UCLouvain

epl

École polytechnique de Louvain

Control Algorithm for an Autonomous Electric Race Car

Authors: **Thomas Brouwers, Louis Libert**
Supervisor: **Julien Hendrickx**
Readers: **Nicolas Docquier, Paul Fisette, Renaud Ronsse**
Academic year 2022–2023
Master [120] in Electro-mechanical Engineering

Acknowledgements

We express our heartfelt gratitude to the Formula Electric Belgium (FEB) team, particularly Guillaume Degroote, Chief Driverless at FEB, for their invaluable insights and expertise throughout our research journey. Their dedication to excellence and willingness to share their knowledge have played a crucial role in shaping the direction and success of this thesis.

Furthermore, we extend our deep appreciation to our thesis supervisor, Prof. Julien Hendrickx, from the Université Catholique de Louvain, for his invaluable guidance, constructive feedback, and mentorship.

We also want to acknowledge the unwavering support of our families, friends, and all those who have stood by us throughout this thesis. Their love, support, and belief in our abilities have been the driving force behind our achievements.

Lastly, we would like to recognize the contributions of all individuals and institutions whose research, publications, and resources have contributed to the knowledge foundation of this thesis. Their groundbreaking work has laid the foundation for our research and provided us with valuable references and inspiration.

Abstract

This paper presents a comprehensive exploration of control algorithms for optimizing performance in autonomous race cars. The control algorithm designed in this research is split into two key components: path planning and path following, with a focus on the path following algorithm and more specifically on the implementation of a Model Predictive Contouring Control (MPCC). The objective of this study is to develop a competitive algorithm that can be applied to the autonomous electric race car built by Formula Electric Belgium for participation in the Formula Student Driverless competition. The research explores the intricacies of path following, optimizing steering, throttle, and brake inputs to maximize the progress of the car while staying within the track boundaries. Various solutions are explored, encompassing different optimization problem formulations, car models, and optimization solvers. The effectiveness of these solutions is evaluated using the Formula Student Driverless Simulator. The study thoroughly discusses the benefits and challenges of the MPCC algorithm, presenting compelling results and performances achieved through various strategies. It provides valuable insights for the implementation of control systems in autonomous race cars and suggests promising avenues for future research to further enhance control performance in this context.

Keywords: control algorithm; autonomous electric race car; path following; path planning; model predictive contouring control

Table of Contents

List of Figures

List of Tables

List of Abbreviations

ADMM Alternating Direction Method of Multipliers 7

CILQR Constrained Iterative Linear Quadratic Regulator 7

DQN Deep Q-Networks 7

FEB Formula Electric Belgium 1, 3, 10, 14

GP Gaussian Process 9

GPMPC Gaussian Process model predictive control 9

GSS Ground Speed Sensor 30

HRHC Hierarchical Receding Horizon Control 10

ILQR Iterative Linear Quadratic Regulator 7

IMU Inertial Measurement Unit 30

LMPC Learning Model Predictive Control 8

LPV Linear Parameter Varying 9

LPV-MPC Linear Parameter Varying - Model Predictive Control 9

LQR Linear Quadratic Regulator 7, 8

MPC Model Predictive Control 7–10, 29, 31, 32, 48, 51, 57

MPCC Model Predictive Contouring Control 9, 10, 29, 51, 52, 57, 58, 62, 66, 67, 75, 80, 83, 84, 86, 91, 98, 100, 105, 106, 125

NMPC Nonlinear Model Predictive Control 8, 9

PANOC Proximal Averaged Newton-type method for Optimal Control 124, 125

PF Path Following 5–8, 10, 14, 15, 19, 20, 22, 24, 29–31, 50, 62, 63, 74, 75, 101–103, 124

PID Proportional-Integral-Derivative 6

PP Path Planning 5, 6, 8, 10, 13–15, 19–23, 25–27, 30, 50, 52, 61, 62, 64, 101, 103, 115, 119

PPO Proximal Policy Optimization 7

ROS Robot Operating System 101

SAC Soft Actor-Critic 7

SLAM Simultaneous Localization And Mapping 14, 16, 20, 22–27, 30, 101, 115, 118, 119

SMC Sliding Mode Control 8

TMPC Tube Model Predictive Control 8

1 | Introduction

1.1 Context

1.1.1 Formula Electric Belgium

The work presented in this paper was carried out in collaboration with Formula Electric Belgium (FEB) [1], with a specific focus on the development of autonomous vehicle algorithms. The project's primary goal was to contribute to the organization's advancements in this area, offering valuable insights and paving the way for future developments.

FEB is a student team based in Leuven that is dedicated to the design, development, and racing of electric vehicles. Every year, they build a new electric race car with the goal of participating in the international competition known as Formula Student. This competition specifically focuses on design and racing for student-built race cars.

The team comprises 52 students from diverse backgrounds, collaborating to design and build fast, efficient electric race cars. Divided into departments responsible for specific aspects of the project, they leverage cutting-edge technology, partnerships, and high-quality parts to ensure their success.

Recently, they have expanded their focus to develop an autonomous vehicle, demonstrating their commitment to pushing the boundaries of electric vehicle technology and autonomous driving. In addition to engineering work, they actively raise public awareness through seminars, events, and collaborations, contributing to the growth of electric vehicle research and development.

1.1.2 Formula Student Competition

Formula Student is an internationally recognized engineering competition that offers students a unique opportunity to showcase their skills in designing, building, and racing formula-style race cars. The competition is structured around a series of static and dynamic events, designed to assess various aspects of the participating teams' vehicles.

Figure 1.1: Student Germany 2017 [2]

These events encompass the key aspects of the competition, combining technical expertise, cost management, business acumen, and performance evaluation. They provide a comprehensive assessment of the teams' race cars and their ability to excel in different areas of engineering and motorsport:

- Static Events:
  - Design analysis: evaluates engineering design processes, innovation, and optimization techniques.
  - Cost analysis: assesses the affordability of the car and the teams' ability to justify design and manufacturing choices.
  - Business plan presentation: challenges teams to present a comprehensive business case, including marketing strategies, financing plans, and sales projections.
- Dynamic Events:
  - Acceleration: measures the car's ability to accelerate in a straight line.
  - Skid-pad: assesses the car's lateral grip and maneuverability.
  - Autocross: evaluates the car's agility, responsiveness, and overall performance on a closed circuit.
  - Endurance: examines the car's durability, reliability, fuel efficiency, and driver endurance over a longer distance.

Formula Student competitions are governed by strict rules and regulations established by organizations such as FSG [5], FSUK [6], FSE [7], etc. These rules ensure

fair play and prioritize safety, defining technical specifications, safety standards, and performance limitations for participating teams. By adhering to these guidelines, Formula Student creates a level playing field where students can showcase their engineering abilities within defined constraints.

1.1.3 Formula Student Driverless

The research presented in this paper seamlessly integrates with the framework of the Formula Student Driverless competition, demonstrating a strong commitment to adhering to its rules and contextual requirements.

The Formula Student Driverless competition is an extension of the Formula Student competition, specifically focused on autonomous vehicles. It provides an exciting platform for student teams to showcase their skills in designing, building, and programming self-driving race cars.

The competition follows a format similar to the traditional Formula Student competition, featuring both static and dynamic events (see Subsection 1.1.2).

Figure 1.2: Formula Electric Belgium's driverless car [3]

The Formula Student Driverless competition offers students hands-on experience in autonomous vehicle development, fostering innovation and preparing them for careers in the field. It showcases advancements in perception systems, decision-making algorithms, and sensor integration while promoting collaboration and problem-solving. This competition propels the future of autonomous vehicles and inspires the next generation of engineers in this dynamic field.

The FEB team has recently embarked on an exciting endeavor by implementing their initial simple, non-optimal solution, specifically designed for participation in the driverless competition. This research significantly contributes to the team's unwavering pursuit of excellence in the field of autonomous vehicle development by establishing a solid foundation of a new optimized control algorithm.

1.1.4 Race track

In the Formula Student Driverless competition, the track is typically designed to simulate real-world racing conditions and challenges. The track layout is carefully planned to include a variety of elements such as straight sections, corners of different radii, chicanes, and slalom sections. These features test the performance and capabilities of the autonomous vehicles in terms of acceleration, braking, cornering, and overall handling.

The track design aims to provide a comprehensive and demanding environment that evaluates the autonomous systems' capabilities in navigation, path planning, and decision-making. It challenges the participating teams to optimize their algorithms and sensor fusion techniques to achieve efficient and precise performance.

It's important to note that the specific track design can vary from one Formula Student Driverless competition to another, as different organizing bodies or locations may have their own unique track configurations.

Some rules remain consistent across competitions in terms of track design. The track typically consists of four orange cones positioned at the start, with two cones on the left and two on the right. Following the orange cones, yellow cones are placed on the right side of the track, while blue cones are positioned on the left (see Figure 1.3). The car is required to commence its run at a distance of 6 meters from the center of the orange cones. The cones themselves are spaced apart by a maximum distance of 5 meters. Furthermore, the track always maintains a minimum width of 3 meters. These regulations ensure a standardized track layout for participants to navigate and optimize their autonomous vehicle's performance. These rules are also applied in the Formula Student Driverless Simulator (see Section 1.1.5).

Figure 1.3: Track design [4]

1.1.5 Formula Student Driverless Simulator

In the context of this work, the Driverless Simulator proved to be an invaluable tool. It provided a realistic virtual environment where comprehensive testing, optimization, and validation of the autonomous systems could be conducted. The simulator enabled the assessment of performance, fine-tuning of the algorithm, and gathering of valuable insights, all without the need for extensive physical testing. The accelerated development process and assurance of accuracy and reliability were facilitated by the simulator's capabilities.

The Formula Student Driverless Simulator [8] is a virtual platform that simulates the experience of racing and testing autonomous vehicles in a realistic, computer-generated environment. It is specifically designed for teams participating in the Formula Student Driverless competition to develop and fine-tune their autonomous driving algorithms before implementing them in a physical race car.

The simulator replicates real-world driving scenarios, enabling teams to experiment, evaluate performance, and refine their autonomous systems. It provides a safe environment to test algorithms, optimize sensor fusion, and enhance decision-making. Teams can simulate race events, analyze data, and virtually compete, fostering collaboration and knowledge sharing.

Figure 1.4: Formula Driverless Simulator interface

Using the Driverless Simulator offers advantages such as saving time and resources, facilitating rapid iteration of algorithms, and providing valuable training opportunities. It reduces the risks associated with testing on actual race tracks and enables efficient fine-tuning of autonomous systems. The simulator serves as a platform for teams to enhance their skills in autonomous vehicle development and racing.

1.2 Design choices and state of the art

1.2.1 Control algorithms for autonomous race cars

The control algorithm for autonomous race cars comprises two critical components: Path Planning (PP) and Path Following (PF).

The PP algorithm serves as the foundation for guiding the autonomous race car along its desired route. It is responsible for determining the reference trajectory based on factors such as track layout, obstacles, and performance objectives. In this work, a particular PP approach has been chosen, which will be elaborated upon in Section 1.2.2.

The PF algorithm aims to enable the autonomous race car to accurately adhere to the predetermined reference trajectory. This involves controlling the car's throttle, steering, and braking to ensure it stays on course, achieves high performance, and

maintains safety within the racing environment. Section 1.2.3 will explore the current state-of-the-art in PF algorithms.

1.2.2 Path Planning

The initial decision in designing the control algorithm involves the integration of the PP algorithm. Two choices are typically considered:

1. The PP algorithm calculates a path at the center of the track, while the PF algorithm computes the optimal car commands to maximize speed while staying within the track boundaries. This approach allows the car to deviate from the reference path to optimize its speed, known as taking the racing line.

2. The PP algorithm determines an optimal path based on predefined criteria, and the PF algorithm focuses on precisely following the reference path while maximizing speed.

In this work, considering the specific application and context (see Section 1.1), the first option is chosen, where the PP algorithm computes the path at the center of the track, and the PF algorithm optimizes the car commands for maximum speed within the track limits. This choice offers a simpler but efficient PP algorithm while ensuring an optimized PF strategy to maximize performance and safety.

1.2.3 Path Following

In the realm of autonomous race cars, PF algorithms play a crucial role in achieving precise control, optimal performance, and safety. These algorithms focus on controlling the car's movements to maximize speed along a planned trajectory. In this section, several notable PF approaches will be explored:

1. **Proportional-Integral-Derivative (PID) Controllers**: PID controllers are widely used in various control systems to regulate a process by adjusting control inputs based on proportional, integral, and derivative terms [9]. They offer a simple yet effective control strategy that ensures stability and responsiveness. In [10], it is concluded that a standalone PID controller provides a satisfactory solution for lateral vehicle control, but more advanced strategies or controllers may be preferable for improved performance. The longitudinal control task is addressed in [11] by implementing adaptive PID control using two different approaches: Genetic Algorithms (GA-PID) and Neural Networks (NN-PID).

2. **Stanley Control**: The Stanley controller combines proportional and feedforward control terms to adjust the vehicle's steering angle based on the lateral error from the reference path. By considering the cross-track error and path tracking gain, the Stanley controller achieves smooth and precise PF [12]. Stanley controller is one of the best for regulating lateral dynamics of a vehicle operating under nominal driving conditions, especially considering the computational complexity at which it offers such accuracy and robustness [10]. In [13], it is illustrated that Predictive Stanley lateral controller performance is

significantly higher than the basic Stanley controller at severe maneuvers and high vehicle speeds.

3. **Pure Pursuit Control**: Pure Pursuit control operates by calculating the curvature of the path ahead [14] and determining the steering angle [15] needed to align the vehicle with a look-ahead point on the path. By continuously updating the look-ahead point and adjusting the steering angle accordingly, the Pure Pursuit controller enables accurate tracking of the reference path and smooth navigation. Pure Pursuit controller is really good for regulating the lateral dynamics of a vehicle operating under nominal driving conditions [10]. In [16], a novel pure pursuit algorithm based on the optimized look-ahead distance named OLDPPA is proposed to improve the tracking accuracy of the Pure Pursuit algorithm.

