
DEGREE PROJECT IN INFORMATION AND COMMUNICATION TECHNOLOGY,
SECOND CYCLE, 30 CREDITS
*STOCKHOLM, SWEDEN 2018*

# Autonomous racing using model predictive control

**FLORIAN CURINGA**

**KTH ROYAL INSTITUTE OF TECHNOLOGY**
**SCHOOL OF ELECTRICAL ENGINEERING**

# Autonomous racing using model predictive control

Master thesis report

Curinga, Florian
curinga@kth.se

January 7, 2018

## Abstract

Autonomous vehicles are expected to have a significant impact on our societies by freeing humans from the driving task, and thus eliminating the human factor in one of the most dangerous places: roads. As a matter of facts, road kills are one of the largest source of human deaths and many countries put the decrease of these casualties as one of their top priorities. It is expected that autonomous vehicles will dramatically help in achieving that. Moreover, using controllers to optimize both the car behaviour on the road and higher level traffic management could reduce traffic jams and increase the commuting speed overall.
To minimize road kills, an approach is to design controllers that would handle the car at its limits of handling, by integrating complex dynamics such as adherence loss it is possible to prevent the car from leaving the road. A convenient setup to evaluate this type of controllers is a racing context: a controller is steering a car to complete a track as fast as possible without leaving the road and by brining the car to its limits of handling.
In this thesis, we design a controller for an autonomous vehicle with the goal of driving it from A to B as fast as possible. This is the main motivation in racing applications. The controller should steer the car with the goal to minimize the racing time.
This controller was designed within the model predictive controller (MPC) framework, where we used the concept of road-aligned model. In contrast with the standard mpc techniques, we use the objective function to maximize the progress along the reference path, by integrating a linear model of the vehicle progression along the centerline. Combined with linear vehicle model and constraints, a optimization problem providing the vehicle with the future steering and throttle values to apply is formulated and solved with linear programming in an on-line fashion during the race. We show the effectiveness of our controller in simulation, where the designed controller exhibits typical race drivers behaviours and strategies when steering a vehicle along a given track. We ultimately confront it with similar controllers from the literature, and derive its strength and weaknesses compared to them.

## Sammfatning

Autonoma fordon förväntas få en betydande inverkan på världen och därigenom eliminera den mänskliga faktorn på en av de farligaste platserna: vägar. Faktum är att dödsfall är en av de största källorna till mänsklig dödlighet och många länder i världen. Det förväntas att autonoma fordon kommer att bidra dramatiskt för att uppnå det. Dessutom använder man kontroller för att optimera både beteende och kommunikationshastighet.

För att minimera vägskador är ett tillvägagångssätt att utforma styrenheter som skulle hantera bilen vid sina gränser för hantering, genom att integrera komplex dynamik, såsom vidhäftningsförlust, är det möjligt att förhindra att bilen lämnar vägen. En praktisk inställning för att utvärdera denna typ av kontroller är ett racing sammanhang: En styrenhet styr en bil för att slutföra ett spår så snabbt som möjligt utan att lämna vägen och genom att bränna bilen till dess gränser för hantering.

I denna avhandling designar vi en kontroller för ett autonomt fordon med målet att driva det från A till B så fort som möjligt. Detta är den främsta motivationen i racingapplikationer. Kontrollern ska styra bilen med målet att minimera racingtiden.
Denna styrenhet utformades inom ramen för Model Predictive Controller (MPC), där vi använde begreppet vägjusterad modell. I motsats till standard mpc tekniker använder vi objektivfunktionen för att maximera framstegen längs referensvägen genom att integrera en linjär modell av fordonsprogressionen längs mittlinjen. Kombinerat med linjär fordonsmodell och begränsningar, ett optimeringsproblem som ger fordonet framtida styr- och gasvärden att applicera formuleras och lösas med linjär programmering i ett onlinemönster under loppet. Vi visar effektiviteten hos vår controller i simulering, där den designade regulatorn uppvisar typiska racerförare beteenden och strategier när du styr ett fordon längs ett visst spår. Vi konfronterar oss slutligen med liknande kontrollanter från litteraturen och härleder dess styrka och svagheter jämfört med dem.

# Contents

# 1 Introduction

The last few years have seen the emergence of autonomous vehicles, both as a research field for laboratories and as a commercial product for vehicles manufacturers such as Tesla and Volvo. If the latter company's product is expected to hit the market in 2017 [1], the other one has already been selling its solution for a couple months [2]. Research laboratories also have notable achievements, for instance Delphi labs deployed an autonomous Audi SQ5 on US roads for a 3500-miles road trip in 2015 [3], as illustrated in Figure 1.


Figure 1: The autonomous Delphi labs car (left) and its decision-making visualization unit.

## 1.1 Motivation

According to a Morgan Stanley study, autonomous cars could save the US $1.3 trillion a year. These earnings would be split between fuel savings from better traffic management, the lowered health care expenses due to the reduction of the number of road kills, and the productivity gains from spending less time driving [4].
On the social side, autonomous cars could allow elderly and disabled people to gain back some mobility, and reduce the need for parking space in cities as well.
A 2016 study by the American Automobile Association (AAA) Foundation for Traffic Safety [5] estimated that the average American motorist was driving around 50 km a day. In a week, this can represent several hours that cannot be used for leisure purposes. By delegating the driving task to an autonomous vehicle, these hours would be freed, resulting in a higher quality of life.
Another benefit of autonomous vehicle hitting the mainstream market is inter-vehicle communication. Nowadays, one human driver can only communicate with its peers using simple tools (directional signals, braking signals, etc). Autonomous vehicles could take advantage of a wireless communication system using a frequency band (IEEE 1609 family) and use traffic optimization algorithms to reduce traffic jams. This is commonly referred

as cooperative driving and theoretical algorithms have been proposed to implement this feature [6]. The conclusions include a 33% reduction of total travel times and a significant reduction of traffic jams, which would have a positive environmental impact since it has been estimated that reducing them could lower the carbon dioxide emissions by 20% [7]. Car manufacturers have realized the potential of cooperative driving for autonomous vehicles and imagine future applications for their new products. For instance, Volvo plans on using inter-vehicle communication systems to allow cars to adapt their speed as a function of other cars' behaviour, such as illustrated in Figure 2.


Figure 2: A visualization of a potential V2V communication application (courtesy of Volvo).

A complementary approach to designing cooperatively driving vehicles is to enable individual car's steering controllers to handle the car at its limits of adherence on the road. Toyota decided to test their new autonomous vehicle in closed environments where "extreme events" will be happening to test the car behaviour in hazardous scenarios [8]. We could imagine that these scenarios include high speed maneuvering control, and a racing context would be very adapted to provide such constraints.

## 1.2 Classification of autonomous vehicles

The current progress of autonomous driving technologies is classified using the SAE levels:

**Level 0:** No automation: the driving task is fully performed by the human driver.

**Level 1:** Driver assistance: the system can handle either steering or acceleration/deceleration using information about the driving environment, and the human driver is expected to perform all remaining aspects of the dynamic driving task.

**Level 2:** Partial automation: the vehicle can now handle both steering and acceleration/deceleration using information about the driving environment.
**From this point, the system is able to monitor the driving environment**

**Level 3:** Conditional automation: the vehicle handles the full driving task but may request the human driver to take control in some specific cases and expect an appropriate response.

**Level 4:** High automation: the car may still request an human intervention in certain cases but now it should not assume an appropriate decision from this person.

**Level 5:** Full automation: the vehicle handles the full driving task.

This classification is extracted from the official SAE international J3016 standard [9]. The previously expressed benefits from autonomous vehicles are not expected to be experienced before level 4 is reached [4]. The Volvo S60 "Drive me" is considered to be level 3 [1].

### 1.3 Challenges

Thus, there is still a long road before autonomous vehicles become as mainstream as the traditional ones, and one of the main tasks is to develop safe and efficient driving controllers.
To begin with, autonomous vehicles are expected to operate in complex urban environments embedding often intricate road maps and many independent actors such as other autonomous cars, human-driven vehicles or pedestrians. Getting all of this data available in real-time is a very complex task as V2V communications are still under-developed. To counterpart this lack of information, the Google Car relies on both laser sensor and video cameras as shown in Figure 3. The price of the first sensor and the computational complexity to extract meaningful data from the second one make this solution sub-optimal.
Then, road data is likely to be updated regularly since speed limits may change, some

roads might close due to heavy works, etc. Thus, relying on an offline map is not optimal and data should be frequently updated to ensure safe driving.

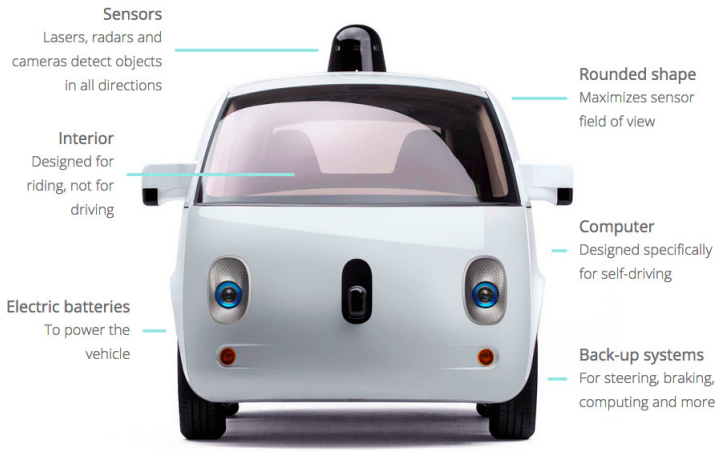


Figure 3: A scheme of the Google Car components (courtesy of Google)

Even with a good environment estimation, with real-time actors' position and up-to-date road maps available, deriving other actors' intentions is a difficult task. Especially it is not possible to know in advance when a human driver is going to be at fault by violating a red light or driving over the speed limit, making it hard to ensure safe autonomous driving. Ultimately, the challenge is also legal, as countries try to figure out the liability in case of a car accident if a human driver decides to let a computer drive the car.
All of these challenges are likely to hinder the deployment of autonomous cars to the consumer market.

## 1.4 Problem formulation

Designing controllers that are able to steer the autonomous car at its limits of handling will decrease the amount of potential road kills, thus increasing the public confidence in autonomous vehicles. A convenient setup to evaluate this type of controllers is a racing context: a controller is steering a car to complete a track as fast as possible without leaving the road and by brining the car to its limits of handling.

The context of this thesis is a racing scenario, where an autonomous car needs to complete the track as fast as possible. The racing task can be divided in two parts:

- Path planning: generating the optimal path (along with its associated speed profile) to be followed by the car in order to minimize the racing time;

- Path tracking: steer the vehicle towards the optimal path.

These two task can either be performed by two different controllers (2-layers approach) or by a single one (1-layer), combining the two parts in one formulation.
Several approaches to this type of controllers design are available, ranging from classic state feedback to model predictive controllers computing the optimal set of inputs (steer, throttle) to be applied to the system in real-time. A popular example of the latter types of methods is the Model Predictive Controller (MPC), because of its optimal problem solving and support of constraints.
The general framework for this type of controllers is:

$$\begin{aligned} &\max && \text{vehicle progress} \\ &\texttt{sub. to.} && \text{vehicle dynamics} \\ & && \text{road constraints} \\ & && \text{input constraints} \end{aligned}$$

In that framework, the goal would be to design a custom 1-layer MPC formulation, combining both path planning and path tracking tasks into one single optimization problem. The aim is maximizing the progress of the car along the centerline. This controller is derived from that study and implemented in a simulation environment and on a physical platform. Another MPC formulation directly extracted from a previous work is implemented as well to compare the performance of this custom controller to other works.
The platform used in this thesis is the F1/tenth RC car, and the driving performance is evaluated by how fast can the controller enable the car to race in a chosen circuit. A metric to evaluate how fast can the car race could be the length of the path tracked in a certain time, constrained by the car having to remain on the road.

## 1.5 Literature review

The design of MPC-based controllers for racing applications have been described on a 1:43 RC cars platform [10]. The work focuses on providing two approaches for the design of real-time optimal racing controllers: a 1-layer MPC and a 2-layers MPC formulation. The setup consists in a controller-equipped car placed in a defined racing track, both previous formulations being implemented and compared in two scenarios: with and without obstacles on the track. The 2-layers approach combines a path planner relying on a car model derived with simplifying assumptions providing both optimal path and associated car velocity, and a path tracker steering the car towards the previously generated path. Both tasks use a MPC formulation: the path planner optimizes the progress along the centerline and the path tracker minimizes the deviation of the car from the optimal path. The

1-layer approach defines a contouring error whose minimization results in maximizing the progress along the centerline. The conclusion of this study suggests that both approaches have pros and cons. The 2-layers approach being sensitive to unfeasible path generation because of the path planner relying on a simplified car model, and the 1-layer approach tendency to track the centerline are amongst the known issues.

