Final Degree Thesis

Bachelor's degree in Industrial Technology Engineering (GETI)

Motion Planning and Control Pipeline for a Formula Student Autonomous Vehicle

January 17, 2024

Author: Oriol Martínez Fité

Director: Vicenç Puig Cayuela

Presentation: 29/01/2024

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

Acknowledgements

First of all, I would like to thank my teammates at BCN eMotorsport, whose collaborative effort and countless hours of work have made the development of the CAT15X race car possible. The dedication and passion exhibited by each member of our team have truly made this project possible.

I would also like to make an special mention to the Electronics, Perception and Control members with whom I've had the privilege to work side by side with the main objective to enhance the driverless performance of CAT15X. Furthermore, I would like to extend my sincere appreciation to the Chassis department, for their true commitment to the team when we needed them the most.

Lastly, I want to express my gratitude to my mentors and fellow team members, Antoni Salom and Oriol Gorriz, who provided essential guidance and support during my initial year at BCN eMotorsport.

I would also like to thank my advisor Vicenç Puig for his wise advises and recommendations throughout the 2022-23 Formula Student season.

Finally, this thesis is dedicated to my family for their incredible support during my two years long Formula Student experience.

Abstract

In this thesis, a comprehensive high-level autonomous driving control pipeline, along with an optimization-based motion planning algorithm, is proposed, detailed, and implemented into BCN eMotorsport's Autonomous Systems Control software stack.

The main goal of the presented algorithms is to improve the competitiveness of BCN eMotorsport's autonomous vehicle, the CAT15X, reaching levels of performance and reliability never seen throughout the team's history.

In section 3 a **Nonlinear Model Predictive Controller** is proposed in order to handle the lateral behaviour of the vehicle using the curvature-based dynamic bicycle model presented in section 2.3.

In section 4 an **Adaptive Cruise Controller** is proposed in order to handle the longitudinal behaviour of the car, closely following the self-computed velocity profile defined in section 4.2.

Finally, an offline **Optimization-based Motion Planning** algorithm is proposed in section 5. This algorithm utilizes the dynamic bicycle model, introduced in section 2.3, as its predictive basis.

The implementation procedure as well as the simulation results for all the presented algorithms are also specified. In addition, the real testing performance of the proposed control pipeline is detailed in section 6.

The complete software designed and implemented within this thesis can be found in:

- Lateral MPC
- Cruise Controller
- Trajectory Optimizer

Contents

Acronyms

AS Autonomous Systems

FS Formula Student

FSG Formula Student Germany competition

FSS Formula Student Spain competition

CV Internal Combustion Vehicle

EV Electric Vehicle

HY Hybrid Vehicle

DV Driverless Vehicle

DC Driverless Cup

UPC Universitat Politècnica de Catalunya

ETSEIB Escola Tècnica Superior d'Enginyeria Industrial de Barcelona

ETSETB Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona

4WD Four Wheel Drive

MPC Model Predictive Control

NLOP Non Linear Optimization Problem

CAN Controller Area Network

PU Processing Unit

PID Proportional-Integral-Derivative controller

CPU Central Processing Unit

ROS Robotic Operating System

ECU Electronic Control Unit

ODE Ordinary Differential Equation

CoG Center of Gravity

SLAM Simultaneous Localization And Mapping

API Application Programming Interface

Nomenclature

$\alpha_{F/R}$	Slip angle at Front/Rear wheel $[rad]$
β	CoG slip angle $[rad]$
δ	Steering angle in wheel frame $[rad]$
κ	Curvature $[m^{-1}]$
μ	Heading with respect to the reference path $[rad]$
θ	Heading of the car on global frame $[rad]$
ζ	Slack variable $[-]$
a_x	Longitudinal acceleration $[\frac{m}{s^2}]$
a_y	Lateral acceleration $[\frac{m}{s^2}]$
$D_{L,R}$	Orthogonal distance from CoG to the Left/Right track limits $[m]$
F_m	Motor force $[N]$
F_x	Longitudinal force $[N]$
$F_{y,F/R}$	Lateral Front/Rear force $[N]$
g	Gravitational acceleration $[\frac{m}{s^2}]$
I_z	Moment of inertia over the z-axis $[kgm^2]$
L	Longitudinal distance from CoG to the furthest corner of the car $[m]$
$l_{F/R}$	Length from CoG to the Front/Rear axle $[m]$
m	Vehicle's mass without driver $[kg]$
M_z	Moment over the z-axis $[Nm]$
n	Orthogonal distance to the reference path $[m]$
s	Progress along the reference path $[m]$
v_x	Longitudinal velocity on vehicle frame$[\frac{m}{s}]$
v_y	Lateral velocity on vehicle frame $[\frac{m}{s}]$
W	Lateral distance from CoG to the furthest corner of the car $[m]$
w	Yaw rate $[\frac{rad}{s}]$
x	Position on the global x-axis $[m]$

y Position on the global y-axis $[m]$

List of Figures

List of Tables

1 Introduction

1.1 Formula Student Competition

Formula Student is a prestigious global engineering competition that tasks university students with the design, construction, and racing of Formula-style race cars. The competition serves as a platform for students to apply their theoretical knowledge to real-world engineering challenges while promoting innovation and practical skills. Teams of university students participating in Formula Student are required to design every aspect of the race car, including the chassis, suspension, powertrain, aerodynamics, electronics, build in software, etc. These designs must adhere to strict safety regulations and budget constraints.

Figure 1: Formula Student competitions around the globe.

The competition is split into Internal Combustion Vehicles (CVs), including Combustion Hybrid Vehicles (HYs), and Electric Vehicles (EVs). Vehicles of both classes can take part in an additional Driverless Cup (DC), which involves providing the car with full autonomous capabilities so it can navigate through cone-delimited tracks without any external communication.

The competition includes both static and dynamic events. Static events evaluate the teams' engineering knowledge and project management skills. These events include presentations on the technical design of the single-seater, cost reports, and a business plan. On the other hand, dynamic events test the car's performance, agility and reliability under racing conditions through the following events: acceleration, skidpad, autocross, endurance and trackdrive races.

The overall score for EV (& CV) and DC competitions are:

	Discipline	EV & CV	DC
Statics	Business Plan	75 points	–
	Cost and Manufacturing	100 points	–
	Engineering Design	150 points	150 points
Dynamics	Acceleration	50 points	–
	Acceleration Driverless	75 points	75 points
	Skidpad	50 points	–
	Skidpad Driverless	75 points	75 points
	Autocross	100 points	–
	Autocross Driverless	–	100 points
	Endurance	250 points	–
	Efficiency	75 points	–
	Trackdrive	–	200 points
Overall		**1000 points**	**600 points**

Table 1: Maximum awarded points.

In summary, Formula Student is a prestigious international competition that challenges university students to design, build, and race Formula-style race cars, providing a unique opportunity to apply theoretical knowledge, foster innovation, and gain practical engineering skills that can significantly benefit their future careers. There exist many competitions held all over the world, although most of the competitions take place in Europe. The most important FS competition called Formula Student Germany (FSG) takes place in the Hockenheimring, Germany.

1.1.1 Driverless dynamic events

The dynamic events in Formula Student are the on-track events where the performance of the teams' vehicles is tested. Every dynamic event consists of a race against the clock as wheel-to-wheel races are absolutely forbidden. Here are the details of each dynamic event involving driverless vehicles:

Autocross: This event tests the car's dynamic ability in a one-lap sprint on an unknown track. The car must finish the event as fast as possible while avoiding any collisions with the cones placed on the track. This event happens to be the most relevant challenge for driverless cars since it requires a reliable cone detection, a fast path planning algorithm, an accurate enough SLAM module and a robust control pipeline.

Trackdrive: This event evaluates the reliability and endurance of autonomous vehicles. The car must navigate a set course on the Autocross track until ten laps are finished. Here the precision of the SLAM module and the reliability of the control pipeline are extensively tested.

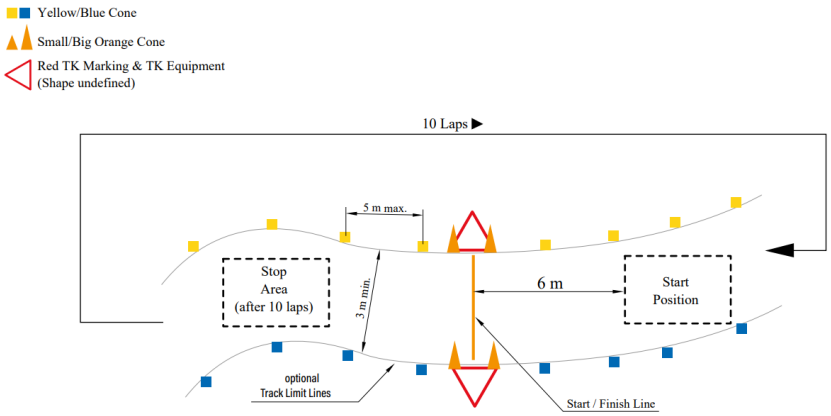


Figure 2: Trackdrive track layout.

Acceleration: The Acceleration event evaluates the car's acceleration in a straight line, from a standing start, over a distance of 75 meters. The performance of the car in this event is evaluated based on the time it takes to reach the finish line.

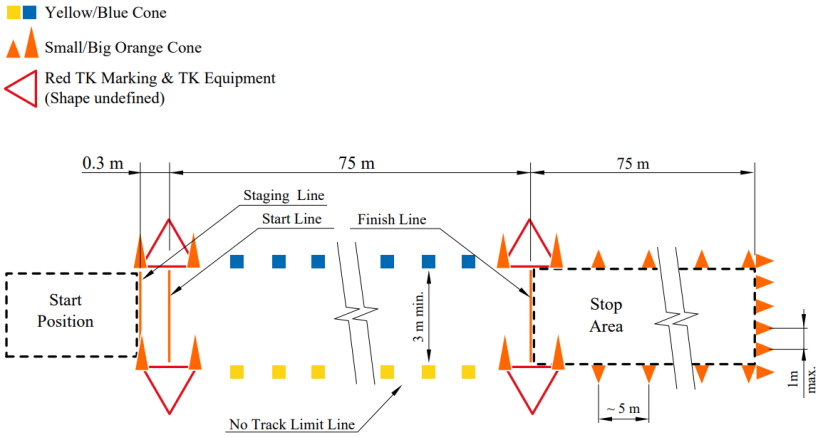


Figure 3: Acceleration track layout.

Skidpad: The Skidpad event measures the car's lateral grip on a flat surface while making a constant radius turn. The skidpad track consists of two pairs of concentric circles in a figure of eight pattern with fixed and previously known dimensions. The performance of the car in this event is evaluated based on the lap time mean of both second turns on each circle.

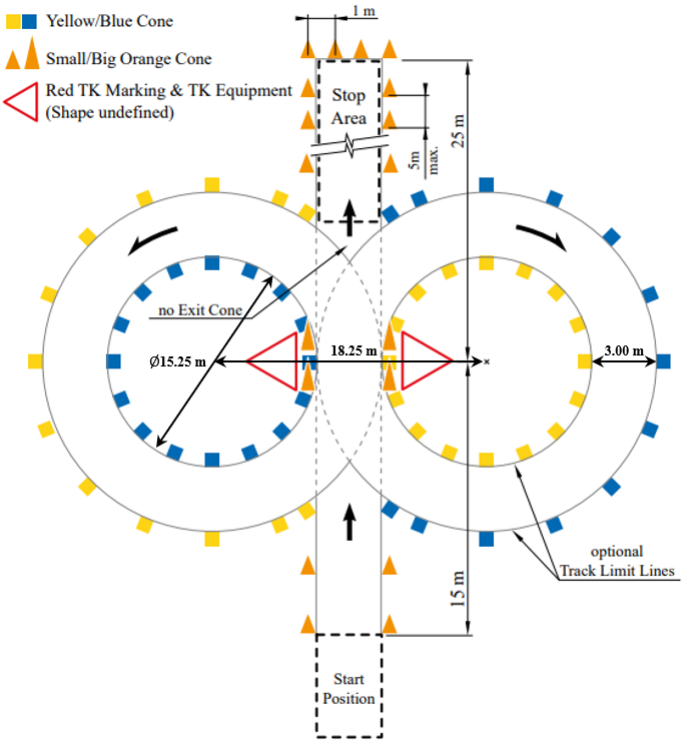


Figure 4: Skidpad track layout.

1.2 BCN eMotorsport

BCN eMotorsport is the Formula Student team from Barcelona, born out of the collaboration between UPC's ETSEIB and ETSETB engineering faculties. It was established in 2007 by the name of *ETSEIB Motorsport*. It started with a combustion prototype and have developed fourteen more cars since then, each with incremental improvements over the years.

In 2011, the team made a significant advancement by creating the first electric Formula Student car in Spain, marking an important evolution in their design concept. This was a testament to their innovative spirit and commitment to staying at the forefront of automotive technology.

In 2018, the team embarked on a new challenge by introducing 4WD capability and creating the first Formula Student autonomous car in Spain. This was a crucial step forward, reflecting the team's ability to adapt to the evolving automotive industry and leverage the latest technology.

The team's roster has expanded over the years, with students from various disciplines joining the team. This has allowed the team to bring in diverse perspectives and skills, enabling them to create some of the best cars in their history.

During the 2021-22 season the first fusion car was developed by the team in order to join together the manual and autonomous technologies in one and only single-seater. The first Formula Stu-

dent car in Spain with manual and driverless capabilities was born.

In summary, BCN eMotorsport is a dynamic team that has made significant strides in Formula Student, from creating a combustion prototype to developing an electric and autonomous car. Their continuous innovation and commitment to using the latest technology have set them apart in the competition.

Figure 5: CAT15X.

Figure 6: BCN eMotorsport team.

1.3 State of the art

The state of the art in Formula Student reflects a dynamic landscape where teams continuously push the boundaries of engineering and innovation. The competition serves as a fertile ground for the development of future automotive technologies, emphasizing sustainability, performance, and the integration of cutting-edge advancements.

Focusing on the driverless category, the best two teams historically have been *AMZ* from ETH Zürich and *KA-RacIng* from Karlsruhe Institut of Technology (KIT). However, in the last two seasons, teams such as *StarkStrom Augsburg e.V.* and *Chalmers FS* have been able to keep the pace and win the most important competition FSG in 2022 and 2023, respectively.

The previous achievements made in BCN eMotorsport's team related to driverless control and motion planning can be found in [1] and [2]. Without these prior developments the work presented in this thesis would have not been possible.

1.3.1 Driverless control

The main approach from the top teams in driverless control usually involves a Model Predictive Controller (MPC) which handles only the lateral behaviour of the car in the case of *KA-RacIng* or the complete behaviour, meaning steering and throttle commands, in the case of *AMZ*.

In [3] *AMZ* presented a novel non-linear MPC controller capable of handling both steering and throttle commands with outstanding performance during the 2019 season. This controller used a cartesian-based dynamic bicycle model, discretized by 50ms, with a solving time mean of 25Hz (40ms). In [4] *AMZ* improved their coupled control solution presenting a curvature-based non-linear MPC, which could run at 40Hz (25ms). This coupled approach reduced the high level AS control into one and only module, which could lead to a more compact pipeline. Moreover, the coupled controller takes into account the vehicle dynamics as a whole, unlocking better performance at high velocities ($\geq 15m/s$), where the non-linearities and correlations

between the lateral and longitudinal behaviour of the car are crucial. However, this control pipeline demands a longer testing time period in order to achieve the same level of reliability than simpler controllers.

In contrast with this approach, in [5] and [6] *KA-RacIng* developed a lateral MPC using a linearized dynamic bicycle model and followed a pre-computed velocity profile with a feedforward PI controller. This decoupled control solution enabled *KA-RacIng* to achieve a remarkable performance, winning the 2021 FSG Driverless competition. In particular, the lateral controller developed by *KA-RacIng* is formulated as a Linear Time Varying MPC (LTV-MPC) using a time-discrete linear vehicle model. The linear optimization problem is then solved by a Quadratic Programming (QP) solver. This fairly simple control pipeline has the advantage of simplifying the complexity behind the longitudinal behaviour of the car, excluding these dynamics from the predictive model used by the lateral controller, achieving a really small solve time (10ms).

Nevertheless, in 2022 season, *StarkStrom Ausburg e.V.* proved that fine-tuned basic controllers such as a Pure Pursuit could achieve really high levels of performance winning the FSG 2022 competition. The Pure Pursuit controller is a geometry-based path-tracking algorithm commonly used in robotics. It guides the vehicle along the desired path by continuously selecting a target point and calculating the steering angle via a closed geometrical formula, deduced from the kinematic bicycle model. With the proper amount of testing, this really simple approach can become a highly reliable lateral controller.

1.3.2 Motion Planning

Regarding FS motion planning approaches a clear distinction must be done between *online* and *offline* software. The former includes all the planning algorithms thought to run in real time, which usually involve receding horizon approaches, re-estimating the output trajectory for every new cone or landmark detected. The latter includes the motion planning algorithms thought to be executed before the car starts driving. These usually involve more CPU demanding tasks and depend on a previously computed initial solution. In these last approaches, the navigation problem is considered solved, and the main objective is to find faster paths for the next laps.

The most widely used approach in FS *online* trajectory planning starts with a discretization of the search space using Delunay triangulation or Voronoi diagram. Aferwards a growing tree structure of possible paths through the already discretized space is performed. Finally an heuristic-ponderated tree search is done in order to retrieve the best path exploiting the generally shared characteristics of FS tracks.

In [6] *KA-RacIng* presents a minimum curvature trajectory planning which runs in parallel to the middle line generation algorithm. *KA-RacIng* claims that a reduction in laptime of more than 10% can be achieved. In [4] *AMZ* presents a novel trajectory optimization algorithm which is able to compute the final path by solving a one-time prediction over the midline trajectory taking into account a dynamic bicycle model.

Notice that *online* strategies fall outside the scope of this thesis. The motion planning approach presented in section 5 is not meant to search for the correct navigation path but to find an optimal trajectory given the detected cones for the whole lap. Thus, an *offline* approach is presented in this thesis.

1.3.3 Outside Formula Student scope

In the field of autonomous driving, a revolutionary fusion of advanced technologies, artificial intelligence and advanced control strategies is propelling the automotive industry toward a future where vehicles navigate complex environments without human intervention. Central to this transformation are sophisticated sensors, such as Lidar and radar, that provide real-time data, coupled with powerful machine learning algorithms enabling vehicles to adapt to diverse driving scenarios. The state of the art in autonomous driving is a rapidly evolving narrative, marked by breakthroughs and a collective push toward a future where driving is not just a task but an immersive and automated experience for passengers.

In order to widen the scope outside of FS applications, the author has considered the following published works: Roborace high level control approaches in [7], Adaptive MPC strategies in [8] and [9], control look-ahead analysis in [10], minimum laptime for electric vehicles in [11] and, finally, optimization-based control for RC cars in [12].

1.4 Objectives

The main purpose of the algorithms presented in this work is to increase the competitiveness of the CAT15X single-seater, the BCN eMotorsport car for the 2022-23 season. The first objective is to obtain a longitudinal and a lateral controller capable of finishing all Formula Student dynamic events with competitive lap times. In second place, to obtain a motion planning module capable of unlocking optimal performance in the Trackdrive event. In order to fulfill this purpose, a set of requirements and specifications, which are detailed below, must be taken into account.

1.5 Specifications

From the Formula Student dynamic events definition, different specifications for Automatic Control and Motion Planning arise. The control pipeline presented in this work has to be able to drive the car through all the dynamic events stated in section 1.1.1 using a different set of parameters for each event, seeking optimal performance for all the events.

In fact, the lateral MPC presented in section 3 aims to achieve maximum performance in Autocross and Trackdrive where the inherent difficulties of both events bring up a major challenge for the control module. Thus, although the controller presented in section 3 must be able to finish all dynamics events, Autocross and Trackdrive performance will be engaged. Less complex solutions (e.g. Pure Pursuit or Stanley controllers) are found to be more appropriate for the rest of the dynamics events, where the fixed and known track dimensions result in more stable planned trajectories and fewer vehicle dynamics non-linearities.

The motion planning approach presented in section 5 is only thought to be used in the Trackdrive event as it depends on (i.e. takes as input) the detected cones positions for the whole track.

Finally, the Cruise Control presented in section 4 has to be able to achieve optimal performance in all the dynamic events. It will be used in all of them.

1.5.1 Control Specifications

The Control module must:

- Remain reliable for all the feasible velocities range, from **0** to **100** $\frac{km}{h}$.
- Run at a sufficient update rate ensuring real-time performance. A minimum loop rate of **40 Hz** is set.
- Follow a racing driving style. The main purpose of the control module is to be able to run as fast as possible keeping the car under control. An average velocity of **12** $\frac{m}{s}$ is set.
- Prioritize easy to tune formulations due to the limited testing schedule.

1.5.2 Motion Planning Specifications

The Motion Planning module must:

- Compute feasible trajectories based on vehicle's **dynamics limitations**.
- Output the resulting trajectory as a **continuous curvature** object. Thus, 2_{nd} derivative continuous spline representation will be used.
- Minimize curvature in order to obtain racing alike paths. In fact, **maximum progress rate** will be ensured, resulting in minimal curvature trajectories.