4. **Linear Quadratic Regulator (LQR)**:

   Linear Quadratic Regulator (LQR) is a control approach that minimizes a quadratic cost function by adjusting control inputs based on the vehicle's current state [17]. Iterative Linear Quadratic Regulator (ILQR) is an iterative version of LQR used for finding optimal trajectories in non-linear systems. To handle constraints in trajectory optimization, the Constrained Iterative Linear Quadratic Regulator (CILQR) algorithm is proposed [18]. It incorporates constraints into the optimization framework, specifically for on-road driving motion planning problems. The integration of Alternating Direction Method of Multipliers (ADMM) with CILQR improves computation efficiency and enables real-time implementation [19].

5. **Machine Learning-Based Approaches**: Machine learning techniques have gained significant attention in developing control algorithms for autonomous race cars. Deep reinforcement learning algorithms, such as Deep Q-Networks (DQN) [20], Proximal Policy Optimization (PPO) [21], [22], [23], and Soft Actor-Critic (SAC) [23] can learn control policies through interactions with the racing environment. These algorithms enable autonomous electric race cars to adapt and optimize their racing strategies based on experience, achieving high-performance racing behavior. In [24], a reinforcement learning (RL)-based framework is suggested for application in autonomous driving systems to maintain a safe distance.

6. **Model Predictive Control (MPC)**: Model Predictive Control (MPC) is a widely adopted PF approach in autonomous racing. It utilizes a dynamic model of the vehicle to predict its future behavior and continuously optimizes control inputs over a defined time horizon. In [25], a custom MPC formulation for racing is derived based on the progress maximisation along the centerline. MPC incorporates track geometry, vehicle dynamics, and racing objectives to plan real-time trajectories and ensure precise trajectory tracking. By considering constraints such as track boundaries and vehicle capabilities, MPC enables the car to achieve maximum speed while maintaining control within the limits of the track. MPC can be regarded as the best control strategy for both simplistic as

well as rigorous driving behaviors [10], [26]. A game-theoric MPC approach for head-to-head autonomous racing and data-driven model identification method is proposed in [27].

While each algorithm brings unique advantages and has proven effective in various applications, the MPC approach stands out for its ability to incorporate a dynamic model, perform real-time optimization, and consider constraints. MPC provides precise trajectory tracking and maneuvering capabilities, maximizing speed while ensuring control within the track limits. Given the specific PP requirements and the application context, MPC is well-suited for the needs of this work.

1.2.4 Model Predictive Control

This section presents an in-depth exploration of the principles, methodologies, and advancements concerning MPC-based PF algorithms. Different MPC variants, optimization techniques, and integration with other control strategies will be examined, providing a comprehensive overview of the state-of-the-art in MPC-based PF. The focus will be on highlighting the contributions and benefits of MPC in achieving high-performance autonomous race car control, including improved speed, accuracy, and safety.

1. **Learning Model Predictive Control (LMPC)**: Learning Model Predictive Control (LMPC) is an advanced control approach that integrates model-based control with machine learning. By leveraging historical data or interactions with the environment, the control policy improves over time [30], [31]. Machine learning algorithms, such as neural networks or Gaussian processes, are employed to approximate system dynamics or cost functions within the MPC framework [30], [32]. This enables LMPC to handle complex and uncertain systems, adapting to changes and unreliable models. For example, in autonomous racing applications, LMPC has been successfully applied to the AMZ [28] Driverless race car gotthard [29], [30]. Another proposed approach suggests using multiple models to enhance robustness and adaptability [33].

2. **Tube Model Predictive Control (TMPC)**: Tube Model Predictive Control (TMPC) addresses model uncertainties and disturbances by considering a range of possible system behaviors within a "tube" around the reference trajectory [34]. Multiple predictions are generated and optimized to ensure that all trajectories within the tube meet operational constraints and reach the target setpoint. Compared to traditional MPC and LQR, TMPC offers improved performance and robustness [35], [36]. The tube width can be adjusted to balance performance and robustness, providing a flexible framework. TMPC enhances the ability of MPC to handle uncertainties, making it valuable for real-world systems with model uncertainties [37]. Additionally, a proposed algorithm combines robust SMC and TMPC [38].

3. **Nonlinear Model Predictive Control (NMPC)**: Nonlinear Model Predictive Control (NMPC) is a control technique for nonlinear systems. It accurately

models complex nonlinear dynamics and formulates the control problem as an optimization task [39], [34]. NMPC handles nonlinearities, constraints, and disturbances, resulting in improved control performance and robustness. Efficient optimization algorithms and numerical techniques are used to address the computational demands [40]. In the context of autonomous electric race cars, NMPC enables accurate modeling and precise control, leading to enhanced performance, stability, and safety [41], [42]. An efficient real-time implementation using a homotopy-based nonlinear interior-point method is presented in [43], achieving significant speedup compared to cold-started implementations.

4. **Linear Parameter Varying - Model Predictive Control (LPV-MPC)**: Linear Parameter Varying - Model Predictive Control (LPV-MPC) is a control approach that combines the concepts of Linear Parameter Varying (LPV) systems and MPC to achieve effective control of nonlinear systems with varying dynamics. LPV systems are a class of models that capture the time-varying behavior of a nonlinear system by describing it as a collection of linear models, where the parameters vary with certain scheduling variables, enabling a more accurate representation of system dynamics [46]. In LPV-MPC, the LPV system representation is utilized within the MPC framework [44], [45], allowing the control algorithm to adapt to changing dynamics and generate control actions that account for the varying parameters, thereby enhancing self-adaptability, robustness, and anti-interference ability under complex and severe working conditions, as demonstrated in [47]. Simulation experiments validate the superior performance of the trajectory tracking controller, exhibiting remarkable comprehensive performance in terms of trajectory tracking accuracy, real-time responsiveness, tracking stability, and driving safety compared to traditional MPC approaches, as highlighted in [44].

5. **Gaussian Process model predictive control (GPMPC)**: Gaussian Process model predictive control (GPMPC) extends MPC by incorporating Gaussian Process (GP) regression, enabling the capture of complex relationships, uncertainties, and nonlinearity in system dynamics. By leveraging GP regression, GPMPC improves control accuracy and adaptability by learning from data and refining predictions, addressing limitations of fixed nominal models and accounting for uncertainties and unmodeled dynamics in real-world systems. Modifications to the traditional MPC formulation are introduced to handle GP regression's stochastic nature, including accurate propagation of uncertainties and modified constraints and cost functions to reduce computational complexity [48], [49], [50].

6. **Model Predictive Contouring Control (MPCC)**: Model Predictive Contouring Control (MPCC) is an advanced control approach for autonomous driving, combining elements of MPC and contouring control. It considers the desired trajectory and road constraints. Using a predictive control framework, MPCC optimizes control inputs based on vehicle dynamics and road geometry to closely track the desired trajectory while maintaining an offset from the

reference path [51]. Contouring control ensures the vehicle maintains a specific distance from the reference path, beneficial in racing applications.

The MPCC algorithm formulates an optimization problem to minimize the error between the vehicle trajectory and the desired path, while adhering to various constraints like vehicle dynamics, safety limits, and road geometry [52]. By solving this optimization problem, MPCC generates control inputs that enable accurate trajectory tracking considering road constraints.

MPCC offers precise and efficient trajectory tracking, enhancing performance, handling, and safety [53]. By combining MPC and contouring control, MPCC enables optimal trajectory tracking in diverse driving conditions. In [54], MPCC is compared to Hierarchical Receding Horizon Control (HRHC).

In [55], authors describe a theoretical implementation of MPCC for an autonomous racing car. The system demonstrated satisfactory performance, but further testing and improvements are suggested. AMZ Driverless and ETH Zurich also developed a similar algorithm for their car gotthard [56].

The MPCC algorithm has emerged as the leading path following solution for the Formula Student Driverless context. Its superiority lies in its ability to optimize speed while accurately following a reference path, considering dynamic constraints, adapting to changing track conditions, without compromising safety and efficiency.

1.3 Contribution and overview

The objective of this project is to develop the most efficient control algorithm for a Formula Student race car, accompanied by a comprehensive analysis of the design, implementation, and validation processes. It provides a detailed explanation regarding the algorithm's implementation, model characterization, performance parameter determination, and expression of the reference path. While the MPCC approach is known to be effective for race cars, the focus is on understanding its functioning, behavior, and achieving optimal solutions.

The aim is to establish a solid foundation for an optimized control algorithm for Formula Electric race cars. Importantly, this project is conducted in collaboration with FEB. Considering the recent introduction of Formula Student Driverless, FEB has initially implemented a simple and inefficient solution. However, this work contributes by providing a comprehensive analysis and an efficient algorithm design that can serve as a fundamental framework for the development of Formula Student Driverless in Belgium.

The first section of this document centers on the design, implementation, and validation of the Path Planning (PP) algorithm. Following that, a preliminary version of the Path Following (PF) algorithm is developed, and its limitations are identified through performance analysis. Building upon this initial solution, an enhanced PF algorithm is designed and validated to achieve improved performance. Additionally, the code architecture is explained, providing insights into the underlying structure of

the algorithms. Finally, the overall performance of the control algorithm is presented, showcasing its effectiveness.

2 | Path Planning algorithm

This chapter provides a comprehensive explanation of the design, implementation, and validation of the PP algorithm. The chapter begins by formulating the problem, clearly defining the algorithm's objective. This involves understanding the available components and selecting an appropriate solution. Next, the various constituents of the algorithm are designed, and their implementation is thoroughly described. Each component is carefully considered and tailored to meet the desired objectives. Finally, a testbench is created to conduct an initial validation of the algorithm. This allows for early assessment and fine-tuning of the algorithm's performance before subjecting it to real-world conditions on the simulator. The validation process ensures that the algorithm functions effectively and reliably in practical scenarios.

2.1 Problem formulation

The objective of the PP algorithm is to find a reference path for the car to follow. To find the best solution to perform this task, it is essential to have a clear understanding of the components available.

2.1.1 Available components

As mentioned in Subsection 1.3, the track is delimited by cones as represented in Figure 2.1. The blue cones represent the left boundary and the yellow cones the right boundary. Additionally, four orange cones are placed between the other cones to represent the starting line.

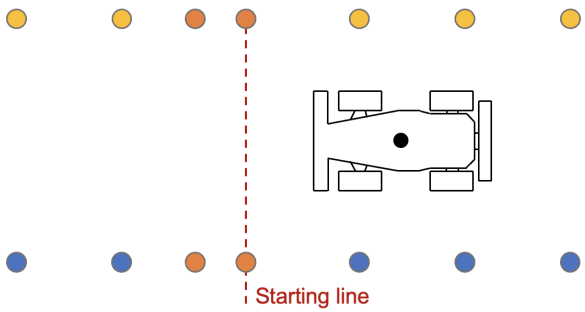

Figure 2.1: Track setup

The cones are placed in pairs, meaning that each blue cone is associated with a yellow cone on the other side of the track. The space between the cones on the same side of the track is maximum $5\,m$, while the space between the two cones inside a pair is minimum $3\,m$. At any moment, the width of the track is therefore at a minimum of $3\,m$ whether the car is moving in a

straight line or around a curve.

To exploit these track properties, a Simultaneous Localization And Mapping (SLAM) algorithm is already implemented on the car. The algorithm was implemented last year by a member of the FEB team. Although not optimal, it provides the materials needed to develop the PP algorithm.

The inertial frame is defined as represented in Figure 3.1, with the initial position $(x, y) = (0, 0)$ and orientation $\phi = 0$ the position and orientation of the car at start. The positions are expressed in two dimensions (there is no interest in the z position) and the orientation is defined positive in the counter-clockwise direction. The position of the car is taken at its center of mass.

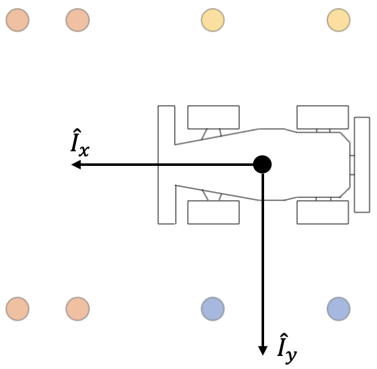

Figure 2.2: Inertial frame

The SLAM algorithm returns the absolute position of the car, as well as the absolute position and color of the cones detected in a range of $6\,m$ around the car. This limited range comes from the limitation of the sensors used to collect the data (a LiDAR for the position and a camera for the color). The informations about the detected cones are sent in two different lists, one with the cones detected in front of the car and the other with the ones behind the car, at a frequency f_{SLAM} varying between 25 and $40\,Hz$.

It is based on these informations and the data returned by the SLAM algorithm than the PP algorithm needs to be designed.

2.1.2 Chosen solution

As discussed in Subsection 1.2.2, it has been decided to implement a simpler but efficient PP algorithm to focus on the PF algorithm. To simplify the representation of the reference path, using the center of the track is the most straightforward approach. This choice ensures that the PF algorithm is consistently aware of a minimum of $1.5\,m$ available on each side of the path for the car to navigate.

To compute the path in the center of the track using the SLAM's output, there are two approaches: one can either rely only on the current output of the SLAM each time it is sent to build a path's piece within the SLAM's range, or keep the elements computed at each SLAM output in memory to gradually build the complete path. The latter option offers the advantage of generating the full path after one lap, allowing to eliminate dependence on the SLAM for subsequent laps. This approach mitigates the risk of error and also overcomes the range limitations of the SLAM in laps beyond the first. The only advantage of the first option is that it eliminates

the need to handle orange cones, as the lap number does not matter. This approach reduces the complexity, but at the cost of decreased efficiency and range in computing the reference path, as well as reduced robustness of the overall system. The second option is therefore chosen.

The final decision to be made concerns how the center of the track is determined. Given that the cones are placed in pairs, it is possible to locate a reference point for the center of the track by finding the midpoint between the two cones in a pair. The algorithm will rely on this straightforward approach to generate a series of reference points that will be transmitted to the PF algorithm.

To compute these points, the cones associated with a given pair need to be found. The most efficient way to achieve this task is by using Delaunay triangulation [57].

Delaunay triangulation is a method of dividing a set of discrete points into triangles in such a way that no point falls inside the circumcircle of any triangle. This means that the obtained triangles connect all the given points with no intersecting edges. For this application, giving a set of cones coordinates would result in the generation of triangles as illustrated in Figure 2.3. With this method, not only one edge crossing the track is generated per pair but two thanks to the triangulation formed with one of the next cones.

