0 & 0\end{bmatrix}^T, \tag{3}$$

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.

As P moves along the path, the orientation of $\{F\}$, given by θ_c, also changes. Denoting $\kappa = \kappa(s)$ as the signed curvature of the path at point $P = P(s)$, the rate of change of θ_c is influenced by the velocity of P and by the curvature of the path κ. It can be deduced using the Frenet-Serret equations that the model for θ_c is given by

$$\dot{\theta}_c = \kappa\,\dot{s} = \omega_c, \tag{4}$$

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

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

The velocity of Q can be given in $\{F\}$ by

$$\left(\frac{d\boldsymbol{q}}{dt}\right)_I = \left(\frac{d\boldsymbol{p}}{dt}\right)_I + \boldsymbol{R}^{-1}\left(\frac{d\boldsymbol{r}}{dt}\right)_F + \boldsymbol{R}^{-1}\left(\boldsymbol{\Omega}\times\boldsymbol{r}\right), \tag{5}$$

where $\boldsymbol{\Omega} = \begin{bmatrix}0 & 0 & \omega_c\end{bmatrix}^T$ denotes the angular velocity vector of $\{F\}$. Multiplying the equation above by $\boldsymbol{R}$ yields

$$\boldsymbol{R}\left(\frac{d\boldsymbol{q}}{dt}\right)_I = \left(\frac{d\boldsymbol{p}}{dt}\right)_F + \left(\frac{d\boldsymbol{r}}{dt}\right)_F + \boldsymbol{\Omega}\times\boldsymbol{r}, \tag{6}$$

which gives the velocity of the vehicle in $\{F\}$.

Replacing

$$\left(\frac{d\boldsymbol{q}}{dt}\right)_I = \begin{bmatrix}\dot{X} & \dot{Y} & 0\end{bmatrix}^T, \left(\frac{d\boldsymbol{r}}{dt}\right)_F = \begin{bmatrix}\dot{s}_1 & \dot{y}_1 & 0\end{bmatrix}^T, \tag{7}$$

and

$$\boldsymbol{\Omega} \times \boldsymbol{r} = \begin{bmatrix} 0 \\ 0 \\ \omega_c = \kappa\,\dot{s} \end{bmatrix} \times \begin{bmatrix} s_1 \\ y_1 \\ 0 \end{bmatrix} = \begin{bmatrix} -\kappa\,\dot{s}\,y_1 \\ \kappa\,\dot{s}\,s_1 \\ 0 \end{bmatrix} \tag{8}$$

in (6), solving for $\dot{s}_1$ and $\dot{y}_1$, and augmenting it with $\dot{\theta} = \dot{\theta}_m - \dot{\theta}_c$ yields

$$\begin{cases} \dot{s}_1 = \begin{bmatrix} cos\,\theta_c & sin\,\theta_c \end{bmatrix} \begin{bmatrix} \dot{X} \\ \dot{Y} \end{bmatrix} - \dot{s}\,(1 - \kappa\,y_1) \\ \dot{y}_1 = \begin{bmatrix} -sin\,\theta_c & cos\,\theta_c \end{bmatrix} \begin{bmatrix} \dot{X} \\ \dot{Y} \end{bmatrix} - \dot{s}\,\kappa\,s_1 \\ \dot{\theta} = \dot{\theta}_m - \kappa\,\dot{s} \end{cases}, \tag{9}$$

The equations in (9) 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 variables $\dot{X}$, $\dot{Y}$, and $\dot{\theta}_m$ must be replaced by the correspondent inertial model equations of the vehicle. This system of equations can also be augmented to include any other equations relevant to the model or problem.

The relevance of (9) 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.

2.3. *Unicycle Dynamic Model*

The inertial dynamic model of the unicycle is given by the following equations

$$\begin{cases} \dot{X} = v\,cos\,\theta_m \\ \dot{Y} = v\,sin\,\theta_m \\ \dot{\theta}_m = \omega_m \\ \dot{v} = \dfrac{F}{m} \\ \dot{\omega}_m = \dfrac{N}{I} \end{cases}, \tag{10}$$

where the state variables X and Y denote the position of the vehicle, θ_m its orientation, v its linear velocity, ω_m its angular velocity, and F and N are given by

$$\begin{cases} F = \dfrac{\tau_1 + \tau_2}{R} \\ N = \dfrac{\tau_1 - \tau_2}{R}\,L \end{cases}, \tag{11}$$

where m is the mass of the unicycle, I its moment of inertia, R the radius of the wheels, L is half the length of the axis that connects the centers of the two wheels, and τ_1 and τ_2 are the torques applied to each wheel.