1.6 Requirements

The existing implementation requirements of the Control and Motion Planning solutions presented in this thesis are detailed in the following subsections.

1.6.1 Control Requirements

- **Processing Unit** powerful enough to execute the C++ implementation of the algorithms under a ROS framework instance on a real-time basis.
- **Actuators** precise enough to act on the car without major jitter. Corresponding to an autonomous steering system, an emergency braking system and continuous communication to the DC-AC inverters for setting torque setpoints.
- **Sensors** for current car state feedback. Specifically, the localization of the car in track (given by the SLAM module), the actuators position (current steering angle) and the current velocity and inertial measurements of the car.
- Fast **communication** between the PU and the ECUs controlling the given actuators or/and receiving data from the multiple sensors. This is done using CAN, so a CAN Decoder and Encoder are needed.
- *Online* **Path Planning** module computing a reference trajectory and its velocity profile. For curvature-based approaches like the one presented in section 3, continuous curvature reference paths are needed.

1.6.2 Motion Planning Requirements

- **Processing Unit**.
- **Perception** system capable of detecting the cone's positions, hence the limits of the track.
- The **current position** and **heading** of the car so the correct part of the computed trajectory is sent over to the control module.

1.7 Workplan

The work presented in this thesis was carried out entirely within the five-month timeframe leading up to the commencement of the Formula Student Competitions. Here a detailed explanation of the time needed for the development of the different algorithms and their corresponding subtaks is presented. The total time span was approximately 20 weeks.

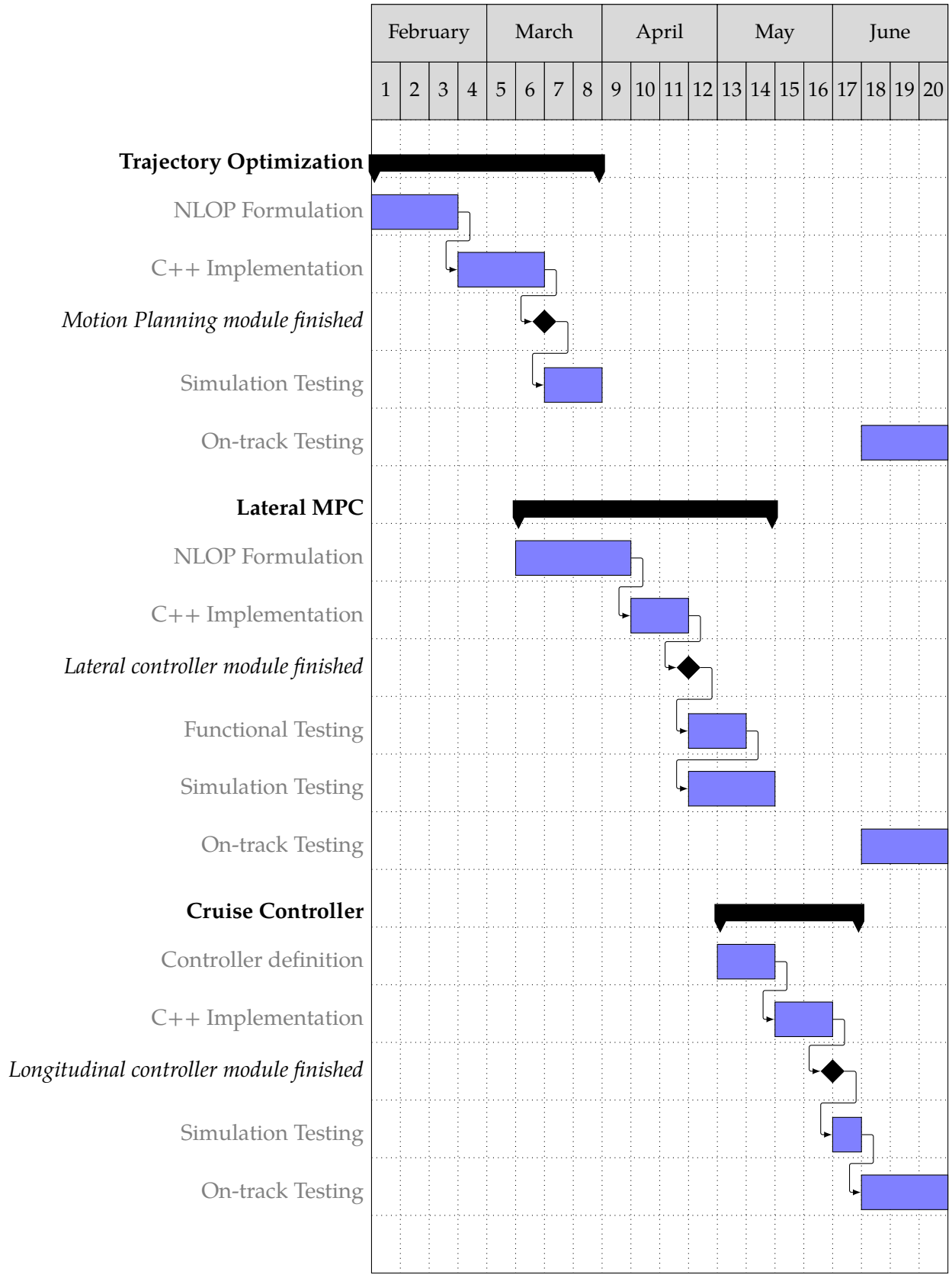


Figure 7: Gantt Chart.

2 Vehicle Model

The quality or precision of the mathematical model used to describe the system plays a critical role in predictive control. Choosing an appropriate model is usually a relevant and critical task which will depend on the current scenario and the control strategy. The system aimed to control in this work is a Formula Student racing vehicle, therefore a vehicle model that describes the behaviour of the car on track is mandatory in order to implement a predictive-based control solution.

Vehicle modeling is an state of the art concern widely studied in multiple engineering disciplines. In control theory, not only the physics principles of the model are important but also its discretization and integration methods as well as the linearization strategy (if applied). In this work, following the requirements stated in section 1.6.1 non linear model approaches will be addressed. Specifically, the kinematic and dynamic bicycle models are presented. Four-wheel drive models, defined in [13] are left apart due to their much higher computational load. In addition, the presented models will always assume that the system evolves in the two dimensional special Euclidean group (SE(2)), so 3 DOF will be considered (2 translation axis X, Y and 1 rotation axis Z). This assumption is consistent within Formula Student environments, where the track is a flat surface without elevation.

2.1 Kinematic Bicycle Model

The kinematic model provides a simple mathematical description of the vehicle movement with a set of trigonometric relationships. This steady-state approach neglects any lateral tire forces and works under the assumption of null skidding. This model is well known in the literature (e.g. [14]) to perform well at low speed profiles (less than $5m/s$) and yields a very simple formulation that can be suitable for basic applications.

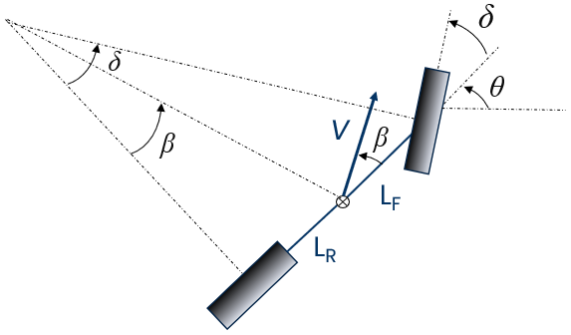


Figure 8: Kinematic bicycle model.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos(\theta+\beta) \\ v\sin(\theta+\beta) \\ w \end{bmatrix} \tag{1}$$

With β as the CoG kinematic side slip angle:

$$\beta = \arctan(\frac{l_R}{l_R + l_F}\tan(\delta)) \tag{2}$$

Where $\dot{x}$, $\dot{y}$ and $\dot{\theta}$ represent the longitudinal, lateral and angular velocities, respectively, in global frame (see Figure 8). The CoG slip angle is represented by β and it's the angle between the velocity vector v and the longitudinal axis from the body frame (or vehicle frame).

Note that, from a numerical perspective, this model is singular at $v = 0$ and therefore not controllable at this point. In [15] a reference-based kinematic model is presented in order to avoid this singularity.

2.1.1 Curvature-based model

For control purposes, a kinematic transformation from Cartesian to Curvature (Frenet-Serret) coordinate frame can be easily applied. This coordinate change can result in a more compact representation (see section 2.3), obtaining the heading and lateral position errors with respect to the reference path.

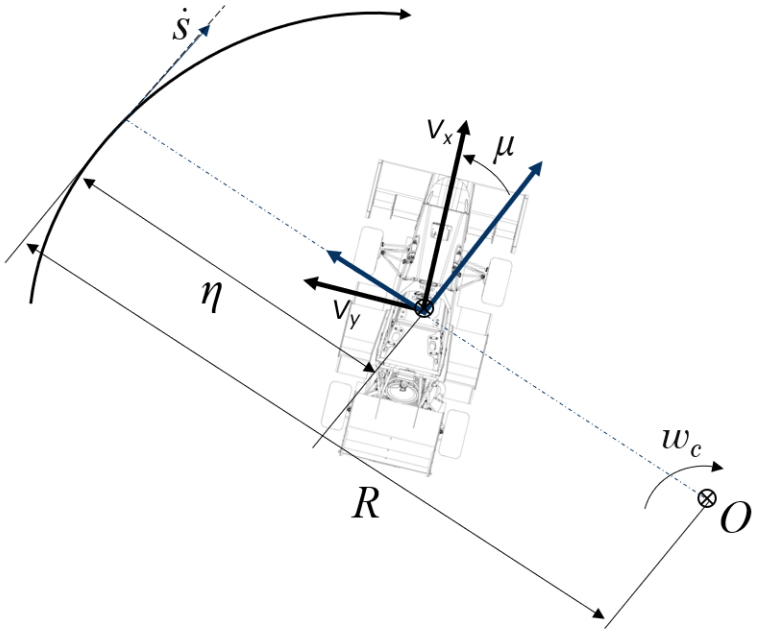


Figure 9: Curvature-based kinematic representation.

From Figure 9, it's possible to deduce two distinct expressions for the car's longitudinal velocity in the trajectory frame. They are set to be equal:

$$(R-n)w_c = v_x\cos\theta - v_y\sin\theta \tag{3}$$

From this, defining $\kappa = 1/R$, the rate of change of the desired heading angle is:

$$w_c = \frac{v_x\cos\theta - v_y\sin\theta}{1-y\kappa}\kappa \tag{4}$$

Using equation (4), the time derivative of the progress along the reference trajectory $\dot{s}$ is:

$$\frac{\partial s}{\partial t} = \dot{s} = Rw_c = \frac{v_x\cos\theta - v_y\sin\theta}{1-y\kappa} \tag{5}$$

With direct inspection, the evolution of the rest of variables is clear:

$$\frac{\partial n}{\partial t} = \dot{n} = v_x\sin\mu + v_y\cos\mu \tag{6}$$

$$\frac{\partial \mu}{\partial t} = \dot{\mu} = w - w_c = w - \kappa(s)\dot{s} \tag{7}$$

Finally, the curvature-based model can be stated as:

$$\begin{bmatrix} \dot{s} \\ \dot{n} \\ \dot{\mu} \end{bmatrix} = \begin{bmatrix} \frac{v_x \cos\mu - v_y \sin\mu}{1-n\kappa(s)} \\ v_x \sin\mu + v_y \cos\mu \\ w - \kappa(s)\dot{s} \end{bmatrix} \tag{8}$$

2.2 Dynamic Bicycle Model

The dynamic representation is characterized by taking into account the vehicle acceleration, so forces are used for the vehicle movement calculation. Thus, this model takes into account the front and rear lateral tire forces as well as the yaw moment caused by them. The dynamic model is more appropriate for working at higher speeds because it includes the tire model which is of special interest in the study of vehicle dynamics. Most of the non-linearities present on the vehicle behaviour are due to the interaction between the tires and the road surface, so a precise tire model plays a critical part in every dynamic model. The dynamic bicycle model is expressed as follows:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{v_x} \\ \dot{v_y} \\ \dot{w} \end{bmatrix} = \begin{bmatrix} v_x \cos(\theta) - v_y sin(\theta) \\ v_x \sin\theta + v_y \cos\theta \\ w \\ \frac{1}{m}(F_x - F_{y,F} \sin\delta + m v_y w) \\ \frac{1}{m}(F_{y,R} + F_{y,F} \cos\delta - m v_x w) \\ \frac{1}{I_z}(F_{y,F} \cos\delta l_F - F_{y,R} l_R + M_z) \end{bmatrix} \tag{9}$$

Here $\dot{v_x}$, $\dot{v_y}$ and $\dot{w}$ represent the longitudinal, lateral and yaw acceleration.

In this model, the longitudinal or motor force F_m is assumed to be applied on the CoG, so the longitudinal force expression is:

$$F_x = F_m - C_r - SC_d v_x^2 \tag{10}$$

Where C_r stands for rolling resistance coefficient and $SC_d = \frac{1}{2}\rho_{air} A C_d$ is the aerodynamic drag coefficient.

2.2.1 Lateral tire model

The lateral force in a tire, or equivalently the tire stiffness, depends on multiple variables such as the slip angle, the tire pressure, the vertical load, the camber angle, the rubber temperature, etc. For control purposes a desired trade-off between computational complexity and accuracy must be considered. Because of this, a very common approach is to express the lateral force as a function of slip angle.

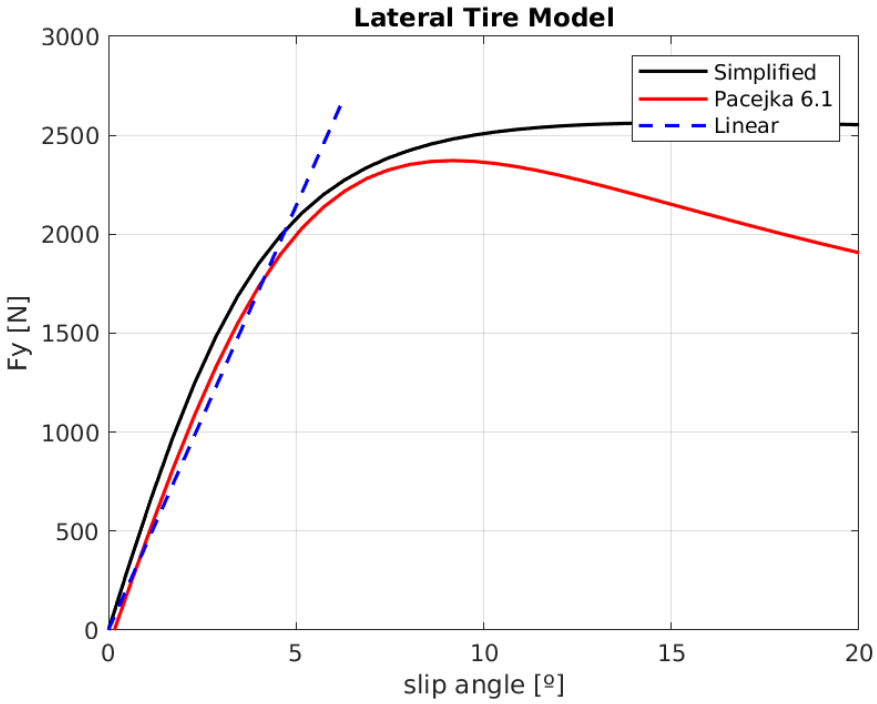


Figure 10: Lateral tire model comparison.

Assuming that the vehicle will always have small slip angles the tire model can be considered a linear function where the lateral force is proportional to the slip angle as follows:

$$F_{y,F/R} = C_{F/R}\alpha_{F/R} \tag{11}$$

Where $C_{F/R}$ are the cornering stiffness coefficients of the front and rear tires.

In Figure 10, the linear approach stays valid for slip angles lower than 5 degrees. However, when this threshold is exceeded, this model assumes that the tire will be able to withstand excessively elevated forces. This assumption will lead to grip loss situations and potentially unstable scenarios.

In Formula Student racing conditions, the linear assumption is too far away from reality because slip angles can go up to 10º. Thus, a more precise approximation for the lateral tire force is needed. The Simplified Magic Formula, a simplification of the Magic Formula by Hans Pacejka [16] provides a non-linear approach to calculate the resulting force from a wider range of slip angles.

$$F_{y,F/R} = F_{z,F/R} D_{F/R} \sin\left(C_{F/R} \arctan\left(B_{F/R}\alpha_{F/R}\right)\right) \tag{12}$$

Where $F_{z,F/R}$ represents the vertical load at each tire and the Pacejka constants $D_{F/R}$, $C_{F/R}$ and $B_{F/R}$ represent the peak value of the curve, the shape factor and the stiffness factor, respectively.

The normal force at the bicycle model front and rear tires is calculated as follows:

$$F_{z,F} = mg\frac{l_R}{l_F + l_R} \tag{13}$$

$$F_{z,R} = mg\frac{l_F}{l_F + l_R} \tag{14}$$

Slip angles are calculated as follows:

$$\alpha_F = \arctan\left(\frac{v_y + l_F w}{v_x}\right) - \delta \tag{15}$$

$$\alpha_R = \arctan\left(\frac{v_y - l_R w}{v_x}\right)$$

This last approach stays close enough to the complete Pacejka Tire model, offering a straightforward expression with only three parameters per wheel/axle. It's the chosen tire model for this thesis. In Figure 11, the Simplified Pacejka model used is shown, staying consistent with the slip angle definition stated above.

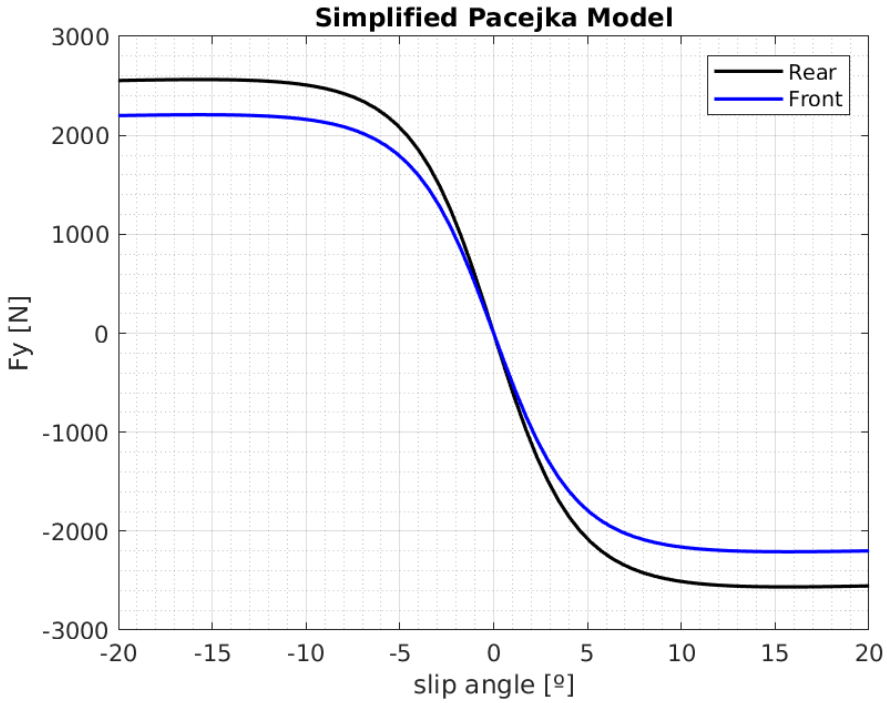


Figure 11: Simplified Pacejka tire model.

It's important to highlight that for significant slip angles, exceeding 10 degrees, both models notably differ as the complete Pacejka model reduces the maximum available lateral force while the simplified approach stays almost constant. This behaviour must be taken into account by the controller in order to avoid high slip situations, where the model trade-off is unacceptable.

2.3 Final approach

In this work, the dynamic bicycle model is used in sections 3 and 5 with some useful modifications in order to make it more suitable for predictive control applications.

As demonstrated in section 2.1.1, following trigonometric/kinematic relationships it's possible to transform any Cartesian-frame to Curvature coordinate frame in order to express spatial variables with respect to the reference path or trajectory. This transformation changes the representation of the 3 DOF from $[x, y, \theta]$ to $[s, n, \mu]$, where s represents the progress along the path, n represents the orthogonal distance from the CoG to a trajectory point (at a specific progress) and μ is the angle between the x-axis of the vehicle and the trajectory direction (or derivative).