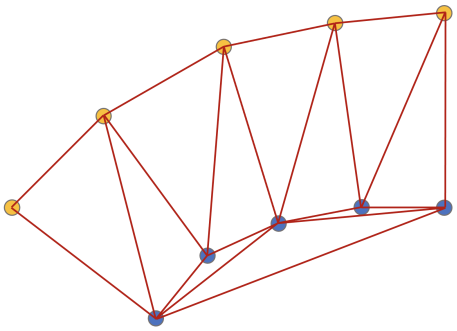

Figure 2.3: Example of Delaunay triangulation

Once the triangles are generated, the edges in which a reference point will be computed (therefore the edges that cross the track) need to be selected. The algorithm will thus consist of filtering the undesired triangles and ordering the consistent ones to generate the successive reference points. Taking the last example, apply this algorithm would result in keeping the triangles and generate the reference points as shown in Figure 2.4.

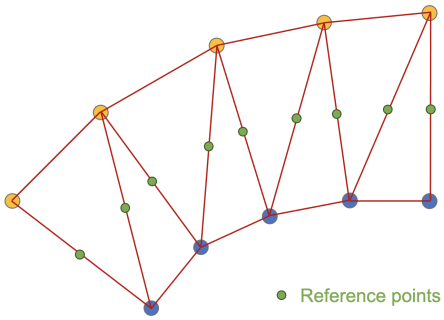

Figure 2.4: Example of triangle filtering and reference points computation wanted

2.2 Design

This section details the PP algorithm's design, which comprises multiple tasks that collectively constitute the overall solution.

2.2.1 Starting line operation

The choice of gradually building the complete path from the successive SLAM outputs leads to the need of a reference starting point from which the next points forming the path will be computed. For that purpose, the four oranges cones can be used since they represent the only landmark to rely on in order to define the beginning of the track.

Initially, the car is situated facing the starting line and is perpendicular to it, as shown in Figure 2.1. The distance between the front of the car and the starting line is $6\,m$, meaning that the SLAM algorithm cannot detect the four orange cones from the initial position due to the limited range. Therefore, it is necessary to move forward to be able to detect them. However, considering that the starting line is always in front of and perpendicular to the car at the start, it is safe to move forward in a straight line until the orange cones are detected.

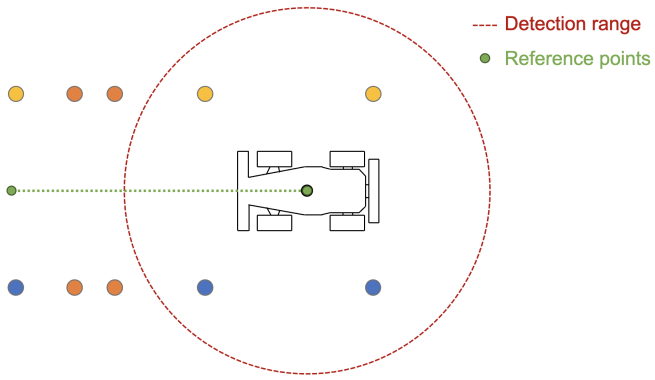

Figure 2.5: Starting procedure - initial position

At the beginning, before any reception of SLAM outputs, two reference points are initialized: the first one corresponds to the initial position of the car and the second one is placed in front of the car at a distance of $8\,m$ (see Figure 2.5).

Afterward, the car can proceed along the path established by the two reference points until it detects the four orange cones. During this procedure, all the other cones are ignored. And once the four orange cones are detected, the process to locate the first reference point can be initiated. Based on the position of the orange cones, the two triangles generated are ordered and the reference points computed by taking the midpoint of the edges crossing the track, as represented in Figure 2.6.

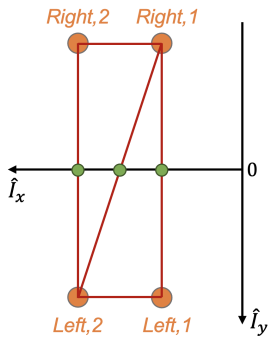

Figure 2.6: Disposition of the four orange cones detected

To build the reference points successively based on the previously constructed elements, a list of the ordered triangles of the track is being updated as well as the reference points computed. The two triangles computed with the orange cones are the basis for the future computed triangles.

For the reference points, an additional procedure is required. The aim is to setup the point on the starting line as first reference point. However, when the four orange cones are detected and the three reference points generated, the car is still positioned before the starting line, as illustrated in Figure 2.7. To handle this situation, the point at the initial position of the car is kept as first reference point until the car has crossed the starting line. This moment is determined by monitoring the car's x position. Once the x position of the car surpasses the x position of the reference point on the starting line, it indicates that the car has crossed the starting line. The point at the initial position of the car can therefore be removed of the list of reference points.

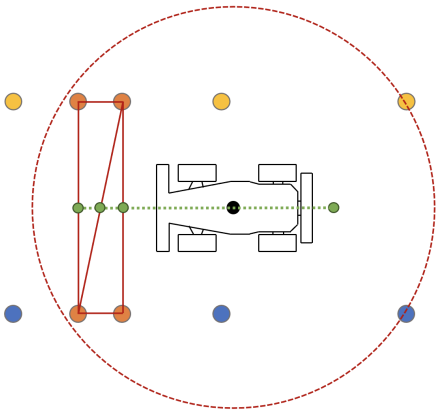

Figure 2.7: Position of the car and reference points kept when the four orange cones have been detected

2.2.2 Cones selection for triangles generation

Once the basis has been setup, the rest of the path can be build up as the car moves along the track and detects new normal cones (blue and yellow cones). The first task is to select the cones among the ones sent by the SLAM algorithm that will be used to generate the next Delaunay triangles and reference points.

For this purpose, the cones already used for previous triangles generations are kept in memory and compared to the ones received. Based on their position, the new cones are determined.

2.2.3 Triangles generation

If one or multiple new cones have been selected, new Delaunay triangle(s) can be generated. for that, not only are the newly selected cones used, but also the two last cones that were used in the previous triangle generation, as illustrated in Figure 2.8.

To determine which previous cones should be used for the next generation from all the cones kept in memory, the last previous triangle is utilized. The two cones of interest have different colors and one of them does not belong to any edge shared with a previous triangle.

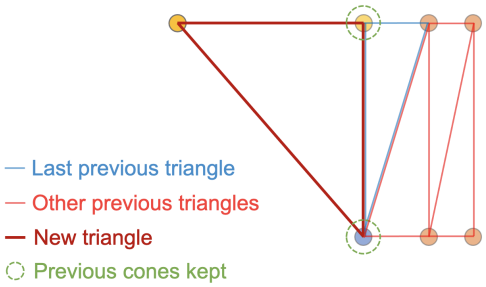

Figure 2.8: Example of triangle generation with one new cone

Special cases are handled when one or two cones of the last previous triangles are orange, as represented in Figure 2.9. In this case, the position within the four orange cones is checked instead of the color to determine on which side the cone is.

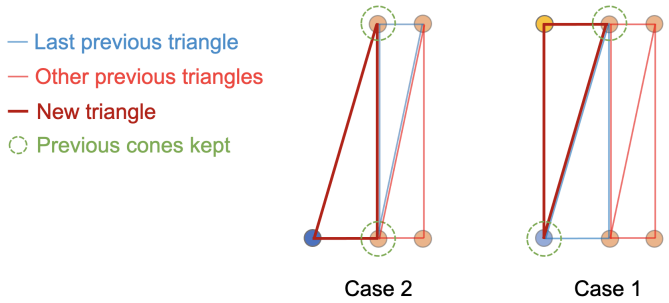

Figure 2.9: Special cases to handle for the selection of the two previous cones to keep for the next triangles generation

2.2.4 Triangles filtering

Now that the triangles have been generated, they need to be filtered to only keep the triangles within the track.

Figure 2.10 shows an earlier example where the triangles within and outside the track are put in evidence. It can be seen that the triangles outside the track (in green) are composed of three cones of the same color (blue in this case). This will always be the case, which makes the filtering straightforward: the triangles with three cones of the same color are removed.

The only special case that can occur is when orange cones are part of a triangle that must be discarded. In this case, the position of the orange cone is checked to determine if it is on the same side as the normal cones of the triangle.

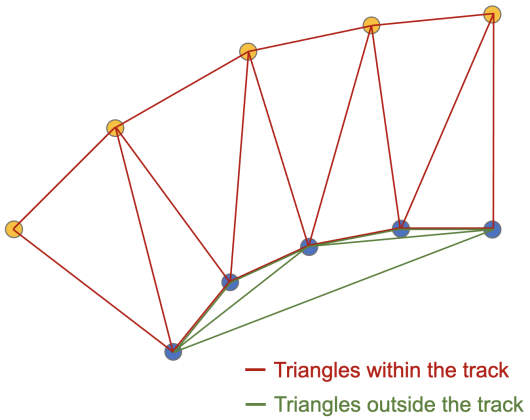

Figure 2.10: Example of generation of triangles within and outside the track

2.2.5 Triangles ordering

After filtering, only the triangles that are within the track are retained, but they are not arranged in any specific order. Consequently, they need to be sorted in order. This process is crucial since the computation of the reference points will rely on the triangles.

Figure 2.11 provides an example to illustrate the procedure. Two new triangles have been generated and must be ordered. The ordering is achieved solely by using the edges of the triangles. The first new triangle to follow the last previous triangle is the triangle that shares an edge with it, as is the case with triangle 1 in this example. Triangle 1 is subsequently added to the list of triangles and becomes the reference for the other new triangles. The next new triangle is again the one sharing an edge with the last selected, which in this case is triangle 2. This process is repeated until all the new triangles have been added. The last triangle added will become the last previous triangle for the next triangles generation, triangle 2 in this example.

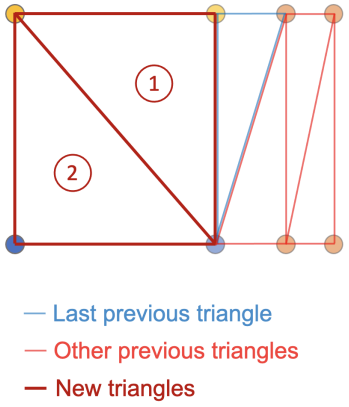

Figure 2.11: Example used to illustrate the triangles ordering procedure

2.2.6 Reference points computation

To compute the reference points, the ordered triangles are utilized. The process is applied once the starting line operation explained in Subsection 2.2.1 has been executed. A new reference point is thus computed each time a new triangle is added.

Each triangle has two edges crossing the track (edges constituted of cones of different colors), 1 and 2 in the example of Figure 2.12. The edge shared with the previous triangle has already been used to generate a reference point (1), therefore the new reference point is the midpoint of the other edge (2).

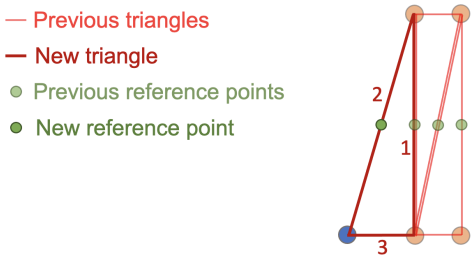

Figure 2.12: Example of a new reference point computation

2.2.7 Lap closure operation

The one remaining task of the algorithm is to handle the lap closure. The lap closure is the action of finishing a lap. As discussed in Subsection 2.1.2, it is an important part of the algorithm since finishing the first lap means that all the reference points have been computed. The SLAM algorithm is not needed anymore and the full track is available for the PF algorithm.

For the PP, lap closure occurs when the full track has been detected, which happens when the car detects an orange cone in front of it, even though it had already passed the four orange cones earlier. The situation is illustrated in Figure 2.13.

Once the first lap closure has occurred, the PP algorithm has finished is main task, which is computing the reference points of the full track. However, it is still used during the next laps to indicate to the PF algorithm when a lap will be closed.

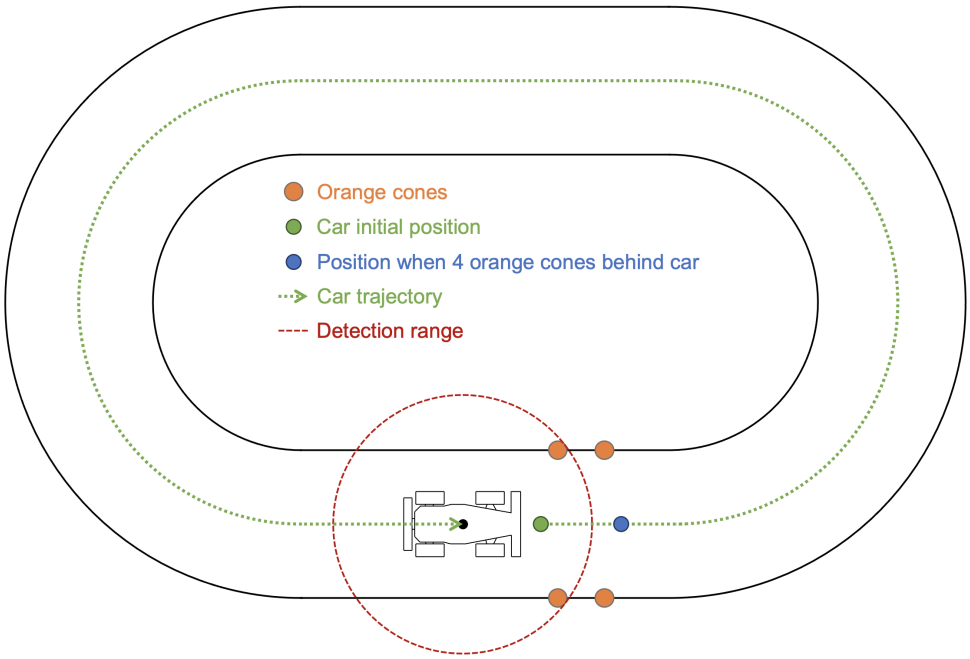

Figure 2.13: Illustration of the moment when the lap closure operation is executed

2.2.8 Summary

All the tasks executed by the algorithm have been explained. Put together, the reference points constituting the center of the track are computed based on the SLAM algorithm's outputs.

In order to launch the other tasks, the starting line operation must be executed and terminated. It defines the basis on which each new element will be computed, as it generates the first triangles and reference points of the track.