A different design is suggested in [11], taking advantage of a reformulation of the whole racing problem to set the racing time as the objective function. The results suggest that this method allows for both an optimal racing behaviour and the tuning of holistic driving strategy variables such as the aggressivity of the driver.
Other nonlinear optimal formulations are described in [12]. The goal is to design a side wind rejection feature using a MPC controller relying on an advanced car model integrating tyre modelling to control the car even in a low adherence scenario (drift). Conclusions indicate that the real-time implementation of such complex formulations is however hard to achieve.
A solution to that issue is suggested in [13]. A simple dynamic model using a second order integrator dynamics is designed and scaled to specific actual vehicles using a sequence of experiments to derive the set of model parameters by curve fitting. The authors indicate that this model can be used for near-limits path planning applications and the conclusions of the study suggest that the accuracy provided by this approach would be sufficient. Another approach is taken in [14], where kinematic and dynamic bicycle models are compared taking into account the discretization phenomena. Results suggest that a controller using the discrete kinematic (simpler) model would perform similarly to its counterpart using a dynamic model, at the condition of using a lower sampling rate.
The topic of vehicle stabilization around a path is addressed in [15], model linearization is used to achieve the same goal as the previous study in a lane-change scenario. Physical limits of the car are taken into account to provide a stable control.
Near-limits dynamic control offer serious challenges because of the complex dynamics of a car tyre in case of adherence loss. Classic control methods are used to tackle this issue in [16]. Combining lateral (path tracking) and longitudinal (speed profile generation) controllers, both taking advantage of feedback and feed-forward schemes, real-time asymptotically stable controllers are designed. Conclusions suggest that it is possible to drive a car at its friction limits using this controller.
Offline methods are suggested in [17] for the path tracking task. A Linear Quadratic Regulation (LQR) controller is derived and implemented to minimize the path tracking error in a lane-change situation. A comparison is made between this methods and other classic designs. The results suggest that a fine tuning of such controllers can provide satisfactory results for different applications such as slow driving or highway driving.
Instead, in [13], a computationally light way of sampling feasible acceleration regions for the car. Starting from a known state, the future car positions can be computed through numerical integration. The first step is to compute a set of feasible longitudinal, lateral

and angular accelerations for various initial states of the vehicles in an off-line fashion. This region is then approximated to a linear convex one. The Figure **??** shows the result of such computation. One of the perks of this approach is not to be limited to stationary trajectories. Finally, an original bicycle kinematic model is derived using a road-aligned approach [18]. The obtained linear system is then used to derive an obstacle avoidance car controller.

## 1.6 Background

In this section, the notations and concepts to be used are introduced. In particular, the research scope and the autonomous vehicle components are detailed. A continuous dynamical system can be defined as

$$\dot{x} = f_l(x, u), \tag{1}$$

with the state $x \in \mathbb{R}^n$, the plant input (and controller output) $u \in \mathbb{R}^p$, and $f_l : \mathbb{R}^n \times \mathbb{R}^p \mapsto \mathbb{R}^n$.

In the same way, discrete system dynamics can be characterized by

$$x(t + T_s) = f(x(t), u(t)), \tag{2}$$

with $t \in R_+$ being the current time and $T_s \in R_+$ being the control algorithm sampling time. The equation (2) can be simplified to (3) by setting $T_s = 1$ and $x_t = x(t)$ without any loss of generality

$$x_{t+1} = f(x_t, u_t), \tag{3}$$

If the system dynamics happen to be linear, the above formulation can be written as:

$$\begin{aligned} x_{t+1} &= Ax_t + Bu_t \\ y_t &= Cx_t + Du_t, \end{aligned} \tag{4}$$

with $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times p}$, $C \in \mathbb{R}^{n\times n}$. This is commonly referred as the open-loop dynamics [19].

In order to perform regulation tasks (i.e., forcing the system state to follow a reference), it is necessary to obtain a closed-loop system. The control output $u$ should be computed as a function of the error between the reference $r$ and the current observed system state $y$.

$$u_k = g(e_0, ..., e_k), \tag{5}$$

with $\forall k \in \mathbb{N},\ e_k = r_k - y_k$.

### 1.6.1 System architecture

A possible autonomous vehicle architecture is detailed in [20] and is shown in Figure 4.

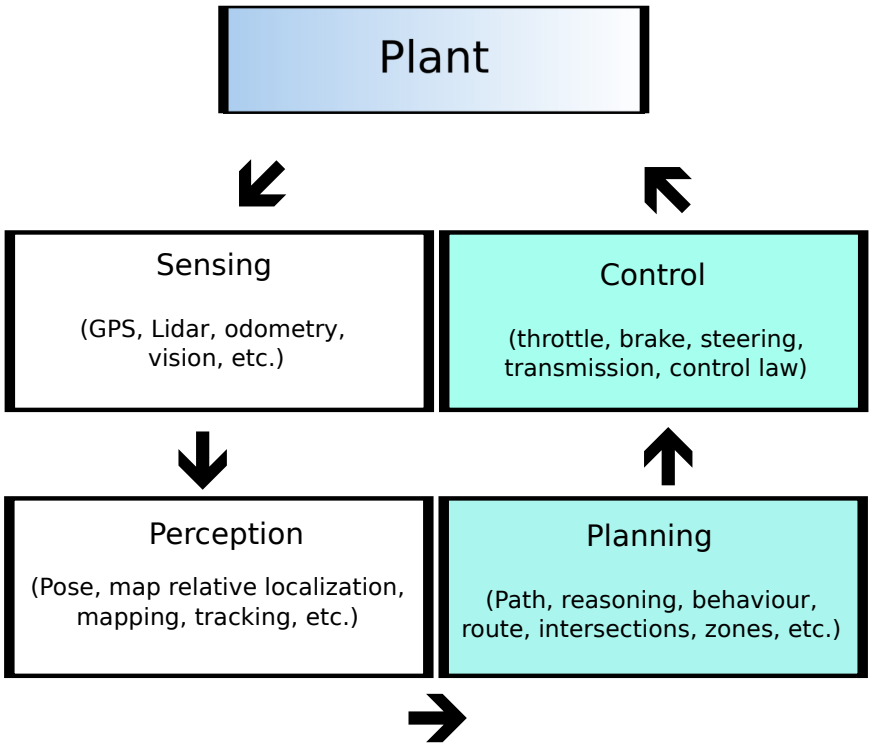


Figure 4: A possible autonomous car architecture [20]

The plant is the physical system to be controlled, i.e., the car. The sensing is designed to collect data from the environment and store it in the car's embedded computer. The perception treats and organizes the previously obtained data in order to estimate physical variables. For instance, the vehicle state (inertial position, velocity, attitude, rates, and the position and shape of other things in the surroundings). The planning uses the previous estimation to compute a reference path to be followed by the car. The controller finally tracks the obtained reference by steering the car inputes to the computed values. For instance, if the planning recommends an acceleration equals to $1m.s^{-2}$, the controller will actuate the engine gas and air intakes (in the case of a combustion engine) to steer the car acceleration to the planned value.

### 1.6.2 Racing theory

Race drivers desire to follow the optimal racing line, i.e., the one that maximizes the progress along the centerline. It is straightforward to compute the racing line when driving in a straight line or a moderate curve, the biggest challenge and where the impact is greatest is in sharp corners. The optimal line should minimize the time spent in a corner and maximize the overall speed through it [21]. The Figure 5 illustrates this theory.

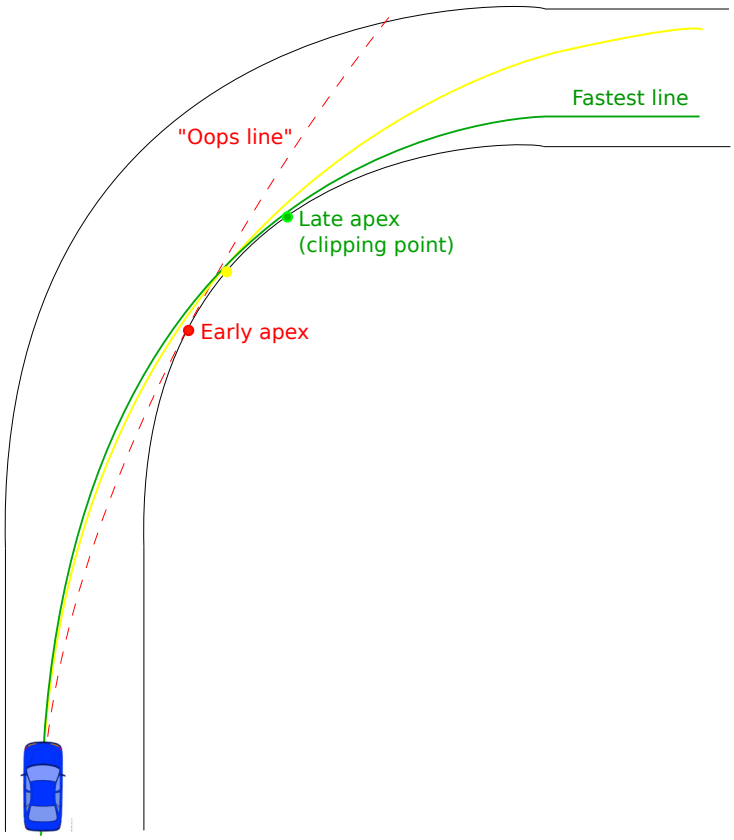


Figure 5: An illustration of driving trajectories in a corner

The apex or clipping point is the point on the inside portion of a corner where a vehicle passes closest to. The apex can also be described as the point of minimum radius and slowest speed achieved in a corner. Two strategies exist to handle a corner: late apex and early apex. An earlier apex will reach the inside of a corner at a higher speed and with a larger radius than a later apex. Ultimately the choice between the two approaches is to be made taking into account the dynamics of the car. It is important to avoid under-steering and over-steering in that case since it can lead the car to leave the road.

### 1.6.3 Classic control approach

A classic control method could consist in defining two state feedback controllers for aligning the car on the road and making it progress along the track. For instance, if the car's input consist in a throttle $u_t$ and steering wheel angle $u_\delta$, we could imagine the formulation

$$\begin{aligned} u_\delta &= K_\delta \left(e_y \ e_\psi\right)^T \\ u_t &= K_t \left(e_y \ e_\psi\right)^T \end{aligned} \tag{6}$$

with $K_t \in \mathbb{R}^{\not\in}$ and $K_\delta \in \mathbb{R}^2$.

To be able to benchmark our control design, classic controllers for path tracking are studied.
The first is extracted from [16] and embeds a kinematic feedback lane keeping controller along with a feed-forward steering controller to mimic drivers' behaviour. Only the first one is studied.

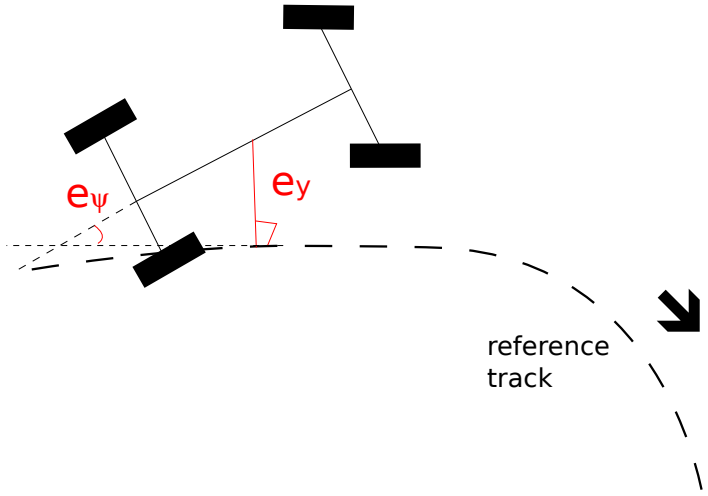


Figure 6: Classic controller: heading error $e_\psi$ and lookahead error $e_y$ definition [16]

The lane keeping control is then

$$u_\delta = -2\frac{k}{C_f}\left(e_y + x_{la} e_\psi\right), \tag{7}$$

The constant $k$ is the lane keeping potential field gain, which is chosen in order to ensure stability. It is worth observing that the parameter $x_{la}$ dictates how sensitive the yaw motion is: a high value allows the vehicle to be responsive but may involve oscillations.

### 1.6.4 MPC framework

Contrary to the classic control theory, Model Predictive Control (MPC) relies on solving an optimization problem on-line by minimizing a cost function $J$ depending on the state $x$ and plant input $u$. This step is repeated at each update time of the control algorithm.

The main idea of MPC is to take advantage of an existing physical model of the system to be controlled to predict the future values its state would take if a specific set of inputs was to be applied to it. This knowledge enables the controller to choose the set of future inputs that satisfies the desired system behaviour. This could be steering the state towards a known reference point or rejecting a disturbance.
An algorithm to implement this controller can be described as:

1. At time $t$, use the system model to predict the $N$ future steps of the plant $y_{t+k},\ k = 1, ..., N$ as a function of the future inputs $u_{t+j},\ j = 0..N$ and measurements known at time $t$.
2. Define a criterion (objective function) based on $y$ and $u$ and optimize it with respect to the future inputs $u_{t+j},\ j = 0..N$.
3. Apply $u_t^*$ to the plant (the first step of the previously computed future inputs).
4. Repeat at the next sampling instant.