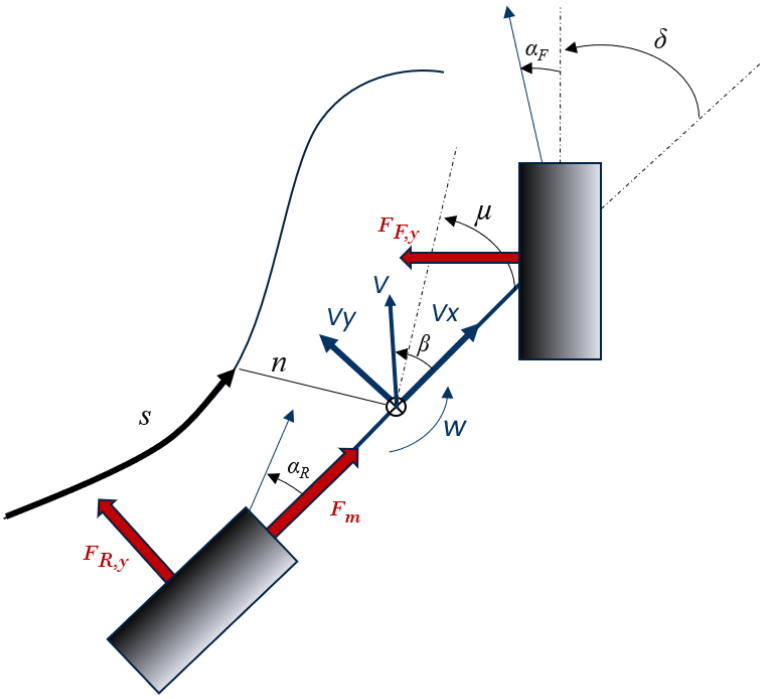


Figure 12: Curvature-based dynamic bicycle model.

The curvature-based dynamic model's expression is as follows:

$$\begin{bmatrix} \dot{s} \\ \dot{n} \\ \dot{\mu} \\ \dot{v_x} \\ \dot{v_y} \\ \dot{w} \end{bmatrix} = \begin{bmatrix} \frac{v_x \cos\mu - v_y \sin\mu}{1 - n\kappa(s)} \\ v_x \sin\mu + v_y \cos\mu \\ w - \kappa(s)\dot{s} \\ \frac{1}{m}(F_x - F_{y,F} \sin\delta + m v_y w) \\ \frac{1}{m}(F_{y,R} + F_{y,F} \cos\delta - m v_x w) \\ \frac{1}{I_z}(F_{y,F} \cos\delta l_F - F_{y,R} l_R + M_z) \end{bmatrix} \tag{16}$$

This transformation makes it possible to remove the progress s state from the model relying on its initial guess to evaluate all quantities depending on it. The progress along the trajectory is given as an input by the path planning algorithm and so is the curvature for each progress $\kappa(s)$.

Because the progress rate $\dot{s} = \frac{\partial s}{\partial t}$ expression depends only on other state variables and the given input $\kappa(s)$, it is possible to calculate the progress evolution over the whole time horizon outside the Optimization Problem module, which reduces its computational load.

The complete vehicle model used in this thesis is:

$$\begin{bmatrix} \dot{\delta} \\ \dot{F_m} \\ \dot{n} \\ \dot{\mu} \\ \dot{v_x} \\ \dot{v_y} \\ \dot{w} \end{bmatrix} = \begin{bmatrix} \Delta\delta \\ \Delta F_m \\ v_x \sin\mu + v_y \cos\mu \\ w - \kappa(s)\dot{s} \\ \frac{1}{m}(F_x - F_{y,F}\sin\delta + m v_y w) \\ \frac{1}{m}(F_{y,R} + F_{y,F}\cos\delta - m v_x w) \\ \frac{1}{I_z}(F_{y,F}\cos\delta l_F - F_{y,R} l_R + M_z) \end{bmatrix} \tag{17}$$

Where the control variables are $u = [\Delta\delta, \Delta F_m, M_z]$ and the state variables are $x = [\delta, F_m, n, \mu, v_x, v_y, w]$. Here, M_z variable is considered an additional or extra moment over the z-axis, that's why is set as a control variable.

3 Lateral Model Predictive Controller

3.1 Concept

Model Predictive Control (MPC) is an advanced control strategy employed in various engineering and industrial applications to optimize the performance of complex systems. It considers a prediction model of the system's behavior over a finite time horizon and seeks to optimize a specific objective function while adhering to system constraints. This predictive nature allows MPC to efficiently handle complex systems with nonlinear dynamics and constraints.

MPC relies on a mathematical model of the controlled system. This model describes how the system's state variables evolve over time and is typically represented by differential equations. The quality of this model is crucial, as it directly impacts the controller's performance. It can be a linear or nonlinear representation of the system dynamics and is typically derived from physical principles (or data-driven approaches [17]).

MPC operates in a receding horizon fashion. It repeatedly solves an optimization problem over a finite future time horizon. The control horizon defines the future time steps over which the control inputs will be optimized, while the prediction horizon extends further to make predictions about the system's behavior. In this work, there won't be any difference between the control and the prediction horizon as they will always have the same length.

Here, a **Curvature-based Nonlinear Model Predictive Controller** is presented in order to control the lateral behaviour of BCN eMotorsport's vehicle.

3.2 Non Linear Optimization Problem Formulation

The Non Linear Optimization Problem, from now on NLOP, is formulated with the objective of achieving, firstly, optimal steering target commands, secondly, to keep the vehicle inside the dynamic bounds constrained by the vehicle model, defined in section 2.3, and the reference trajectory, and finally, to apply soft control commands in order to avoid damaging the hardware.

The state vector x is defined as $x = [\delta, n, \mu, v_y, w]^T$ and the control vector u is defined as $u = [\Delta\delta, M_{tv}]^T$. Note that the s-state is decoupled as explained in 2.3 and the variables related to the longitudinal behaviour of the car are excluded. Thus, the optimization variables vector z is defined as $z = [u, x]^T = [\Delta\delta, M_{tv}, \delta, n, \mu, v_y, w]^T$.

In order to achieve softer steering commands, the steering derivative (or steering rate) $\Delta\delta$ is used as a control variable, leaving the actual steering δ as a state variable. This way, a direct ponderation over the rate of change of the output command is considered, resulting in less abrupt changes. Nevertheless, the rate of change of the torque vectoring moment M_{tv} is not considered, as it's meant to be an additional target moment which will act as a setpoint to the low level control torque allocation module.

The Non Linear Optimization Problem (NLOP) formulation is as follows:

$$\underset{u,\,x}{\text{minimize}} \quad \sum_{k=0}^{N} J(u_k, x_k) \tag{18a}$$

subject to

$$x_{k+1} = F_t(u_k, x_k), \tag{18b}$$

$$x_k \in \chi,\ u_k \in \Upsilon, \tag{18c}$$

$$g_{Track}(x_k) \leq 0, \tag{18d}$$

$$k = 0, \ldots, N \tag{18e}$$

Where N is the prediction horizon and $F_t(u_k, x_k)$ represents the integrated vehicle model (17).

3.2.1 Equality Constraints

The vehicle model acts as an equality constraint so each state variable evolves following its differential equation. The vehicle model considered for this controller is presented in section 2.3, excluding the longitudinal behaviour. The control variable ΔF_m is removed as well as the state variables F_m and v_x. However, the longitudinal velocity v_x must still be taken into account as a real time parameter (input), which changes along the progress $v_x(s)$. The model expression is:

$$\begin{bmatrix} \dot{\delta} \\ \dot{n} \\ \dot{\mu} \\ \dot{v}_y \\ \dot{w} \end{bmatrix} = \begin{bmatrix} \Delta\delta \\ v_x(s)\sin\mu + v_y\cos\mu \\ w - \kappa(s)\dot{s} \\ \frac{1}{m}(F_{y,R} + F_{y,F}\cos\delta - m v_x(s) w) \\ \frac{1}{I_z}(F_{y,F}\cos\delta l_F - F_{y,R} l_R + M_z) \end{bmatrix} \tag{19}$$

This differential equations are integrated using the explicit 4th order Runge-Kutta method (RK4).

3.2.2 Inequality Constraints

The state and control variables domains are represented as χ and Υ respectively. This hard constraints represent the mechanical and dynamic limits of the car. One of the main advantages of MPC is that it ensures this limitations won't be exceeded in any feasible solution, which adds rebustness to the controller.

On the other hand, in order to restrict the car's position to the inside of the track, soft constraints (18d) are added:

$$\begin{aligned} n + Lsin(|\mu|) + Wcos(\mu) &\leq \mathcal{D}_L(s) \\ -n + Lsin(|\mu|) + Wcos(\mu) &\leq \mathcal{D}_R(s) \end{aligned} \tag{20}$$

Here L and W represent the distances from the car's CoG to the furthest corner point of the car, and $\mathcal{D}_{L,R}(s)$ represent the orthogonal distance from the reference trajectory to the left or right track limits for each progress/point s. Both equations are then reformulated in order to match with expression (18d). In addition, a slack variable ζ is introduced because in the event of the

car exiting the track boundaries, the controller must not lead to an infeasibility but re-enter the track as soon as possible. The final track constraints are stated as follows:

$$\begin{aligned} n + Lsin(\measuredangle\mu|) + Wcos(\mu) - \mathcal{D}_L(s) - \zeta \leq 0 \\ -n + Lsin(\measuredangle\mu|) + Wcos(\mu) - \mathcal{D}_R(s) - \zeta \leq 0 \end{aligned} \tag{21}$$

Adding slack variables to the NLOP increases its computational load because they are treated as control variables. Here one and only slack variable is added, taking part in both track constraints. This can be done because there cannot be a situation where both constraints are active. E.g., if the car moves towards the left side of the track, the first constraint will be near activation $\left|n - \mathcal{D}_L(s)\right| \simeq 0$ while the second one will move away from activation $\left|-n - \mathcal{D}_R(s)\right| \ll 0$. If there was no such duality between the given constraints, one slack variable per constraint must have been added.

3.2.3 Cost Function

The cost (or objective) function is the expression that is going to be minimized over the whole horizon length until a local minimum is found. In another words, the controller's performance mostly depends on the terms considered in this function and their ponderation (weights). It is stated as follows:

$$J(u_k, x_k) = -Q_s\dot{s}_k \; + \; R_d\Delta\delta_k^2 \; + \; R_{M_{tv}}M_{tv}^2 \; + \; Q_\mu\mu^2 \; + \; Q_n n_k^2 \; + \; Q_{slip}\beta_{x,k}^2 \tag{22}$$

Where $-Q_s$ is added in order to maximize the progress rate over the given path, the control softening weights R_d and $R_{M_{tv}}$ are included in order to penalize abrupt control changes, path following weights Q_μ and Q_n are included to ensure the controller actually follows the reference trajectory achieving small values of μ and n, and finally, Q_{slip} ponderates over the kinematic and dynamic slip angles difference β, stated as follows:

$$\beta_{x,k} = \beta_{dyn,k} - \beta_{kin,k} \tag{23}$$

$$\begin{aligned} \beta_{dyn,k} &= \arctan(\frac{v_{y,k}}{v_{x,k}}) \\ \beta_{kin,k} &= \arctan(\frac{\delta l_R}{l_R + l_F}) \end{aligned}$$

This last condition is added so aggressive driving has a higher cost than smoother driving maneuvers, where the kinematic and dynamic slip angles usually take similar values. Moreover, as explained in section 2.2.1, the presented controller must avoid too high slip angle values because of the existing trade-off in the simplified tire model.

3.3 Implementation

All Autonomous Systems, Control and Electronics software implemented on BCN eMotorsport's car communicate through ROS Noetic framework. Thus, all the algorithms presented in this thesis are implemented in C++ and run in a self-assembled Processing Unit with Ubuntu 20.04 GNU x86_64 as the operating system. The PU has an Intel i9 main CPU with 20 cores in order to execute in real time the complete driverless pipeline.

The implementation of the lateral MPC algorithm in C++ consists on a wrapper module that circles around the C-generated solver library. The MPC module is in charge of receiving the real-time data from all the available sensors of the car together with the path planning output and handle this information in structured arrays to the solver library on run-time. After the optimization problem is solved, the given solution is read and the optimal commands are published to the ROS network for the low level control algorithms to follow the resulting target commands. This commands are then sent directly to the actuators.

From a software development point of view, several procedures must be considered in order to close the control loop. Before entering the ROS loop, the MPC object must be initialized, setting up all the static parameters and the data structures used during run-time. Within the loop, the car state and path planning output are received from the ROS network via callbacks. Afterwards, the NLOP variable boundaries together with the real time parameters are set. Then the MPC initial state feedback, defined as $x_{init} = [\delta, n, \mu, v_y, w]$, is updated. The steering position δ is directly received from the potentiometer whereas the lateral velocity v_y and yaw rate w are received from the state estimation module. The lateral distance n and heading μ with respect to the trajectory must be calculated knowing the actual car's position $[x, y, \theta]$ received from the SLAM module. After that, the initial guess array must be filled. This array contains the values from which the solver will start to iterate for each state variable for each stage of the prediction horizon. This values taken by the solver are of vital importance as they can greatly affect the solving time, depending on how far they are from the optimal state. That's why the initial guess array is filled following the procedure explained in section 3.3.1. Finally, the solver instance is called, the final command is picked from the solution array and is sent over to the ROS network.

NOTE: a *ROS Dynamic Reconfigure* instance is added to the MPC module enabling online parameter tuning, so an extra callback is defined in order to overwrite the real time parameters values at run-time.

3.3.1 Solver

In order to solve the NLOP problem in real time, a fast and reliable solver is mandatory. Many state of the art open source and commercial solvers exist. However, Embotech FORCESPRO solver was chosen for its fast solving capabilities in tailored and highly non-linear MPC formulations as the one presented in section 3.2. The chosen NLOP solving algorithm is the Nonlinear Primal-Dual Interior-Point method because of its robustness and reliability for most nonlinear problems.

FORCESPRO provides a MATLAB and/or Python API where the complete optimization problem formulation must be specified. Afterwards, the code is C-generated in order to be able to use it in embedded systems unlocking a much faster performance on solving time.

Moreover, in order to speed up the solver convergence, an heuristic procedure can be followed. If an optimal solution was found in the last solver call, the initial guess of the optimization variables for the next call will be this very same solution. Otherwise, the initial guess will be each boundary midpoint. This strategy stays consistent because due to the high frequency of the control loop (40Hz), the optimal solution z_k will often be found near the already known z_{k-1} solution.

3.3.2 Algorithm pseudocode

Algorithm 1 Tailored MPC

$p \leftarrow get\_parameters()$ ▷ Read parameters
$mpc \leftarrow mpc :: mpc(p)$ ▷ Call mpc constructor

$state \leftarrow ros :: callback\_state()$ ▷ Set up car state callback
$planner \leftarrow ros :: callback\_planner()$ ▷ Set up planner callback

$dyn\_reconfig\_sub \leftarrow ros :: callback\_dynConf()$ ▷ Dynamic Reconfigure callback

$r \leftarrow ros :: Rate(p.freq)$ ▷ Control loop frequency
while $ros :: ok()$ **do**

    $ros :: spinOnce()$ ▷ Update callbacks with received msgs

    $ub, lb \leftarrow mpc :: nlop :: get\_boundaries()$ ▷ Set up variables boundaries

    $x0 \leftarrow mpc :: current\_pose(state)$ ▷ Get current car state

    if $exit\_flag == 1$ **then**
        $init\_guess \leftarrow mpc :: last\_solution()$ ▷ Set up initial guess
        $s \leftarrow mpc :: progress\_prediction()$ ▷ Predict progress values
        $v_x, \kappa \leftarrow mpc :: progress\_sampling(s, planner)$ ▷ Pick progress dependant values
    else
        $init\_guess \leftarrow mpc :: mid\_bounds()$
        $v_x, \kappa \leftarrow mpc :: equal\_sampling(planner)$
    end if

    $solution, exit\_flag \leftarrow mpc :: nlop :: solve(x0, v_x, \kappa, init\_guess, ub, lb, p)$ ▷ Solver call

    $commands \leftarrow mpc :: prepare\_msg(solution)$

    if $mpc :: is\_finished(p.mission)$ **then** ▷ Check if mission is finished
        $commands \leftarrow mpc :: safe\_stop()$
    end if

    $ros :: publish(commands)$ ▷ Publish target commands

    $ros :: sleep(r)$ ▷ End point to ensure control loop frequency
end while

3.3.3 Module architecture

The complete MPC diagram is shown here:

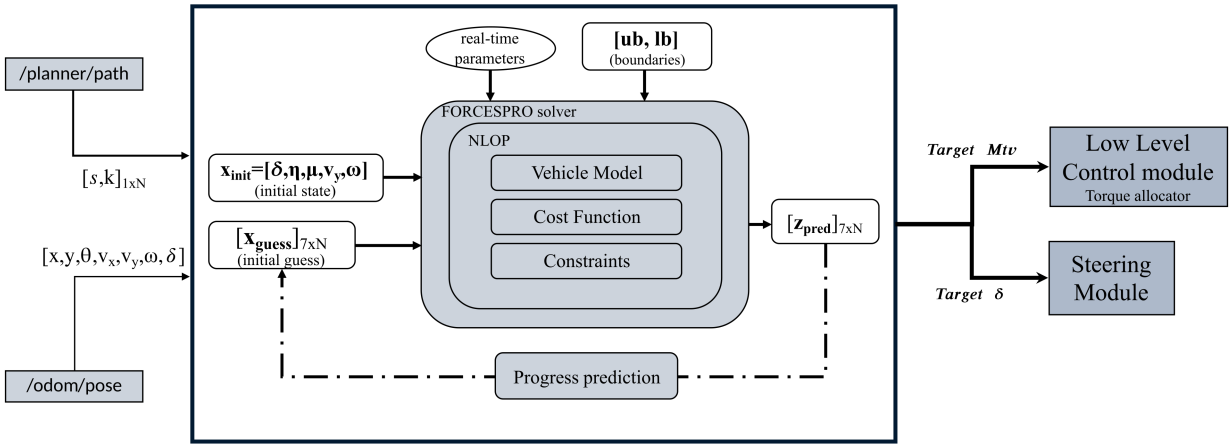


Figure 13: Lateral MPC diagram.

3.4 Results

In order to be able to test and tune the Lateral MPC cost function weights, a simulation software capable of computing precise vehicle dynamics interactions is mandatory. The IPG CarMaker multibody simulation software is chosen.

CarMaker provides a high-performance, real-time capable physics engine based on C/C++, with a highly efficient vehicle model. Moreover, a complete parametrization of the given model is possible, adapting every part of the vehicle to match with the studied one, in this case the CAT15X. IPG's software also enables access to the simulation loop via a C++ API, so a real-time communication between the ROS network and IPG's physics engine is set in order to close the simulation loop. The complete AS pipeline can be executed through a ROS instance while interacting with the simulation physics, evaluating this way the different algorithms reliability and performance in the same framework in which they will be running in the vehicle's PU.

To test the controller, the AS *online* path planning pipeline is executed in order to tune the MPC while preserving the stability of the complete software stack. Although an initial tuning is made with a standalone instance of the controller, the final tuning procedure is carried out taking into account the path planning variability, as it has proved to be of great influence when choosing the optimal cost function weights. Depending on the number of cones seen by the perception module, the path planning algorithm can drastically change the output trajectory which could lead to aggressive steering commands and unstable car behaviours. The presented controller

must be robust to these input changes.

The objective of the following simulation tests is to understand how the cost parameters affect the car behaviour on track in order to choose the parameters set which achieves the lowest lap time.

Embotech's FORCESPRO solver provides an exit flag to every solver call. This way, the exit status of the solving iterative process can be checked before sending the output solution to the ROS network. When a local optimal solution is found (i.e. the point satisfies the KKT optimality conditions to the requested accuracy) the exit flag returned value is 1. If the solver reaches the maximum number of iterations specified without finding an optimal solution the exit flag is 0. Finally, if the solver could not proceed due to infeasibility issues the exit flag is -7. Hence, it is crucial to monitor the solver exit flags to guarantee the acquisition of optimal commands in every iteration of the controller. Notice that if the maximum number of iterations are reached during the run, the output commands could also be used because the solving procedure ensures these will always be inside the given constraints. However, lap time and control smoothness will be compromised.

The following simulation results have been achieved using **Ubuntu 20.04** with an **AMD Ryzen 9** CPU, **IPG CarMaker** v11.0 and **ROS Noetic**.

An integration step of **0.025s** is used for all the following tests, with an horizon length of **40** stages. Thus, a **1.0s** prediction horizon is used.

3.4.1 Weight analysis

Cost function weights play a crucial role in optimization problems because they determine the relative importance or priority assigned to different objectives within the optimization process.

By adjusting the weights assigned to each component of the cost function, preferences and priorities can be defined to focus on more specific goals. Consequently, cost function weights enable fine-tuning of the optimization process in order to align with the desired trade-offs and objectives of the presented controller.

In practical terms, the appropriate selection of cost function weights can lead to solutions that better align with real-world requirements, constraints and preferences.

The **progress rate maximization term** Q_s ensures minimal curvature traces, linking together μ, n and v_y state variables. Note that due to the lateral feature of this controller, the v_x is set to be a constant value for each stage of the NLOP and can not be iterated by the solver. In order to maximize the progress rate expression in (5) v_y and μ should be minimized. However, the lateral deviation n should be maximized, which goes against intuition. In fact, this condition is what actually causes approximate minimal curvature traces where the car tends to cut in corners, resulting in racing alike paths although higher lateral deviation distances are achieved in curves. In Figure 14 this correlation between the lateral deviation and the trajectory's curvature is shown. Higher values of curvature enable higher values of lateral deviation, translating into a racing driving style.