The dynamic model of the unicycle in the moving Frenet-Serret is obtained by substituting (10) in (9), yielding

$$\begin{cases} \dot{s}_1 = -\dot{s}\,(1 - \kappa\,y_1) + v\,cos\,\theta \\ \dot{y}_1 = -\kappa\,\dot{s}\,s_1 + v\,sin\,\theta \\ \dot{\theta} = \omega_m - \kappa\,\dot{s} \\ \dot{v} = \dfrac{1}{m}\dfrac{\tau_1 + \tau_2}{R} \\ \dot{\omega}_m = \dfrac{L}{I}\dfrac{\tau_1 - \tau_2}{R} \end{cases}. \tag{12}$$

The state variables of the unicycle dynamic model in $\{F\}$ are $\begin{bmatrix} s_1 & y_1 & \theta & v & \omega_m \end{bmatrix}^T$ and the control inputs are $\begin{bmatrix} \tau_1 & \tau_2 & \dot{s} \end{bmatrix}^T$.

3. Motion Control with MPC

3.1. *Model Predictive Control*

Model Predictive Control (MPC)[4] is a control method that makes explicit use of a system model to predict its future outputs and states over a finite horizon, and, based upon that knowledge, solve an online optimization problem to obtain a control decision.

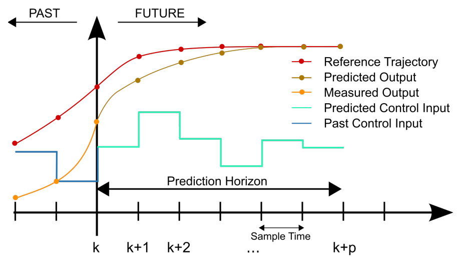


Figure 2: Model Predictive Control strategy. Available at [1].

Figure 2 illustrates the typical MPC control strategy. At each sampling instant, the optimization problem is solved by minimizing a cost function within a finite prediction horizon, while considering the latest available state measurement or estimate as the initial condition. The solution to this 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, the problem is repeated with the new available system state and with the prediction horizon shifted towards the next sampling instant. The shifting forward of the prediction horizon is also known as the receding or moving horizon strategy.

The formulation as an optimization problem 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. Furthermore, MPC can handle tightly coupled multivariate systems, even if these systems are non-linear (Non-Linear MPC or NMPC). The main disadvantage of MPC is the high computational load, since it has to solve an online open-loop iterative optimization problem for every sampling step.

3.1.1 Model Predictive Control Formulation

For a system whose state variables and control inputs are denoted by $\boldsymbol{x}$ and $\boldsymbol{u}$, respectively, the MPC optimization problem can be formulated, for discrete-time invariant systems, as follows

$$\begin{aligned}
\underset{\overline{\boldsymbol{x}}, \overline{\boldsymbol{x}}}{\text{minimize}} \quad & J(\overline{\boldsymbol{x}}, \overline{\boldsymbol{x}}) = \sum_{i=1}^{N} \ell(\overline{\boldsymbol{x}}_i, \overline{\boldsymbol{u}}_i) && \text{(13a)}\\
\text{subject to} \quad & \overline{\boldsymbol{x}}_0 = \boldsymbol{x}_t \ , && \text{(13b)}\\
& \overline{\boldsymbol{x}}_i = f(\overline{\boldsymbol{x}}_{i-1}, \overline{\boldsymbol{u}}_{i-1}), && \text{(13c)}\\
& \overline{\boldsymbol{x}}_i \in \boldsymbol{\mathcal{X}}_i, \ i = 1, \ldots, N, && \text{(13d)}\\
& \overline{\boldsymbol{u}}_i \in \boldsymbol{\mathcal{U}}_i, \ i = 1, \ldots, N-1, && \text{(13e)}
\end{aligned}$$

where $\overline{\boldsymbol{x}}$ and $\overline{\boldsymbol{u}}$ represent the predictions of $\boldsymbol{x}$ and $\boldsymbol{u}$, N denotes the length of the prediction horizon, and $\ell(.)$ is the instantaneous cost function to be minimized. The optimization problem is subject to the constraints (13b) to (13e), where constraint (13b) imposes the initial condition on the state ($\boldsymbol{x}_t$ represents the state at time t), and constraint (13c) imposes the system model. The state is also subject to a set of constraints represented by $\boldsymbol{\mathcal{X}}$ in (13d), and, likewise, the control inputs are subject to the constraints represented by $\boldsymbol{\mathcal{U}}$ in (13e).