The criterion at step 2 can include constraints on both the system state and its inputs [22]. These constraints are one of the strongest advantages of MPC compared to other techniques, as they allow for a high-level and efficient way of preventing the system state or its inputs from reaching certain values. The computed solution is still optimal with respect to these constraints, and the computational complexity is not significantly increased if the problem is convex and the constaints linear.

Let us consider the non-linear discrete-time system (2) with unitary sampling time.

$$x_{t+1} = f(x_t, u_t),$$

where $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ is $C^1$.
Both state $x$ and input $u$ are subject to constraints:

$$x_t \in X,\ u_t \in U,\ \forall t \in \mathbb{Z}^+$$

with the input constraints $U \subseteq \mathbb{R}^m$ and the state constraints $X \subseteq \mathbb{R}^n$.
Let us set $N > 0$ the prediction horizon, and $J : \mathbb{R}^n \times \mathbb{R}^{m \times N} \to \mathbb{R}^+$ the cost function defined as:

$$J(x_t, U_t) = \sum_{k=t}^{t+N-1} l(x_k, u_k), \tag{8}$$

where $U_t = [u_t, ..., u_{t+N-1}]$ is the sequence of $N$ future inputs to be optimized, and $x_k,\ \forall k \in [t, ..., t+N]$ is the resulting state trajectory. Ultimately,
$l : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^+ \in C^1$ is the stage gain.

From the previous definitions, it is possible to set the MPC general optimization problem as:

$$\begin{aligned}
\min_{u} \quad & J(x_t, u_t) && && \text{(9a)}\\
\text{s.t..} \quad & x_{k+1} = f(x_k, u_k), && \forall k \in [t, ..., t+N-1] && \text{(9b)}\\
& x_t = x(t) && && \text{(9c)}\\
& u_k \in U, && \forall k \in [t, ..., t+N] && \text{(9d)}\\
& x_k \in X, && \forall k \in [t, ..., t+N] && \text{(9e)}
\end{aligned}$$

where $u = [u_t, ..., u_{t+N-1}]$ is the sequence of the next $N$ inputs to be optimized. The equation (9b) is the prediction of the MPC: a physical model of the car is used to derive the next car states as a function of the inputs. The equation (9c) is the initialization of the MPC problem: the current state of the system $x(t)$ is assigned to the first element of the sequence of $N$ next predicted states of the system. The equation (9d) represents the input constraints and the equation (9e) represents the state constraints.

The solution of this optimization problem gives us the optimal inputs $u_k^*,\ \forall k \in [0, ..., N-1]$. We apply $u_0^*$ and repeat the resolution for the next step.

Although the MPC framework is adapted to both continuous and discrete systems, and both linear and non-linear systems, practical implementations on computer systems require the sampling of continuous physical dynamics and therefore take advantage of the discrete formulation. The Figure 7 illustrates the architecture of this solution.

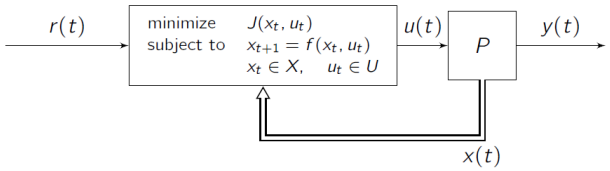


Figure 7: MPC system architecture [19]

Furthermore, the solving of optimization problems can be made computationally cheap if it is convex and if the prediction model $f$ is linear time-invariant. Also, it is then guaranteed that the solution converges to a global optimum.
Practically, $f$ is linearized around an equilibrium point to allow this simplified formulation.

$$\delta x(k+1) = \frac{\partial f(x,u)}{\partial x}\Big|_{(x_0(k),u_0(k))} \delta x(k) + \frac{\partial f(x,u)}{\partial u}\Big|_{(x_0(k),u_0(k))} \delta u(k), \tag{10}$$

where $\delta x(k) = x(k) - x_0(k)$ and $\delta u(k) = u(k) - u_0(k)$, and we set:

$$A(k) = \frac{\partial f(x,u)}{\partial x}\Big|_{(x_0(k),u_0(k))} \tag{11}$$

$$B(k) = \frac{\partial f(x,u)}{\partial u}\Big|_{(x_0(k),u_0(k))} \tag{12}$$

If we reuse the model (4)

$$\begin{aligned} x(t+1) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) + Du(t) \end{aligned}$$

The optimization problem is defined as

$$\begin{aligned} \min_{U} \quad & J(x_0, U_0) \\ \text{s.t. } & \delta x_{k+1} = A(k)\delta x_k + B(k)\delta u_k, \quad \forall k \in [0, ..., N-1] \\ & x_0 = x(t) \\ & u_k \in U, \qquad \forall k \in [0, ..., N] \\ & x_k \in X, \qquad \forall k \in [0, ..., N] \end{aligned} \tag{13}$$

The cost function $J$ can now be written in a parabolic formulation for reference tracking, i.e. steering the state $x$ towards a reference $x_r$.

$$J(x(k), U(k)) = \sum_{k=t}^{t+N-1} (x(k) - x_r)Q(x(k) - x_r)^T + u(k)Ru(k)^T \tag{14}$$

with $Q \in \mathbb{R}^n$ and $R \in \mathbb{R}^m$ [23].

## 1.7 Outline

Chapter 2 is dedicated to summarizing the vehicle models that have been described in the literature and how they differ in terms of fidelity and complexity. Also, the problematic of the physical fitting is quickly explored, in particular for the tire identification.

Chapter 3 focuses on the existing MPC formulation for racing applications and how they differ. Their advantages and drawbacks are detailed. Then, a custom MPC formulation is derived from the idea of maximizing the progress along the centerline over the prediction horizon.

Finally, chapter 4 details the experimental setup and actual implementation of the latter controller. Furthermore, a comparison is set up in order to evaluate the controller racing performance.

# 2 Vehicle modeling

## 2.1 Kinematic model

In order to solve the optimization problem (13), the dynamics of the vehicle have to be investigated. This section focuses on presenting the bicycle model that describes the car behaviour on the road as reported in [10] and [12]. It consists in simplifying the car body to a rigid body of mass $m$ and inertia $I_z$, with a front wheel and a rear wheel only. The dynamics are 2D (in-plane motion), thus the pitch, roll dynamics and load changes are not considered.

The vehicle model is used both to simulate the car behaviour on the road and by the MPC during the optimization problem solving step. In the latter case, the model is often simplified from the simulation case since using more complex models with more states has a dramatic impact on the optimization problem solving complexity.

### 2.1.1 Bicycle model

**Kinematic model** Figure 8 shows notations for the bicycle model kinematics.

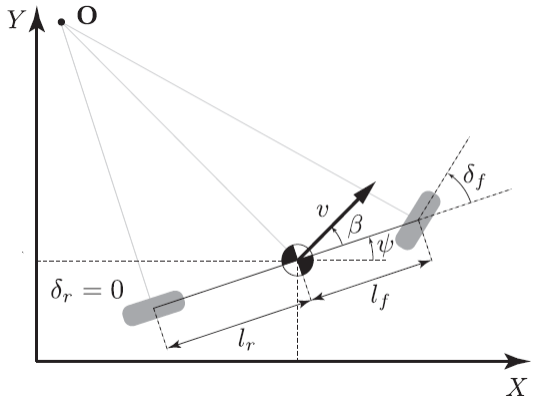


Figure 8: A scheme of the bicycle model [14]

The frame is body-fixed to the bicycle and the origin is set at its centroid. The vehicle dynamics equations are described by [14] and [11].

$$\begin{cases} \dot{X} = v\cos(\psi+\beta) \\ \dot{Y} = v\sin(\psi+\beta) \\ \dot{\psi} = \dfrac{v}{l_r}\sin(\beta) \\ \dot{v} = a \\ \beta = \tan^{-1}\left(\dfrac{l_r}{l_f+l_r}\tan(\delta_f)\right) \end{cases} \tag{15}$$

The state is

$$x = (X\ Y\ \psi\ v)^T \tag{16}$$

and the input vector is

$$u = (a\ \delta_f)^T \tag{17}$$

The kinematic model focuses on the the geometrical properties of the vehicle and, by definition, does not take into account the dynamics of the vehicle (tire forces, yaw rate, etc). It assumes that the car adheres perfectly to the road (no slip). This model gives satisfactory results when the car runs at low speed.

The previous formulation (15) can be significantly simplified if the speed is set as a constant [18]. The idea is to define the following errors:

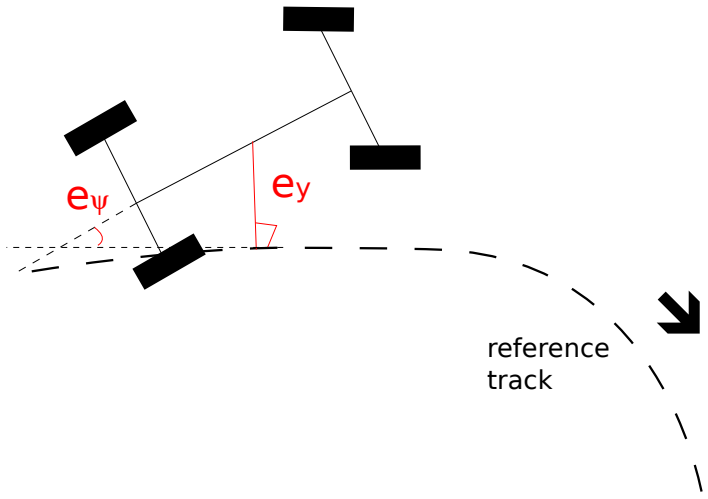


Figure 9: The lateral and heading errors

The lateral error $e_y$ is computed as the euclidian distance between the car position and its projection on the reference track $z_{ref}$. The heading error $e_{psi}$ is the difference between the car heading $\psi$ and the reference track heading at the projection point $\psi_{ref}$. Ultimately, $\rho_s$ denotes the reference track's curvature radius at the same point.

Starting from the nonlinear kinematic model previously introduced, with constant speed $v$ and a center of gravity located at the rear axle to simplify the model:

$$\begin{cases} \dot{X} = v\cos(\psi) \\ \dot{Y} = v\sin(\psi) \\ \dot{\psi} = \dfrac{v}{l}\tan(\delta) \end{cases} \tag{18}$$

where $l$ denotes the wheelbase.
We introduce the time and spatial derivatives as: $\dot{x} = \frac{dx}{dt}$ and $x' = \frac{dx}{ds}$. It is straightforward to derive that $\dot{e}_\psi = \dot{\psi} - \dot{\psi}_s$ ($\psi_s$ being the heading angle of the track centerline), $\dot{e}_y = v\sin(e_\psi)$, and:

$$\dot{s} = \frac{\rho_s v\cos(e_\psi)}{\rho_s - e_y} \tag{19}$$

The **spatial representation** is obtained by using $e'_\psi = \frac{\dot{e}_\psi}{\dot{s}}$ and $e'_y = \frac{\dot{e}_y}{\dot{s}}$ instead of their time counterparts. The result is a two-states non-linear system with $\delta$ as input [18].

$$\begin{cases} e'_\psi = \dfrac{(\rho_s - e_y)\tan(\delta)}{\rho_s l\cos(e_\psi)} - \psi'_s \\ e'_y = \dfrac{\rho_s - e_y}{\rho_s}\tan(e_\psi) \end{cases} \tag{20}$$

This model is used to build a custom controller in the next sections. It is simplified even further with additional simplifying assumptions and linearization near from the centerline.
This would lead to the linear discrete-time model :

$$\begin{pmatrix} e_y \\ e_\psi \end{pmatrix}_{k+1} = \begin{pmatrix} 1 & \Delta s \\ -1/\rho_s^2 & 1 \end{pmatrix}\begin{pmatrix} e_y \\ e_\psi \end{pmatrix}_k + \begin{pmatrix} 0 \\ \Delta s \end{pmatrix}\tilde{\kappa}_k \tag{21}$$

with $\Delta s = vT_s$ and $\tilde{\kappa} = \frac{\tan(\delta)}{l} - \psi'_s$ being the curvature input of the vehicle path.

## 2.2 Dynamic model

A kinematic model offers satisfactory results when the vehicle speed and steering angle are low enough, but it becomes inadapted when the vehicle is brought to its limit of adherence and tires start to lose grip on the road (this is referred as drifting). To simulate a realistic vehicle behaviour, it is necessary to integrate these complex dynamics in our simulation model. Moreover, a simplified version of these dynamics can be integrated to the prediction model to allow the controller to predict when the tires are about to lose grip.
The model's equations are derived from [10] and [24].
The figure 10 details the notations used for the bicycle model description.