Then the other tasks can be executed. The cones selection determines which cone among the ones sent by the SLAM algorithm is a newly detected cones. With these cones and the two last previous ones, Delaunay triangles are generated. These triangles are filtered and ordered to keep only the triangles within the track and put them in order for the new reference points to be computed.

These actions are repeated each time the SLAM algorithm delivers an output until the first lap closure operation is executed. At this moment, all the cones of the track have been detected, and all the reference points have been computed. Every task besides the lap closure operation is therefore shut down. An example of a full track completed by the PP algorithm is represented in Figure 2.14.

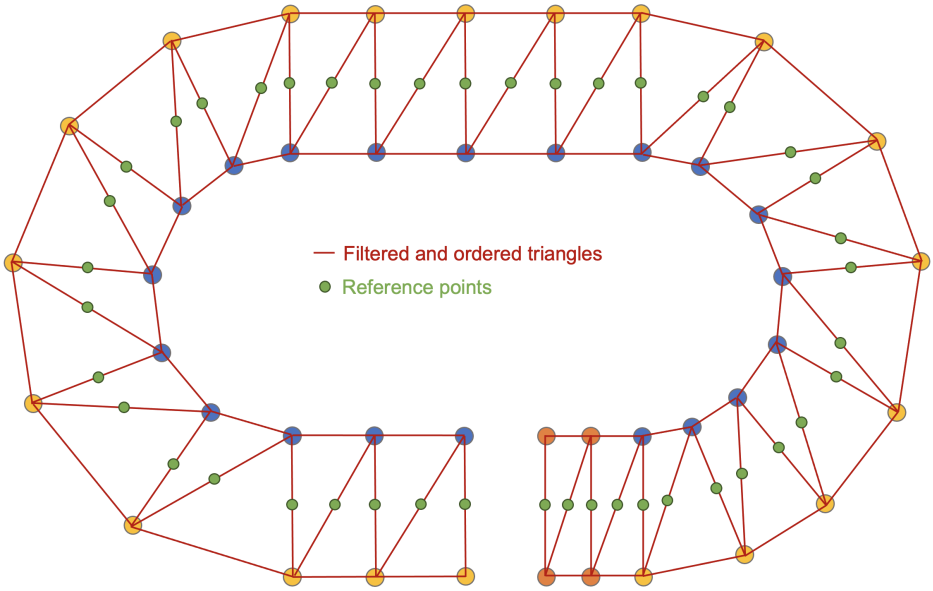

Figure 2.14: Example of a full track completed by the PP algorithm

2.3 Implementation

The algorithm is implemented in C++ which is the best language for a large project with a lot of dependencies that needs to run in real-time. The architecture of the full code and the links between the algorithms will be explained later in Chapter 5. This section focuses only on the implementation of the PP algorithm.

2.3.1 Creation of the necessary tools

To simplify the implementation of the algorithm, some tools in the form of structures in C++ are created. These tools will be used in the different tasks of the algorithm to improve the overall efficiency and clarity of the code.

- A first tool is created to define a `Location` with x and y positions as arguments. This is a basic tool that will facilitate the storage of positions.

- A `Cone` structure is then created to represent a cone with a certain `Location` and color. It also defines the comparison of two cones based on their position.

- To represent the four orange cones and define the methods useful for their utilization, an `OrangeCones` structure is created. It enables the storage of the orange `Cones` with their place on the map, as explained in subsection 2.2.1 and illustrated in Figure 2.6. Two methods are implemented: one to add an orange cone to the structure by determining its place among the four possible ones and another to determine whether or not a given cone is in the structure.

- An `Edge` structure is defined with the two `Cones` forming it as arguments. In addition, the information about its length, midpoint `Location` and whether or not it crosses the track are stored. It also defines the comparison operator for 2 edges by comparing their respective cones.

- All the necessary tools to define a `Triangle` have been created. In this structure, the three `Cone`s and three `Edge`s forming it are stored. A method determines whether or not a given edge is part of the triangle and another determines whether or not a given triangle shares an edge with the triangle.

- To represent the output of the SLAM algorithm using the created tools, a `ConeArray` structure is defined. It stores as arguments a list of `Cone`s in front of the car, as well as the position of the car.

- The last structure created is called `Polynomials` and represents the output of the PP algorithm. The computed reference points are stored in a list, as well as the car position and information about lap closure. It is called `Polynomials` because it contains the materials that will be used by the PF algorithm to create polynomials.

2.3.2 Path Planning class and methods

The algorithm is implemented as a class called `PathPlanning` containing attributes and methods to perform the different tasks. Using the previously defined tools makes the implementation of the class more efficient and clear, thereby enabling easier debugging.

Algorithm 1 PP main method

Require: *cone_array*, SLAM algorithm's output converted into `ConeArray` structure
  if lap closure operation not executed yet **then**
    cones selection for triangles generation ← `delaunay_cones_gen(`*cone_array*`)`
    if new cone(s) selected **then**
      generation of selected cone(s) coordinates for `delaunator` library ← `delaunay_cones_coord_gen()`
      triangles generation, filtering and ordering ← `filtered_ordered_triangles_gen()`
      if starting line operation finished **then**
        reference points computation ← `polynomials_gen()`
      end if
    end if
    if starting line crossed for the first time **then**
      removal of first reference point
    end if
  else
    cones selection to check lap closure ← `delaunay_cones_gen(`*cone_array*`)`
end if

The main method `coneArrayCallback()` is the method called to execute the full PP algorithm. It takes the output of the SLAM algorithm as argument and then uses the other methods to generate the reference points. The pseudo-code of the

method is represented in Algorithm 1.

The actions taken refer to the ones described in the design Section 2.2. The second method used, `delaunay_cones_coord_gen()`, transforms the list of selected cones into the appropriate format of list of cones coordinates for the delaunator library [58]. This is the library used to generate the Delaunay triangles from the coordinates of the selected cones.

2.4 Validation

In order to finalize the development of the PP algorithm, it is necessary to validate its design and implementation. This involves testing every possible scenario that the PP algorithm could encounter and confirming the accuracy of the results. The first step is to create an environment where the results obtained depend solely on the performance of the PP algorithm, and are not influenced by external factors. By using directly the Formula Student simulator to run the tests, the results could be influenced by the performance of the SLAM algorithm providing the data of the track to the PP. The Formula Student Driverless Simulator is also constrained by the available track configurations, which limits the scope of possible tests that can be conducted. To address these issues, it is necessary to construct a testbench that provides a completely secure environment for conducting any required tests. Once the algorithm's performance has been validated using the testbench, the next step is to verify its functionality in the target environment represented by the simulator. This step is essential for detecting any potential errors that may arise due to the non-ideal environment and finding solutions to address them. By validating the algorithm in the target environment, it is possible to ensure that it performs correctly and meets the required specifications under realistic conditions, rather than just in a "perfect world" scenario.

First, a brief description will be provided regarding the design and implementation of the testbench, while the detailed calculations will be provided in appendix. Next, the validation process will be described, which involves utilizing both the testbench and the simulator.

2.4.1 Validation with testbench

Testbench and validation process

The objective of the testbench is to provide an ideal environment to perform all the necessary tests on the PP algorithm. The problem formulation is used to build that environment, as well as the needs in terms of testing cases to define the different functions of the testbench. Since the results produced by the algorithm are based on the SLAM output, the latter must be replicated. In addition, the tests rely on the track configurations. It is therefore necessary to have a testbench allowing the simple creation and configuration of various tracks and replicating the SLAM output based on the built track. The design and implementation of the testbench, constituted of

the track builder and SLAM output replicator is described in Appendix A.

The validation with the testbench consists of verifying the functioning of each method and the final result in all the different cases than can happen. All the possible cases described in the design section are tested and the following data are analyzed to confirm the good functioning of each method,

- the new cones selected by the `delaunay_cones_gen(`*cone_array*`)` method;
- the triangles generated by the `delaunator` library and the triangles filtered and ordered by the `filtered_ordered_triangles_gen()` method;
- the reference points computed by the `polynomials_gen()` method;
- the cones added to the list of detected cones to verify that a cone detected but not used is not considered as a detected cone.

In addition, the algorithm is tested in different situations (large and sharp turns, full track) to confirm its overall functioning and detect potential particular situations. For more details about the conducted tests, a benchmark is presented in Appendix B.

The validation is straightforward because of the limited possible cases and the exactitude of the expected results. Inputs and outputs are precisely defined, thus validating the algorithm implies verifying the obtained results with the exact expected results. The algorithm design is not too complex and does not rely on parameters that require optimization or experimental testing, resulting in binary outcomes. The results are either good or bad, without any measurement of quality or intermediate nuance. The implementation is the only determinant, and the validation process mainly comprises debugging. The clarity and effectiveness of implementation facilitated this process. In terms of computational time, the algorithm takes approximately 60 μs on average to be executed, providing quasi instantaneously the reference points to the PF algorithm when data are received from the SLAM. However, when testing the overall algorithm performance in various tracks, an additional special case that requires specific actions has been revealed.

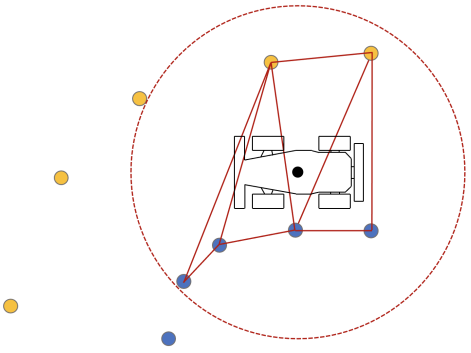

Figure 2.15: Example of the additional special case to handle

Additional special case to handle

The additional special case to handle can occur in a sharp curve, for instance, where the total number of blue cones detected can exceed the number of yellow cones by two (or vice versa). This circumstance is depicted in Figure 2.15, where four blue cones were

detected while only two yellow cones were
identified. Such an occurrence leads to the formation of a narrow triangle, which may result in a less accurate reference point and cause disruption in the generation of subsequent triangles.

Once all the new triangles have been added, the number of cones on each side of the track are compared. If the number of cones on one side exceeds by two the number of cones on the other side, triangles and thus cones have to be removed until this is not the case anymore.

Result for a complete track

An example of result of the PP algorithm for a complete track is shown in Figure 2.16. At the beginning of the track, the car is located at $(0, 0)$, and the orange cones are not initially detected. Subsequently, the car is moved forward along the track. The generation of triangles starts with the orange cones, and the initial reference point at $(0, 0)$ is removed when the car crosses the starting line. The normal cones located between the starting point and the orange cones are not added in the detected cones at the beginning, allowing for the generation of triangles with them at the end of the lap. All the cones have been used to compute as much as reference points possible, beginning from the starting line and ending just before it to form the complete track.

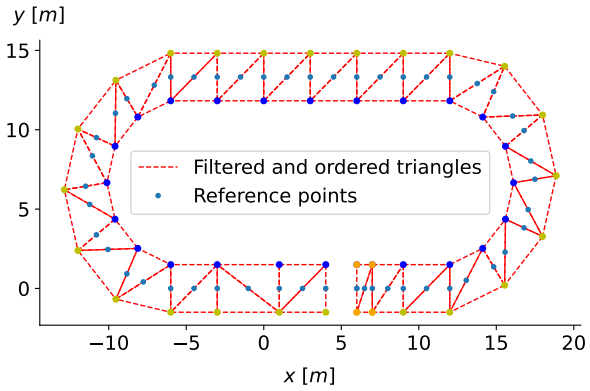

Figure 2.16: Result of the PP algorithm for a complete track

2.4.2 Validation on Formula Student Driverless Simulator

Now that the PP algorithm has been validated in an ideal environment, it can be tested in the target environment to confirm its functioning under real conditions. To this extend, the algorithm is tested on the Formula Student Driverless Simulator in manual mode. In manual mode, the car can be moved on the map with the arrows of the keyboard, allowing easy displacements free from constraints. The car can be moved step by step to observe particular cases or smoothly along the track to confirm its accuracy under normal functioning. The main method of the PP algorithm is executed each time the SLAM algorithm sends an output. Contrary to the testbench, the real SLAM algorithm relies on sensors to find the cones positions and is therefore subject to uncertainties. The objective of this validation on the simulator is therefore to ensure the global functioning of the PP algorithm when run along side with the

SLAM algorithm in the target environment.

Visualization tools are implemented and used to represent in real-time the cones detected by the SLAM algorithm, the car position, the filtered and ordered triangles and the reference points computed by the PP algorithm. The outputs send by the SLAM can thus be verified and the reactions of the PP to these outputs checked.

A minor issue is encountered when the SLAM algorithm fails to detect a cone. In such cases, the PP algorithm simply generates one less triangle, but this does not affect the computation of reference points or future results, as shown in Figure 2.17. The special case handling added to manage the addition of too much detected cones on one side compared to the other allows to avoid disrupting the triangle generation sequence.

A more significant issue arises when the SLAM algorithm returns a cone with an incorrect position. If the returned position is not consistent with the position of other cones, such as being in the middle of the track, then the triangles are still generated, but the reference points computed using those triangles may not be centered on the track and/or may not follow the track's direction. Figure 2.18 illustrates an example of such a situation.

The first issue has already been addressed by the designed PP algorithm, but the second issue could potentially cause the full control algorithm to fail, and therefore needs to be handled. For this purpose, some security mechanisms have been added to the algorithm.

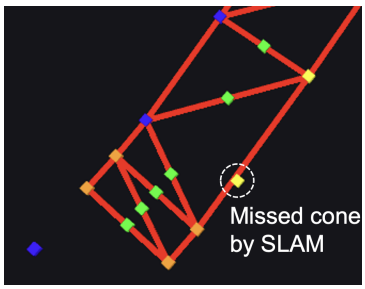

Figure 2.17: Illustration of the SLAM algorithm failing to detect a cone

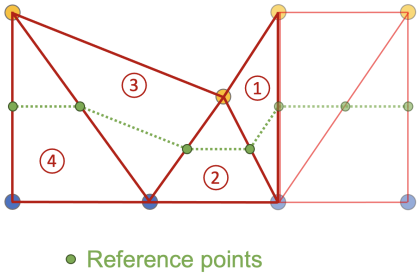

Figure 2.18: Illustration of the issue encountered when the SLAM algorithm returns a cone with an incorrect position

2.4.3 Security mechanisms

In this subsection, a cone sent by the SLAM algorithm in an incorrect position will be called a failed cone. A first security mechanism is added to ignore the failed cones positioned too close of an already detected cone. To achieve this, the distance range in which a cone is considered to be the same as another cone is extended. Therefore, when a failed cone is positioned too close to an already detected cone, it is not considered a new cone and is consequently not used by the algorithm.