3.1.2 Linear 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 linear quadratic cost function

$$\ell(\boldsymbol{x}, \boldsymbol{u}) = \boldsymbol{x}^\top \boldsymbol{Q} \boldsymbol{x} + \boldsymbol{u}^\top \boldsymbol{R} \boldsymbol{u}, \tag{14}$$

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 where each diagonal entry is the weight attributed to the corresponding state or input variable. The weights in $\boldsymbol{Q}$ are used to set the relative importance between each state variable. Furthermore, increasing a value of $\boldsymbol{R}$ relatively to $\boldsymbol{Q}$ penalizes the control action of the corresponding control variable.

A MPC controller using the cost function (14) is known as a "linear quadratic regulator" (LQR), and it controls the system to the origin since it directly penalizes non-zero states and control inputs .

3.1.3 Reference Tracking

Controlling the system towards a reference state $\boldsymbol{x}_{ref}$ other than zero can be achieved with the cost function

$$\ell(\boldsymbol{x}, \boldsymbol{u}) = (\boldsymbol{x} - \boldsymbol{x}_{ref})^\top \boldsymbol{Q} (\boldsymbol{x} - \boldsymbol{x}_{ref}) + \boldsymbol{u}^\top \boldsymbol{R} \boldsymbol{u}, \tag{15}$$

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 (15), 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 (15) that allows $\boldsymbol{u}$ to be non-zero in steady state. The resulting cost function is

$$\begin{aligned}
\ell(\boldsymbol{x}, \boldsymbol{u}) = {} & (\boldsymbol{x} - \boldsymbol{x}_{ref})^\top \boldsymbol{Q} (\boldsymbol{x} - \boldsymbol{x}_{ref}) \\
& + (\boldsymbol{u} - \boldsymbol{u}_{ref})^\top \boldsymbol{R} (\boldsymbol{u} - \boldsymbol{u}_{ref}) \, , \qquad (16)
\end{aligned}$$

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

3.2. *Path Boundaries*

For every point of the path, a boundary is defined as the normal distance to the path beyond which the vehicle is not allowed to go. 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 enforced via inequality constraints on the state y_1 as

$$l_{lower} \leq y_1 \leq l_{upper} \ , \tag{17}$$

where l_{lower} and l_{upper} denote the lower and upper limits of the path, with $l_{lower} < l_{upper}$.

However, this formulation can create a problem when $s_1 \neq 0$, since the virtual target is not the closest point of the path to the vehicle in this situation. It can happen, especially in curves, that the path boundaries do not accurately portrait the limits of the path in the vicinity of the vehicle, as depicted in Figure 3.

The proposed solution is to increase the focus the controller puts on making $s_1 \approx 0$, since the errors in the path boundary constraints decrease as s_1 tends to zero. Therefore, when $s_1 \approx 0$, the boundary conditions defined by (17) approximately match the path limits when viewed from the vehicle position.

To increase the importance of s_1, the relative weight of s_1 is increased in the cost function. An alternative method to keep the virtual target and the vehicle close together is presented in Section 3.4.

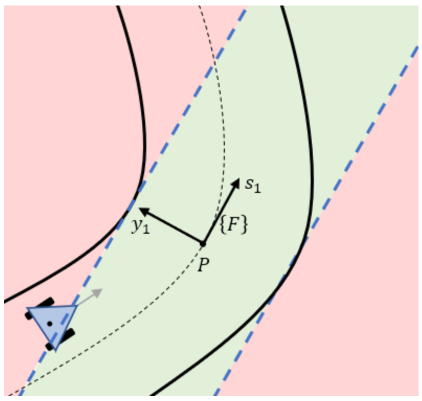


Figure 3: Path boundaries error

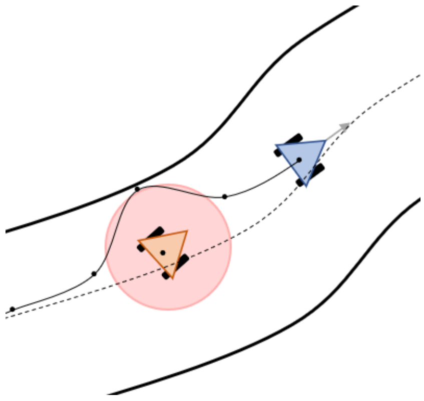


Figure 4: Obstacle Avoidance Violation

3.3. *Obstacle Avoidance*

Obstacle avoidance can be achieved by restricting the region of positions where the vehicle is allowed to be.