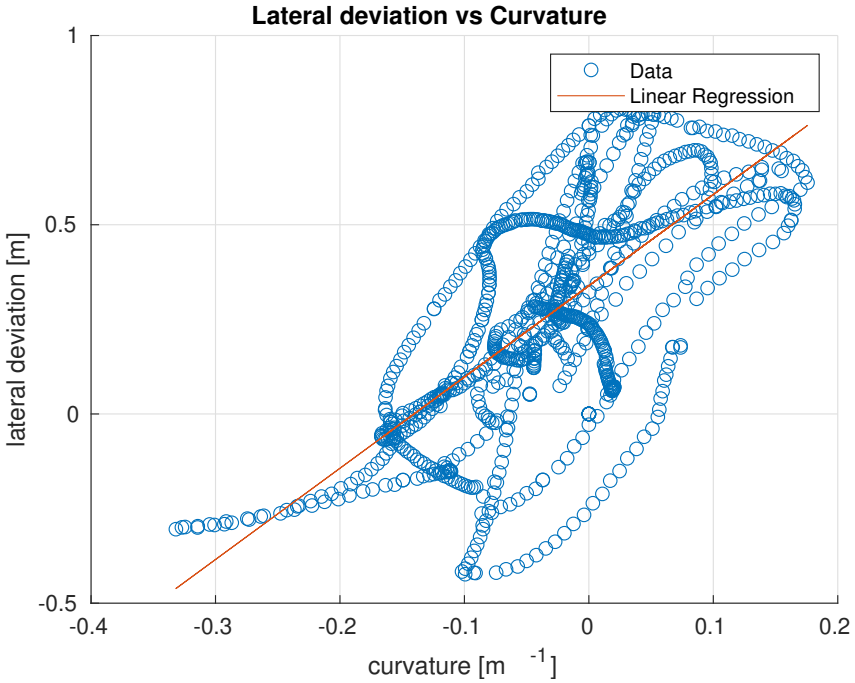


Figure 14: Lateral distance vs Curvature correlation.

The **lateral deviation weight** Q_n penalizes high lateral distances to the reference trajectory to ensure the controller actually follows the reference path. As shown in Figure 15 higher values of Q_n reduce the lateral deviation, ensuring a better path following behaviour. Because of the progress rate maximization term explained above, smaller values of lateral deviation do not certainly mean better performance. The existing trade off between low deviation errors and the corner tracing behaviour must be considered. This decision is greatly influenced by the path planning algorithm being used. If the input trajectory is set to be close to the midline path of the track, the curve tracing behaviour of the controller may be prioritized. However, if the received path is already tracing in curves, seeking small deviation errors may be the right choice.

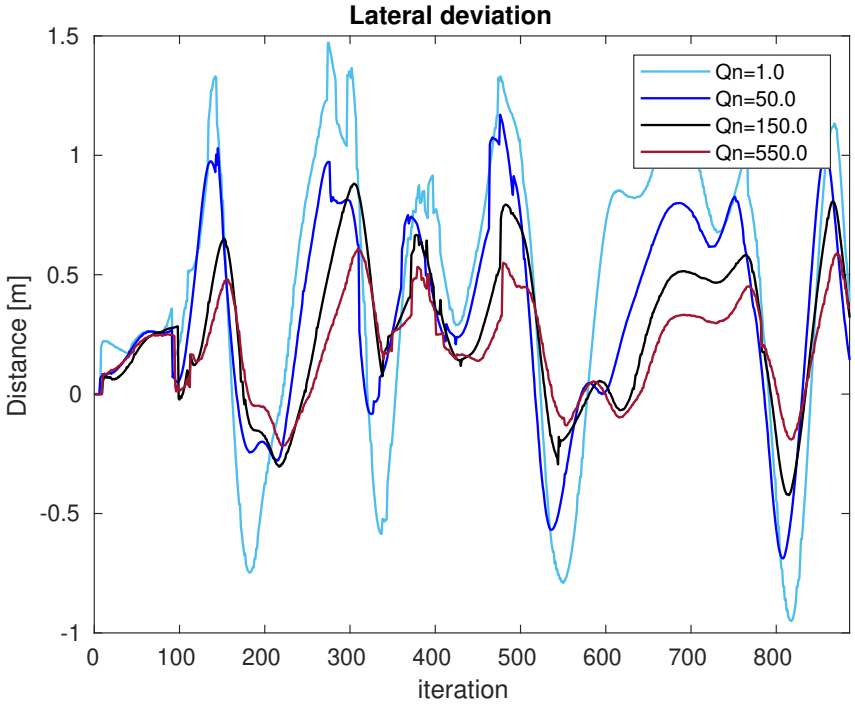


Figure 15: Lateral distance comparison.

The **torque vectoring extra moment weight** $Q_{M_{tv}}$ lets the controller use an additional moment around the z axis hugely affecting the vehicle dynamics in curves. This extra moment is then used as a setpoint by the Torque Allocation module, coupling this way the high level lateral controller here presented with the low level control pipeline of BCN eMotorsport vehicle. In Figure 16 the target extra moment is shown for different tuning weights. For really small weight values, the controller starts using higher values of this additional moment improving its cornering performance.

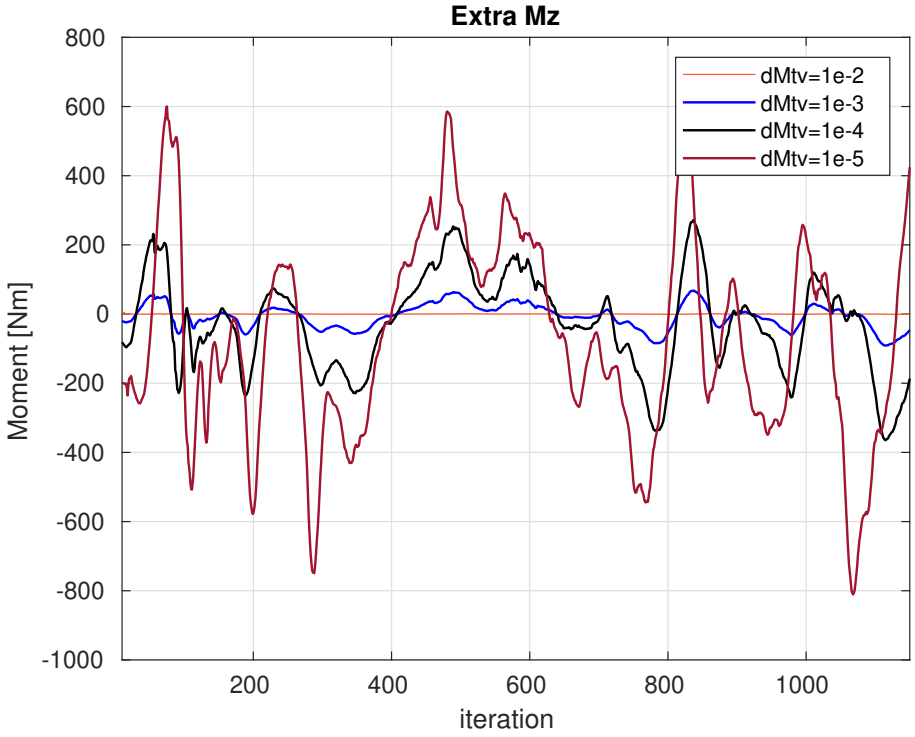


Figure 16: Torque Vectoring additional moment setpoints.

The **steering rate softening weight** R_d is meant to penalize abrupt steering changes, parameterizing this way the maximum rate of change accepted by the actuators. Figure 17 shows that for higher costs the steering rate stays low, which translates to more stable steering commands in Figure 18. Nevertheless, really high values of this weight could penalize the controller responsiveness in really tight turns.

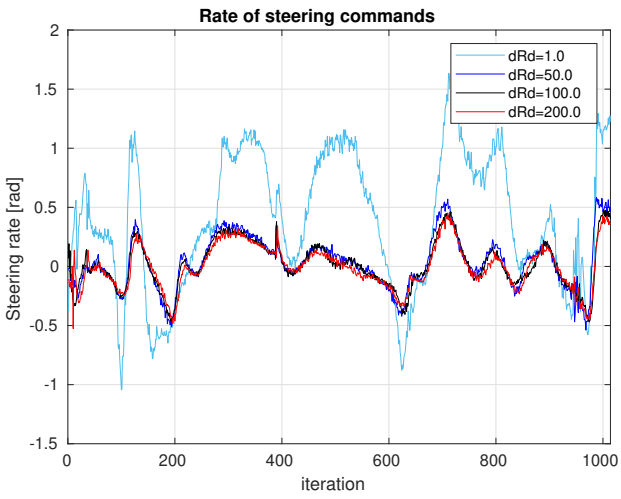


Figure 17: Steering rate comparison.

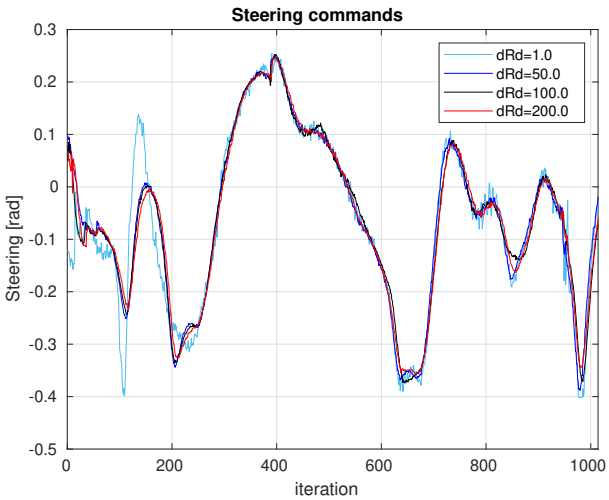


Figure 18: Steering commands comparison.

3.4.2 Performance

Within this section a fraction of the simulation performance of the Lateral MPC is presented with the results obtained from an autocross run in two distinctive FS tracks: Formula Student Germany (FSG 2019) competition and Formula Student Spain (FSS 2019) competition.

In order to be consistent with the existing limitations on the real vehicle, the presented test runs have been performed without executing the Torque Vectoring software and using only both rear driving wheels. Nevertheless, the low level control pipeline from BCN eMotorsport is executed, so Traction Control and Regenerative Braking modules are present. In addition, as mentioned in section 3.4 the team's Track Limits detection and online Path Planning modules are also running.

Regarding the **FS Spain** test run, in Figure 19 is clearly visible that the controller is able to keep the car's yaw rate within the predicted values, with the output steering commands shown in Figure 21. The actual lateral velocity of the car also fits its target predicted values in Figure 20, although a small miss match is present. This variation in lateral velocity can be explained by its high sensitivity towards vehicle model changes. Due to the acknowledged (and contrasted) superior accuracy of the simulation model compared to the dynamic bicycle model stated in 2, Figure 20 highlights the limitations of the employed prediction model.

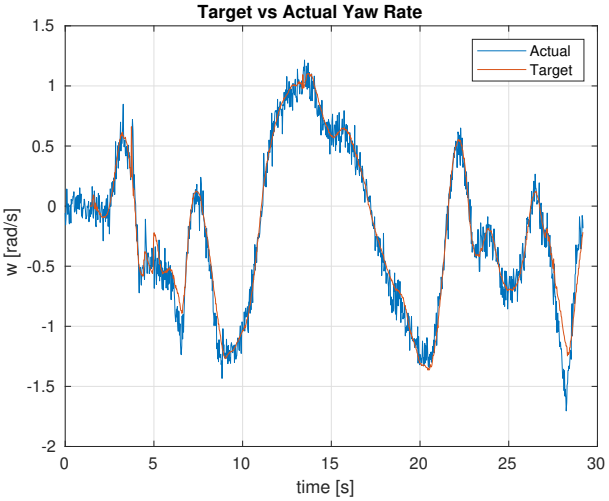


Figure 19: FSS Actual vs Target yaw rate.

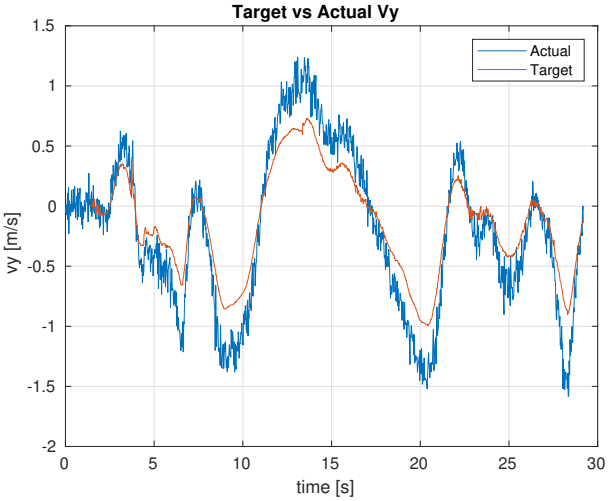


Figure 20: FSS Actual vs Target lateral velocity.

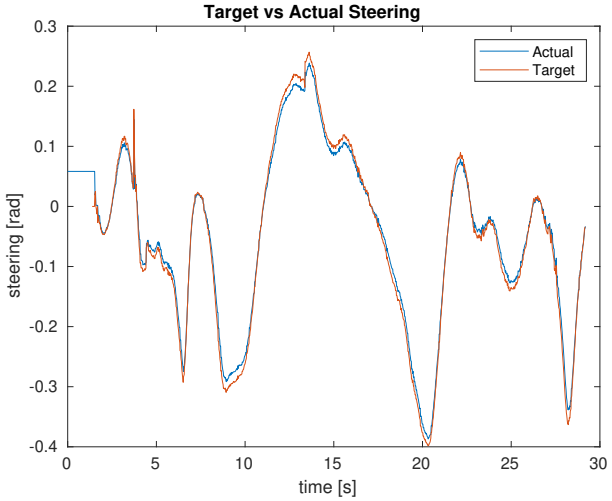


Figure 21: FSS Actual vs Target steering angle.

The same conclusions are extracted from the FS Germany test run, which can be seen in Figures 22, 23 and 24.

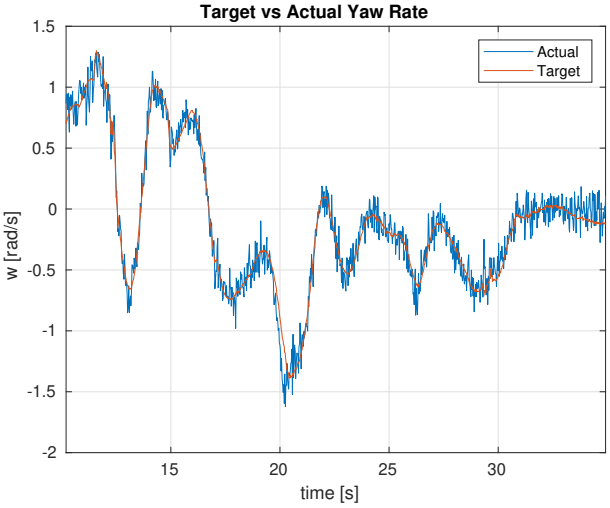


Figure 22: FSG Actual vs Target yaw rate.

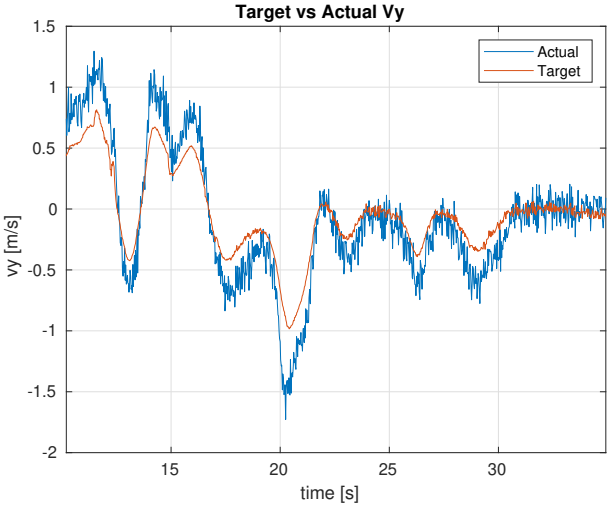


Figure 23: FSG Actual vs Target lateral velocity.

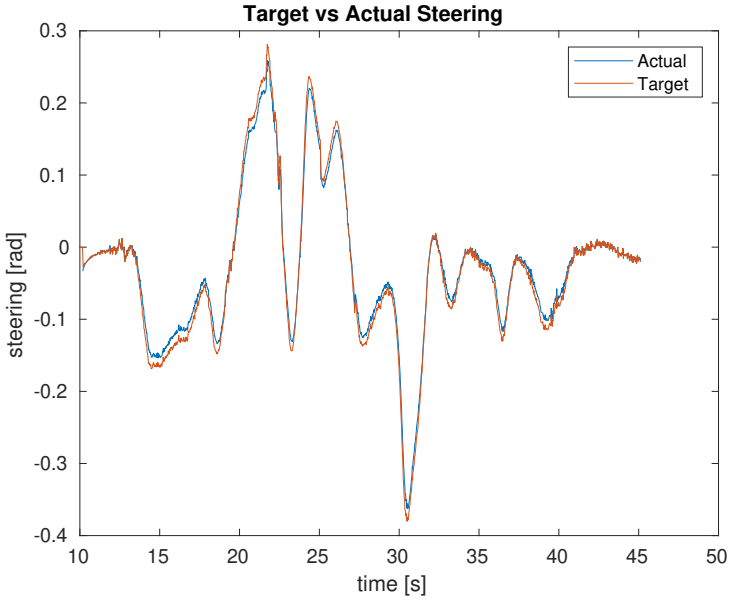
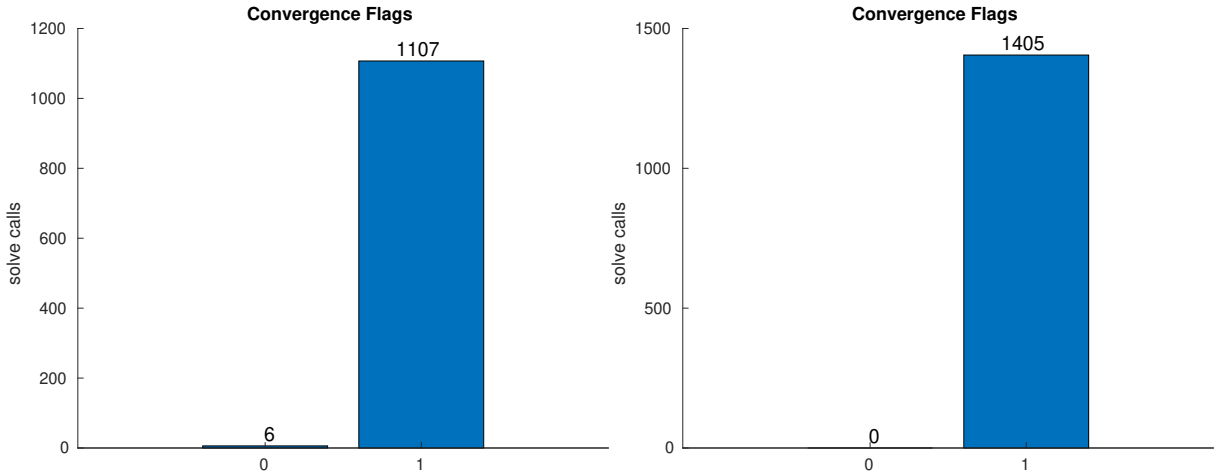


Figure 24: FSG Actual vs Target steering angle.

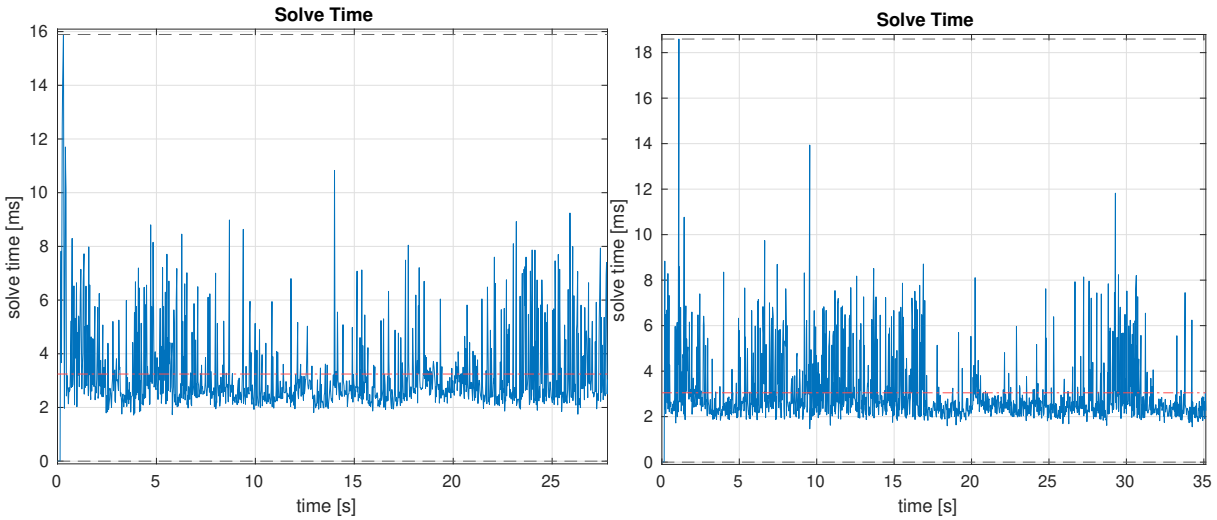
Another important aspect of any test run are the number of optimal solutions found. If the solver reaches the maximum number of iterations for a long period (typically more than 4 iterations) the quality of the output commands can highly decrease, loosing steering movement continuity and potentially destabilizing the vehicle. A great example is shown in Figure 21, where near the first five seconds of the run the controller outputs an outlier steering command due to 6 consecutive maximum iterations exit flags, shown in Figure 25. The exit flags for the complete FSS and FSG runs are shown in Figure 25, with a **0.53%** and **0.0%** of maximum iterations reached respectively.