The second security mechanism eliminates the failed cones that are too close from an already detected cone of another color. Indeed, the distance between two cones of opposite color is minimum $3\,m$. Therefore, before adding a new cone in the list of detected cones, the distance between the cone and the already detected cones is checked. If this distance is under $2\,m$ (a security margin is taken to not accidentally eliminate a good cone), the cone is added to the list of failed cones. This security mechanism would for example be able to eliminate the failed cone of Figure 2.18.

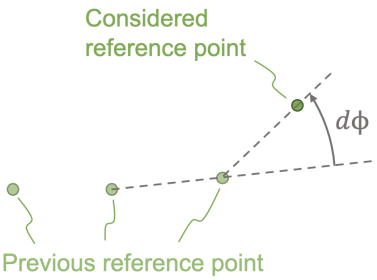

Figure 2.19: Definition of the angle between successive reference points

As a final security measure, if a failed cone has not been eliminated by the previous mechanisms, an additional step is added at the end of the process. This involves utilizing the computed reference points to determine if a failed cone was used to generate a triangle. To do this, a maximum angle $d\phi_{max}$ is defined between the successive reference points. The angle $d\phi$ is calculated as the difference in orientation between the line formed by the two previous reference points and the line formed by the current reference point and the one preceding it, as shown in Figure 2.19. If $|d\phi| > d\phi_{max}$, the change in orientation of the new reference point is considered too large to be an acceptable reference point. The cone that was used to generate the triangle and, therefore, the reference point, is added to the list of failed cones and the process is stopped.

These security mechanisms have been validated with the testbench and then tested on the simulator. The issues of the SLAM algorithm no longer prevent the proper functioning of the PP algorithm. This closes the design and implementation of the algorithm. A simulation showing the functioning of the PP algorithm in real-time on a full track of the simulator in manual mode is available here [60]. The cones detected by the SLAM algorithm are represented in their respective color, the filtered and ordered triangles and the reference points computed by the PP algorithm are represented in red and green respectively.

3 | First developed Path Following algorithm

The solution developed must find the optimal control commands in real-time in order to meet the objective of maximizing the car's speed along the path while staying within the track boundaries. The system to control, the car, is a physical system with a complex and dynamical behavior. In racing applications, the dynamic behavior of the vehicle is pushed to its limits in order to maximize progress along the track, and the control algorithm must be able to take into account the limits of the car's dynamics to ensure safe operations.

It is in this perspective that Model Predictive Control (MPC) has been chosen as basis for the developed control algorithm. MPC is an advanced control strategy that uses a model of the system to predict its future behavior and optimize a control signal over a finite time horizon. The control signal is computed by solving an optimization problem that takes into account the current state of the system, the model predictions, a performance criterion and possibly constraints. In the particular case of a vehicle progressing along a reference path, the MPC formulation can be extended to Model Predictive Contouring Control (MPCC). The contouring formulation symbolizes the fact that a contour (the reference path) must be followed.

During this work, an initial formulation of the PF algorithm was developed and tested. After a thorough analysis, limitations of the initial solution were identified, leading to the development of an improved solution in Chapter 4 based on the initial one. The final solution builds upon the initial solution, and the analysis conducted is valuable for understanding the final solution. Consequently, the design and validation of the initial solution will be presented in sufficient detail to trace the path that led to the final solution.

The problem will be formulated to clarify the algorithm's objective, inputs, and outputs. The subsequent section will delve into the MPC theory that forms the basis of the algorithm. Furthermore, the initial vehicle model employed will be described, along with its adaptation to this particular application, culminating in the model's characterization and validation. Finally, the design and implementation of the MPCC problem will be described, and the testing and validation of the PF algorithm elucidated, followed by an acknowledgment of its limitations, which spurred the development of the final solution.

Regarding the optimization problem, a solver was employed in order to focus on the design of the problem. The initial solver had limited performance but was utilized for the analysis of the initial solution. For the improved solution, a more effective solver was employed. The usage of the initial solver will not be extensively detailed in the main document, focusing instead on the utilization of the final solver in Chapter 4.

3.1 Problem formulation

3.1.1 Objective

The primary goal of the PF algorithm is to make the car follow the given reference path. The execution of PF may involve adhering to certain constraints, and the specific manner in which the path is followed depends on the desired performance objectives. In the case of a race car evolving on a track, the algorithm must maximize the car's speed along the path while staying within the track boundaries. This creates a trade-off between productivity (maximizing speed) and accuracy (maintaining proximity to the reference path while remaining within the track boundaries).

3.1.2 Inputs

The reference path is sent by the PP algorithm in the form of a list of reference points (x, y positions) at the center of the track. During the first lap, the farthest reference point in the track direction has a maximum lead of 6 meters ahead of the car's current position. For subsequent laps, the reference path covers the entire track. The width of the track is minimum $3\,m$, so there is minimum $1.5\,m$ between the reference path and the left and right boundaries.

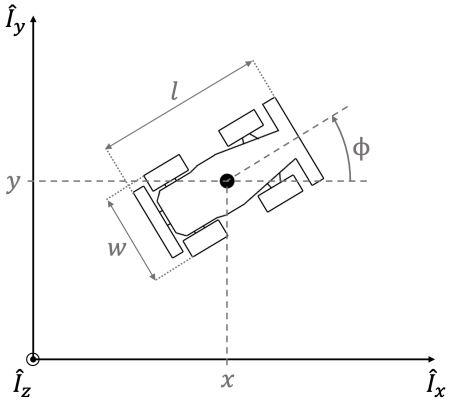

Figure 3.1: Inertial frame, car position, orientation and dimensions

The car is equipped of sensors to retrieve informations about the state of the car. Some of them were already used by the SLAM algorithm to compute the position of the car. The information is sent to the PP algorithm, which itself sends it to the PF algorithm at a frequency $25 \leq f_{SLAM} \leq 40\,Hz$.

In addition, odometry information (aquired by Ground Speed Sensor (GSS) and Inertial Measurement Unit (IMU) sensor) provides the linear velocities $\boldsymbol{V} = (V_x, V_y, V_z)$, orientation ϕ and the angular velocity around the z axis ω of the car in the inertial frame at frequency $100 \leq f_{odo} \leq 200$. The inertial frame, as well as the car position (x, y), orientation ϕ and dimensions (length $l = 2.261\,m$ and width

$w = 1.130\,m$), are represented in Figure 3.1. As stated in the previous chapter, the inertial frame is defined on the starting position and orientation of the car, which are taken at its center of mass. Finally, the mass of the car is $m = 255\,kg$ and its maximum velocity $\sim 27\,m/s$.

3.1.3 Outputs

In order to control the car, the PF algorithm must compute three control commands: throttle, brake, and steering. The throttle command corresponds to the acceleration of the car (throttle pedal) and is expressed as a value between 0 and 1, where 0 indicates no acceleration and 1 indicates full throttle. The brake command is similar, with a value between 0 and 1, where 0 indicates no braking and 1 indicates full braking. The steering command represents the angle of the car's front wheels and is used to change its direction. It is expressed as a value between -1 and 1, where -1 corresponds to a full steering angle to the left, 1 corresponds to a full steering angle to the right, and 0 corresponds to no steering. The maximum steering angle is 25°.

3.2 MPC theory

The idea behind MPC is to optimize a cost function J over a finite time horizon T, subject to constraints, in order to determine the optimal control command $\boldsymbol{u}$ to apply to the system being controlled. The optimization problem is solved iteratively over the finite time horizon T, and at each time step Δt, the first control action of the resulting sequence is applied to the system. Then, the optimization problem is re-solved using the current state measurements, and the process is repeated.

MPC consists of several steps that form the overall MPC problem formulation:

1. System modeling:

   MPC requires a mathematical model of the system being controlled. The model describes how the system responds to control inputs and external disturbances. The model can be a linear or nonlinear representation of the system.

2. Cost function definition:

   MPC requires a cost function that describes the performance criterion to be optimized. The cost function includes terms that represent the system's desired performance and constraints on the system's behavior, such as staying within the track boundaries.

3. Prediction:

   MPC predicts the system's future behavior over a finite time horizon. The prediction is based on the current state of the system and the model of the system.

4. Optimization:

MPC optimizes the control sequence over the finite time horizon subject to constraints. The optimization problem is solved using mathematical techniques such as gradient descent or quadratic programming.

5. Control commands:

   The first control commands of the optimized sequence is applied to the system. The process is repeated at each time step.

In a classic formulation, the measured system's output is the state of the system, denoted by $\boldsymbol{x} \in \mathbb{R}^{n_x}$, and the system's input are the control command, denoted by $\boldsymbol{u} \in \mathbb{R}^{n_u}$. The control command is found by the MPC based on the state of the system and a set of parameters $\boldsymbol{p} \in \mathbb{R}^{n_p}$ at current time t. This set of parameters could be for example the parameters to define the reference path that must be followed by the system, or the weights of the cost function. The state of the system is measured and the parameters are computed at every time step Δt to be sent to the MPC in order to compute the new control commands. In a real-time application, the MPC must therefore compute the control commands within the time Δt.

Inside the MPC, the cost function J is optimized over the finite time horizon T at each time step Δt. T represents the time over which the optimal control command sequence $\boldsymbol{u_{seq}}$ is computed. The MPC uses its own time step Δt_{MPC} to discretize the problem and obtain the sequence of $N = \frac{T}{\Delta t_{MPC}}$ control commands, denoted as $\boldsymbol{u_{seq}} \in \mathbb{R}^{n_u \times N}$, over the finite time horizon. N is called the prediction horizon, with $k = 1, 2, ..., N$ the current prediction. The model of the system is discretized and used to express the next predicted state $\boldsymbol{x}_{k+1}$ as a function of the current predicted state $\boldsymbol{x}_k$ and control command $\boldsymbol{u}_k$. The real state of the system received as input is used as initial state $\boldsymbol{x}_1 = \boldsymbol{x}_{init}$ for the predictions. An example of MPC scheme to represent the described quantities and signals is shown in Figure 3.2.

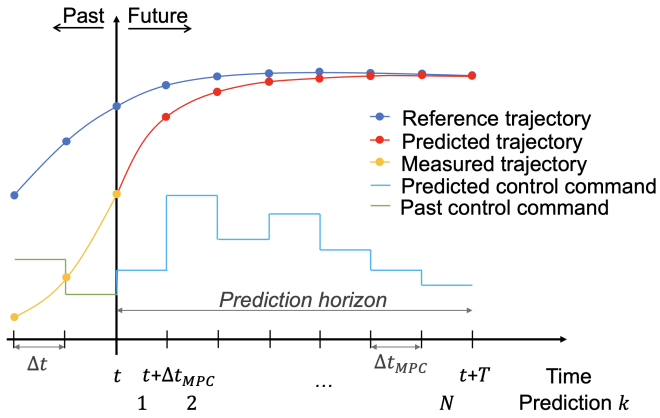

Figure 3.2: Example of MPC scheme

For the optimization problem, the cost function J to minimize is expressed as a sum of cost functions $J_k : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_p} \to \mathbb{R}$ over the prediction horizon,

$$J = \sum_{k=1}^{N} J_k(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{p}_k),$$

and depends on the state, control command and parameters at each prediction. The

update of the system's state with the model is expressed as equality constraints,

$$\boldsymbol{x}_{k+1} = f_k(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{p}_k),$$

with $f_k : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_p} \to \mathbb{R}^{n_x}$. The optimization can also be subject to inequality constraints, possibly non-convex and nonlinear, applied at each prediction,

$$\underline{h}_k \leq h_k(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{p}_k) \leq \overline{h}_k,$$

with the function $h_k : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_p} \to \mathbb{R}^{n_h}$.

The optimization problem can therefore be expressed, in its basic form, as:

$$\begin{aligned} \underset{\boldsymbol{u} \in \mathbb{R}^{n_u}}{\text{minimize}} \quad & \sum_{k=1}^{N} J_k(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{p}_k) \\ \text{subject to} \quad & \boldsymbol{x}_1 = \boldsymbol{x}_{init} \\ & \boldsymbol{x}_{k+1} = f_k(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{p}_k) \\ & \underline{h}_k \leq h_k(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{p}_k) \leq \overline{h}_k \end{aligned}$$

When the optimization is finished, the N optimal control commands $\boldsymbol{u}_k$ have been found and form the control command sequence $\boldsymbol{u}_{\boldsymbol{seq}}$. The control command applied to the system is the first control command of the sequence: $\boldsymbol{u} = \boldsymbol{u}_1$.

3.3 Vehicle kinematic model

In order to predict the motion of the car during the optimization process, a mathematical model that describes the behavior of the car when subject to control commands must be developed. A racing car is a complex dynamic system with a highly nonlinear behavior, especially when pushed to its limits. The more accurate the model will be, the better the prediction of its motion will be closed to the real motion of the car, allowing to find better optimized control commands for better performance and safe operations. However, increasing the complexity of the model also means increasing the computation time of the optimization. In a real-time application, the computation time is limited by the frequency at which the control commands are sent to the car. A higher sending frequency can improve the control and reactivity of the car. A trade-off must therefore be found between a reduced complexity to match the time constraints and a high enough accuracy to be able to describe the system real behavior.

For this first developed solution, the most basic model will be used and is called the kinematic bicycle model [59]. The objective is to evaluate the performance of the algorithm with a basic model in order to point its limitations and develop an improved solution. For the design, only the important characteristics of the basis of the model and the assumptions made to obtain the model will be described. The details of the calculations will be given in Appendix C. The model is then adapted to cater to the specific requirements of this application by developing a drivetrain model. Finally, the model will be characterized and validated.

3.3.1 Design

The car can be represented by a 4-wheels model with a steering angle for each front wheel, δ_r and δ_l, as shown in black in Figure 3.3. This model is first simplified by combining the two front wheels and two rear wheels to obtain a single front wheel with one steering angle δ and single rear wheel, as shown in red in the figure. This is why the model is called a bicycle model.