By knowing the coordinates of the center of the obstacle in $\{F\}$, an exclusion zone around the obstacle can be defined by a circle centered on the obstacle, with a radius large enough to encompass the entire obstacle or the vehicle, whichever is bigger. Then, the obstacle exclusion zone can be enforced with the following nonlinear inequality constraint

$$(s_1 - C_{s_1})^2 + (y_1 - C_{y_1})^2 \geq r^2, \qquad (18)$$

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.

It is remarked that, as a result of the movement of the virtual target and $\{F\}$, the position of the obstacle, when expressed in $\{F\}$, changes between time instants. Any existent movement of the obstacle itself also contributes to the change of position, although moving obstacles are out of the scope of this work. Since MPC predicts future states to obtain a control decision, the constraint that enforces the exclusion zone must be updated each step of the prediction window with the new position of the obstacle, computed from its fixed inertial position.

Due to the discrete nature of the problem, it can happen that the obstacle constraints are satisfied in each time step, but the vehicle actually violates the constraints between sampling instants. Figure 4 depicts one of these situations.

Two complementary approaches that mitigate this problem are to incorporate a margin of safety into the radius of the exclusion zone and to decrease the sampling period of the controller. If sized properly, the margin of safety allows the vehicle to violate the obstacle constraint without entering the physical area of the obstacle. The downside is that it can only be so big until the problem becomes infeasible in overtake situations. The latter solution also allows the margin of safety to be smaller, but requires more computational power available, therefore a balance of these two solutions must be achieved.

However, it is remarked that, adding obstacle constraints greatly increases the difficulty of the optimization problem, and thus it increases the computational requirements of the controller. For this reason, it is recommended that the obstacle avoidance be built into the path itself, instead of leaving this job to the controller.

3.4. *Virtual Target Speed Control*

The arc-length path parametrization introduced the ability to control the speed at which the virtual target moves along the path via the control variable $\dot{s}$. This ability can be exploited to keep the virtual reference close to the vehicle when the latter is not able to keep up. One advantage of controlling the speed of the virtual target is that it is a virtual reference, *i.e.*, it has no dynamics.

The virtual target must behave in such a manner that when the vehicle is moving close behind, or on top, the virtual target moves at a predefined reference speed, and, when the vehicle starts to lag behind, the virtual target starts slowing down, and stops if necessary. When the vehicle starts regaining ground, the virtual target gradually starts to speed up again. It is also expected that, if the vehicle overtakes the virtual target for some reason, the virtual target must quickly speed up to catch up with the vehicle. The objective is to make the vehicle always be close behind, or on top of, the virtual target, since the latter sets the speed at which the vehicle will move through the path, effectively pulling the vehicle towards it. The reference velocity can be set to a velocity close to the maximum achievable velocity of the vehicle, for example.

In this work, it is proposed to use the state variable s_1 as the variable that sets the speed of the virtual target. The state s_1 partially provides 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. However, information regarding the normal distance of the vehicle to the path is lost, but that does not present a problem because the controller always works to minimize both s_1 and y_1 at the same time, if configured properly.

One candidate function that fulfills all the criteria mentioned above is the exponential function, therefore, the virtual target speed can be given as a function of s_1 by

$$\dot{s}(s_1) = \dot{s}_{ref}\ \exp\left(\frac{s_1}{\lambda}\right), \tag{19}$$

where $\dot{s}_{ref}$ is the reference speed when $s_1 = 0$ and λ is a tuning parameter. Figure 5 shows resulting virtual target speed profiles for different values of λ.

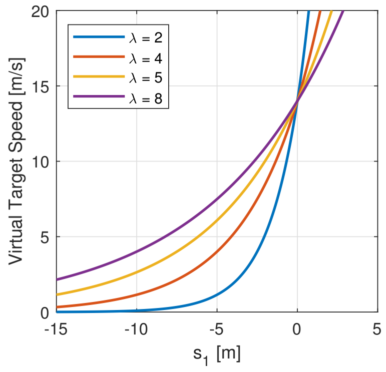


Figure 5: Virtual target speed profiles for a reference speed of 14 m/s.

It is remarked that, the virtual target speed controller can be incorporated into the vehicle model, and thus eliminates one optimization variable from the problem.