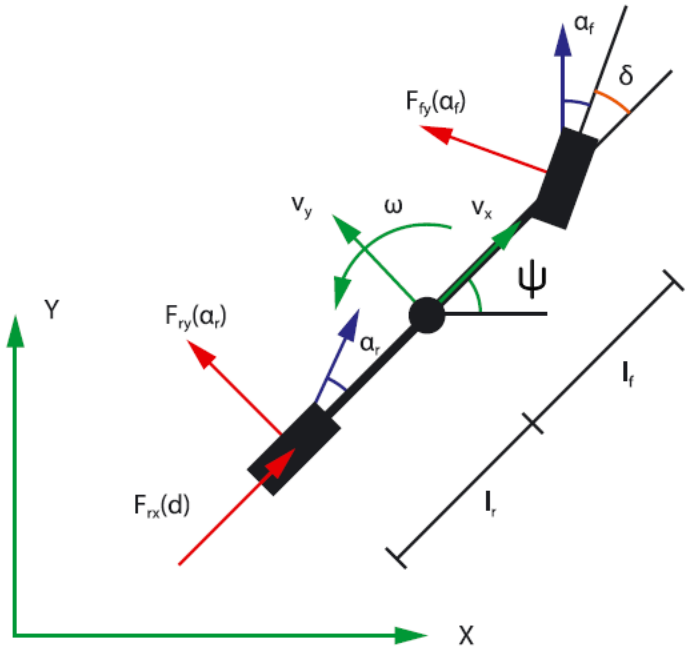


Figure 10: A scheme of the bicycle model [10]

The frame is still body-fixed to the bicycle and the origin is set at its centroid.

$$\begin{cases} \dot{X} = v_x \cos(\psi) - v_y \sin(\psi) \\ \dot{Y} = v_x \sin(\psi) + v_y \cos(\psi) \\ \dot{\varphi} = \omega \\ \dot{v_x} = \dfrac{1}{m}\left(F_{r,x} - F_{f,y} \sin(\delta) + m v_y \omega\right) \\ \dot{v_y} = \dfrac{1}{m}\left(F_{r,y} - F_{f,y} \cos(\delta) - m v_x \omega\right) \\ \dot{\omega} = \dfrac{1}{I_z}\left(F_{f,y} l_f \cos(\delta) - F_{r,y} l_r\right) \end{cases} \tag{22}$$

The state is:

$$x = (X \ Y \ \psi \ v_x \ v_y \ \omega)^T \tag{23}$$

It is now time to take into account the forces applied by the tires to the road as well as the yaw dynamics. In [10] and [11], the model includes two more states that translate the increased complexity of the system, that could lead to more accuracy when the car is close to its limits of handling. The $F$ forces model the tires interaction with the road and are determined in a semi-empiric way, as detailed in the section 2.2.1.

### 2.2.1 Tire forces

The figure 11 details the notations used to describe the tires interaction with the road. This is needed to fully characterize the dynamics of the vehicle and thus drive it at the limits of handling: under some conditions, the vehicle can lose adherence and then slip.

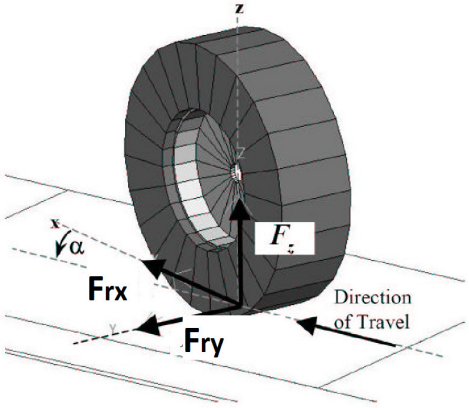


Figure 11: A scheme of the tire forces, notations are for the rear wheel [12]

**Pacejka tire model** The tire forces are modelled using a Pacejka tire model [10].

$$\begin{aligned} F_{f,y} &= D_f \sin\left(C_f \arctan\left(B_f \alpha_f\right)\right) \\ F_{r,y} &= D_r \sin\left(C_r \arctan\left(B_r \alpha_r\right)\right) \\ F_{r,x} &= \left(C_{m1} - C_{m2} v_x\right) d - C_r - C_d v_x^2 \end{aligned} \tag{24}$$

where:

$$\begin{aligned} \alpha_f &= -\arctan\left(\frac{\dot{\varphi} l_f + v_y}{|v_x|}\right) + \delta \\ \alpha_r &= -\arctan\left(\frac{\dot{\varphi} l_r - v_y}{|v_x|}\right) \end{aligned} \tag{25}$$

The parameters $B$, $C$ and $D$ are to be determined empirically by performing a set of experiments on the car. The $F_{r,y}$ and $F_{f,y}$ equations from the system (24) model tire friction on the road and the last one uses a DC motor coupled to friction models to describe the rolling resistance and the drag of the rear wheel.

It is critical to have a realistic tire model to take the slip phenomenon into account, without that it would not be possible to drive the car at its limits of handling and race efficiently. However, fitting a physical tire behaviour to its corresponding parameters value in a Pacejka model can be hard because of the model complexity.

**Fiala tire model** The Fiala tire model takes a simpler approach, requiring less parameters to fit a physical tire to a mathematical model. The first assumption is that lateral and longitudinal stiffness $C_f$ and $C_r$ are equal. The total slip $\sigma$ is computed:

$$\sigma = \sqrt{\sigma_y^2 + \sigma_x^2} \tag{26}$$

With $\sigma_x$ and $\sigma_y$ being longitudinal and lateral slip, respectively.

$$\begin{aligned} \sigma_x &= \frac{r\omega_w - v_x}{r\omega_w} && \text{during acceleration} \\ \sigma_x &= \frac{r\omega_w - v_x}{V_x} && \text{during braking} \\ \sigma_y &= \frac{v_x}{r\omega_w} \tan(\alpha) \end{aligned} \tag{27}$$

with $\omega_w$ being the rotational speed of the wheel, and $r$ the radius of the wheel.
The fundation of the Fiala tire model is to assume that the pressure distribution on a tire's contact patch is parabolic,

$$\begin{aligned} F_t &= \mu F_z \left(3\theta\sigma - 3(\theta\sigma)^2 + (\theta\sigma)^3\right) && \text{if } \sigma \leq \sigma_m \\ F_t &= \mu F_z && \text{if } \sigma \geq \sigma_m \end{aligned} \tag{28}$$

with $\theta = \frac{1}{\sigma_m} = \frac{C_r}{3\mu F_z}$. Note that $C_f = C_r$. Sliding is assumed to begin when $F_t = \mu F_z$. From that, the lateral and longitudinal tire forces can be obtained.

$$\begin{aligned} F_{r,x} &= \frac{\sigma_x}{\sigma} F_t \\ F_{r,y} &= \frac{\sigma_y}{\sigma} F_t \end{aligned} \tag{29}$$

In case of pure lateral slip, $\sigma_y = \tan(\alpha)$ and $\sigma_x = 0$. In case of pure longitudinal slip, $\sigma_y = 0$.

The simpler approach the Fiala tire model takes motivates us to use it as our tire model for the dynamic simulation model.

# 3 MPC for racing

In this section, we first detail how previous studies tackled the MPC for racing application challenge, in terms of architecture and formulation. Then, a custom MPC formulation is derived, aiming at maximizing the progress of the car along the centerline over the prediction horizon, this formulation will be named Progress Maximization Model Predictive Controller (PM-MPC).

## 3.1 Two-layers MPC

In [10], the authors propose the division of the problem can be divided in two smaller subproblems: the path planning and path tracking. .
First, it is necessary to generate an optimal *racing line* along with its speed profile. In fact, as developed in [21], simply following the centerline is a terrible strategy since a significant time would be lost in curves (where the track length decreases with the proximity from the internal boundary of the curve) and could even lead to significant speed decreases if the corner is sharp enough. So it is necessary to know which path should the car follow and at which speed.
Then, it is necessary to compute from this optimal path a set of inputs (steering and throttle) that would allow the car to remain close enough from the generated racing line. This is a classic control problem that is referred to as line tracking.

### 3.1.1 Path planning

As introduced in [13], the path planning step must generate a reference path to be followed by the car. In a racing context, this path should minimize the lap driving time, or equivalently maximize the progress over a certain time. Thus the reference can be generated by solving an on-line optimization problem subject to the car's dynamics and the constraint that the vehicle should not go off-road.
However, as detailed in [10] and [13], this approach leads to highly non-linear, non-convex objective functions and overall generates an optimization problem current solvers can not handle in most cases. Thus a workaround is needed.
In [10], a library of vehicle motion primitives is store and used for online integrations. The vehicle motion primitives are consisted of set constant speed and constant steering angles. An illustration is shown in Figure 12. The formulation of such MPC is:

$$
\begin{aligned}
x^* &= \max_{j \in [1,\ ...,\ N_0]} P(x_k^j) \\
s.t.\ x_t^j &= x(t) \\
x_{k+1}^j &= f(x_k^j, u_k^j),\ \forall k \in [t,\ ...,\ t+N-1] \\
x_k^j &\in X_{track} \\
u_k^j &\in U(x)
\end{aligned}
\tag{30}
$$

where the $j$ index belongs to the set $[1,\ ...,\ N_0]$ of all the possible trajectories arising from the input set the controller might choose. And $x^* = [x_0^*,\ ...,\ x_{N-1}^*]$ is the optimal trajectory, maximizing the progress along the centerline. This problem formulation differs from (13) by providing a set of optimal states to be tracked as reference instead of a set of inputs to be applied. The operator $P : x \to P(x)$ is the projection of a state towards the centerline.
Moreover, $U(x)$ is the set of all admissible inputs the car can accept. This set is a function of the current state and is computed in an offline fashion. It basically is a simplified dynamic model of the car, a library linking the possible steering angles as a function of the current speed of the car. This is a computationally cheap way of adding dynamics to the model but it makes bold assumptions about the car behaviour: the speed is assumed constant over a whole prediction horizon, as well as the trajectory curvature.

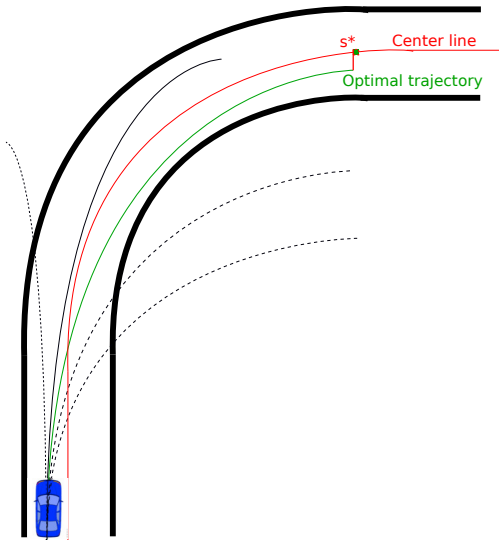


Figure 12: A representation of a path planner algorithm: the path with that offers the largest progress along the centerline is chosen [10].

### 3.1.2 Path tracking

In the path tracking layer, an MPC is used to track the reference path/trajectory planned in the path planning layer.

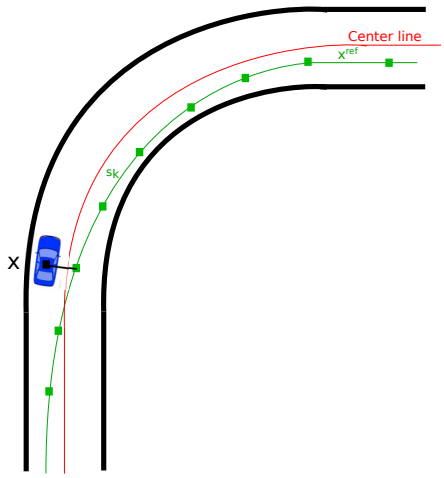


Figure 13: path tracking errors illustration [10]

This controller steers the car towards the generated optimal trajectory and tries to minimize the deviation from it. The MPC is formulated as follows [10]:

$$
\begin{aligned}
&\min_{x,u} \; ||x_{t+N} - x^{ref}_{t+N}||^2_P + \sum_{k=0}^{N-1} ||x_{t+k} - x^{ref}_{t+k}||^2_Q + ||u_{t+k} - u^{ref}_{t+k}||^2_R \\
&s.t.\ x_t = x(t) \\
&x_{k+1} = f(x_k, u_k), && \forall k \in [t, ..., t+N-1] \\
&F_k x_k \leq f_k + s_k, && \forall k \in [t+1, ..., t+N] \\
&s_k \geq 0, && \forall k \in [t+1, ..., t+N] \\
&x_m \leq x_k \leq x_M, && \forall k \in [t+1, ..., t+N] \\
&u_m \leq u_k \leq u_M, && \forall k \in [t, ..., t+N-1],
\end{aligned}
\tag{31}
$$

where $s_k,\ \forall k \in [1,\ ...,\ N]$ is positive a use to soften the constraint on the state in order to enhance stability. Also, $\forall P \in M^n(\mathbb{R})$, the $P$ norm is defined as:

$$
||.||_P :\ x \rightarrow P^T x P, \tag{32}
$$

where $x \in \mathbb{R}^n$. In (31), $P, Q \in M^n(\mathbb{R})$, $R \in M^p(\mathbb{R})$ and $x_k \in \mathbb{R}^n$, $u_k \mathbb{R}^p$, with $p, n \in \mathbb{N}$.