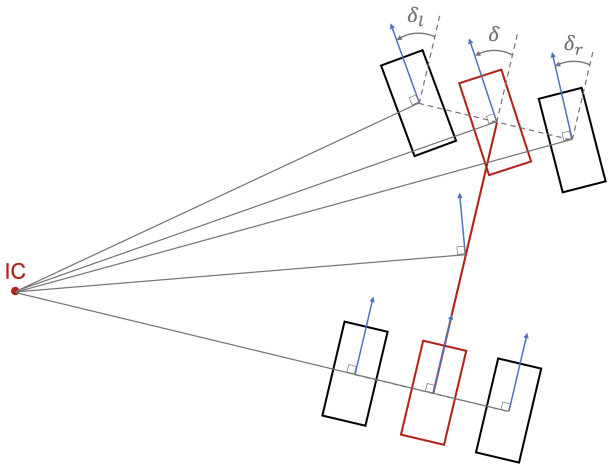

Figure 3.3: Instantaneous center of rotation (IC) comparison of the 2-wheels and 4-wheels models

This simplification is built on the concept of instantaneous center of rotation (IC). If the vehicle is frozen at a point in time, a line perpendicular to the velocity vector can be drawn at any point on the vehicle, as represented in Figure 3.3. The velocity vectors at the rear wheel, front wheel and center of mass have been drawn. If no slip is considered for the wheels, meaning that the velocity of the wheel acts in the same direction that the wheel is facing in, all the lines meet at the same point called the IC. At that instant in time, it is the point around which the vehicle is rotating. by combining the front and read wheels into one, the vehicle still rotates around the same point.

Then, to keep the model basic, some important assumptions are made:

- The vehicle is operating in a 2D plane;
- The vehicle has a lumped mass all acting at the center of mass.

The first assumption implies that the vehicle cannot roll over, and thus eliminates the need to model the vehicle's third rotational degree of freedom. This significantly reduces the complexity of the model, making it more computationally efficient.

However, this assumption neglects the effects of lateral weight transfer, which occurs when the vehicle undergoes cornering or other maneuvers. In reality, the vehicle's weight is distributed across its tires, and the weight distribution changes as the vehicle moves. This can cause changes in the vehicle's handling, particularly in high-speed or high-performance driving scenarios.

Additionally, the 2D assumption assumes that the vehicle's suspension system is perfectly vertical and cannot move laterally. In reality, the suspension system allows for lateral movement, which can affect the vehicle's handling and stability.

The second assumption implies that the vehicle is assumed to be a rigid body, meaning that there is no deformation of the body or suspension during motion. Also, all of the mass is assumed to be concentrated at a single point, which is the center of mass, and the center of mass is assumed to be fixed in space, meaning that it does not move during motion.

Again, this assumption simplifies the modeling process, but can also lead to inaccuracies in the model. For example, in reality, the mass of the vehicle is distributed throughout the body and is not concentrated at a single point. Additionally, the suspension and tires can deform during motion, which affects the handling and stability of the vehicle. Finally, the assumption that the center of mass is fixed in space may not hold true in all situations, such as during acceleration or braking.

These assumptions can have an important impact on the modeling accuracy in certain cases, but are essential to develop a model capable of being used in real-time applications by an optimization algorithm. Additionally, a third assumption is made concerning the slip of the tires on the road. No slips are assumed, meaning that the direction of the front and rear wheels velocities are aligned with the orientation of their respective wheel. This assumption has not a big impact when operating at low speed, when the tires keep a good grip on the road, but can lead to important differences in the car's behavior at higher speeds.

The model's goal is to generate a state-based model, meaning that it must generate a set of equations that can fully describe the model at any point in time. The model with the different quantities of interest are represented in the inertial frame in Figure 3.4. The position (x, y) of the car is determined by its center of mass (black circle). The car's orientation is denoted as ϕ and the angle between the car's orientation and its velocity vector v is the slip angle β. The steering angle of the front wheel is δ, with the front wheel located at a distance l_F from the center of mass and the rear wheel at a distance l_R. The model is a kinematic model, meaning that it is a simplified representation of the vehicle's motion that considers only the position and orientation of the vehicle. It assumes that the vehicle moves without any forces or torques acting on it and focuses on the relationship between the vehicle's position and its control inputs.

The state of the system is described by $\boldsymbol{x} = [x\ y\ \phi\ v]^T$ and the state equations are derived in Appendix C to obtain,

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\phi} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} v\cos(\phi+\beta) \\ v\sin(\phi+\beta) \\ \frac{v}{l_R}\sin(\beta) \\ a \end{bmatrix}, \tag{3.1}$$

with a the acceleration of the car, β the slip angle,

$$\beta = \arctan\left(\frac{l_R}{l_R + l_F}\tan(\delta)\right),$$

and δ the steering angle used as control input of the system.

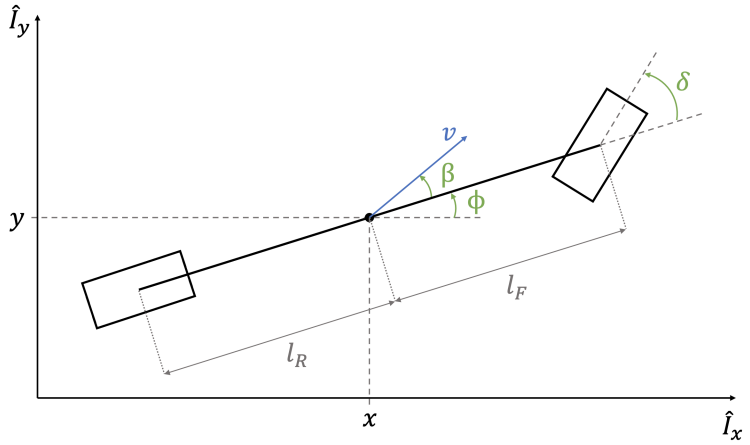

Figure 3.4: kinematic bicycle model representation

The rate of change of the car's position and orientation can now be computed by the state of the car $\boldsymbol{x} = [x\ y\ \phi\ v]^T$, its known dimensions l_R and l_F and its control input δ. To adapt the model to our application, throttle and brake commands must appear in the model. These are two different commands but both act on the acceleration of the car. The throttle controls the amount of electric power supplied to the motor, which in turn determines the torque and power output of the engine, and hence the acceleration of the car. Increasing the throttle results in more electric power being supplied to the engine, which leads to an increase in the torque and power output, and consequently, the acceleration of the car. On the other hand, the brake acts to reduce the acceleration of the car by applying a resisting force to the wheels. Therefore, the velocity's rate of change, which is the acceleration a of the car,

$$\dot{v} = a,$$

depends on the throttle and brake applied to the car. In order to express the acceleration in terms of throttle and brake, a drivetrain model is developed. The drivetrain model represents the relationship between throttle and brake inputs and the resulting acceleration behavior of the vehicle. This can be expressed as a force applied to the car. For this kinematic bicycle model formulation, this force F is defined as the force acting at the center of mass of the car in the same direction as its velocity to put the car of mass m in motion. The Newton's second law is then used to express the acceleration in terms of this force,

$$a = \frac{F}{m}$$

The force F does not only depend on the throttle or brake applied, losses must also be taken into account. Three different force components will be considered:

1. Traction (motor) or brake force:

   First of all, throttle and brake are two distinct control commands but they will never be used simultaneously as it would result in energy loss. Instead, they can be combined into a single control command denoted as D (driver command).

Throttle and brake inputs, defined between 0 and 1, control the force applied to the car, with positive force from throttle and negative force from brake. When neither is used, no force is applied. The combined input, D, ranges from -1 to 1: if $D \geq 0$, throttle command equals D and brake command equals 0; if $D < 0$, brake command equals $-D$ and throttle command equals 0.

With this new control command definition, the component of the force applied by the motor or brake can be defined as a function depending only on D, denoted as $f_m(D)$. This function must satisfy the following constraints to be valid:

$$\begin{cases} f_m(D) & > 0 \quad \text{if } D > 0 \\ f_m(0) & = 0 \\ f_m(D) & < 0 \quad \text{if } D < 0 \end{cases} \tag{3.2}$$

2. Rolling resistance:

   Rolling resistance is the force that opposes the motion of a vehicle's wheels when they roll on a surface. It is caused by the deformation of the tire and the surface it is rolling on. The major contribution to rolling resistance comes from the deformation of the tire, which is mostly influenced by the weight of the vehicle thus its mass. The mass of the car being fixed, the rolling resistance force can be defined as a constant to simplify the model. This constant will be denoted as C_r.

3. Drag:

   In the context of vehicle dynamics, drag is the resistance force that opposes the motion of the vehicle due to its interaction with the surrounding air. The drag force can be affected by various factors such as the air density or the vehicle's orientation and speed relative to the wind, but it mostly depends on the speed of the car. The drag force increases with speed and is null when the speed is null. A simplified model for the drag force can be defined as a function only depending on speed v, denoted as $f_d(v)$. This function must satisfy the following constraints to be valid:

$$\begin{cases} f_d(v) & > 0 \quad \text{if } v \neq 0 \\ f_d(0) & = 0 \end{cases} \tag{3.3}$$

The force F can therefore be expressed as,

$$F = f_m(D) - C_r - f_d(v), \tag{3.4}$$

which concludes the derivation of the vehicle model. The functions f_m and f_d and the constant C_r must be found experimentally.

The final formulation of the kinematic bicycle model is shown below,

$$\underbrace{\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\phi}\\ \dot{v}\end{bmatrix}}_{\dot{\boldsymbol{x}}} = \underbrace{\begin{bmatrix} v\cos(\phi+\beta) \\ v\sin(\phi+\beta) \\ \frac{v}{l_R}\sin(\beta) \\ \frac{1}{m}\left(f_m(D) - C_r - f_d(v)\right)\end{bmatrix}}_{f_{kin}(\boldsymbol{x},\boldsymbol{u})}, \tag{3.5}$$

with,

$$\beta = \arctan\left(\frac{l_R}{l_R + l_F}\tan(\delta)\right), \tag{3.6}$$

and $\boldsymbol{u} = [\delta\ D]^T$ the control commands.

3.3.2 Characterization and validation

The vehicle model, especially the drivetrain model, must be characterized in order to have a complete model. Then it must be validated to verify the behavior of the car model compared to the real car.

Drivetrain model characterization

To find the functions and the constant C_r of the drivetrain model,

$$a = \frac{F}{m} = \frac{1}{m}(f_m(D) - C_r - f_d(v)), \tag{3.7}$$

some experiments must be performed on the simulator. Since its the acceleration that must be characterized, the experiments will be performed in straight line, with the steering command kept at 0.

$f_d(v)$ and C_r can be decoupled from $f_m(D)$ by setting up $D = 0$ to obtain $f_m(0) = 0$. Additionally, the drag, modeled by $f_d(v)$, increases with the speed v. the rolling resistance, on the other hand, stays constant and is probably small. At high speed, the term $f_d(v)$ can therefore be considered dominant compared to C_r. But at low speed, C_r might have a certain impact because $f_d(v)$ will be small.

The two first experiments will aim at determining $f_d(v)$ and C_r respectively, while the term $f_m(D)$ will be determined by the third and fourth experiment (third for the throttle and fourth for the brake). For the four experiments, the following data will be recorded:

- Time t;
- Speed v;
- Acceleration a. The acceleration is computed based on the two last speed measures v, v_{prev} and the time that elapsed between the two measures,

$$a = \frac{v - v_{prev}}{t - t_{prev}};$$

- Throttle/brake command D.

- **Experiment 1:** Determination of $f_d(v)$

To reach high speed, full throttle ($D = 1$) is applied to the car at the start. When $v = 15\,m/s$, no more throttle ($D = 0$) is applied and the data recording is stopped when the car is almost at rest. During the time where $D = 0$, equation (3.7) becomes,

$$a = -\frac{1}{m}(C_r + f_d(v)) \tag{3.8}$$

where $C_r << f_d(v)$ can be considered when the speed is not too low. a can therefore be expressed in terms of v to find the function f_d,

$$a = \frac{-f_d(v)}{m} \tag{3.9}$$

The recorded data are shown in Figure 3.5. As expected, when $D = 0$, the deceleration decreases with time since the speed decreases, making the $f_d(v)$ term increasingly smaller. To find the relation between a and v, only the relevant data for $t \in [3, 18]s$ are kept, as shown in Figure 3.6. A Savitzky-Golay filter is used on the acceleration to smooth the data and have a better representation of a. Then, in Figure 3.7, a is expressed in terms of v. Their relation is approximately linear in the whole range of speed considered. A polynomial fitting of order 1 is therefore used to find the expression of the line best fitting the data, resulting in,

$$a = -0.29023v + 0.00304$$

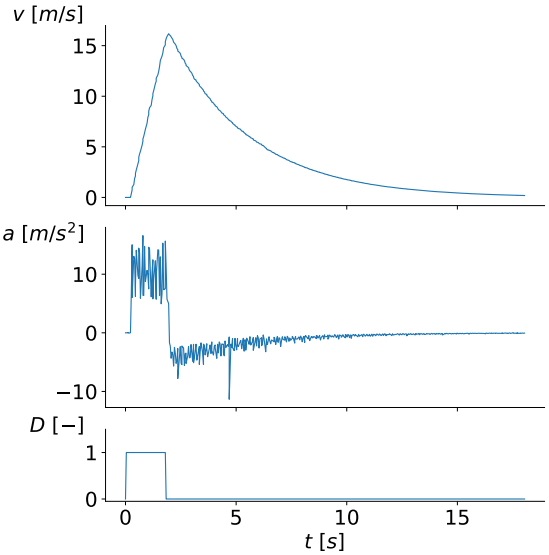

Figure 3.5: Raw data of experiment 1

By considering $0.00304 \approx 0$, this equation can be identified to equation (3.9) to find the expression of $f_d(v)$,

$$f_d(v) = 0.29023m\,v = \underbrace{74.01}_{C_d}\,v \tag{3.10}$$

with $m = 255\,kg$ the mass of the car. By suppressing the independent term 0.00304, the condition $f_d(0) = 0$ is respected.