(a) FSS (b) FSG

Figure 25: Exit flags bar plot.

Finally, the solving time for each test run is shown in Figure 26. Notice that although some modules of the AS pipeline together with the low level control complete pipeline are being executed during the simulation, the greatest computational burden in the car comes from the AS Perception pipeline, which processes big quantities of LiDAR point clouds in real time; and it's not present in the simulation. This computational load must be taken into consideration when analyzing the controller's solving time, as it probably will be much higher when executed in the onboard PU. Both solving time means are found near 3 ms, with maximums of 16 and 18 ms. It's clear that the Lateral MPC loop can be safely executed at 40 Hz, complying with the control specifications defined in section 1.5.1.



(a) FSS (b) FSG

Figure 26: Lateral MPC solve time.

4 Cruise Controller

4.1 Concept

Cruise control is a technology commonly found in automobiles that allows drivers to maintain a consistent speed while driving on highways or open roads. It works by automatically adjusting the throttle or engine power to keep the vehicle moving at a set speed, which the driver selects and activates. Nonetheless, within the context of this thesis, the term "cruise control" will be employed to describe the longitudinal controller's function of adjusting the throttle to follow a specified velocity profile, as opposed to maintaining a constant speed.

In order to accomplish this objective, the selected controller is a PID controller. PID controllers, short for Proportional-Integral-Derivative controllers, enjoy extensive usage in control systems across diverse industrial sectors due to their effectiveness in overseeing and governing processes and systems. Within the context of this thesis, their most relevant advantages include their ability to swiftly respond to changes, adapt to various conditions, and maintain system stability. These attributes align closely with the prerequisites outlined in section 1.6.1, making the PID controller an ideal choice for the task at hand. Furthermore, considering the constraints of limited testing time and the inherent reliability challenges associated with Formula Student prototypes, the use of a straightforward controller that relies on a minimal set of adjustable parameters for optimal tuning is prioritized.

Here, an **Adaptive Cruise Controller** is presented in order to control the longitudinal behaviour of this year's BCN eMotorsport car.

4.2 Velocity Profile Calculation

In order to compute the velocity profile for each new path found by the path planning module in real time, a parametrized GG diagram approach is used.

The GG diagram is a commonly used plot in vehicle dynamics to understand and improve vehicle performance. The x-axis represents lateral acceleration values while the y-axis usually represents longitudinal acceleration, all expressed in units of "g". This plot is a valuable tool to asses tire performance, optimize vehicle handling and often make design decisions related to tire and suspension systems.

The parametrized GG diagram expression is stated as follows:

$$(\frac{a_x}{a_{x,max}})^2 + (\frac{a_y}{a_{y,max}})^2 \leq 1 \tag{24}$$

Therefore, this dynamic equation transforms into a pair of semi-ellipse equations, where the inner region defines the operational work zone, and the outer region is the area where slipping or reaching the limit is not advisable. To push the car to its performance limits, the focus must be set on the limit zone, incorporating a safety margin to ensure safe operation.

The maximum lateral acceleration of the car is previously set and remains constant during the run while the maximum longitudinal and braking accelerations are dynamically adjusted taking into account the Low Level Control positive and negative torque limitations. This way, the cruise controller is always following a velocity profile that stays consistent with the maximum torque capabilities of the car.

In order to calculate the velocity profile tracing the given ellipse-shaped constraint a two-step procedure must be followed.

A previous approximation of the velocity profile is calculated assuming that the car stays in pure cornering behaviour for the whole reference trajectory. This assumption states that the vehicle is always under maximum lateral acceleration, so:

$$v_{cornering,i} = \min(\sqrt{\frac{a_{y,max}}{|\kappa_i|}}, v_{max}) \tag{25}$$

Where $i \in [0, size(path))$, the given curvature at trajectory point i is represented as κ_i and v_{max} is included for safety reasons.

Afterwards, the velocity profile is computed taking into account the longitudinal acceleration and braking capabilities of the single-seater. Firstly, a forward calculation is done taking into consideration the maximum longitudinal acceleration followed by a backwards calculation taking into account the maximum braking acceleration. Finally, the minimum velocity for each trajectory point is picked. The procedure is as follows:

Accelerating: $i \in [0, size(path))$

$$a_{y,i} = \min(|\kappa_i|\, v_{accel,i}^2, a_{y,max}) \tag{26}$$

$$a_{x,i} = a_{x,accel,max}\sqrt{1 - (\frac{a_{y,i}}{a_{y,max}})^2} \tag{27}$$

$$v_{accel,i+1} = \sqrt{2a_{x,i}\Delta s + v_{accel,i}^2} \tag{28}$$

Braking: $j \in (size(path), 0]$

$$a_{y,j} = \min(|\kappa_j|\, v_{brake,j}^2, a_{y,max}) \tag{29}$$

$$a_{x,j} = a_{x,brake,max}\sqrt{1 - (\frac{a_{y,j}}{a_{y,max}})^2} \tag{30}$$

$$v_{brake,j+1} = \sqrt{2a_{x,j}\Delta s + v_{brake,j}^2} \tag{31}$$

Final velocity profile:

$$v_{final,i} = \min(v_{cornering,i}, v_{accel,i}, v_{brake,i}) \tag{32}$$

Note that because this is an online (on run-time) approach, the first velocity v_0 is set to match with the current velocity of the vehicle.

To generate a velocity profile for the entire track rather than a partial profile, it is necessary to replace the initial velocity condition with a periodic one. It is crucial to initiate the profile from a curvature apex, as these points represent local maxima, and their velocity is already known to be $v_{cornering}$. This ensures the final velocity profile is the highest possible. Thus, the selected apex will define the point where the first and last velocities match.

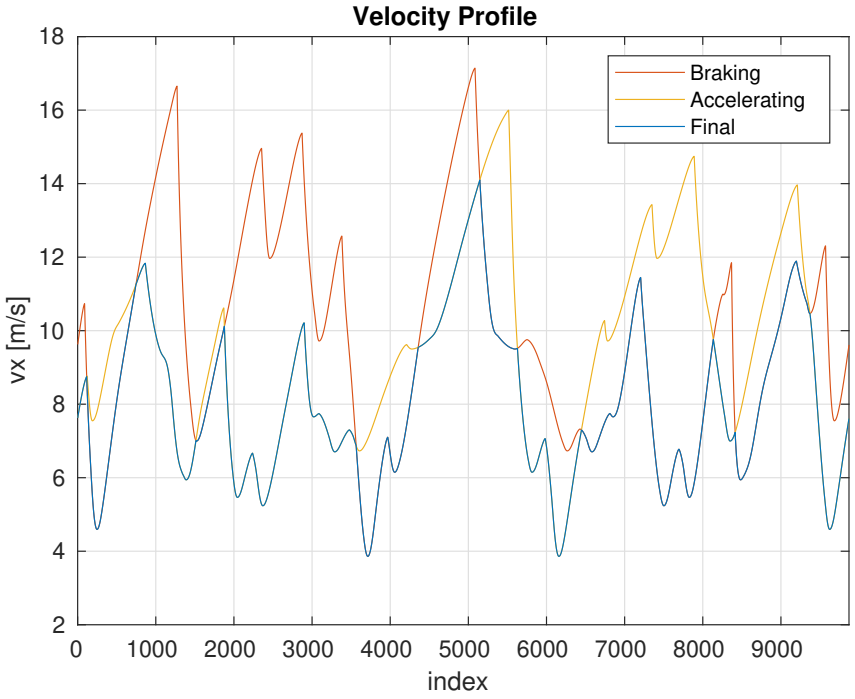


Figure 27: Closed velocity profile of FSS track.

The velocity profile shown in Figure 27 was computed taking into account the following restrictions: $a_{x,accel,max} = 4.0m/s$ and $a_{y,max} = a_{x,brake,max} = 6.0m/s$.

4.3 PID Formulation

The PID controller uses three components to adjust the control output based on the error, which is the difference between the desired setpoint and the actual process variable, in this case the target velocity previously calculated as stated in section 4.2 and the current velocity of the vehicle:

- **Proportional (P)**: The proportional term is directly proportional to the current error. It multiplies the error by a constant (the proportional gain, often denoted as K_p) and provides an immediate response to changes in the error. A larger K_p value leads to a stronger and faster response but may introduce oscillations if it's too high.

- **Integral (I)**: The integral term accumulates the error over time and is designed to eliminate any steady-state error. It multiplies the integral of the error by a constant (the integral gain, often denoted as K_i). It is effective in eliminating long-term error, but too high a K_i value can cause instability and slow response.

- **Derivative (D)**: The derivative term is proportional to the rate of change of the error. It multiplies the derivative of the error by a constant (the derivative gain, often denoted as K_d). The derivative term provides damping and helps to reduce overshoot and oscillations. However, a high K_d value can lead to excessive noise amplification.

4.3.1 Anti-windup

An anti-windup strategy is also added to the cruise controller. Anti-windup is a technique used in PID controllers to prevent or mitigate the issue of integral windup. Integral windup occurs when the integral term of the PID controller accumulates a large error over time due to the controller being saturated, i.e., when the control output is limited by physical constraints or saturation limits of the system being controlled. In this use case, the controller's output (or throttle) is comprised between $[-1, 1]$ where -1 represents maximum braking (maximum negative torque) and +1 represents maximum acceleration (maximum positive torque). This command is later sent over to the Low Level Control module, which is responsible to linearly transform this value to a final torque command. In order to see the anti-windup's architecture, see Figure 28.

4.3.2 Adaptive PID

For security reasons, it is common that the maximum longitudinal acceleration set in the GG diagram is lower than the actual maximum acceleration capabilities derived from the maximum torque available. In order to be consistent with this situation, the PID's proportional gain K_p is linearly ajusted as follows:

$$K_{p,final} = K_p(\frac{a_{x,gg}}{a_{x,max}}) \tag{33}$$

Where K_p stands for the initial gain value, $a_{x,gg}$ represents the GG diagram's maximum longitudinal acceleration taken into account when computing the velocity profile and $a_{x,max}$ is the actual maximum acceleration derived from the torque limitations of the car.

4.4 Implementation

The cruise control module, developed in C++, is designed to function by receiving the desired longitudinal velocity from the path planning module and the actual velocity of the vehicle from the state estimation module. Afterwards, it calculates the current error and employs the PID logic to derive the final throttle command. Additionally, the module keeps track of the motor torque limitation, representing the maximum achievable torque. To adapt the proportional gain K_p, the linear adjustment (33) is made after converting the maximum torque at the motor axis to the maximum available longitudinal acceleration of the car. Finally, the throttle command is sent to the Low Level Control module, which linearly transforms it to torque commands for each driven wheel.

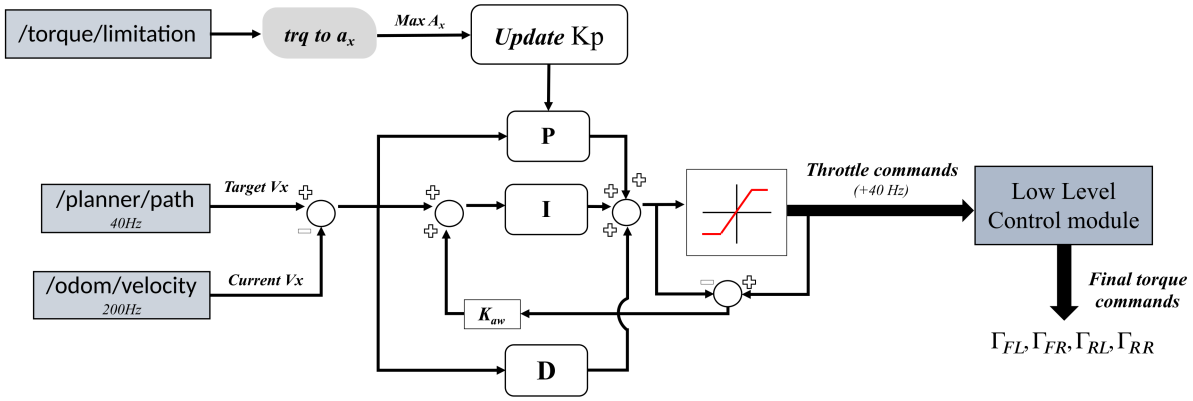
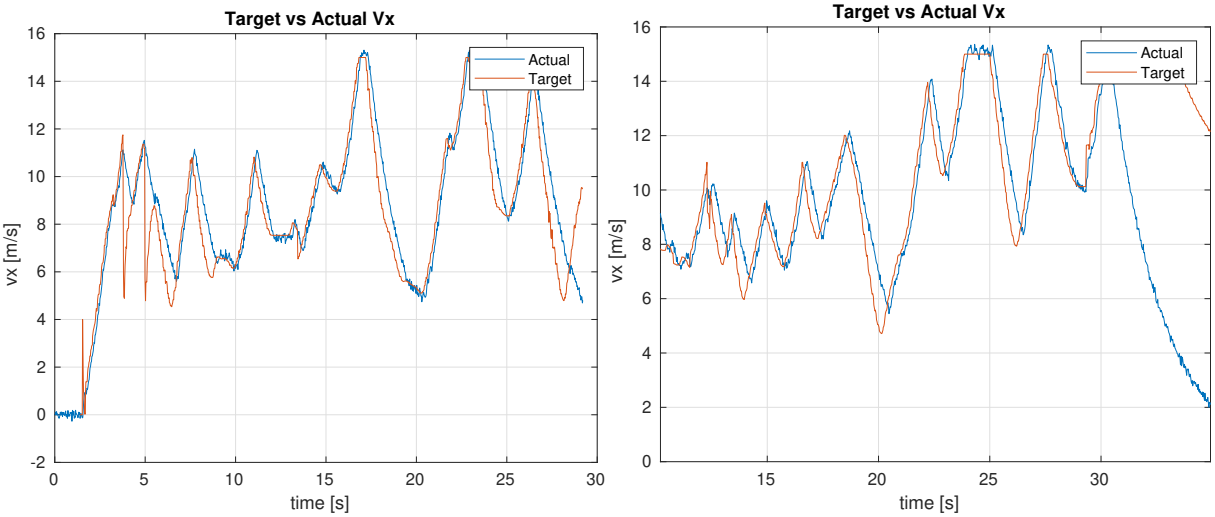


Figure 28: Cruise Controller diagram.

4.5 Results

In order to evaluate the Cruise Controller performance in a simulation environment, the same tracks as in section 3.4 will be used. Furthermore, the same simulation conditions as the ones presented in section 3.4 apply throughout all this section as well as the same modules are executed during the runs.

In Figure 29 the accumulated target velocity setpoints over time are shown together with the actual car's longitudinal velocity. Focusing now on the FSS run, the accumulated target profile shows aggressive changes in the velocity setpoint near the five seconds timeline. This behaviour is due to the unsteady path planning trajectory, which radically changes two times in a row because of track limits corrections. In other words, as more cones are being detected, the track limits module updates its estimation of the drivable path which triggers a path planning trajectory recalculation. This behaviour is usually seen at the start of the run because of the limited number of detected cones, which leads to a wrong track limits detection, quickly fixed as the car starts moving. In contrast, in the FSG test run this behaviour is not seen, as the corrections are made over the final points of the trajectory, not changing it completely. Following the completion of the FSG autocross run, the braking curve after crossing the finish line can be seen.



(a) FSS (b) FSG

Figure 29: Actual vs Target longitudinal velocity.

In order to fulfill the longitudinal control specifications defined in section 1.5.1 the throttle commands must be smooth enough not to damage the hardware. In Figure 30 the output commands of the Cruise Controller are shown for FSS test run.

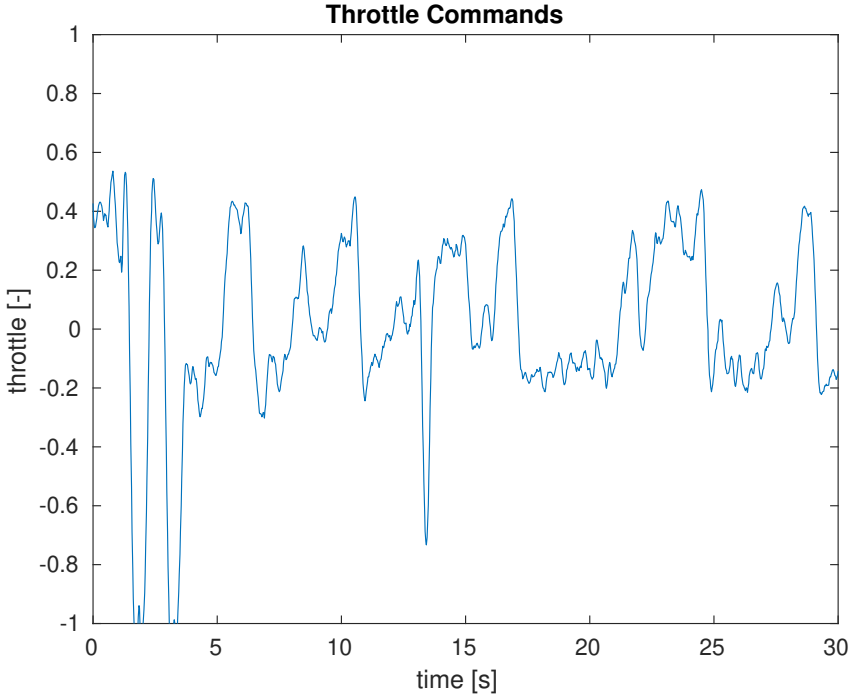


Figure 30: FSS throttle commands.

4.5.1 Velocity profile verification

The velocity profile calculation presented in section 4.2 has been verified through different runs, iterating over different longitudinal and lateral maximum accelerations. Afterwards, the GG plot is used to confirm the vehicle isn't surpassing the limitations imposed. Ensuring that the vehicle actually complies with the acceleration ellipse constraint also proves the performance of the Cruise Controller in keeping a small velocity following error. Figure 31 shows that the maximum lateral and longitudinal acceleration values from each run tend to follow the given limitations. However, the lateral acceleration maximum values overpass at some points its limit. This behavior is expected, given its inherent correlation with the lateral controller's performance and the dynamics of the tires.

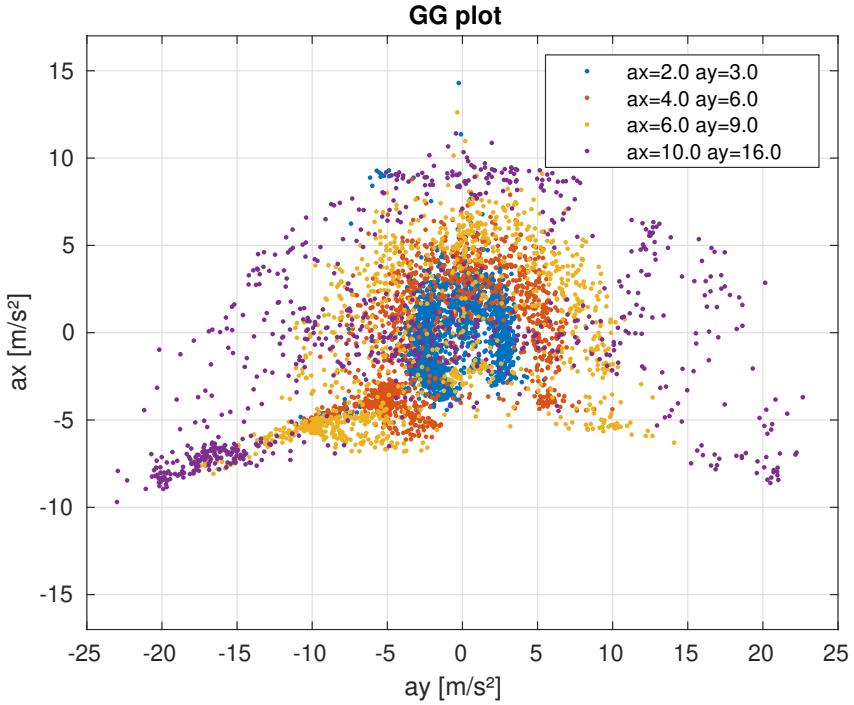


Figure 31: FSS track GG diagram (in SI units).