This formulation is based on minimizing the deviation from the previously generated reference path $[x_0,\ ...,\ x_N]$. But other formulations could be used, such as the one from

[11], that aims at directly minimizing the race time. Using a spatial reformulation, the time becomes an independent variable in the MPC formulation the optimization problem is then based upon.

### 3.2 One-layer MPC

Another approach arises in [10]. The two latter problems identified in the previous section as path planning and path tracking could be merged into a single formulation, denoted as Model Predictive Contouring Control (MPCC). The Figure 14 illustrates this principle.

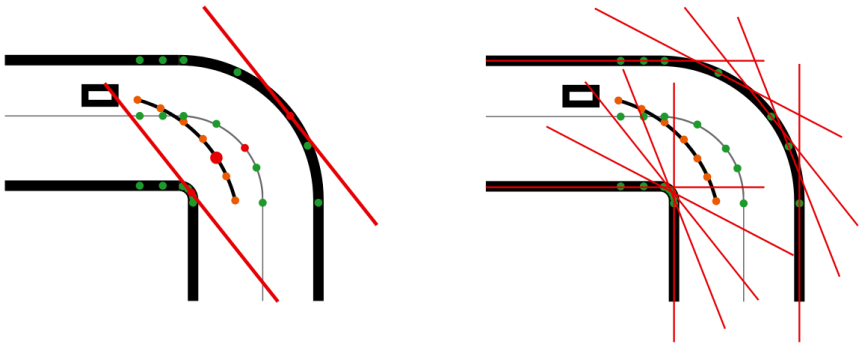

Figure 14: Illustration of one-layer MPC [10]

First, the trajectory is parametrized by its arc length $s$. It allows us to to obtain any point $x = \left(x^1_{ref},\ x^2_{ref}\right)$ on the centerline by evaluating a third-order polynomial on $s$.

$$\Phi(s) = \arctan\left(\frac{\partial x^2_{ref}(s)}{\partial x^1_{ref}(s)}\right) \tag{33}$$

The error measure for the lateral deviation is computed using the previously introduced projection operator $P$ to obtain the centerline-projected state $x^P$ but this would lead to a nested optimization problem, as an orthogonal projection is already an optimization problem by itself, causing a low efficiency.
Instead, a lag error $e^l$ is defined.

$$e^l(x, s^A) = |s^A - s^P| \tag{34}$$

And it is approximated by:

$$\begin{aligned}\hat{e}^c(x^1, x^2, s^A) &= \sin(\Phi(s^A))(x^1 - x^1_{ref}(s^A) - \cos(\Phi(s^A))(x^2 - x^2_{ref}(s^A) \\ \hat{e}^l(x^1, x^2, s^A) &= -\cos(\Phi(s^A))(x^1 - x^1_{ref}(s^A) - \sin(\Phi(s^A))(x^2 - x^2_{ref}(s^A)\end{aligned} \tag{35}$$

The Figure 15 illustrates these entities.

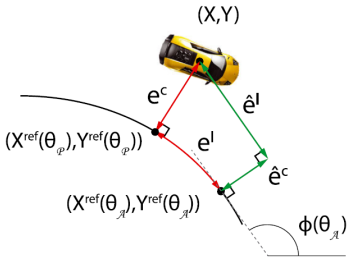


Figure 15: One layer MPC errors illustration [10]

The final formulation can then be obtained.

$$\begin{aligned}\min \sum_{k=1}^{N} &||\hat{e}^c_k(x_k, s_k)||^2_{q_c} + ||\hat{e}^l_k(x_k, s_k)||^2_{q_l} - \gamma(v_k - v_0)T_s + ||\Delta u_k||^2_{R_u} + ||\Delta v_k||^2_{R_v} \\ \text{s.t. } & s_0 = s \\ & s^A_{k+1} = s^A_k + v_k T_s, \ \forall k \in [0, \ ..., \ N-1] \\ & 0 \leq s_k \leq L, \ \forall k \in [0, \ ..., \ N-1] \\ & 0 \leq v_k \leq \bar{v}, \ \forall k \in [0, \ ..., \ N],\end{aligned} \tag{36}$$

where $\Delta u_k = u_k - u_{k-1}$ and $\Delta v_k = v_k - v_{k-1}$, the norms $||.||_{q_c}$ and $||.||_{q_l}$ are defined in (32). $T_s$ is the sampling time of the controller and $\gamma \in \mathbb{R}$ is a tuning weight. The third line of (36) expresses the approximation made to estimate $s$, as it is considered that the car is always having a low heading error $e_\psi$ as defined in Figure 9.

### 3.3 Criticism of the previous controllers

The previously described controllers aim at minimizing the racing time along a track. However, they have design issues that hinder their performance in that challenge.

The 2-layers MPC has to solve two optimization problems per step instead of one, which results in an increased computational complexity. To counter that, the authors have

decided to rely on an offline library of vehicle motion primitives, with constant speed and constant steering angles for the path planning path. This reduced the computational complexity of that step but introduced other issues. First, the number of available paths is limited by the size of the library, implying that the best possible path to take might not be accessible for path generation. Then, the fact that both steering angles and throttle are set constants along the prediction horizon limits the validity of the available paths, as the vehicle might end up having a different speed a few steps in the future. Ultimately, keeping a constant steering angle over the prediction horizon implies that the available trajectories consists in circle arcs that can become full circles if the prediction horizon becomes too large.

The 1-layer MPC's cost function penalizes deviation from the centerline by integrating the norm of $e^c$, which implies that the controller will be encouraged to steer the car towards the centerline. In [21], the author states that this is unlikely to be an optimal racing strategy, as seeking for the internal road boundaries in gently curvy portions and aiming for the late-apex in sharp curves provides the largest progress along the centerline, as shown in Figure 5. This can be mitigated by tuning the weights $q_c$, $q_l$ and $\gamma$, but it requires the designer to find a compromise between the racing performance and the road exit risk mitigation.

As a result, a custom controller is designed to attempt to offer another way of tackling the racing challenge.

## 3.4 Design of a custom 1-layer racing MPC controller

In this section are exposed the notations and the solving strategy of the problem.
To start with, the controller to be designed is chosen to have a 1-layer architecture in order to limit the required computational complexity. Also, it is chosen not to rely on off-line libraries describing the vehicle dynamics but instead integrate a simplified model of the vehicle dynamics to the optimization problem.

### 3.4.1 The racing challenge

The approach consisting in maximizing the progress of the racing car along the centerline is kept from the 2-layers formulation because it is a rather intuitive way of formulating the racing objective. This idea is equivalent to minimizing the racing time. The figure 16 illustrates this concept.

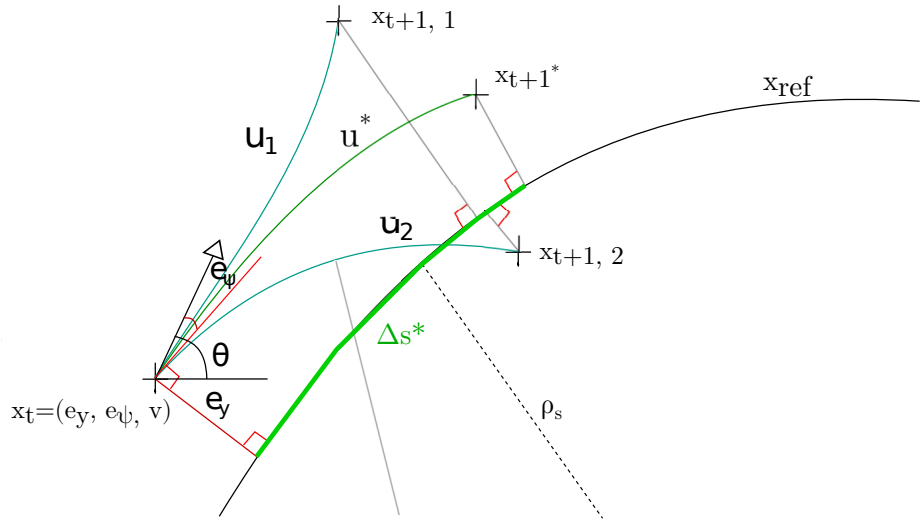


Figure 16: A scheme detailing the notations

The state of the system (i.e. the racing car) is denoted as $x$, the input is $u$. The racing track's centerline is $x_{ref}$ and $\rho_s$ is its curvature radius.
The errors $e_y$ (lateral) and $e_\psi$ (heading) are computed and are detailed in the next sections. The goal is then to select the input $u^*$ that maximizes the progress along the centerline between the projection of $x_i$ and the projection of $x_{i+1}$, this value is denoted as $\Delta s^*$.

### 3.4.2 Modelling

It is necessary to express the progress along the centerline during a discrete step as a function of the system states $x = (e_y\ e_\psi\ v)$. Let us start from the equation (19), that provides $\dot{s}$ and discretize it to obtain $\Delta s = \dot{s} T_s$.

$$\Delta s = T_s v \frac{\rho_s \cos(e_\psi)}{\rho_s - e_y}, \tag{37}$$

The prediction model is referred as $f$:

$$x_{k+1} = f(x_k, u_k), \tag{38}$$

For now, $f$ is a nonlinear function whose explicit expression is obtained by transformation of 20.

### 3.4.3 Custom racing MPC formulation

The resulting formulation is then:

$$\begin{aligned} \min_{u} \; & -\sum_{k=0}^{N-1} v_k \frac{\rho_{s,k}\cos(e_{\psi,k})}{\rho_{s,k} - e_{y,k}} \\ \text{s.t. } & x_0 = x(t) \\ & x_{k+1} = f(x_k, u_k) \\ & x_k \in X \\ & u_k \in U \end{aligned} \tag{39}$$

The formulation (39) is limited by several factors:

- The goal function is nonlinear, making the optimization problem non-convex and thus computationally expensive to solve;
- The prediction model is nonlinear.

Also, the constraints need to be specified later on.

We start over from the model (20). The linearization strategy for this model is based on the fact that when a model is linearized around an equilibrium point, the resulting approximation becomes less and accurate when the system is far from this latter point. The car is neither expected to always drive near from the centerline nor any of the road limits, but we can assume that the predicted trajectory from the previous optimization step will accurately predict the next positions.
If the vehicle starts from a known position, the controller can predict the $N$ next positions, and then re-use them as linearization points for the next steps. At each new computation, $N-1$ linearization points are then available (all the previous ones but the first one since it is now behind the car) and the last point can be obtained from the new MPC computation. The Figure 17 illustrates this behaviour. The equilibrium states are denoted as $\forall k \in [0, \ ..., \ N]$, $\bar{e}_y(k)$ for the lateral deviation and $\forall k \in [0, \ ..., \ N]$, $\bar{e}_\psi(k)$ for the heading deviation.

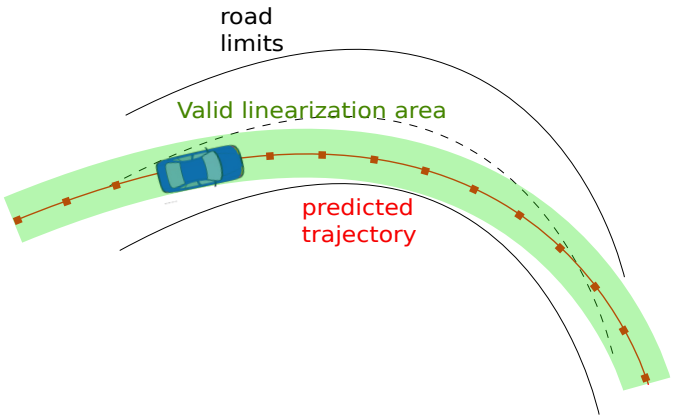


Figure 17: Scheme illustrating the linearization strategy: the equilibirum points are chosen on the predicted vehicle path

We set the curvature of the car trajectory $\kappa = \tan(\delta)/l$ as an input and $x = \begin{pmatrix} e_\psi \\ e_y \\ v \end{pmatrix}$.

$$x' = f(x,u) = \begin{pmatrix} \frac{(\rho_s - e_y)\kappa}{\rho_s \cos(e_\psi)} - \psi'_s \\ \frac{\rho_s - e_y}{\rho_s} \tan(e_\psi) \\ 0 \end{pmatrix} \tag{40}$$

We then discretize the system using $\Delta s = vT_s$ as the elementary step, where $v$ is the vehicle speed and $T_s$ is the controller's sampling time.

$$x_{k+1} = x_k + \Delta s \left( \sum_{i=1}^{M_x} \frac{\partial f}{\partial x_i}(\bar{x}_k, \bar{u}_k)(x_{i,k} - \bar{x}_{i,k}) + \sum_{i=1}^{M_u} \frac{\partial f}{\partial u_i}(\bar{x}_k, \bar{u}_k)(u_{i,k} - \bar{u}_{i,k}) + f(\bar{x}_k, \bar{u}_k) \right) \tag{41}$$

with $M_x$ and $M_u$ being the number of states and inputs, respectively. In order to account for the speed variation of the car, we introduce the speed of the vehicle $v$ as a state of the system. The new state is denoted as $x = \begin{pmatrix} e_\psi \\ e_y \\ v \end{pmatrix}$.