3.5. *Variable Orientation Error Weight*

A fixed weight for the orientation error θ in the MPC cost function means that the controller is always trying to correct the orientation of the vehicle, regardless of the vehicle position relatively to the reference. When the vehicle is offtrack, reducing the orientation error is of less importance than reducing the normal distance, moreover, these two actions oppose each other.

By dynamically decreasing the weight of θ as $|y_1|$ increases, the controller is going to prioritize decreasing $|y_1|$ first. As $|y_1|$ gets smaller, *i.e.*, as the vehicle orthogonally approaches the reference, by gradually increasing the weight of θ, the controller is going to start adjusting the orientation of vehicle too.

The proposed method allows the controller to, at a beginner stage, aggressively reduce $|y_1|$ since the weight of θ is very small compared to the weight of y_1. Furthermore, at a final stage, when y_1 is small enough, the gradually increased importance of θ forces the controller to align the vehicle with the path, resulting in a smooth approach.

To achieve this behavior, in this work, it is proposed that the weight of θ follow the following Gaussian function of y_1

$$\rho_\theta(y_1) = \alpha\ \exp\left(\left[\frac{y_1}{\beta}\right]^2\right), \tag{20}$$

where α and β are a tuning parameters. Figure 6 shows the variation of the weight of θ for different values of α and β.

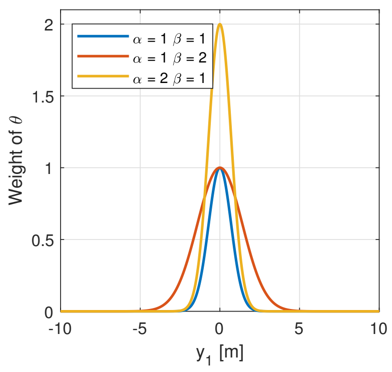


Figure 6: Variation of the θ weight coefficient *vs* y_1

The Gaussian function is considered a suitable candidate function because it presents symmetry relatively to y_1 and is a bounded continuous function. The weight of θ has its maximum α when $|y_1| = 0$, *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 as a function of $|y_1|$, within an interval controlled by β, outside of which it can be considered to be zero.

To incorporate this functionality in the MPC controller, the entry corresponding to θ in the diagonal matrix of fixed weights $\boldsymbol{Q}$ is changed to zero. Furthermore, the cost function

$$\ell_\theta(y_1, \theta) = \rho_\theta(y_1)\,\theta^2, \tag{21}$$

is added to the cost function in (13), resulting in a new cost function

$$\ell'(\boldsymbol{x}, \boldsymbol{u}) = \ell(\boldsymbol{x}, \boldsymbol{u}) + \ell_\theta(y_1, \theta), \tag{22}$$

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

4. Results

The unicycle being considered in the following experiments has a rectangular shape measuring 2.8 by 1.3 meters in length and width, respectively, and the vehicle parameters are shown in Table 1. The unicycle model is defined by equations (12) and 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.

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

Table 1: Unicycle physical parameters.

Unless otherwise stated, the following simulations are obtained with a prediction horizon $N = 10$ and a controller sampling period $T_s = 0.125$ seconds. The unicycle initial conditions are $x_0 = [0\ 15\ 0\ 7\ 0]^\top$, 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 starting at position (35, 0) in $\{I\}$. Also, $\theta = 0°$means that the vehicle starts aligned with the tangential axis of $\{F\}$, and thus the initial orientation of the vehicle is $\theta_m = 90°$. Finally the vehicle also starts with a linear velocity of $v = 7$ m/s and with no angular velocity, *i.e.*, $\omega_m = 0$ rad/s. The initial conditions are kept the same throughout most of the experiments to provide a basis of comparison between experiments.

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.

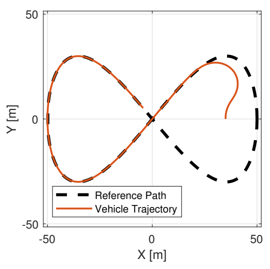
4.1. *Constant Virtual Target Speed*

In this experiment the virtual target moves at a constant velocity equal to 14 m/s (50.4 km/h). The weights assigned to each state and control variables in the MPC cost function are shown in Table 2.

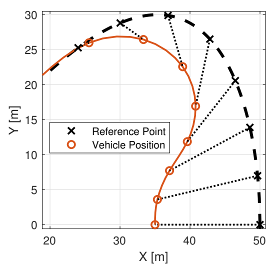
States					Inputs	
s_1	y_1	θ	v	ω_m	τ_1	τ_2
1	1	1	0	0	0	0

Table 2: Controller weights assigned to each variable.