The lap time achieved with every set of acceleration limits is shown in Figure 32.

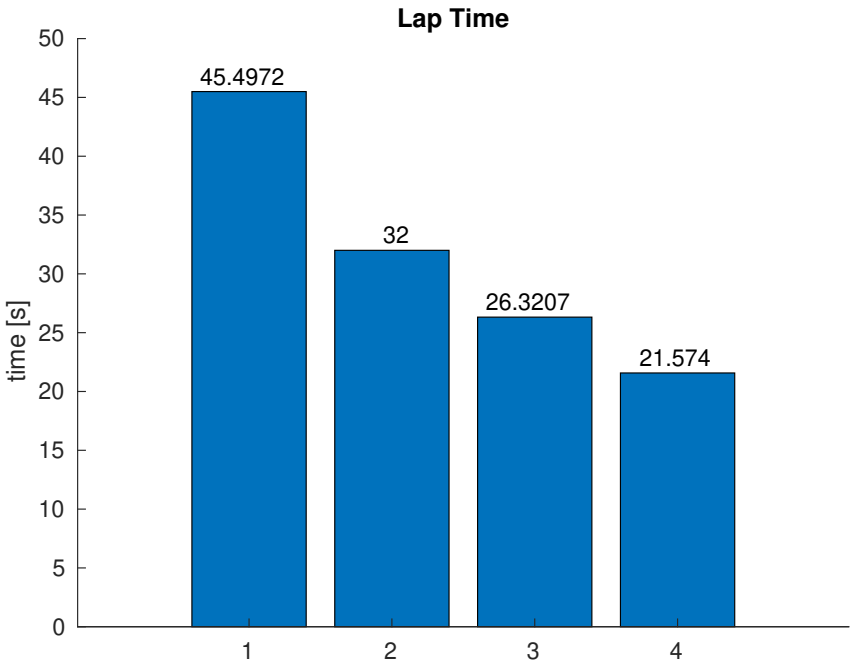


Figure 32: Lap time in FSS track.

5 Trajectory Optimization

5.1 Concept

Path planning is a critical aspect of every autonomous systems pipeline. It involves calculating a safe and optimal route for the vehicle to travel from its current location to a desired destination. This process relies on various components, including perception, localization, map representation, route planning, trajectory generation, collision avoidance, decision-making, optimization, and continuous real-time updates.

Path planning in autonomous vehicles is a complex and computationally intensive task that usually combines machine learning, sensor data processing, and control systems. Its success is essential for the safe and efficient operation of autonomous vehicles, enabling them to navigate urban and highway environments, adhere to traffic regulations, and avoid accidents.

Formula Student tracks are always delimited by cones (see the rules stated in [18]) and there can't be any dynamic objects inside the track, so collision avoidance is not taken into account in this thesis. For this reason, the most commonly found approaches in FS competitions make use of geometrical relationships and hardly ever take any vehicle dynamics restrictions into consideration.

In the context of this project, the task of the path planning module involves calculating the reference path that the vehicle will follow. This path is computed considering the track boundaries identified by the perception stack but also taking into account the vehicle model outlined in section 2.3.

To clarify, the path planning algorithm detailed in this thesis is designed as an offline strategy, needing prior knowledge of the entire track. This approach is adopted in order to start the Trackdrive event with a pre-computed, optimal trajectory.

The trajectory optimization presented in this section is based on a one-time prediction of the optimal state of the car at each point of the track. The final trajectory will be derived from the NLOP solution and the reference/input path.

5.2 Pre-processing

Before starting the optimization problem procedure, a previous trajectory candidate is needed. Given the cones that define the left and right track limits a pre-processing algorithm is defined in order to compute an approximation of the midline path. These midpoints are actually gates midpoints, defining a gate as the straight line that crosses a pair of points each one belonging to one track limit.

In order to compute such gates, the shorter track limit is chosen as reference and, from that, straight lines perpendicular to the tangent of this reference are obtained. Where this perpendicular straight lines (a.k.a gates) intersect the other track limit, new points are created. Repeating this procedure for the whole reference cones, gates for the whole track are defined.

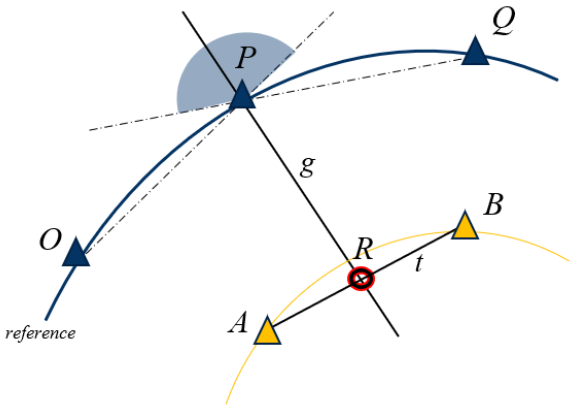


Figure 33: Gate generation diagram.

As shown in Figure 33, the main idea of the presented algorithm is to iterate over the whole reference cone positions P_i. Knowing the previous and next cones $O_i = P_{i-1}$ and $Q_i = P_{i+1}$ from the reference track limit, the angle bisector is calculated in order to obtain the perpendicular gate direction. The gate endpoint R_i will be the point where it crosses the other track limit. To find point R_i coordinates, the following vectors are defined:

$$\vec{m} = \frac{\vec{PO}}{\left\|\vec{PO}\right\|} + \frac{\vec{PQ}}{\left\|\vec{PQ}\right\|} = (m_x, m_y) \tag{34}$$

$$\vec{AB} = (ab_x, ab_y) \tag{35}$$

From this, the straight line t and the actual gate g are obtained:

$$t : y = \frac{ab_y}{ab_x}x + a_y - \frac{ab_y}{ab_x}a_x \tag{36}$$

$$g : y = \frac{m_y}{m_x}x + p_y - \frac{m_y}{m_x}p_x \tag{37}$$

Where $P = (p_x, p_y)$ and $A = (a_x, a_y)$. The new point $R = (R_x, R_y)$ is found where t and g cross:

$$R_x = \frac{a_x - p_y + \frac{m_y}{m_x}p_x - \frac{ab_y}{ab_x}a_y}{\frac{m_y}{m_x} - \frac{ab_y}{ab_x}a_y} \tag{38}$$

$$R_y = t(R_x) = g(R_x) = \frac{m_y}{m_x}R_x + p_y - \frac{m_y}{m_x}p_x \tag{39}$$

Once each P_i and R_i are found for each gate g_i, the midline points are computed:

$$MP_i = \frac{P_i + R_i}{2} \tag{40}$$

Keep in mind that a previous spline interpolation of both track limits could be made, so the procedure defined above doesn't necessarily have to be applied to cone positions but a better discretization of the reference track limits can be achieved, leading to more gates and a better defined midline trajectory.

Afterwards, the midline points are interpolated via C^2 parametric splines so a smaller and uniform discretization (0.025m) is achieved. With this parametric representation, the midline is defined as $\gamma(t) = (x(t), y(t))$ and the curvature at each point is computed:

$$\kappa(t) = \frac{x'(t)y''(t) - y'(t)x''(t)}{(x'(t)^2 + y'(t)^2)^{\frac{3}{2}}} \tag{41}$$

NOTE: For the C++ implementation of this pre-processing, the most critical operation is to find the closest two points on the other track limits to point P. In order to perform this operation in an efficient way, a *KD-tree* structure is defined with the discretized points from the non-reference track limit. This way, a fast nearest neighbors search can be made iteratively, returning both nearest points A and B for each P.

A *KD-tree* is a data structure for organizing points in a k-dimensional space. It recursively partitions the space along different dimensions, creating a hierarchical tree structure. This allows for efficient multidimensional searches, such as finding nearest neighbors. The tree is built by selecting dimensions, splitting the space based on median values, and repeating the process recursively. *KD-trees* are commonly used in computer science for faster multidimensional searches.

5.3 Non Linear Optimization Problem Formulation

The motion planning NLOP diverges from the one stated in section 3.2 so that it takes into account the complete vehicle model defined in section 2.3. Here, not only the lateral behaviour but also the longitudinal behaviour of the car are considered. Thus, a new inequality constraint can be added so the predicted states remain within a friction ellipse. Because the tire model explained in section 2.2.1 does not consider combined slip, this new constraint will prevent the high level layer from demanding unrealistic accelerations. In this case, the resultant trajectory will never request too high accelerations.

The state vector x is defined as $x = [\delta, F_m, n, \mu, v_x, v_y, w]^T$ and the control vector u is defined as $u = [\Delta\delta, \Delta F_m, M_{tv}]^T$. Note that the s-state is decoupled as explained in 2.3. Thus, the optimization variables vector z is defined as $z = [u, x]^T = [\Delta\delta, \Delta F_m, M_{tv}, \delta, F_m, n, \mu, v_x, v_y, w]^T$.

The NLOP description is as follows:

$$\underset{u,\,x}{\text{minimize}} \quad \sum_{k=0}^{N} J(u_k, x_k) \tag{42a}$$

subject to

$$x_{k+1} = F_s(u_k, x_k), \tag{42b}$$
$$x_N = x_0, \tag{42c}$$
$$x_k \in \chi,\ u_k \in \Upsilon, \tag{42d}$$
$$g_{Track}(x_k) \leq 0, \tag{42e}$$
$$g_{Ellipse}(x_k) \leq 0, \tag{42f}$$
$$k = 0, \ldots, N \tag{42g}$$

Where N is the total length of the input trajectory discretized by Δs, $F_s(u_k, x_k)$ represents the integrated space dependant vehicle model defined in section 5.3.1, $g_{Ellipse}(x_k)$ is the new friction ellipse constraint and $x_N = x_0$ represents a cyclic constraint so the prediction for the last and first states match.

5.3.1 Spatial Transform

The model presented in section 2.3 is expressed as a system of continuous time ODEs. For trajectory optimization purposes this model is transformed into the spatial domain with the progress s as the running variable instead of time t. This way, the predicted states will evolve with respect to space, so every new trajectory point can be computed for each Δs of the midline (input) path.

This transformation can be achieved as follows:

$$\dot{x} = \frac{\partial x}{\partial t} = \frac{\partial x}{\partial s}\frac{\partial s}{\partial t} \tag{43}$$
$$\frac{\partial x}{\partial s} = \frac{1}{\dot{s}} f_t(u(s), x(s)) = f_s(u(s), x(s)) \tag{44}$$

The final model used is:

$$\begin{bmatrix} \dot{\delta} \\ \dot{F_m} \\ \dot{n} \\ \dot{\mu} \\ \dot{v_x} \\ \dot{v_y} \\ \dot{w} \end{bmatrix} = \frac{1}{\dot{s}} \begin{bmatrix} \Delta\delta \\ \Delta F_m \\ v_x \sin\mu + v_y \cos\mu \\ w - \kappa(s)\dot{s} \\ \frac{1}{m}(F_x - F_{y,F}\sin\delta + m v_y w) \\ \frac{1}{m}(F_{y,R} + F_{y,F}\cos\delta - m v_x w) \\ \frac{1}{I_z}(F_{y,F}\cos\delta l_F - F_{y,R} l_R + M_z) \end{bmatrix} \tag{45}$$

It's worth pointing out that the time-to-space transformation remarkably increases the computation load of the NLOP as the complexity of the model increases. However, because this motion planning solution is thought to run offline it is not a major concern.

5.3.2 Cost Function

The objective function for trajectory optimization purposes differs from the one presented in section 3.2.3 in that it doesn't account for the path following weights Q_n and Q_μ. Because the optimal trajectory is obtained as a consequence of solving the NLOP, lateral deviation n and path heading μ states must be freely defined by maximizing the progress rate $\dot{s}$. Output commands weights are added so that realistic steering and throttle profiles can be achieved. Finally, the slip cost weight is left unchanged.

$$J(u_k, x_k) = -Q_s \dot{s}_k \; + \; R_d \Delta\delta_k^2 \; + \; R_{M_{tv}} M_{tv}^2 \; + \; Q_{slip} \beta_{x,k}^2 \tag{46}$$

5.3.3 Periodic Constraint

The newly introduced equality constraint in equation (42c) holds particular significance by ensuring alignment between the predicted states for the final state N and the prediction for the initial state 0. This terminal consideration eliminates the degree of freedom associated with the vehicle's starting pose. By enforcing this periodic behavior, the output states become independent of the initial state. Consequently, identical optimal states are computed for each track, regardless of the initial conditions. Furthermore, it guarantees the absence of any discontinuity between the final and initial states, being consistent with the motion planning objective defined in section 1.4: running the Trackdrive event, consisting of 10 laps.

5.3.4 Inequality Constraints

The track limits inequality constraint is defined identical as in section 3.2.2 so it won't be specified here.

As said before, the ellipse of forces inequality constraint prevents the motion planning module from demanding unrealistic accelerations, complying with an ellipse-shaped constraint stated as follows:

$$\rho_x F_x^2 + F_{y,F/R}^2 \leq \lambda D_{F/R}^2 \tag{47}$$

Where ρ_x defines the shape of the ellipse and λ determines the maximum combined force.

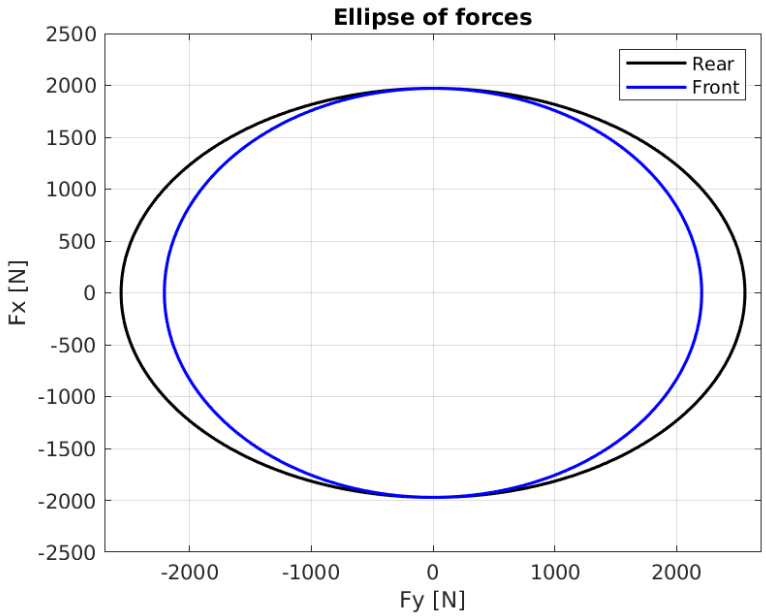


Figure 34: Ellipse of forces constraint.

However, this formulation is not very intuitive when it comes to tuning. Because of that, a reformulation of the forces ellipse can be made to obtain a GG diagram instead:

$$(\frac{F_x}{ma_{x,max}})^2 + (\frac{F_{y,F/R}}{ma_{y,max}})^2 \leq 1 \tag{48}$$

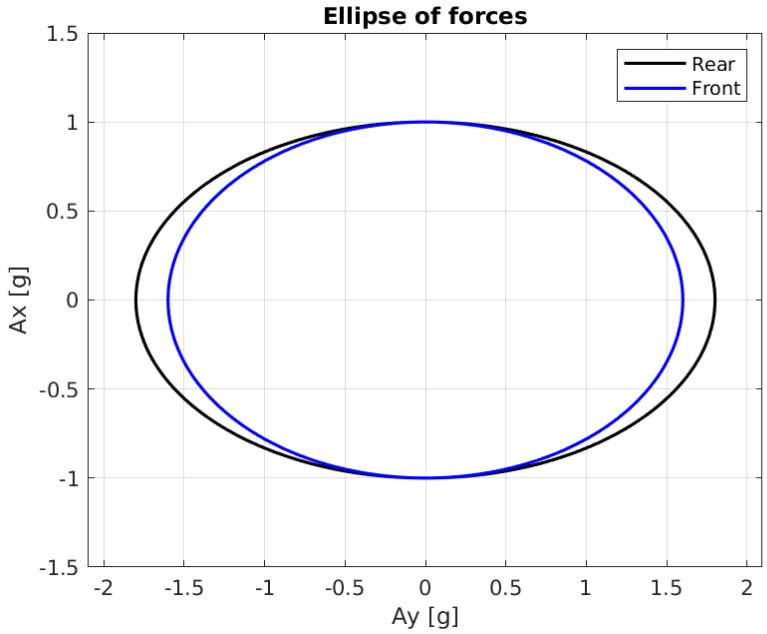


Figure 35: GG plot constraint.

With this formulation, the forces ellipse constraint ensures the predicted optimal states won't exceed the dynamic limitations of the car limiting the maximum longitudinal and lateral acceleration, which are set as constant parameters for the complete run.

5.4 Implementation

The motion planning module is implemented in C++. The software is divided into three different submodules or executables: pre-processing, optimization and planning, with the latter being the only module thought to run online, publishing the partial trajectory into the ROS network. The other two modules must be executed before the run takes place, saving their results in local memory.

The *pre-processing* module is in charge of computing the logic defined in section 5.2 and saving the fully discretized midline trajectory in local memory. Afterwards, the *optimization* module should be executed. This executable is in charge of setting up the NLOP and solving it taking as input the curvature values from the midline trajectory. The predicted states and control variables are then saved in local memory. Finally, the *planning* module should be executed together with the rest of the AS pipeline. This module is in charge of loading the NLOP solution and computing the optimal trajectory. Knowing the optimal μ and n states for each point of the midline path, the new trajectory points are obtained. The computed trajectory is then published into the ROS network so the Lateral MPC follows it.

The motion planning approach here presented does not only output the new trajectory points but also the optimal values for each state variable taken into account in the NLOP, which are $x = [\delta, F_m, n, \mu, v_x, v_y, w]^T$. This way, an optimal velocity profile is also obtained from each v_x solution. Moreover, v_y and w values for each trajectory point may be used as upper boundaries for the Lateral MPC in order to ensure the controller is taking into account the optimal motion, using the motion planning solution as a long term guidance.

5.4.1 Solver

In contrast to the Lateral MPC presented in section 3, the solver chosen for solving the NLOP defined above is Ipopt, included within the algorithmic differentiation software CasADi. This open-source software tool was chosen instead of Embotech's FORCESPRO mentioned in section 3.3.1 because of it's customization and flexibility capabilities. Due to the significantly tailored path planning solution here presented, CasADi's software together with Ipopt solver offered a much greater adaptability although inferior solve time performance could be expected. Because of the *offline* architecture of the algorithm, higher solve times are not a big concern. Ipopt solver, short for Interior Point Optimizer, implements an interior point line search filter method that aims to find a local solution of the NLOP. The mathematical details of the algorithm can be found in several publications (e.g. [19] and [20]).

5.5 Results

In order to tune the offline path planning approach presented in this section, a similar procedure as in Lateral MPC's 3.4 section has been followed because of the analogous vehicle model used in both algorithms. A discretization of 0.5 meters is used for all the presented results.

The final parameters set used for both tracks can be found in appendix B.

The **progress rate maximization term** Q_s now becomes more sensitive because the v_x variable is one of the state variables of the NLOP, meaning for higher Q_s higher longitudinal velocities will be achieved. As explained in section 3.4.1, progress rate maximization causes approximate minimal curvature paths with racing alike final trajectories. Thus, the progress rate weight becomes the most sensitive tuning parameter of this path planning algorithm, as it directly defines the degree of tracing of the optimal trajectory. In consequence, the curve tracing is tightly coupled with the longitudinal velocity profile. For greater curve tracing, higher velocities for the whole track.

In Figures 36 and 37 the final trajectory is shown for both simulation tracks presented previously. The blue path represents the midline trajectory obtained following the pre-processing explained in section 5.2. The black path represents the final optimal trajectory after the NLOP has been solved.

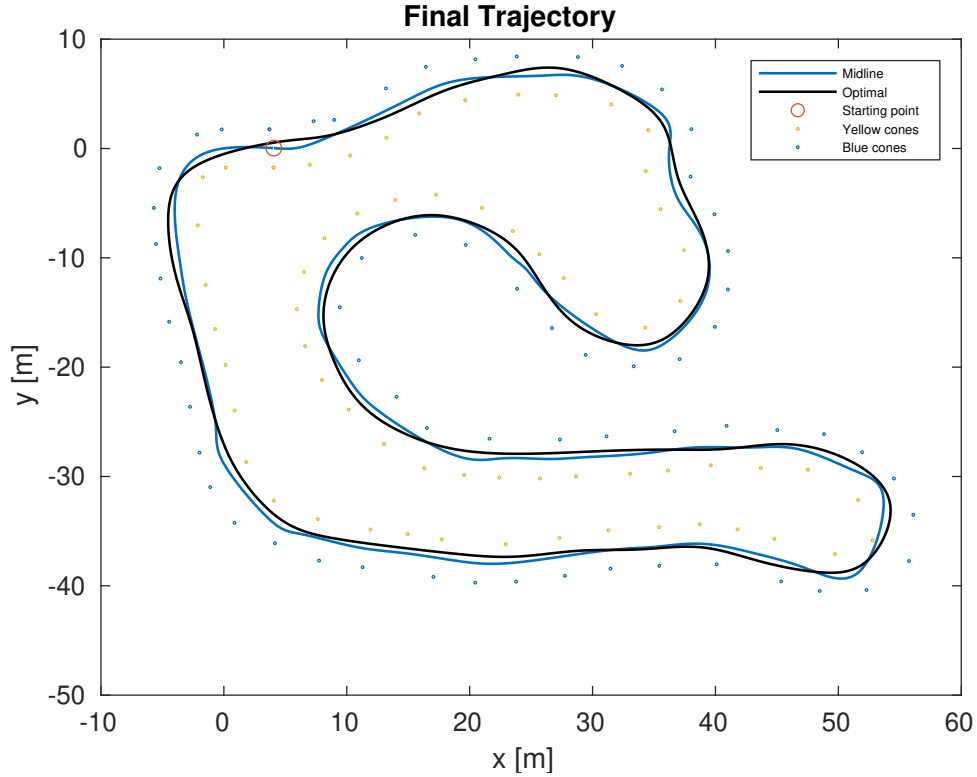


Figure 36: FSS Midline vs Optimal trajectory.