$$x_{k+1} = x_k + T_s \left( \sum_{i=1}^{M_x} \frac{\partial g}{\partial x_i}(\bar{x}_k, \bar{u}_k)(x_{i,k} - \bar{x}_{i,k}) + \sum_{i=1}^{M_u} \frac{\partial g}{\partial x_i}(\bar{x}_k, \bar{u}_k)(u_{i,k} - \bar{u}_{i,k}) + g(\bar{x}_k, \bar{u}_k) \right) \tag{42}$$

The final formulation is then:

$$\forall k \in \mathbb{N}, x_{k+1} = A(x_k - \bar{x_k}) + B(u_k - \bar{u_k}), \tag{43}$$

where:

$$\begin{aligned} A &= \frac{T_s}{\rho_s} \begin{pmatrix} \frac{\rho_s}{T_s} - \tan(\bar{e}_\psi)\bar{v} & \frac{(\rho_s - \bar{e}_y)\bar{v}}{\tan^2(\bar{e}_\psi)+1} & \tan(\bar{e}_\psi)(\rho_s - \bar{e}_y) \\ -\frac{\bar{\kappa}\bar{v}}{\cos(\bar{e}_\psi)} & \frac{\rho_s}{T_s} + \frac{\sin(\bar{e}_\psi)\bar{\kappa}(\rho_s - \bar{e}_y)\bar{v}}{\cos^2(\bar{e}_\psi)} & \frac{\bar{\kappa}(\rho_s - \bar{e}_y)}{\cos(\bar{e}_\psi)} - \rho_s\psi_s \\ 0 & 0 & 0 \end{pmatrix} \\ B &= \begin{pmatrix} 0 & 0 \\ \frac{(\rho_s - \bar{e}_y)\bar{v}}{\cos(\bar{e}_\psi)\rho_s} & 0 \\ 0 & a_k \end{pmatrix} \end{aligned} \tag{44}$$

It is the prediction model to be used from now on. It is important to note that there exist a function $F$ such that: $\forall k \in \mathbb{N},\ x_k = F(x_0, u)$ where $u = (u_0,\ ...,\ u_{N-1})$ and $F : \mathbb{R}^3 \times \mathbb{R}^N \to \mathbb{R}^3$.

**Constraints** The constraints can be divided in 3 groups:

- The physical limits of the car (maximum throttle, curvature of the trajectory, etc);
- The racing constraints (the car must not leave the road, go in reverse, etc);
- The controller-designed constraints (soft constraints, etc).

The physical constraints include:

- The car has physical limits for throttle and braking: $\forall k\ \in \mathbb{N}, a_{min} \leq a_k \leq a_{max}$;
- The car trajectory has physical curvature limits: $\forall k\ \in \mathbb{N}, \kappa_{min} \leq \kappa_k \leq \kappa_{max}$

The racing constraints include:

- The car should not leave the road: $\forall k\ \in \mathbb{N}, -L_{road} \leq e_y(k) \leq L_{road}$;
- The car should not not go in reverse: $\forall k\ \in \mathbb{N}, -\pi/2 \leq e_\psi(k) \leq \pi/2$

**Cost function** The cost function has to be linearized too in order to generate a convex optimization problem. As exposed in (39), the goal is to maximize the progress along the centerline, $\Delta s$.

$$J_l(u, x_t) = -\sum_{k=t}^{t+N-1} v(k)\rho_s(k)\frac{\cos(e_\psi(k))}{\rho_s - e_y(k)} \tag{45}$$

where $x_t = \begin{pmatrix} e_\psi(t) \\ e_y(t) \\ v(t) \end{pmatrix}$ is the current system state. Here, we use the cost function as the opposite of the objective function, it depends on the sequence of inputs $u = (u_t, ..., u_{t+N-1})$ and the current state $x_0 = x(t)$. Let us linearize the objective function following the same process as for the prediction model. The result is:

$$\begin{aligned} J_l(u, x_t) = -\sum_{k=t}^{t+N-1} \Bigg( & \bar{v}(k)\bar{\rho}_s(k)\frac{\cos(\bar{e}_\psi(k))}{\bar{\rho}_s(k) - \bar{e}_y(k)} + \frac{\cos(\bar{e}_\psi(k))\bar{\rho}_s(k)\bar{v}(k)}{(\bar{\rho}_s(k) - \bar{e}_y(k))^2}(e_y(k) - \bar{e}_y(k)) \\ & - \frac{\sin(\bar{e}_\psi(k))\bar{\rho}_s(k)\bar{v}(k)}{\bar{\rho}_s(k) - \bar{e}_y(k)}(e_\psi(k) - \bar{e}_\psi(k)) + \bar{\rho}_s(k)\frac{\cos(\bar{e}_\psi(k))}{\bar{\rho}_s(k) - \bar{e}_y(k)}(v(k) - \bar{v}(k)) \Bigg) \end{aligned} \tag{46}$$

**Soft constraints** The cost function will tend to drive the car near from the road boundaries in order to maximise progress as illustrated in Figure 5. This can generate issues if the prediction model is unable to predict the future steps accurately, as the car can find itself on the boundary of the road and a hard constraint such as $\forall k \in \mathbb{N}, -L_{road} \leq e_y(k) \leq L_{road}$ (the car should not leave the road) could prevent the optimisation problem from generating any solution as a hard constraint would be violated at step $k = 0$.
To solve this issue, it is necessary to add soft constraints to the formulation. In [10], a soft constraint is added to the MPCC, penalizing the deviation from the centerline by using a cost function such as:

$$J(u, x_t) = J_l(u, x_t) + \sum_{k=t}^{t+N-1} \epsilon e_y(k) \tag{47}$$

where $\epsilon$ is a user-defined coefficient preventing the car from being too near from road boundaries. This approach is described as sub-optimal since a high value for $\epsilon$ would prevent the car from adopting an optimal driving, by forcing it to steer towards the centerline. And a low value would cancel the benefits of a soft constraint.

A different approach is chosen here. If is desirable to penalize when the car is near from the road boundaries but let the controller optimize the trajectory in other cases. Let us choose a safe road semi-width $L_{safe} < L_{road}$ where the controller is free to operate without soft constraints.

$$\begin{cases} J(u, x_t) = J(u, x_t) & \text{, if} \quad |e_y(0)| < L_{safe} \\ J(u, x_t) = J(u, x_t) + \sum_{k=t}^{t+N-1} \epsilon e_y(k) & \text{, if} \quad |e_y(0)| > L_{safe} \end{cases} \tag{48}$$

Such an approach allows the controller to steer the car towards the optimal trajectory in any cases but when "risky" trajectories such as near off-road ones are involved. In thatcase, the cost function penalizes for near-road-limits trajectories using the second formulation of $J$.

**Complete formulation** The full formulation can now be expressed.

$$\begin{aligned} \min_u \; & J(u, x_t) \\ \text{sub to } & x_t = x(t) \\ & x_{k+1} = A(x_k - \bar{x_k}) + B(u_k - \bar{u_k}), \quad \forall k \in [t, \; ..., \; t+N-1] \\ & |e_y(k)| \leq |e_y(k) + \epsilon_r L_{road}|, \qquad \forall k \in [t, \; ..., \; t+N] \\ & -\pi/2 \leq e_\psi(k) \leq \pi/2, \qquad \forall k \in [t, \; ..., \; t+N] \\ & a_{min} \leq a_k \leq a_{max}, \qquad \forall k \in [t, \; ..., \; t+N] \\ & \kappa_{min} \leq \kappa_k \leq \kappa_{max}, \qquad \forall k \in [t, \; ..., \; t+N] \end{aligned} \tag{49}$$

It is necessary to note that the constraint on $e_y$ has been softened: there is no longer any hard constraint preventing the car from leaving the road. However, if the controller steers the car over $L_{safe}$, the cost function penalizes this behaviour and encourages the controller to keep the car on the road. Such an approach requires to tune $\epsilon$ and $L_{safe}$ in order to obtain to ensure safety without hindering the controller too much. The coefficient $\epsilon_r$ allows to virtually trick the controller into thinking the car is near the border of the road, but not over it. So feasibility is ensured while keeping the controller aware that there is danger ahead and that keeping on steering in that direction is not an option.

**Practical considerations** The matrices $A$ and $B$ defined in the equation (44) depends on both $\rho_s$ and $\psi_s$, that are respectively the curvature radius and curvature of the reference track. They are evaluated at the nearest point on the reference track relatively to the vehicle position, which is the orthogonal projection of the vehicle position to the reference track. As exposed in the 1-layer MPC formulation from [10], a projection operation comes

down to solving an optimization problem, so it is not possible to estimate the projection point in that way since we would then have two nested optimization problems.

We take inspiration from [10]'s 1-layer MPC by estimating the projection point using the vehicle progress:

1. At the beginning of the race, an orthogonal projection of the vehicle position to the reference track is performed. This gives the desired projection point at $t = 0$ denoted as $x_s(0)$, and $(e_y(0)\ e_\psi(0))$. Since we can know the initial speed $v_0$, we thus obtain $x_0$.

2. At each step, the optimal input $u^*$ is obtained by solving (49). This input is then fed to (43) which provides us with an estimation of $x(t+1) = (e_y(t+1)\ e_\psi(t+1)\ v(t+1))$. This is then used to compute $\Delta s(t)$ using (37), which gives us $x_s(t+1) = x_s(t)+\Delta s(t)$.

3. Using an offline sampled representation of the track, the previous $x_s(t+1)$ can be used to find the next $\rho_s$ and $\psi_s$.

# 4 Experimental setup

In order to implement the previously derived MPC controller, a simulation environment has been developed in MATLAB. It consists in a car racing on an user-defined track using an user-designed controller for throttle and steering control.

## 4.1 Setup

### 4.1.1 Reference track

The reference track is the centerline of the track defined by the user as the racing road. It can be generated by fitting a spline to a set of waypoints.

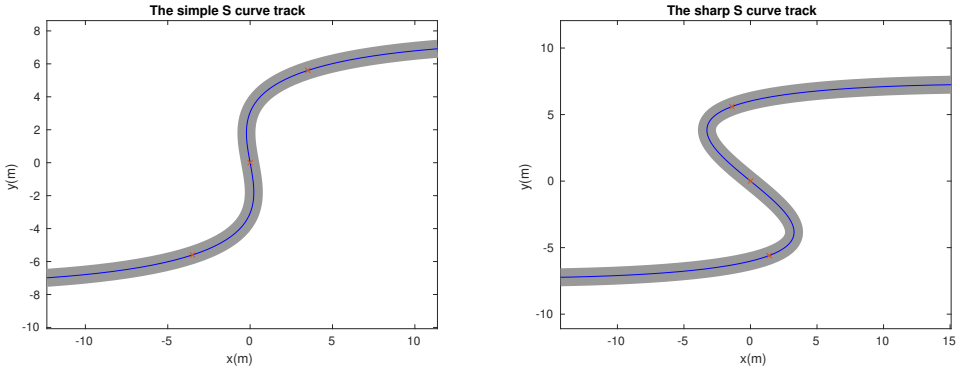


Figure 18: Scheme illustrating the reference track generation for 2 different setups: and S-curve and a 8-shaped track, the yellow crosses are the waypoints

This method allows the user the generate a wide range of tracks with specific curvature and shapes. A starting point and a finish line need to be set too.
For the controller to use this reference information, the generated track needs to be sampled. A constant step of 10 $cm$ is being used in our case. This data is stored and made available to the controller, in order to enable the computation of $\rho_s$ and $\psi_s$ as detailed in Section 3.4.3.
The track length is around $25m$ and its total width is $80cm$. The latter dimension has been computed from actual racing tracks (such as Nürburgring) whose width is usually around $8m$, this size has been divided by 10 in order to adapt it to our platform: the F1tenth.

It is chosen to focus on the two different tracks shown in Figure 18. The second offers significantly sharper curves than the first one in order to represent a more complex racing challenge for the controller. In both cases, the car starts in the bottom left of the track map and must reach the upper right part.

### 4.1.2 The car

The car is the subject to be steered by a user-defined controller. It is simply represented by a white rectangle whose dimensions are fitted to the width and the length of the physical car that is being simulated.

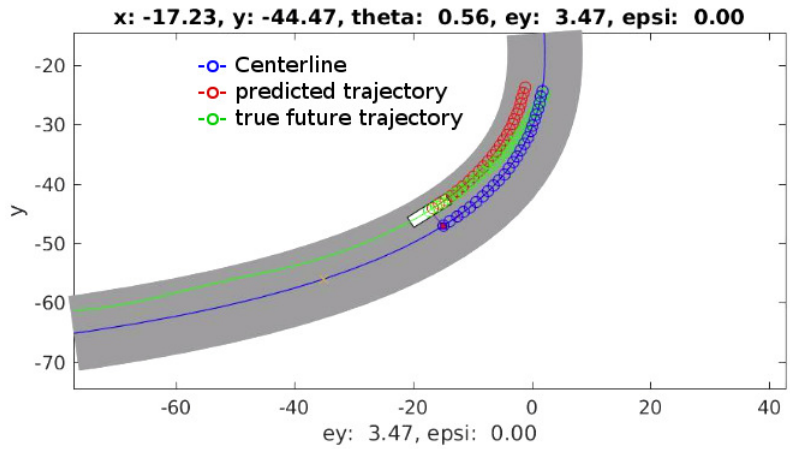


Figure 19: A visual representation of the car (white rectangle) in the simulation environment.