Figure 7 shows that, despite starting with an offset and with half the velocity of the virtual target, the controller is able to drive the vehicle to smoothly converge to the moving reference. From then on, vehicle and reference move together, and thus path following is achieved for the rest of the simulation.



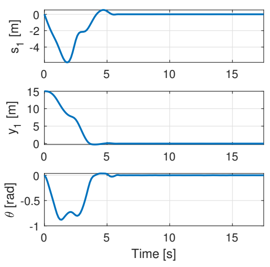
(a) Resulting trajectory.



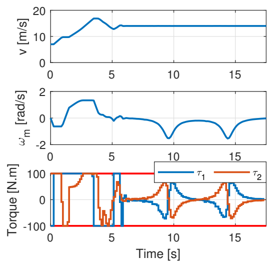
(b) Zoomed-in details of the trajectory spaced every 0.5. s.

Figure 7: Resulting trajectory of the unicycle using a constant virtual target speed of 14 m/s.

The evolution of the state and control variables displayed in Figure 8 shows the position and orientation errors converging to zero as wanted. Furthermore, Figure 8 shows that the vehicle velocity converges to the virtual target constant velocity of 14 m/s, as expected.



(a) States s_1, y_1, and θ.



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

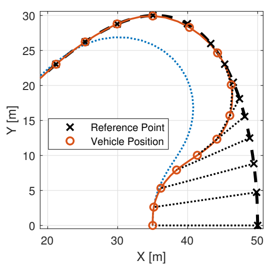
Figure 8: Evolution of the state and control variables over time.

4.2. *Controlled Virtual Target Speed*

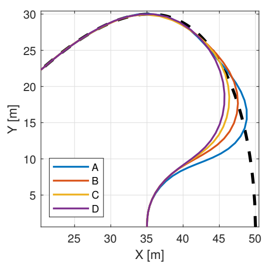
In this experiment, the virtual target speed controller presented in Section 3.4 is tested. The aim of this experiment is to demonstrate the benefits of controlling the virtual target velocity based on the position of the vehicle relatively to the virtual target. To provide a basis of comparison, the initial conditions and controller weights are the same as in the previous experiment.

Regarding the virtual target speed controller, the reference speed is set to the same value of 14 m/s when $s_1 = 0$ for all controllers. The tuning parameter λ, which controls the aggressiveness of the virtual target speed profile, is set to 2, 4, 6, and 8 in controllers A, B, C, and D, respectively.

Figure 9(a) shows the comparison between two trajectories, one that is the same as in the previous experiment obtained using a constant virtual target velocity, and another obtained using controller C which utilizes a virtual target speed controller with $\lambda = 6$. It can be seen that controlling the virtual target speed allows the vehicle to converge to the path much sooner, since the virtual target slows down to "wait" for the vehicle to get close.



(a) Trajectory comparision between constant and controlled virtual target speeds.



(b) Resulting trajectories for diffently tuned virtual target speed controllers.

Figure 9: Resulting trajectories using and not using a controlled virtual target speed.

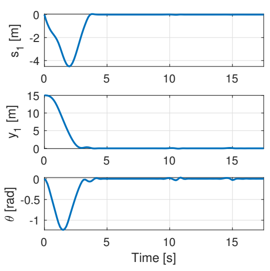
Figure 9(b) shows the comparison between trajectories of four controllers using differently tuned virtual target speed controllers. It can be seen that, as λ increases in value (from controller A to D), the vehicle converges later to the path, since the virtual target speed profile becomes less aggressive, *i.e.*, the change in speed responds more slowly to changes in s_1. Conversely, smaller values of λ allow the vehicle to converge to the path sooner, because the virtual target has lower velocities for smaller negative values of s_1, which allows the virtual target to wait closer in front of the vehicle. On the other hand, it also picks up speed very quickly, which results in overshoots as the virtual target gains velocity very quickly when the vehicle approaches it.

The evolution of the state and control variables displayed in Figure 10 shows the position and orientation errors converging to zero, and also shows the evolution of the virtual target speed. At first, when $s_1 = 0$, the virtual target moves at its reference speed of 14 m/s. As soon as the vehicle starts to lag behind, it gradually slows down until the vehicle stars to recover ground again and picks up speed again, as desired.