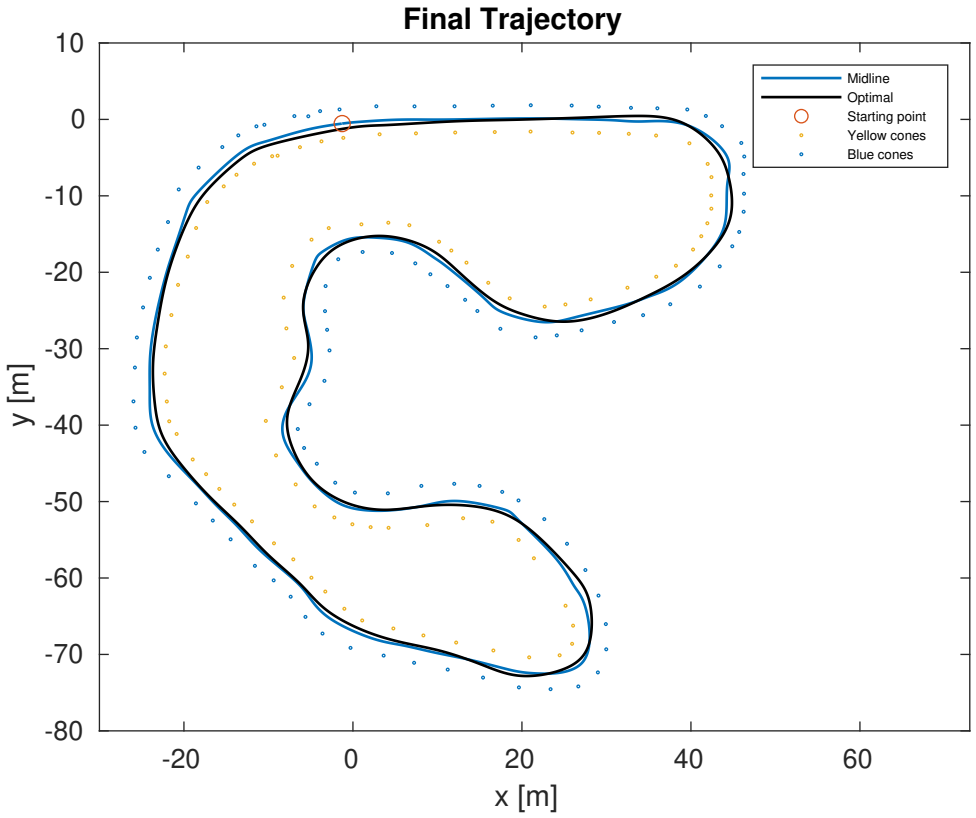
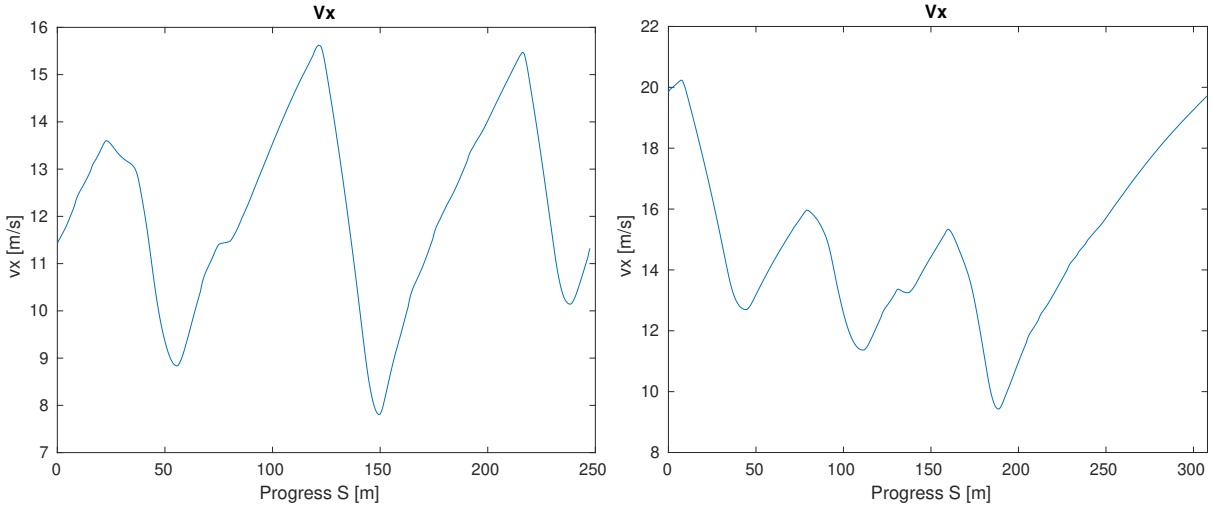


Figure 37: FSG Midline vs Optimal trajectory.

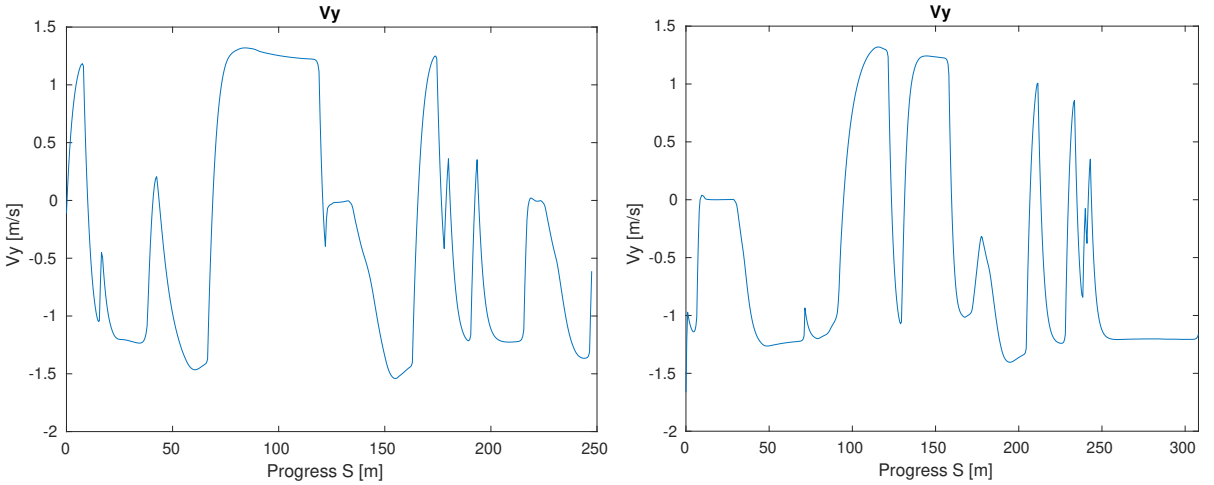
Although this algorithm is thought to run offline, some conclusions can be extracted from its solving time. Because of the spatial discretatization presented in section 5.3.1 the NLOP's solving time directly depends on the total length of the track. For longer tracks, more midpoints will have to be evaluated so the horizon length of the total prediction will be higher. For FSS track the solving time is 11.47 seconds and for FSG track is 20.09 seconds. FSS track is 240 meters long while FSG track is 320 meters long.

Finally, the main advantage of this trajectory optimization algorithm is that it does not output an optimal trajectory but it is extrapolated from the optimal states. The NLOP solution comprises optimal values for the state vector $x = [\delta, F_m, n, \mu, v_x, v_y, w]^T$ for each stage of the horizon length, in this case the length of the track. Thus, apart from the final trajectory computed with n and μ states, odometry profiles also arise from the solution. In Figures 38, 39 and 40 the odometry profiles for FSS and FSG tracks are shown.



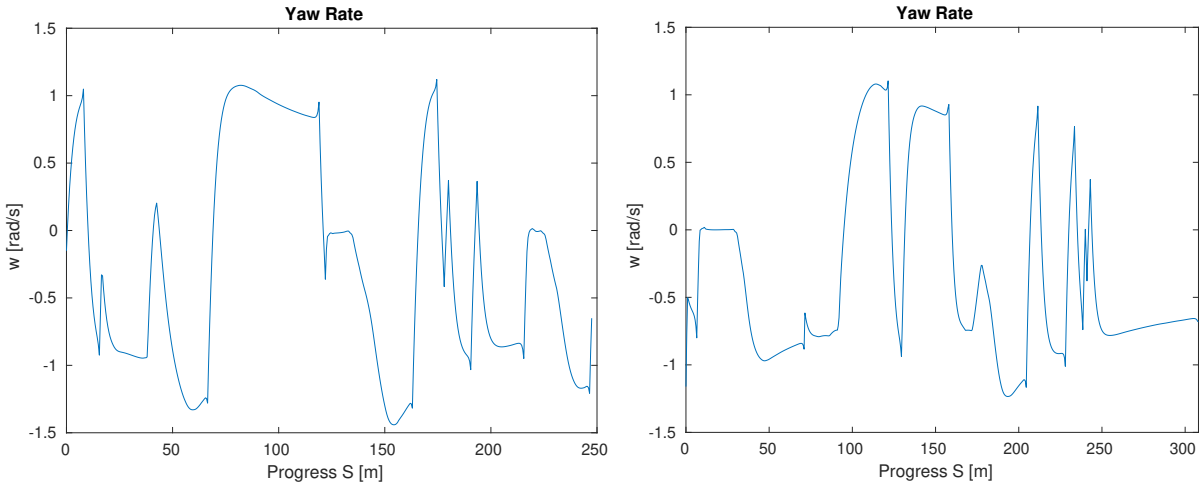
(a) FSS (b) FSG

Figure 38: Longitudinal velocity profile.



(a) FSS (b) FSG

Figure 39: Lateral velocity profile.



(a) FSS (b) FSG

Figure 40: Yaw rate profile.

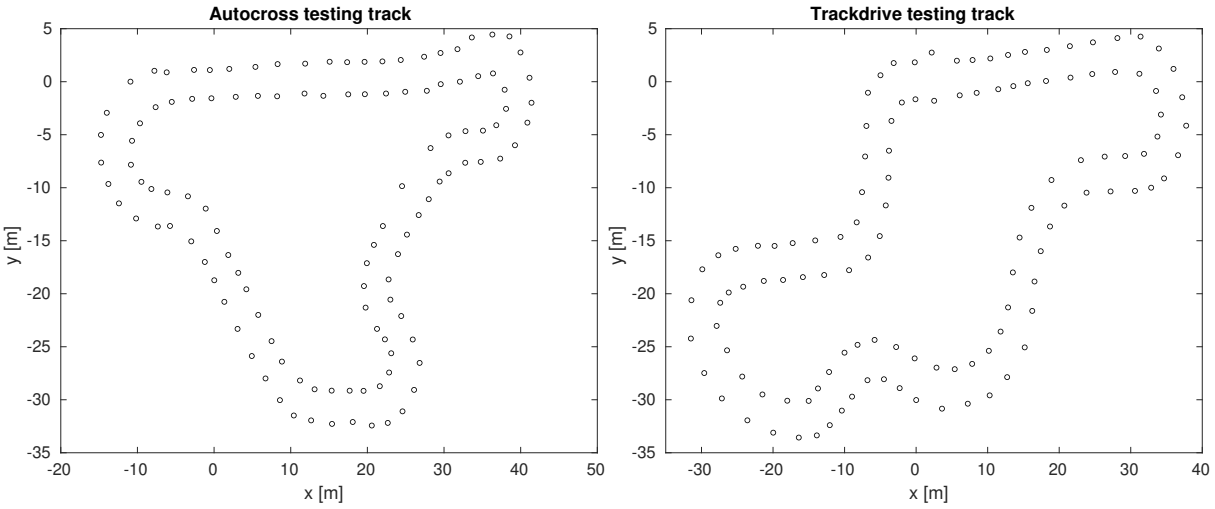
Compared to the "kinematic" velocity profile calculation defined in section 4.2, smoother profiles are achieved with less dependency on curvature's variability. Calculating curvature for splines presents multiple challenges, which include numerical instability, potential parameterization issues, sensitivity to the choice of basis functions or complexities in handling closed splines. As a consequence, it is common to compute curvature profiles with inherent noise that may not accurately represent the reality. Thus, the velocity profile in Figure 27 greatly differs the one presented in Figure 38, the former being much less steady than the latter due to curvature's distortion.

Thanks to the periodic constraint presented in (42c), every profile starts and finishes with the same exact value, enabling this way a loop closure without any discontinuities.

6 Overall Testing Results

At last, the testing performance of the Lateral MPC together with the Cruise Controller is presented. The results are divided into the data collected in an autocross and a trackdrive run, both conducted in Parc del Fòrum, Barcelona. Different tracks are chosen for each one of the events. In Figure 41 the 145 meters long autocross track is shown as well as the trackdrive track, with an overall distance of 171 meters.

The complete trackdrive run can be found in the following video: How the Autonomous Pipeline of a Formula Student Car Works - BCN eMotorsport.



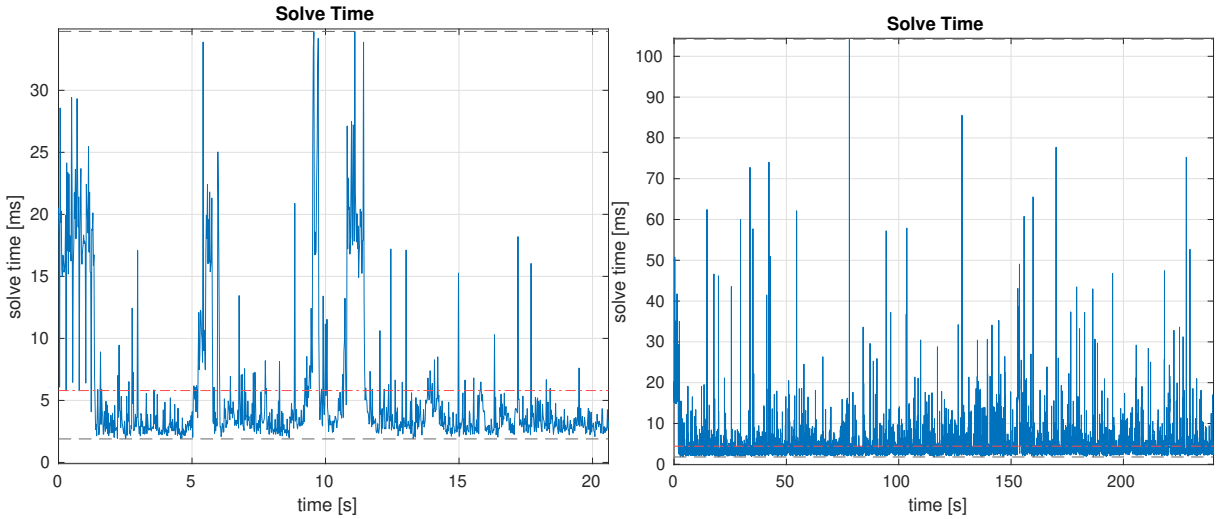
(a) Autocross (b) Trackdrive

Figure 41: Testing tracks.

The parameters used for both tests are summarized in appendix A.

6.1 Solve time

In Figure 42 the computing times of the Lateral MPC for both runs are presented. Although the solving time is higher compared to simulation performance, the controller can keep up with its 40 Hz main loop without major problems. The solve time mean of the autocross run is **5.74** ms and **4.43** ms is the solve time mean for all 10 laps of the trackdrive event. During the autocross run peak values of 40 ms can be found while some peaks over 60 ms are found in the trackdrive event.

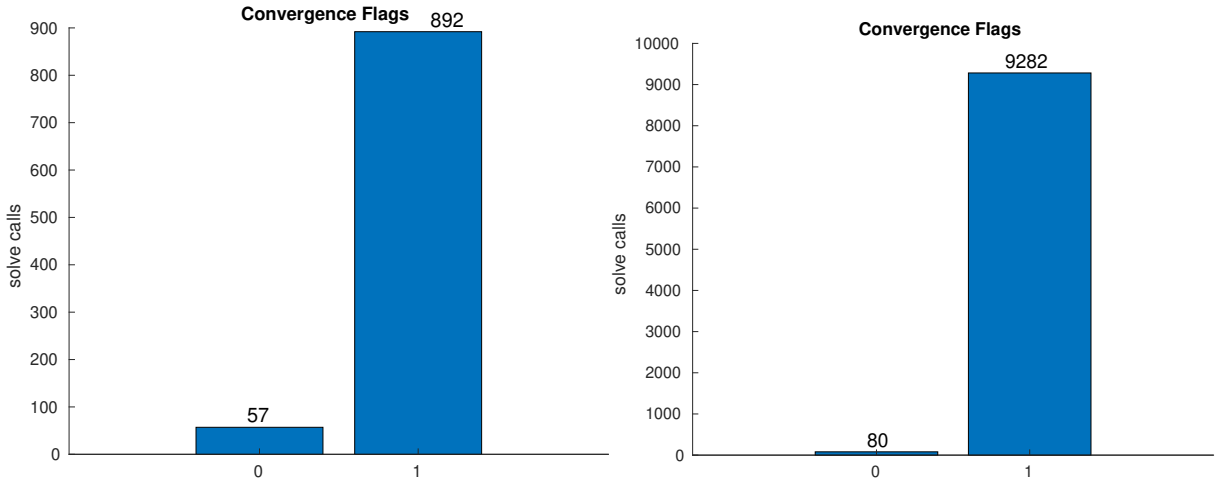


(a) Autocross (b) Trackdrive

Figure 42: Solve time.

6.2 Exit flags

The percentage of maximum number of iterations reached with respect to the total number of solver calls for the autocross run is of **6.0%**. For the trackdrive run this percentage is much lower: **0.85%**. That's because during the trackdrive event the planned trajectory is much more stable, which greatly benefits the NLOP convergence.



(a) Autocross (b) Trackdrive

Figure 43: Exit flags.

6.3 Performance

Here the autocross target and actual odometry values are presented. As expected, a greater missmatch exists compared to the simulation results. However, both controllers are able to drive the car with outstanding performance maintaining the vehicle's stability with a maximum velocity of **12 m/s**.

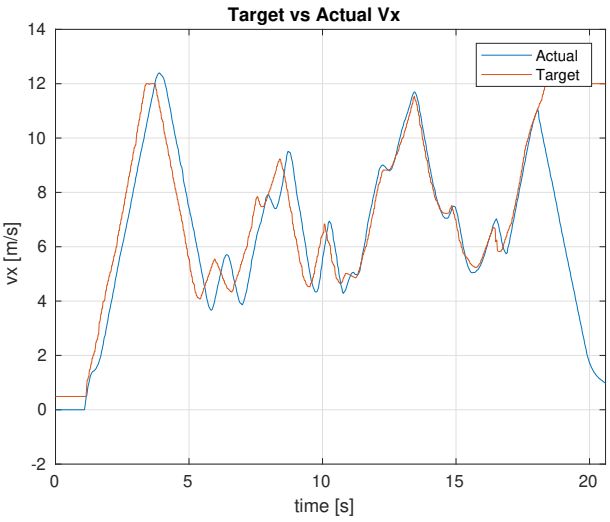


Figure 44: Actual vs Target long. velocity.

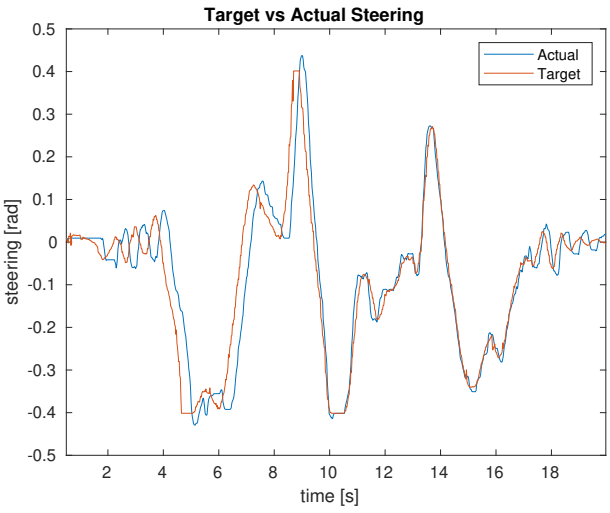


Figure 45: Actual vs Target steering angle.