At each simulation step, the next car's position is determined by feeding the computed input $u_0$ to a physical model of the actual car that is being simulated. This model can be kinematic or dynamic (with Fiala tire model). This model is non-linear and is therefore different from the predicition model shown in (44).

The green line behind the car represents the history of its previous positions, while the blue line represents the centerline.
The red circles that spans in front the car are showing the predicted $N$ next steps according to the MPC controller, these are the future positions of the car that are used in as linearization points $[\bar{x}_t,\ ...,\ \bar{x}_{t+N+1}]$.
The green circles are the actual positions the car would occupy if the current set of inputs is to be applied, they are computed by applying the input sequence to a vehicle model that is more complex than the one used for prediction. The controller is unaware of these "true" positions and they are simply computed for the user information: the more they differ from the predicted states (in red), the less the prediction model is accurate.
Ultimately, the blue circles are the reference states corresponding to the predicted ones. They are used to get the $\psi_s$ and $\rho_s$ data at each step and are obtained by orthogonal projection of the predicted states towards the centerline.

### 4.1.3 The optimization problem and solver

The Matlab toolbox Yalmip [25] is used to provide the user with an abstraction layer for optimization problem formulation and solving. The library has an easy syntax that enables the control designer not to worry about the lower level problem solving.
The optimization problem to be solved is (49). Since the function $C$ defined as:

$$C : (Z_1,\ ...,\ Z_{N-1}) \rightarrow -\sum_{k=0}^{N-1} (a + b_1(e_y(k) - c_1) - b_2(e_\psi(k) - c_2) + b_3(v(k) - c_3)) \quad (50)$$

is linear $\forall a, b_1, b_2, b_3, c_1, c_2, c_3 \in \mathbb{R}$ and the constraint sets are convex, then linear programming can be used to solve (49). This is a computationally cheap process compared to non-convex problem solving.

## 4.2 Methodology

In the experiment one, we change the simulation model to a non-linear kinematic model such as (15). This will give us insights about the way the controller reacts to errors between the prediction model and the simulation model. The goal of this setup is to obtain a proof of concept: the controller needs to steer the car in a way that maximises its progress along the centerline. It is expected that the car behaves like described in [21]. To obtain that proof of concept, the boundary values $\kappa_{min}$ and $\kappa_{max}$ are chosen low in order to mimic the high-speed dynamic model limitations in terms of trajectory curvature.
Ultimately, we set the simulation model to be a dynamic bicycle model with Fiala tire model (22). The tire parameters are set to fit the F1tenth vehicle.
The last experiment to be run confronts several controllers in the 8-shaped track: our custom Progress Maximization MPC (PM-MPC), and the Hierarchical Receiding Horizon Controller (HRHC) from [10].

| **Experiment** | **track** | **simulation model** | **controllers** |
|---|---|---|---|
| 1a | Simple S | Nonlinear kinematic model | PM-MPC, HRHC |
| 1b | Sharp S | Nonlinear kinematic model | PM-MPC, HRHC |
| 2 | Simple S | Nonlinear dynamic model, Fiala tires | PM-MPC, HRHC |

Table 1: Summary of the experiments to be presented

## 4.3 Results

The experiment one is first performed and proves that the controller is able to steer the car in a way that maximizes the progress along the track. This includes exhibiting behaviours described in [21] while not leaving the road and always keeping the speed as high as possible.

In all experiments, the car starts with an initial speed of $12m.s^{-1}$, next to the centerline. Its speed is capped by $18m.s^{-1}$ (which is the F1tenth top speed). Unless specified otherwise, the predicition horizon $N$ is 25 steps.

### 4.3.1 Experiment 1 - Nonlinear kinematic simulation model

In this experiment, the simulation model is the nonlinear kinematic bicycle model (18) with an extra constraint on the steering rate. This one has been set to a low value in order to mimic the dynamic model behaviour and force the controller to adopt a "realistic" trajectory.

The Figure 20 illustrates the result of the experiment 1a. We can observe that the controller manages to steer the car without leaving the road and completes the race in 21.30 seconds.

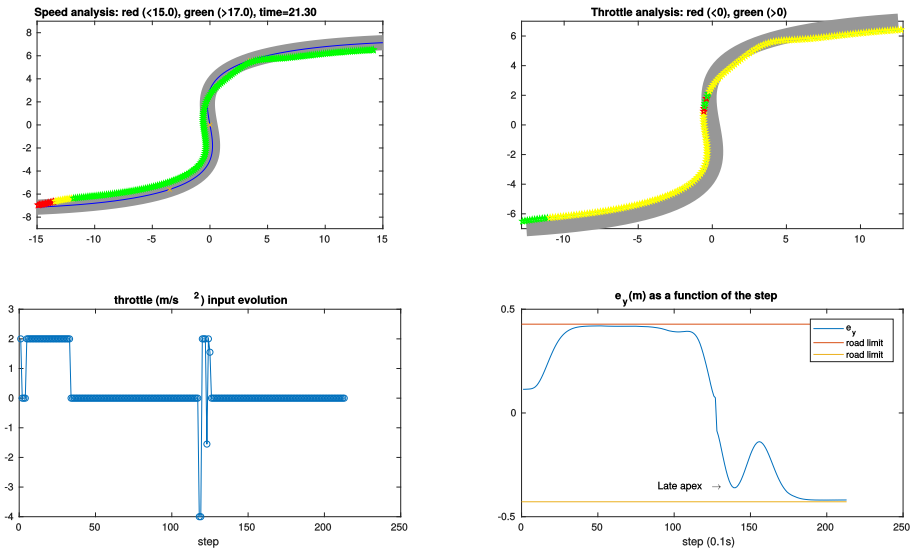


Figure 20: PM-MPC performance on the Simple S curve (experiment 1a)

In the first curve, the controller manages to always keep the car near the interior bound-

ary (which is the fastest path) without leaving the track and by keeping the speed as high as the top limit. However, when the second curve is about to be reached, the controller slightly reduces the speed and adopt a behaviour that resembles the practices described in [21]. We can notice in the Figure 20 that the controller steers the car towards the border of the internal part of the road in a corner, aiming for the apex. In the second corner, the car is near from the road limits ($e_y = -40\ cm$) just after the middle of the corner, resulting in a late apex, which is desirable. The trajectory is still not perfect as the apex could have been later, resulting in a significant "overshoot" that brings the car near from the centerline after the second curve. It is also noticeable that the car come very close to the road limits (the wheels end up around $2cm$ away from the boundary). This is due to a fine tuning of the soft constraints related to this aspect. A disavantage of this technique is that badly tuned value could easily let the car go offroad.

The experiment on the sharp S curve (1b) offers similar results, as the car is now forced to take advantage of racing techniques to pass both curves. In the first curve, an early apex results from the controller choices, which is not desirable since it makes the car lose time and increases the risks to leave the road. From the fourth curve in 21, we can see that the controller attempted to leave the interior lane before the turning point but lacked time. This could be due to either a low prediction horizon or a wrong prediction model that becomes inaccurate when the curvature radius is low.

The second curve offers better results as a late apex is obtained and off-road is avoided, allowing the car to keep its high speed.

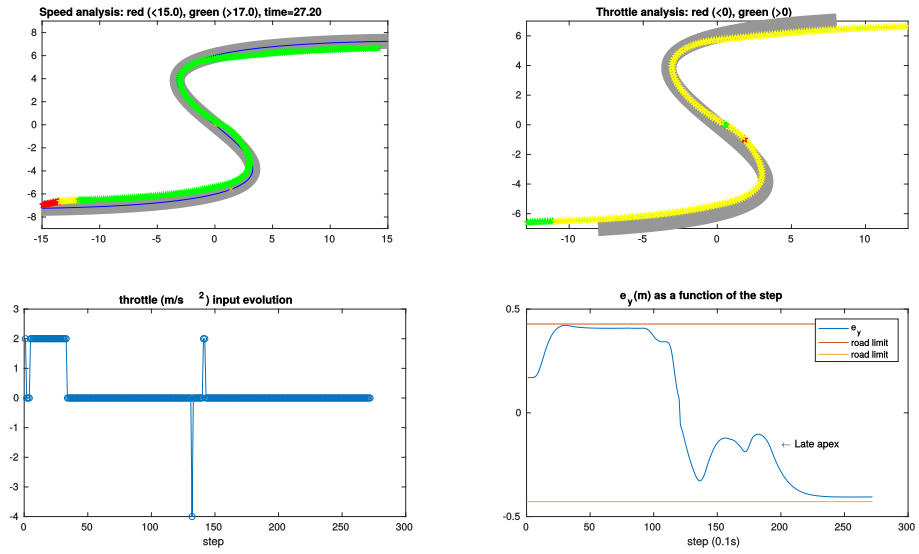


Figure 21: PM-MPC performance on the Sharp S track (experiment 1b)

### 4.3.2 Experiment 2 - Nonlinear dynamic simulation model

The simulation model now shifts to the dynamic bicycle model (22). The PM-MPC controller is first tested out-of-the box (i.e., without changing any parameter from the previous experiment), but fails to complete the race as its relatively high speed and trajectory curvature make the car lose adherence and drift before going off-road. So the controller needs to be adapted.
A simple solution is to add constraints to both car speed and trajectory curvature:

$$\begin{aligned} v &\leq g(\kappa_s) \\ \kappa &\leq h(v), \end{aligned} \tag{51}$$

With $h, v : \mathbb{R} \leftarrow \mathbb{R}$. Thus, the controller cannot generate a trajectory with both high curvature and high speed, limiting the drift risk. The centerline curvature is denoted as $\kappa_s$ and the car curvature input as $\kappa$. Such a controller is implemented for the experiment 2, the result is shown in Figure 22.

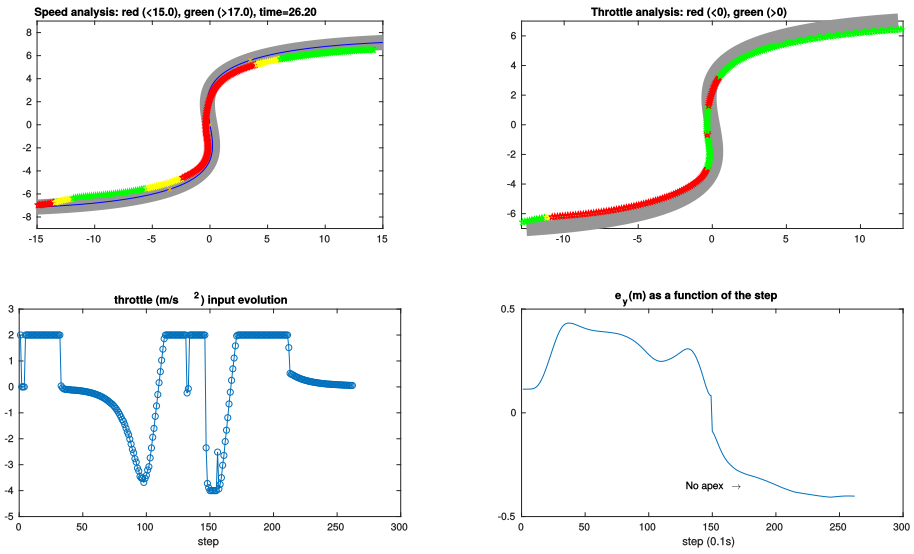


Figure 22: PM-MPC performance on the Simple S track with dynamic simulation model

The car can now remain on the road, but the price to pay is a lower speed. As a result, the racing practices described in [21] are not observable anymore, resulting in a suboptimal driving. In this experiment, the consequences of having a inaccurate predicition model are noticeable. The Figure 23 shows how the controller is tricked into thinking the car has more adherence than what it actually has.

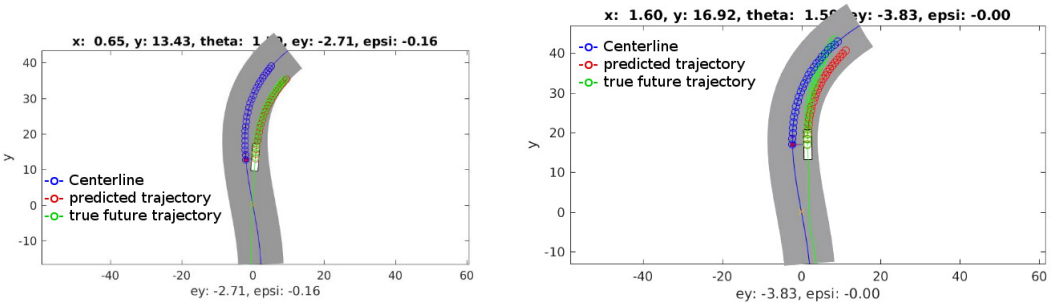


Figure 23: Picture of the car behaviour in the second corner: for kinematic and dynamic cases (experiment 2)

This dynamic simulation model case is the one that "matters" since it is meant to reproduce the car behaviour with the maximum fidelity, while the kinematic case is virtual. The issue illustrated in Figure 23 shows that the prediction model currently in use is not

accurate enough to account for the complex dynamics of the car, impacting the racing performance. A solution to this problem is to derive a better prediction model that would integrate drift behaviours.