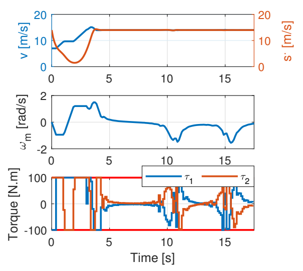
4.3. *Variable Orientation Error Weight*

The previous experiment shows that decreasing λ allows the vehicle to turn to the path more agressively, but it also generates an overshoot in the trajectory. In this experiment, the variable orientation error weight method presented in Section 3.5 is tested as a complementary feature to the virtual target speed controller to achieve better smooth convergences with the path using smaller λ.

In this experiment, three controllers are tested and their parameters and weights are shown in Ta-



(a) States s_1, y_1, and θ.



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

Figure 10: Evolution over time of the state and control variables of controller C ($\lambda = 6$).

ble 3. It is remarked that controller A is the same of the previous experiment.

	Weights*			
Controller	θ	λ	α	β
A	1	2	-	-
B	15	2	-	-
C	0	2	15	3

*The remaining weights remain unchanged and are present in Table 2.

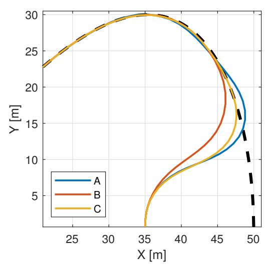
Table 3: Controller weights and parameters.

Figure 11(a) shows that controller C achieves a faster and smoother convergence without overshoots when compared to the other controllers. Increasing the fixed weight of θ in controller B eliminates the overshoot in the trajectory, although it causes the vehicle to converge more slowly to the path, and thus the benefit of using a smaller λ is lost. Controller C, on the other hand, uses the same maximum θ weight, given by α, of controller B, but since this weight decreases with increasing $|y_1|$, it first allows the vehicle to turn more aggressively to the path and, at a later stage when the vehicle is close to the virtual target, the gradually increasing value of θ makes the controller align the vehicle with the path, and thus eliminates the overshoot.

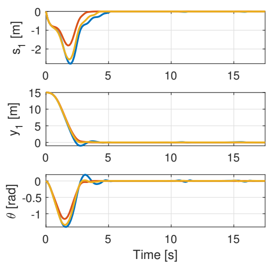
The evolution of the key state variables displayed in Figure 11(b) also shows that the variable θ weight method produces smoother signals throughout the simulation, especially when compared to Controller A.

4.4. *Path Boundaries and Obstacle Avoidance*

In this experiment, the methods proposed to enforce path boundaries and obstacle avoidance in Sections 3.2 and 3.3, respectively, are tested. The controller used is the same as controller C of the previous experiment and, unlike the previous experiments, the vehicle starts aligned with the path and on top of the virtual target with a velocity of 14 m/s.



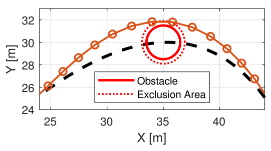
(a) Trajectories.



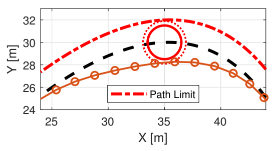
(b) Evolution of the error states over time. states

Figure 11: Resulting trajectories and error states evolution of the differently tuned controllers.

Figure 12(a) shows that, when an obstacle with an exclusion radius of 2 meters is placed on the path, the controller is able to avoid it and return the vehicle to the path afterwards. As discussed in Section 3.3, the vehicle violates the exclusion zone but not the physical area of the object between controller sampling instants, therefore demonstrating the importance of adding a margin of safety to the obstacle.



(a) Obstacle avoidance with no path boundaries.



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

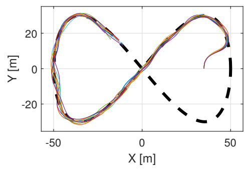
Figure 12: Obstacle avoidance with virtual target speed control and variable θ weight.

Figure 12(b) shows the resulting trajectory of the same vehicle when a single path boundary of 2 meters is placed on the right side of the path, when viewed from a vehicle moving from the right to the left of the figure. The distance between the path and the boundary is made equal to the exclusion zone radius to force the vehicle to take an alternative path around the obstacle, when compared Figure 12(a). As expected, the vehicle is shown to take another path around the obstacle in Figure 12(b), and thus the path boundaries are demonstrated to work successfully.

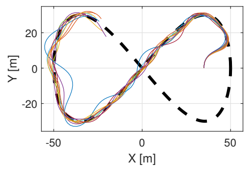
4.5. *Robustness Experiments*

The previous experiments are performed under perfect conditions, where every state, control input and vehicle parameter is assumed to be known. To assess to some degree the robustness of the designed controller, this experiment introduces uncertainty in the position or orientation of the vehicle with respect to an inertial frame, since these states are the ones measured in a real world application. The controller being tested is controller C of Section 4.3, with the initial conditions and vehicle parameters remaining the same. 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.