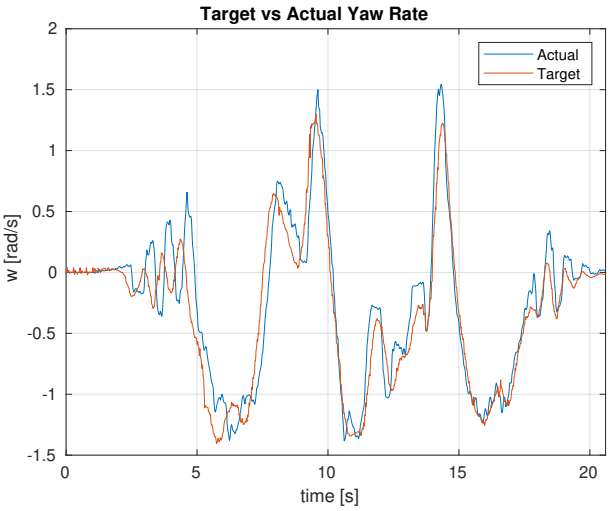


Figure 46: Actual vs Target yaw rate.

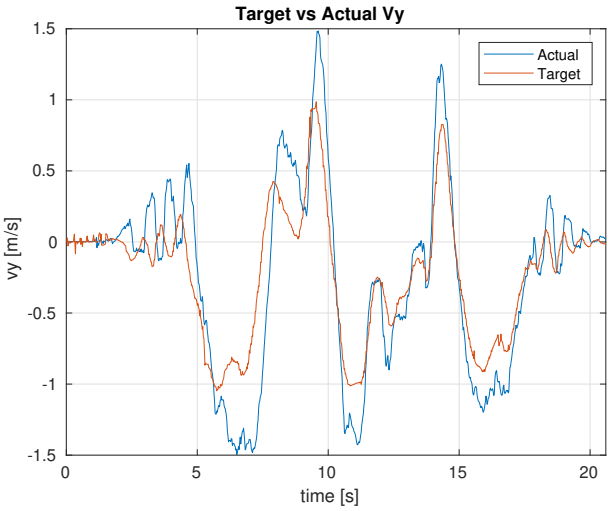
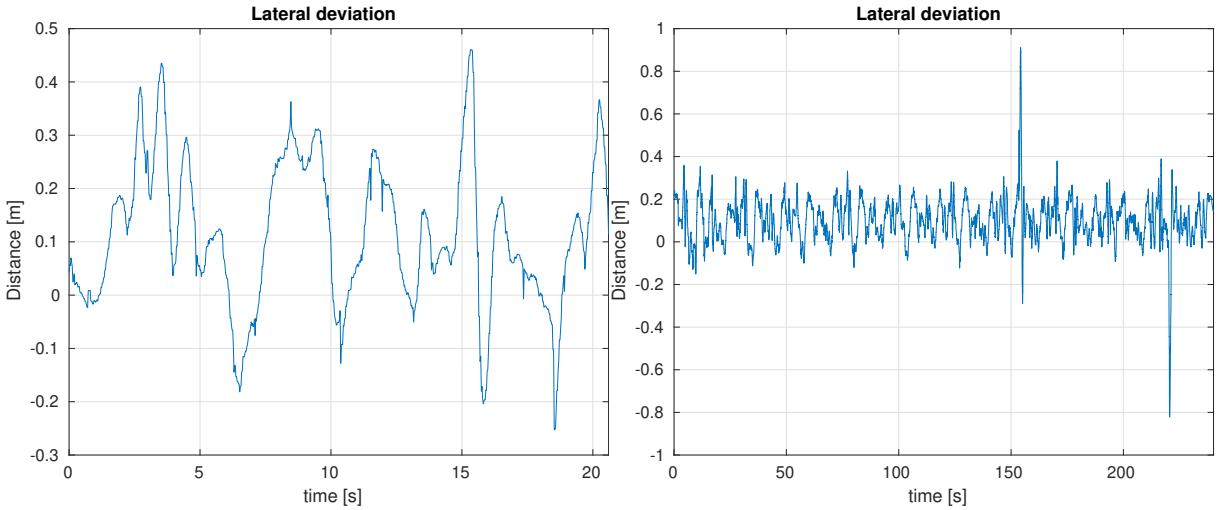


Figure 47: Actual vs Target lateral velocity.

In Figure 44 it is clearly visible the correction made by the integral component of Cruise Controller's PID reducing the velocity tracking error over time.

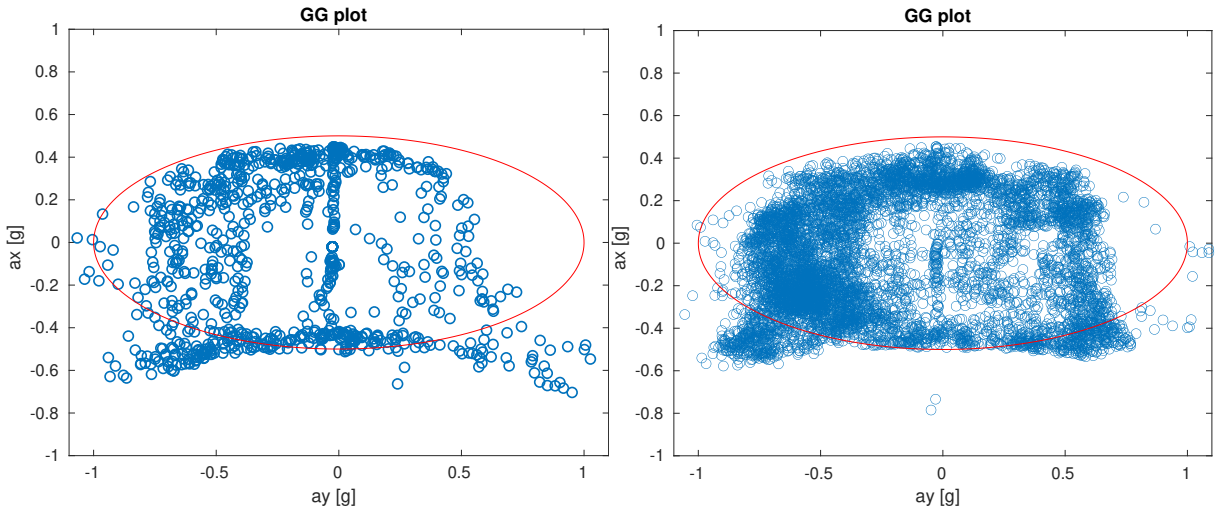
The lateral deviation of the car with respect to the reference trajectory is kept within admissible values for both tests, as it can be seen in Figure 48.



(a) Autocross (b) Trackdrive

Figure 48: Lateral deviation.

Finally, the GG plots of the autocross and trackdrive runs are shown. The acceleration boundaries imposed during both tests can be found in appendix A.2. Similar to the behaviour seen in simulation tests, the lateral acceleration exceeds the defined boundaries at some working points. On the countrary, the longitudinal acceleration perfectly fits the given constraints.



(a) Autocross (b) Trackdrive

Figure 49: GG diagram.

7 Budget

The implementation of this project requires both hardware and software equipment. Here all the hardware dependencies of the complete autonomous systems pipeline are addressed as well as the used software licenses. The CAT15X autonomous driving FS single-seater employs one 32-channel LiDAR sensor, a high-performance Dual Antenna GNSS-Aided Inertial Navigation System, a self-assembled Processing Unit, an Emergency Braking System and a Brushless DC motor for steering actuation. Regarding the software dependencies, open source distributions were mainly used except for Embotech's FORCESPRO solver and IPG CarMaker simulation framework.

Notice that the materials and manpower needed to build the monocoque, dynamics, aerodynamics, electronics and powertrain of the CAT15X prototype won't be included in this budget.

The personal cost is divided into the wages for a junior engineer and a degree thesis tutor, of 10€/h and 30€/h respectively.

The final budget of this project is detailed in Table 2. Note that the products obtained via sponsorships agreements result in a final cost of 0€. It's possible that some product costs do not match with the actual prices as the initial market prices are the ones considered in the following table.

Product	**Q**	**€/unit**	**Total Cost [€]**	**Final Cost [€]**
Velodyne VLP-32C	1	13.000,00	13.000,00	0,00
Vectornav VN300	1	4.850,00	4.850,00	0,00
Intel i9 10900	1	444,07	444,07	444,07
Motherboard IMB 1222 VW	1	348,00	348,00	348,00
Emergency Brake System	1	3.100,00	3.100,00	3.100,00
Remote Emergency System	1	1.735,50	1.735,50	1735,50
Maxon BLDC motor	1	300,00	300,00	210,58
FORCESPRO Engineering License *360-day subscription*	3	2.947,28	8.841,84	0,00
FORCESPRO SW License *360-day subscription*	5	982,43	4.912,15	0,00
FORCESPRO HW License *360-day subscription*	3	11.789,13	35.367,39	0,00
IPG CarMaker License	1	-	-	0,00
Junior Engineer (10€/h)	1080	10.800,00	10.800,00	10.800,00
Degree thesis tutor (30€/h)	15	450,00	450,00	450,00
TOTAL			**84.148,95**	**17.088,15**

Table 2: Budget

8 Environmental Impact

Formula Student Competitions are deeply involved with the electrification of vehicles. In fact, the first Formula Student competition for electric cars took place in 2010 setting up a new major challenge for the student engineering community. From then on, most of the teams started moving into electric energy sources and drivetrain, enhancing the competitiveness of their prototypes. In 2023 the last Formula Student Germany CV competiton was held, pushing the final combustion engine teams into electrification.

Nowadays, Electric Vehicles (EVs) have emerged as a promising solution to address environmental concerns associated with traditional internal combustion engine vehicles. The widespread adoption of electric vehicles is seen as a pivotal step towards reducing greenhouse gas emissions, mitigating air pollution, and promoting a more sustainable future.

One of the primary environmental advantages of electric vehicles is their significantly lower tailpipe emissions. Unlike conventional vehicles powered by gasoline or diesel, electric vehicles produce zero tailpipe emissions during operation. This shift to electric power contributes to improvements in air quality, particularly in urban areas where vehicular emissions are a major source of pollution.

The autonomous vehicles industry has ignited discussions about its environmental impact, sparking both optimism and concerns. On the positive side, autonomous vehicles have the potential to revolutionize transportation, making it more efficient and reducing the overall carbon footprint. Autonomous vehicles, equipped with advanced sensors and communication technologies, can navigate traffic patterns more efficiently, minimizing congestion and subsequently lowering fuel consumption. This improvement in traffic management could lead to a reduction in greenhouse gas emissions and air pollutants.

9 Conclusions and Future Work

In conclusion, within this work a reliable high level autonomous driving control pipeline has been accomplished. The Lateral MPC and the Adaptive Cruise Controller have proven to achieve an outstanding performance in both simulation and testing environments. In fact, throughout the testing period lasting one month and a half, the CAT15X drove **19.36 km** in driverless mode.

The autonomous systems pipeline of CAT15X has exceeded all prior team achievements by securing a **P1** in the Driverless Skidpad event and achieving a **P2** finish in the Trackdrive event during the Formula Student Spain 2023 competition. In the latter event, both the Lateral MPC and the Cruise Controller proposed in this thesis were in charge of the high level commands of the single-seater. In the Skidpad event, as outlined in section 1.5.1, the car was guided by the Cruise Controller together with a Pure Pursuit.

In terms of the Lateral MPC algorithm, it is able to safely run at the objective loop rate of 40 Hz, fulfilling the control specifications defined in section 1.5.1. In addition, also complying with the control specifications, it ensures a really high level of reliability for all the feasible velocities range of CAT15X. Furthermore, the high-fidelity simulation software developed this season has enabled to fine tune the controller before the actual testing started, achieving remarkable results despite the short testing period available in Formula Student's schedule. Finally, the presented advanced controller has demonstrated to be robust in front of path planning re-calculations, an essential feature when racing in unknown tracks (as in the autocross event).

The Adaptive Cruise Controller presented in this thesis has proved to achieve a small velocity following error when employing GG-diagram based dynamic velocity profiles. Furthermore, its PID architecture has substantially reduced the tuning complexity of the overall control pipeline. This represents a great advantage in Formula Student field, where the testing time is extremely limited and shared between driverless and manual modalities.

Regarding the Motion Planning algorithm, although it has not been tested on the real car it has shown promising results in simulation. Because of its *offline* facet, simulation results extensively prove its performance as its results are unique for each track.

Although the three algorithms proposed in this thesis have proven to perform well while adhering to the specifications defined in section 1.5, several improvements could be introduced to enhance its overall capabilities.

On the one hand, a coupled control solution could be developed in order to handle both steering and throttle commands. Coupled MPC solutions have already been developed in BCN eMotorsport's previous seasons. However, the extremely limited testing schedule makes the tuning process implausible, ending with controllers with big reliability issues. The future coupled control solution should be developed together with an automatic tuning strategy so that optimal cost function weights were guaranteed. An adaptive MPC approach could also be explored, so the controller could dynamically adapt its tuning at the same time that is being tested. For state of the art works on this topic see section 1.3.3.

On the other hand, due to the continuously changing *online* trajectory's length, a spatial MPC approach could be designed. The main idea behind this solution is to be able to adapt the prediction horizon with respect to the overall distance of the received trajectory. During the autocross event, the number of detected cones defines the total length of the trajectory. As you

progress along the event, more cones are detected and usually the overall detection range increases (e.g. when the car is on a straight line) triggering longer planned trajectories. With the spatial dynamic bicycle model described in 5.3.1 an MPC with dynamic horizon length could be developed so that the discretization step changes with respect to the path length. The output control commands should be transformed into the time domain afterwards. An initial implementation of this approach can be found here.

Finally, an enhancement to the Adaptive Cruise Controller involves incorporating a feedforward component. By calculating an initial throttle estimate based on longitudinal acceleration values throughout the velocity profile, the feedback action of the PID controller would be then primarily focused on refining this initial estimate.

References

[1] Antoni Salom Llabrés. *Non-linear Model Predictive Control for a Formula Student Racing Autonomous Vehicle*. http://hdl.handle.net/2117/372963, 2022.

[2] Albert Gassol Puijaner. *Development of the Model Predictive Controller and Simultaneous-Localization-And-Mapping Algorithm of the Control System*. http://hdl.handle.net/2117/358190, 2021.

[3] Juraj Kabzan; Miguel I. Valls; Victor J.F. Reijgwart; Hubertus F.C. Hendrikx; Claas Ehmke; Manish Prajapat; Andreas Bühler; Nikhil Gosala; Mehak Gupta; Ramya Sivanesan; Ankit Dhall; Eugenio Chisari; Napat Karnchanachari; Sonja Brits; Manuel Dangel; Inkyu Sa; Renaud Dubé; Abel Gawel; Mark Pfeiffer; Alexander Liniger; John Lygeros and Roland Siegwart. *AMZ Driverless: The Full Autonomous Racing System*. 2019.

[4] Sirish Srinivasan; Sebastian Nicolas Giles; Alexander Liniger. *A Holistic Motion Planning and Control Solution to Challenge a Professional Racecar Driver*. 2021.

[5] Sherif Nekkah; Josua Janus; Mario Boxheimer; Lars Ohnemus; Stefan Hirsch; Benjamin Schmidt; Yuchen Liu; David Borbély; Florian Keck; Katharina Bachmann and Lukasz Bleszynski. *The Autonomous Racing Software Stack of the KIT19d*. 2020.

[6] Andres Alvarez; Nico Denner; Zhe Feng; David Fischer; Yang Gao; Lukas Harsch; Sebastian Herz; Nick Le Large; Bach Nguyen; Carlos Rosero; Simon Schaefer; Alexander Terletskiy; Luca Wahl; Shaoxiang Wang; Jonona Yakupova; Haocen Yu. *The Software Stack That Won the Formula Student Driverless Competition*. 2021.

[7] Danilo Caporale, Alessandro Settimi, Federico Massa, Francesco Amerotti, Andrea Corti, Adriano Fagiolini, Massimo Guiggiani, Antonio Bicchi, and Lucia Pallottino. Towards the design of robotic drivers for full-scale self-driving racing cars. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 5643–5649, 2019.

[8] Yassine Kebbati; Vicenç Puig; Naima Ait-Oufroukh; Vincent Vigneron; Dalil Ichalal. Optimized adaptive mpc for lateral control of autonomous vehicles, 2021.

[9] Maximilian Brunner; Ugo Rosolia; Jon Gonzales and Francesco Borrelli. *Repetitive Learning Model Predictive Control: An Autonomous Racing Example*. 2017.

[10] András Mihály; Balázs Németh and Péter Gáspár. *Analysis of driver behavior related to look-ahead control*. 2012.

[11] Stan Broere and Mauro Salazar. *Minimum-lap-time Control Strategies for All wheel Drive Electric Race Cars via Convex Optimization*. 2021.

[12] Alexander Liniger; Alexander Domahidi and Manfred Morari. *Optimization-Based Autonomous Racing of 1:43 Scale RC Cars*. 2017.

[13] Sebastiaan van Aalst. *Virtual Sensing for Vehicle Dynamics*. PhD thesis, Arenberg Doctoral School. Faculty of Engineering Science, 2020.

[14] Jason Kong, Mark Pfeiffer, Georg Schildbach, and Francesco Borrelli. Kinematic and dy-

namic vehicle models for autonomous driving control design. In *2015 IEEE Intelligent Vehicles Symposium (IV)*, 2015.

[15] Eugenio Alcalá Baselga. *Advances in planning and control for autonomous vehicles*. PhD thesis, UPC, Departament d'Enginyeria de Sistemes, Automàtica i Informàtica Industrial, 2020.

[16] Hans B. Pacejka and Egbert Bakker. *The magic formula tyre model. Vehicle System Dynamics*. 1992.

[17] Prof. Dr.-Ing. Frank Allgöwer. Data-driven mpc: From linear to nonlinear systems with guarantees. https://www.youtube.com/watch?v=9GP1dmj58cw, 2022.

[18] Formula Student Germany. Formula student rules v1.1. https://www.formulastudent.de/fileadmin/user_upload/all/2023/rules/FS-Rules_2023_v1.1.pdf, 2023.

[19] A. Wächter. *An Interior Point Algorithm for Large-Scale Nonlinear Optimization with Applications in Process Engineering*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 2002.

[20] A. Wächter and L.T. Biegler. *On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming*. 2006.

Appendices

A Testing parameters set

A.1 Lateral MPC

Parameter	Summary	Value [SI]
Q_s	Progress rate weight	10.0
Q_n	Normal distance weight	550.0
Q_μ	Heading (track reference) weight	0.1
Q_{slip}	Slip difference weight	2.0
Q_{slack}	Track constraint slack weight	1000.0
R_d	Steering rate weight	100.0
$R_{M_{tv}}$	Additional moment weight	1.0
B_f	Pacejka Constant	10.5507
B_r	Pacejka Constant	10.5507
C_f	Pacejka Constant	-1.2705
C_r	Pacejka Constant	-1.2705
D_f	Pacejka Constant	2208.0635
D_r	Pacejka Constant	2563.599
L_f	Distance from CoG to front axis	0.708
L_r	Distance from CoG to rear axis	0.822
L	Total car's length	2.72
W	Total car's width	1.5
m	Car's mass	210.0
I_z	Inertial moment around z axis	180.0

Table 3: Lateral MPC parameters set

A.2 Cruise Controller

Parameter	Summary	Value [SI]
K_p	PID's proportional gain	1.2
K_i	PID's integral gain	0.1
K_d	PID's derivative gain	0.0
$a_{y,max}$	Maximum lateral acceleration	7.0
$a_{x,max}$	Maximum longitudinal acceleration	4.0
$a_{x,brake,max}$	Maximum braking acceleration	6.0

Table 4: Cruise Controller parameters set

B Motion Planning parameters set

Parameter	Summary	Value [SI]
Q_s	Progress rate weight	1.0
Q_{slip}	Slip difference weight	0.1
R_d	Steering rate weight	1.4
R_a	Acceleration rate weight	0.3
$R_{M_{tv}}$	Additional moment weight	1.0
B_f	Pacejka Constant	10.5507
B_r	Pacejka Constant	10.5507
C_f	Pacejka Constant	-1.2705
C_r	Pacejka Constant	-1.2705
D_f	Pacejka Constant	2208.0635
D_r	Pacejka Constant	2563.599
L_f	Distance from CoG to front axis	0.708
L_r	Distance from CoG to rear axis	0.822
L	Total car's length	2.72
W	Total car's width	1.5
m	Car's mass	210.0
I_z	Inertial moment around z axis	180.0
ρ_{air}	Air's density	1.255
A_{aero}	Drag area	1.0
C_d	Drag coefficient	1.2727
C_{motor}	Motor Constant (max. N)	4283.4645
C_r	Rolling resistance (% of car's weight)	0.45%

Table 5: Trajectory Optimizer parameters set