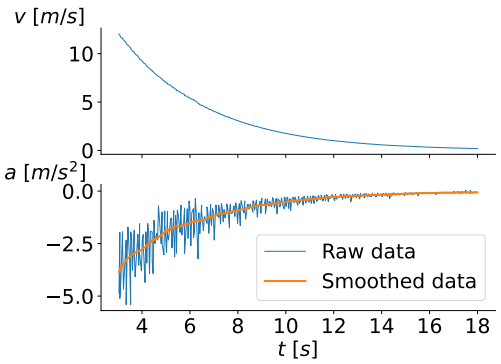

Figure 3.6: Relevant data kept for experiment 1

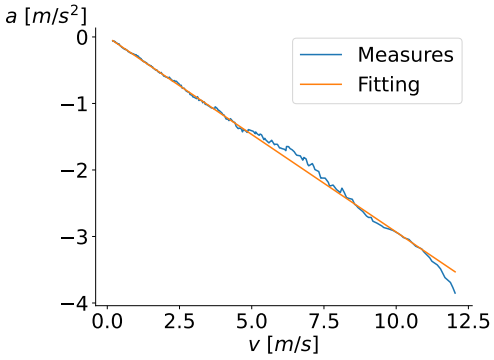

Figure 3.7: Linear relation between acceleration a and speed v in experiment 1

- **Experiment 2:** Determination of C_r

To reach low speed, small throttle ($D = 0.2$) is applied to the car at the start. When $v = 2\,m/s$, no more throttle ($D = 0$) is applied and the data recording is stopped when the car is at rest. During the time where $D = 0$, the obtained equation is the same as for the first experiment (3.8) but this time C_r is not neglected. Knowing $f_d(v)$, the impact of C_r can be analyzed to determine it.

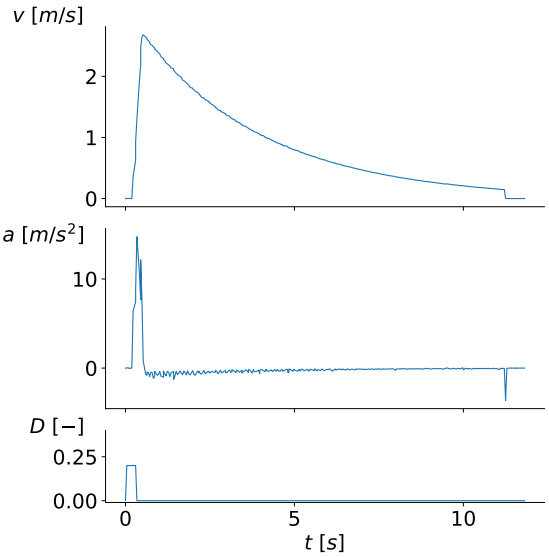

Figure 3.8: Raw data of experiment 2

The raw data recorded during the experiment are shown in Figure 3.8. The speed and acceleration have the same behavior as for experiment 1 but with smaller amplitude. The relevant data are kept for $t \in [0.5, 11]s$ and the acceleration is again smoothed, as shown in Figure 3.9. By plotting a as a function of v in Figure 3.10, It can be observed that the relation is again linear. A polynomial fitting of order 1 results in the expression,

$$a = -0.28453v - 0.00078$$

This expression is very close to the one obtain in experiment 1, and the independent term is again almost null. It can be concluded that there is no impact from rolling resistance even at very low speed, meaning that,

$$C_r = 0 \tag{3.11}$$

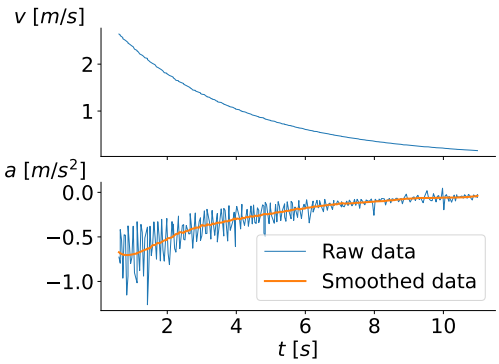

Figure 3.9: Relevant data kept for experiment 2

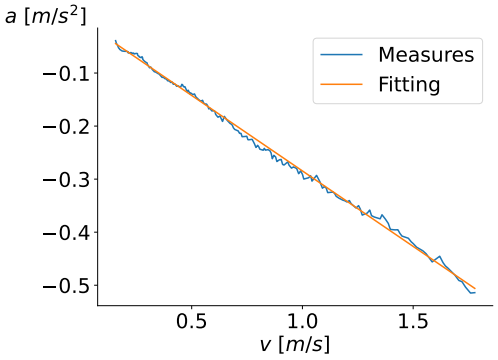

Figure 3.10: Linear relation between acceleration a and speed v in experiment 2

- **Experiment 3:** Determination of $f_m(D)$ for $D \geq 0$ (throttle)

Set a constant D during the whole time of the experiment, and repeat it for different values of D in its positive range. Typically,

$$D = [0.025,\ 0.05,\ 0.075,\ 0.1,\ 0.2,\ 0.3,\ 0.4,\ 0.5,\ 0.6,\ 0.7,\ 0.8,\ 0.9,\ 1.0]$$

$f_d(v)$ and C_r being known, $f_m(D)$ can be determined by looking at the evolution of a and v for the different values of D.

The data recorded during the experiment for every D are shown in Figure 3.11, where $\hat{a}$ are the accelerations that have been smoothed. The accelerations abruptly increase at the start then decrease until being null. Therefore, the speeds increase until reaching a plateau. It is explained by the fact that at a certain speed, the term $f_m(D)$ is totally compensated by the term $-f_d(v)$ in the expression of the acceleration. In other words, the force generated by the motor is equal to the force exerted by drag.

An important characteristic of the throttle has been discovered during this experiment. To set the car in motion when it is completely stationary, a minimum throttle of 0.2 is required during 300 ms before applying the desired throttle. Therefore, a command $D = 0.2$ is

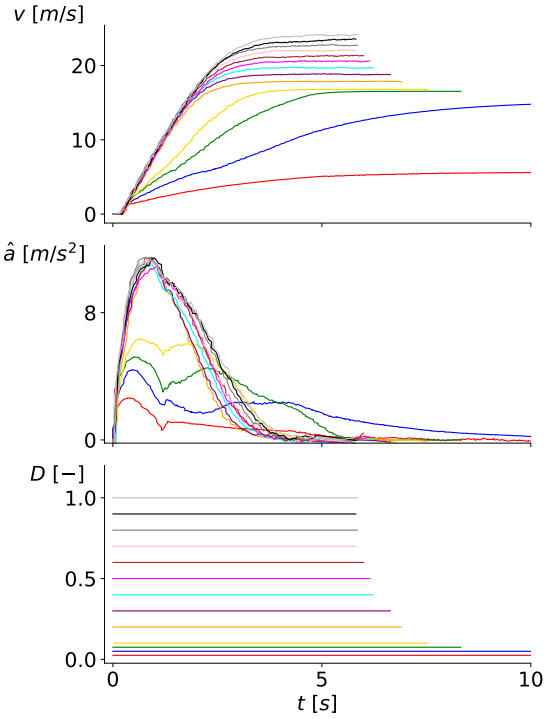

Figure 3.11: data of experiment 3 for the different values of D

applied at start during 300 ms then the desired value of D is applied.

The accelerations are almost the same for $D > 0.2$ but differ for $D \leq 0.1$. The relation between a and v is therefore very different depending on the value of D, as it can be seen with the measures in Figure 3.12. To perfectly model the acceleration depending on the throttle applied, it would be necessary to have a different function depending on the speed for every D. However, such approach would result in a very complex model not suitable for this application. Therefore, it has been decided to approximate the a, v relation by a line for each value of D, as shown in Figure 3.12. Each line is expressed as (remembering than $C_r = 0$),

$$a = \frac{1}{m}(c - C_d v), \tag{3.12}$$

where $c \in \mathbb{R}$ is the coefficient obtained by the better fitting of the line with the measures. For each D, a coefficient c is found, which represents the value of $f_m(D)$. By looking at their relation, the function $f_m(D)$ for all positive D can be determined.

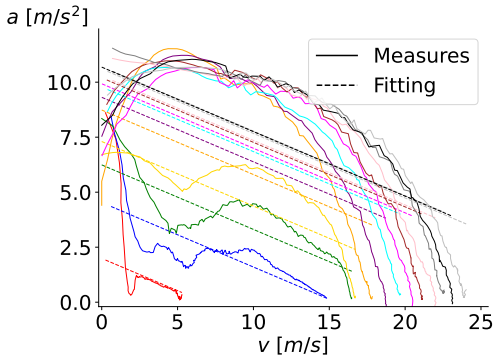

Figure 3.12: Relation between the accelerations and speeds measured for the different values of D in experiment 3, and their fitting line

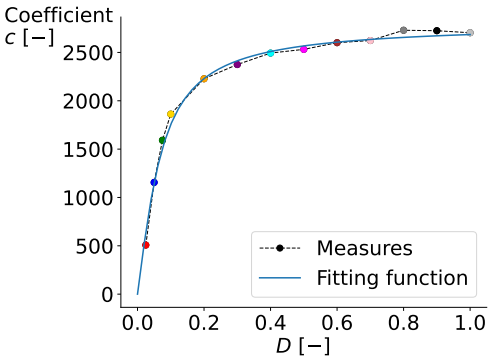

Figure 3.13: Relation between the coefficients c computed with the fitting lines and D in experiment 3, and the fitting function $f_m(D) = C_{m1} \arctan(C_{m2} D)$

The different coefficients c as a function of D are plotted in Figure 3.13, and it can be observed that the curve formed by the points resembles an arctan function. The arctan function respects the condition $\arctan(0) = 0$, thus it can be chosen to model $f_m(D)$, where two coefficients C_{m1}, C_{m2} must be determined to obtain the better fitting possible,

$$f_m(D) = C_{m1} \arctan(C_{m2} D) \tag{3.13}$$

By taking,

$$\begin{cases} C_{m1} = 7m = 1785 \\ C_{m2} = 15 \end{cases} \tag{3.14}$$

the fitting function in Figure 3.13 is obtained.

By using this approach, the motor model is not computationally too complex, which is necessary to be able to be used in real-time, but does not perfectly describes the behavior of the acceleration reacting to the throttle command. It could degrade the quality of the prediction, therefore the model must still be validated to verify that the difference is not to big.

- **Experiment 4:** Determination of $f_m(D)$ for $D < 0$ (brake)

To reach high speed, full throttle ($D = 1$) is applied to the car at the start. When $v = 15\,m/s$, brake is applied ($D < 0$) and the data recording is stopped when the car is at rest. This procedure is repeated for different values of D in its negative range. Typically,

$$D = [-0.05,\ -0.1,\ -0.2,\ -0.3,\ -0.4,\ -0.5,\ -0.6,\ -0.8,\ -1.0]$$

The recorded data are shown in Figure 3.14. The speed decreases more and more rapidly as D becomes more negative until it reaches $D = -0.6$. For $D \leq -0.6$, the speed decreasing it identical. In Figure 3.15, the relation between the acceleration and the speed in plotted. Applying the same line fitting as in experiment 3 produces very good fittings for $D \geq -0.6$ thanks to the quasi linear behavior between a and v. For $D \leq -0.6$, the quality of the fittings decreases.

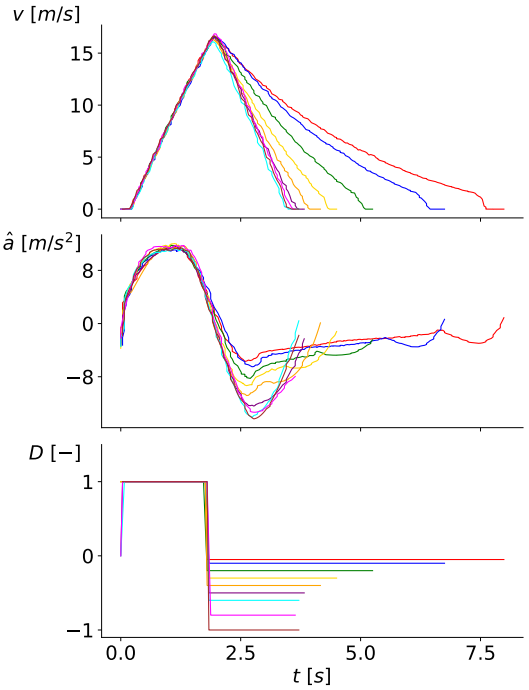

Figure 3.14: data of experiment 4 for the different values of D

The distinction for $D \geq$ or ≤ -0.6 is even clearer in Figure 3.16 where the coefficients c evolve in two different ways on each side of $D = -0.6$. The function $f_m(D)$ computed in experiment 3 has also been plotted to verify that $f_m(D)$ correctly models both throttle and brake. It can be seen that this is not the case. Although the difference is not too big for $D \leq -0.6$, it is for $D \geq -0.6$ which could really degrade the quality of the braking of the car.

The fitting error for the brake could be reduced by changing the coefficients C_{m1} and C_{m2} but it would also increase the fitting error for the throttle. Since it is important to have a good model for both throttle and brake, an alternative must be found. Looking again at the computed coefficients in Figure 3.16, the full brake could not be easily modeled because of the discontinuity at $D = -0.6$. But if the brake is limited to $D = -0.6$, the model would be a line perfectly fitting the measures, as

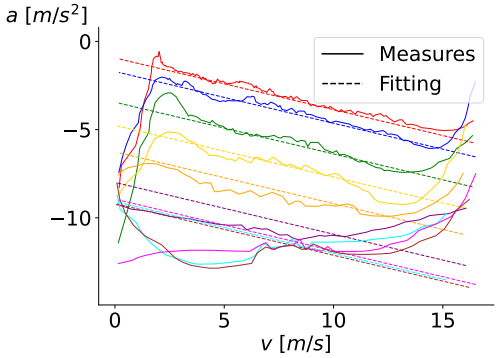

Figure 3.15: Relation between the accelerations and speeds measured for the different values of D in experiment 4, and their fitting line

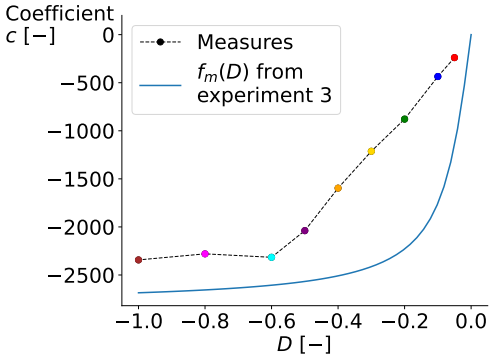

Figure 3.16: Relation between the coefficients c computed with the fitting lines and D in experiment 4, and the function $f_m(D) = C_{m1} \arctan(C_{m2} D)$ found in experiment 3

represented in Figure 3.17. With this approach, a model for the brake is computed,

$$f_b(D) = 16mD = \underbrace{4080}_{C_b} D \tag{3.15}$$

The throttle and brake are coupled by D for the model of the car, meaning that a single function must describe both in the model. But when the computed control command D is applied to the car, it is decoupled into either throttle or brake. Therefore, if $f_m(D)$ is kept for the model, D can directly be applied to the car when it is positive (throttle) because it perfectly fit the throttle coefficients. But when D is negative (brake), before applying it to the car, it can be adapted to the brake model $f_b(D)$.