### 4.3.3 Comparison of controllers

Let us now implement and test the HRHC controller in the same environment. As shown in Figure 24, the HRHC exhibits a more conservative behaviour than the PM-MPC: before sharp curves, the controller has a tendency to slightly decrease the speed of the car. However, the racing time is comparable to the PM-MPC in the same environment: 21.1$s$.

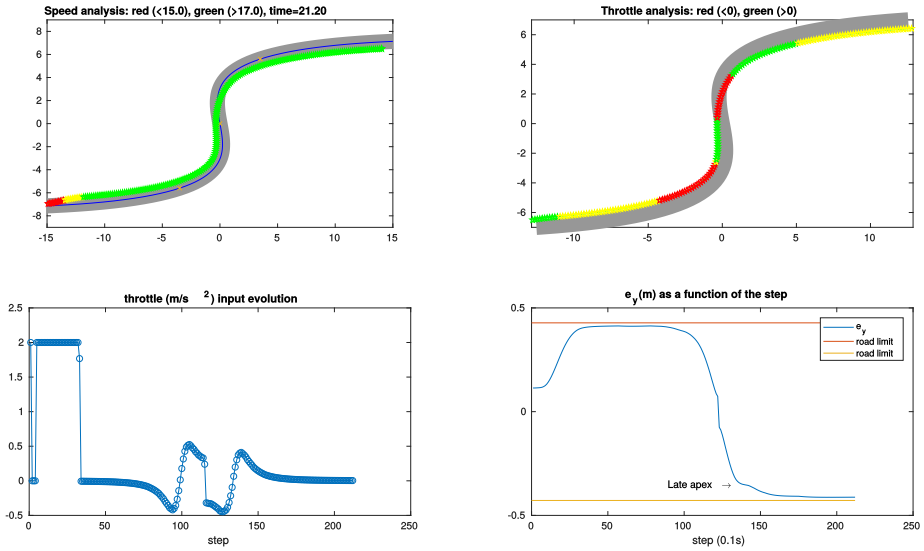


Figure 24: HRHC performance on the Sharp S track

Such a behaviour can be explained by the constraint that each planned path needs to have the same curvature over the prediction horizon, limiting the range of feasible trajectories. This drawback was also evocated by the original author of the controller [10]. Besides that, the overal trajectory resembles the PM-MPC's one, and also exhibits racing practices.

The Figure 25 offers a zoomed-on-second-corner comparison between these two controllers. While the PM-MPC reaches a medium-late apex and overshoots towards the centerline, the reduced speed of the HRHC allows for no overshoot. Theoretically, the best racing line should be obtained with an in-between behaviour, with a later apex.

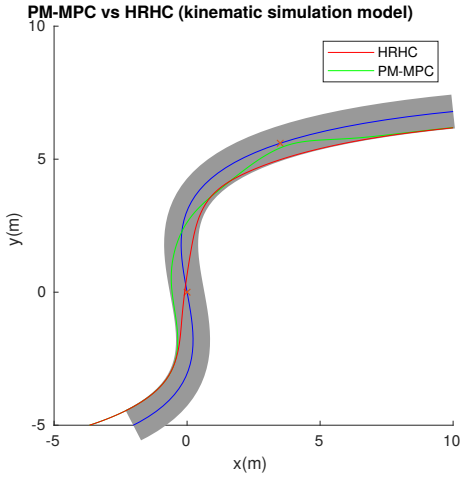


Figure 25: HRHC vs PM-MPC with kinematic simulation model

The Figure 26 illustrates the same comparison with a dynamic simulation model. This time, there is little difference between the two trajectories. The racing times are almost equivalent between the two controllers, the PM-MPC being $0.2s$ faster. This could be explained by the constraints (51), which can hinder the controllers' decisions by lowering the allowed speed. Both trajectories are thus suboptimal. In that setup, the ability of the PM-MPC to generate more complex trajectories is hindered by the low speed and curvatures that the extra constraints enforce. In that case, it could be expected that the performance between the controllers would be very similar.

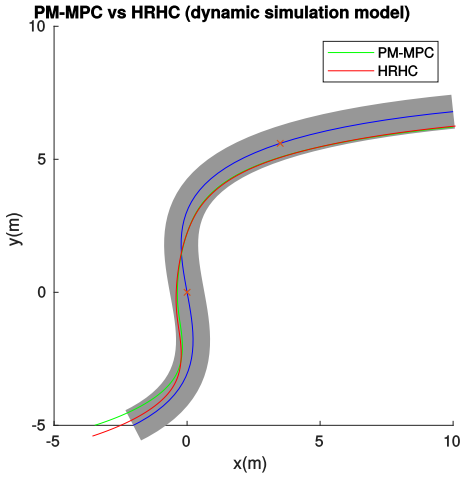


Figure 26: HRHC vs PM-MPC with dynamic simulation model

The Figure 27 illustrates the difference between the PM-MPC with $N = 25$ and $N = 50$. In both cases, the speed profile is the same and the PM-MPC completes the race in $20.70s$, which is slightly faster than the shorter-sighted version. It seems that increasing the prediction horizon allows for a later apex when the prediction horizon is increased. Since the controller has a larger prediction horizon, the corners are taken into account earlier, which allows for more accurate planning.

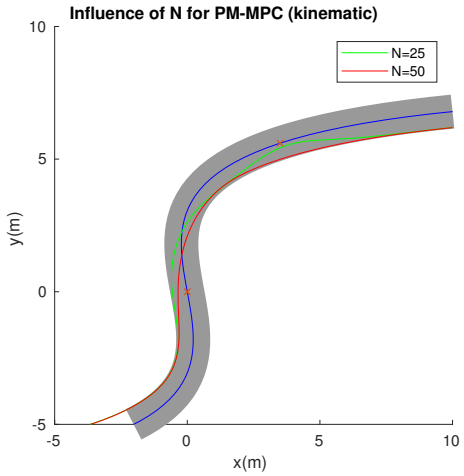


Figure 27: PM-MPC with $N = 25$ vs PM-MPC with $N = 50$ with kinematic simulation model

The same experiment has been done with HRHC, however it has been discovered that increasing $N$ resulted in lower performances. Infact, the same conclusion has been made in the original paper [10]: since the planned path's curvatures need to be constant over the prediction horizon, increasing $N$ leads to less feasible trajectories as it is harder to keep the same curvature over a larger time span. The PM-MPC does not seem to suffer from that problem. Moreover, since the PM-MPC uses linear programming to solve its optimization problem, using larger $N$ should not result in high CPU load. Also, it could be possible to increase the performance of the PM-MPC with $N = 25$ by using a proper final cost.

In addition to this result, the PM-MPC framework offers the advantage of being more holistic and the lack of performance difference between the HRHC and PM-MPC could be changed if the predictions model would integrate more realistic dynamics of the car.

# 5 Conclusions and future work

In this study, the topic of Model Predictive Control for racing applications has been explored. The state of the art revealed that several MPC formulations already existed and that they has their own assets and drawbacks.
Based on that, a custom MPC formulation for racing has been derived based on the progress maximisation along the centerline. It has the advantage of being a 1-layer formulation that offers an intuitive approach to the racing problem. Also, after having linearized both the prediction model and the cost function, the optimization problem can be solved by simple linear programming.
However, the current prediction model is lacking dynamics information about the car, particularly the tires. Because of that, the controller's performance is severely reduced when using a simulation model integrating tire dynamics.

A way to go further with this approach could be design a set of constraints that would ensure that the car always remain in the kinematic model case by only allowing lower steering angles with the increase of the speed. This would requires to have $\kappa_{max}$ and $\kappa_{min}$ as functions of $v$ and $u$. Also, it would be interesting to add the slip angles from the dynamic model to the problem states: as they indicate when the tires are losing adherence, it would be possible to include a constraint of these states to ensure that the car never drifts.

Since the PM-MPC's optimization problem can be solved by linear programming, its implementation should be possible on low-power CPUs, and with no need for expensive commercial non-convex solvers. A simple sampling of the track should be performed beforehand, and the state of the vehicle needs to be known at each step. This can be achieved by using motion capture setups or relying on sensor data to estimate the current state.

# References


[1] Tim Stevens. Volvo details autonomous drive me cars, on sale in 2017. *cnet.com*, February 2015.

[2] Rhett Jones. Tesla to take its biggest step toward fully autonomous cars tomorrow. *gizmodo.com*, August 2017.

[3] Angela Swartz. Delphi's autonomous car proof of concept: driving itself across the u.s. *Silicon Valley Business Journal*, March 2015.

[4] Benjamin Zhang. Autonomous cars could save the us 1.3 trillion dollars a year. *Business Insider*, September 2014.

[5] Sandra Rosenbloom Brian Tefft Tim Triplett, Rob Santos. American driving survey: 2014 – 2015. *AAA foundation for traffic safety*, 2016.

[6] Byungkyu Park Joyoung Lee. Development and evaluation of a cooperative vehicle intersection control algorithm under the connected vehicles environment. *IEEE transactions on intelligent transportation systems, vol. 13, no. 1, March 2012*, 2012.

[7] Matthew Barth. Real-World CO2 Impacts of Traffic Congestion. *University of California Transportation Center University of California*, March 2008.

[8] Stephen Edelstein. Toyota autonomous vehicles will soon take part in hazardous driving tests. *The Drive*, October 2017.

[9] SAE international. Automated driving, levels of driving automation are defined in new SAE international standard J3016. September 2016.

[10] Alexander Liniger, Alexander Domahidi, and Manfred Morari. Optimization-based autonomous racing of 1:43 scale RC cars. *Optimal Control Applications and Methods*, 36(5):628–647, 2015.

[11] Robin Verschueren, Stijn De Bruyne, Mario Zanon, Janick V. Frasch, and Moritz Diehl. Towards time-optimal race car driving using nonlinear MPC in real-time. In *Proceedings of the IEEE Conference on Decision and Control*, volume 2015-Febru, pages 2505–2510, 2014.

[12] Tamás Keviczky, Paolo Falcone, Francesco Borrelli, Jahan Asgari, and Davor Hrovat. Predictive control approach to autonomous vehicle steering. *2006 American Control Conference*, pages 4670–4675, 2006.

[13] Florent Altché, Philip Polack, and Arnaud de La Fortelle. A Simple Dynamic Model for Aggressive, Near-Limits Trajectory Planning. *IEEE IV 2017 conference*, June 2017.

[14] Jason Kong, Mark Pfeiffer, Georg Schildbach, and Francesco Borrelli. Kinematic and Dynamic Vehicle Models for Autonomous Driving Control Design. pages 2–7, July 2015.

[15] F Borrelli, P Falcone, T Keviczky, J Asgari, and D Hrovat. MPC-based approach to active steering for autonomous vehicle systems. *International Journal of Vehicle Autonomous Systems*, 3(2/3/4):265, 2005.

[16] Krisada Kritayakirana and J. Christian Gerdes. Autonomous vehicle control at the limits of handling. *International Journal of Vehicle Autonomous Systems*, 10(4):271–296, 2012.

[17] Jarrod M Snider. Automatic Steering Methods for Autonomous Automobile Path Tracking. *Work*, (February):1–78, 2009.

[18] Mogens Graf Plessen, Pedro F. Lima, Jonas Martensson, Alberto Bemporad, and Bo Wahlberg. Trajectory Planning Under Vehicle Dimension Constraints Using Sequential Linear Programming. April 2017.

[19] Mikael Johansson. Lecture notes in EL2700 model predictive control. *KTH (Kungliga Tekniska Högskolan, Stockholm)*, 2016.

[20] Jonathan P. How Mark Campbell, Magnus Egerstedt. Autonomous driving in urban environments: approaches, lessons and challenges. *The Royal Society Publishing*, September 2010.

[21] Brian Beckman. The Physics of Racing. April 1991. available at: http://phors.locost7.info/contents.htm.

[22] Lennart Ljung Torkel Glab. *Control theory, multivariable and nonlinear methods.* Taylor and Francis, 2000.

[23] Pedro F. Lima. Predictive control for autonomous driving with experimental evaluation on a heavy-duty construction truck. *licenciate thesis*, pages 46–49, May 2016.

[24] Emilio Frazzoli Efstathios Velenis and Panagiotis Tsiotras. On steady-state cornering equilibria for wheeled vehicles with drift. 2009.

[25] J. Löfberg. Yalmip : A toolbox for modeling and optimization in matlab. In *In Proceedings of the CACSD Conference*, Taipei, Taiwan, 2004.

TRITA 2017:175
ISSN 1653-5146