In the first experiment, zero mean white Gaussian noise with a variance of 1 meter is introduced in the inertial position of the vehicle, given by X and Y. The second experiment introduces zero mean white Gaussian noise with variance of 1 degree into the orientation of the vehicle θ_m. The results of these experiments are presented in Figure 13 and Table 4.



(a) Noise in X and Y with a variance of 1 meter.



(b) Noise in θ_m with variance of $1°$.

Figure 13: Trajectories of the unicycle under uncertainty of the position or orientation of the vehicle using a fully featured controller.

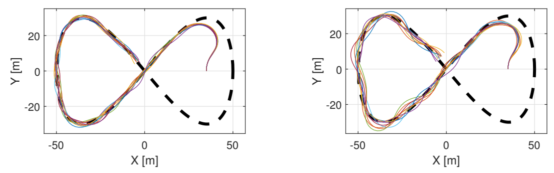
	$\lvert s_1\rvert$ [m]		$\lvert y_1\rvert$ [m]		$\lvert\theta\rvert$[°]	
Fig.	Mean	Var.	Mean	Var.	Mean	Var.
13(a)	1.23	1.43	1.06	0.68	10.73	19.40
13(b)	0.15	0.02	1.20	1.56	16.50	22.19
14(a)	1.04	0.62	1.07	0.57	9.69	17.96
14(b)	0.87	1.34	1.14	0.98	16.90	26.85

Table 4: Mean and variance of s_1, y_1 and θ in the different simulations.

Figure 13 shows that, despite a considerable uncertainty in key states, such as the position or orientation of the vehicle, the controller proves to be robust to this degree of uncertainty in both scenarios, since it is able to follow the path.

Table 4 shows that an uncertainty of 1°in the orientation of the vehicle does not produce significant errors in s_1, although it causes worse orthogonal tracking than a 1 meter uncertainty in the position of the vehicle.

Figure 14 and Table 4 show the results obtained under the same degree of uncertainty using a controller with no added features, such as the controller of Section 4.1. It can be seen that the added 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 under uncertainty on the orientation of the vehicle.



(a) Noise in X and Y with a variance of 1 meter.

(b) Noise in θ_m with variance of 1°.

Figure 14: Trajectories of the unicycle under uncertainty of the position or orientation of the vehicle with a featureless controller.

5. Conclusions

The main objective of developing a controller using Model Predictive Control to drive an autonomous racing vehicle to follow a reference path is achieved in this work. Based on the work developed in [3], 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. The inspiration from [3] 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 ability to control the speed of the virtual target 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 to specify the virtual target speed using an exponential function is shown to produce very positive results.

Designing the controller using Model Predictive Control allows constraints to be placed on both the state and control variables. This ability is successfully explored in this work to implement path boundaries and basic obstacle avoidance.

This work also proposes a method to dynamically change the weight of the orientation error based on its orthogonal distance to the reference by using a Gaussian function of this distance. This method gradually increases the weight on orientation error as the vehicle gets closer to the path, and vice-versa, and thus allows the vehicle to more aggressively reduce the orthogonal distance to the path at an initial stage, while at a later stage forces the controller to align the vehicle with the path for a smooth convergence. This method is shown to produce positive results.

Finally, some robustness experiments are also performed to test the controller by introducing uncertainty in the vehicle position or orientation. Despite considerable uncertainties, the controller proves to be robust, although the scenarios tested are limited, and thus a more comprehensive battery of robustness tests must be performed in future work.

Future steps of this work include designing the controller utilizing a more realistic and complete vehicle model, that accounts for additional dynamics, such as tire grip and motor dynamics. Other steps are to use MPC to generate the path the vehicle will follow and expand on the obstacle avoidance formulation. Finally, a fundamental step is to test the controller with real hardware, running the controller in real time to control a test vehicle.

Acknowledgements

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.

References

[1] M. Behrendt. File:mpc scheme basic.svg — wikimedia commons, the free media repository, 2020. [Online; accessed 3-November-2020].

[2] 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.

[3] A. Micaelli and C. Samson. Trajectory tracking for two-steering-wheels mobile robots. *IFAC Proceedings Volumes*, 27(14):249 – 256, 1994. Fourth IFAC Symposium on Robot Control, Capri, Italy, September 19-21, 1994.

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

[5] 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.

[6] 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.