The coefficient computed by the solver that must be apply to the car is known, it is described by $f_m(D_{solver})$ where D_{solver} is the D computed by the solver,

$$c = f_m(D_{solver}) = C_{m1} \arctan(C_{m2} D_{solver})$$

Then, to actually apply this coefficient to the car, the command D_{apply} that must be given corresponds to the D computed with the brake model $f_b(D)$ knowing the coefficient c wanted,

$$D_{apply} = \frac{c}{C_d} = \frac{C_{m1}}{C_b} \arctan(C_{m2} D_{solver}) \tag{3.16}$$

The conversion of D_{solver} (D) into D_{apply} (brake command) using $f_m(D)$ and $f_b(D)$ is shown in Figure 3.18.

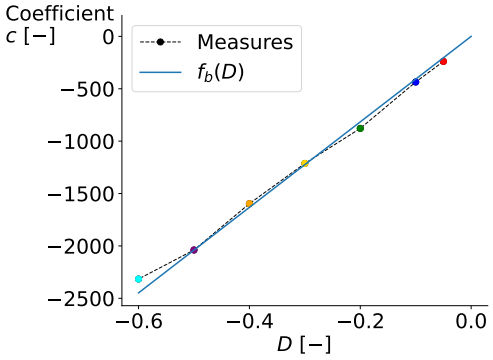

Figure 3.17: Fitting of the measured coefficients c in experiment 4 by the brake model $f_b(D)$ for $D \in [-0.6, 0]$

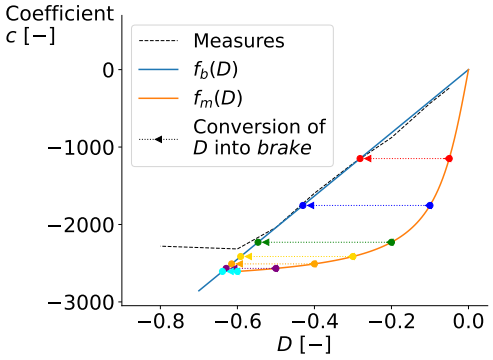

Figure 3.18: Conversion of D into brake command using $f_m(D)$ and $f_b(D)$

To resume, the control inputs sent to the car are,

$$throttle = \begin{cases} D & if D \geq 0 \\ 0 & if D < 0 \end{cases}$$

$$brake = \begin{cases} 0 & if D \geq 0 \\ \frac{C_{m1}}{C_b} \arctan(C_{m2} D) & if D < 0 \end{cases}$$

and the drivetrain model used to expressed the acceleration is,

$$a = \frac{1}{m}(C_{m1} \arctan(C_{m2} D) - C_d v) \tag{3.17}$$

with,

$$\begin{cases} C_{m1} & = 1785 \\ C_{m2} & = 15 \\ C_b & = 4080 \\ C_d & = 74.01 \end{cases} \tag{3.18}$$

Drivetrain model validation

To validate the drivetrain model, the state model has been implemented in its discrete form (Euler order 1),

$$\boldsymbol{x}_{k+1} = \underbrace{\boldsymbol{x}_k + \Delta t f_{kin}(\boldsymbol{x}_k, \boldsymbol{u}_k)}_{f(\boldsymbol{x}_k, \boldsymbol{u}_k)} \tag{3.19}$$

with the values $l_R = 0.783\,m$, $l_F = 0.435\,m$ and is used to perform the experiment 3 for throttle and experiment 4 for brake. The results obtained with the model are then compared to the measures.

In Figure 3.19 and 3.20, the speeds are compared for low throttles and higher throttles respectively. It can be observed on both figures that the model well describes

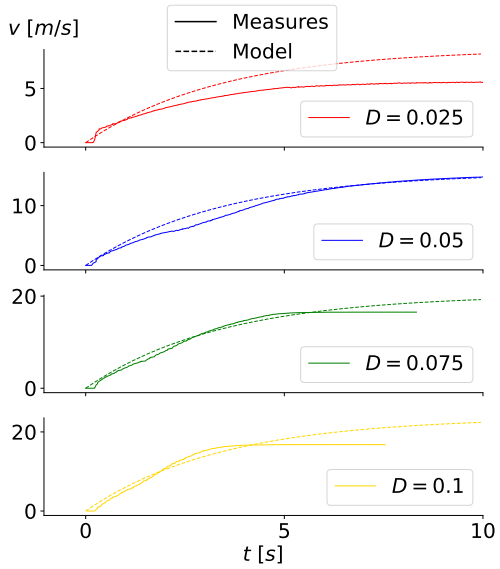

Figure 3.19: Validation of the drivetrain model at low throttle

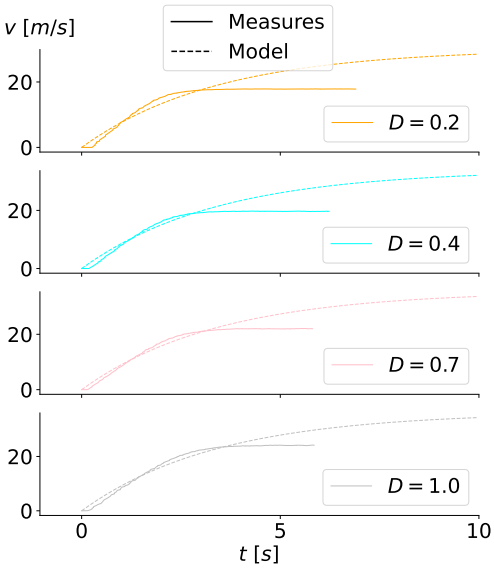

Figure 3.20: Validation of the drivetrain model at higher throttle

the behavior of the car in the increasing part of the speed. Then, when the measured speed reaches a plateau, the quality of the model is degrading depending on D. The bigger D is, the bigger the difference between the measures and the model. For low D, especially for $D = 0.05$ and $D = 0.075$, the model matches very well the measures. In this range of D, the maximum plateau (constant speed) reaches almost $20\,m/s$, which is already a high speed. Therefore, as long as the target speed given to the car does not exceed $20\,m/s$, the model is well describing the car behavior. Of course, even in this situation, the controller could apply a higher D to accelerate quickly but in this case the speed would be in its increasing part, which is well described by the model even at higher D.

In Figure 3.21, the speeds are compared when brake is applied, in the range of $D \in [-0.6, 0]$ covert by the brake model $f_b(D)$. The experiment 4 is per-

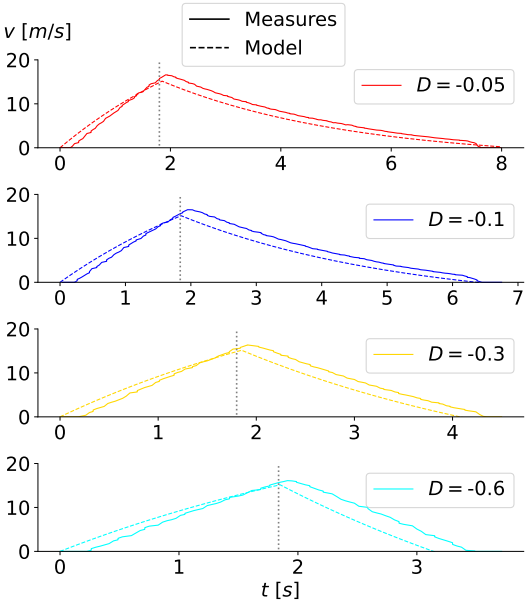

Figure 3.21: Validation of the brake model for $D \in [-0.6, 0]$ (the dotted vertical lines indicate the time at which the brake is applied)

formed with the model, meaning that the brake is applied when $v = 15\,m/s$ (the dotted vertical lines indicate the time at which the brake is applied). It can be observed that for the model, the speed directly starts decreasing when the brake is applied. But on the measures, there is a time interval in which the car's speed continues to increase, then the brake takes effect. This phenomena cannot be modeled but should not cause any problems since the time interval is relatively small ($\sim 100\,ms$).

Then, looking at the curves after the brake is applied, it can be verified that the brake model $f_b(D)$ is very accurate. The small gap between measures and the model is only due to the brake effect's interval time. Without it, the curves almost perfectly match.

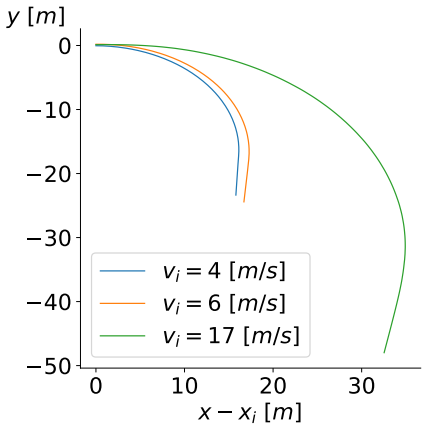

Figure 3.22: Comparison of the three measured curves taken by the car at different speeds but same steering angle (10°)

Full model validation

Now that the drivetrain model has been characterized and validated, the rest of the model can be validated. In particular the behavior in a curve, since strong hypothesis have been made at this level (no slip).

First of all, the behavior of the car is observed when applying the same steering angle at different speeds to check if the car is subjected to slip. The results for three different speeds and a steering angle of 10° to the right are shown in Figure 3.22. The car is brought to a speed v_i in straight line by applying a certain throttle command at the start. The moment the steering angle is applied, the x position of the car is x_i. The three curves are started when $x = x_i$ for them to start when the steering angle is applied and be able to compare them. When the orientation of the car has reached −90°, the steering angle is reset to 0. If there was no slip, the three curves would be approximately the same. Whatever the speed, if there is full grip between the tire and the road, the car takes the same trajectory. In Figure 3.22, it can be observed that there is slip.

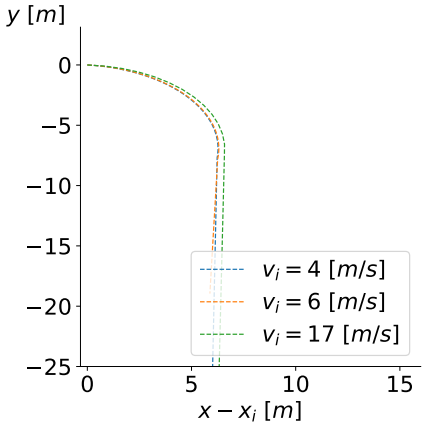

Figure 3.23: Comparison of the three curves taken at different speeds but same steering angle (10°) computed with the model

To compare the measures with the model, the same procedure is applied and the results are compared in Figure 3.24 for the three different

speeds. The differences are important because of the hypothesis of no slip made to build the model. The higher the speed, the higher the slip and therefore the differences. Without slip, the car is able to turn more efficiently. If the three curves computed with the model are compared, it can be seen that no slip is considered in the model since the three curves are almost identical (see Figure 3.23).

The prediction performed with the model by the solver will therefore be much more idealistic than the real behavior of the car. If the difference is to big, typically for higher speeds, the quality of the control will greatly be decreased. It can be predicted that the kinematic model will allow to have a relatively accurate control at low speeds but will not be efficient at higher speeds.

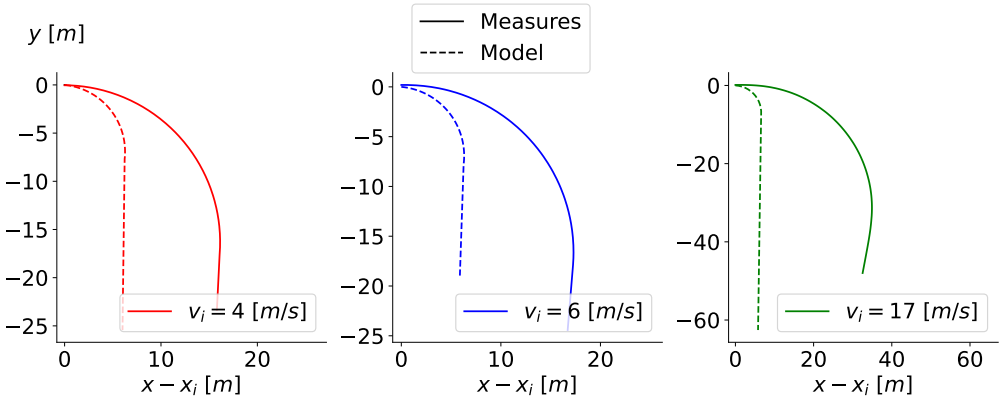

Figure 3.24: Validation of the full model in a curve taken at different speeds but same steering angle (10°)

3.4 MPCC problem

3.4.1 Design

Contouring formulation

The contouring formulation aims at linking the MPC problem to a contour following. The contour to follow is the reference path formed by the reference points at the center of the track. The reference path formulation will be described in the next Subsection 3.4.1 based on the contouring formulation needs.

To follow the path, the position of the car (x, y) must be link to the path. By projecting the car's position (x, y) into the reference path (at the center of the track), the distance between the two positions, called the contouring error ϵ^c, can be found, as shown in Figure 3.25. This distance is defined as the normal deviation from the reference path. It can be expressed by,

$$\epsilon^c = \sin(\phi_c(s_r))(x - x_c(s_r)) - \cos(\phi_c(s_r))(y - y_c(s_r)),$$

where $(x_c(s_r), y_c(s_r))$ is the position at the projection, $\phi_c(s_r)$ is defined as the angle formed by the tangent at the projection and the x axis,

$$\phi_c(s_r) = \arctan2\left(\nabla y_c(s_r), \nabla x_c(s_r)\right),$$

and $s_r(x, y)$ is the value of the path parameter where the distance between the point $(x_c(s_r), y_c(s_r))$ and (x, y) is minimal.

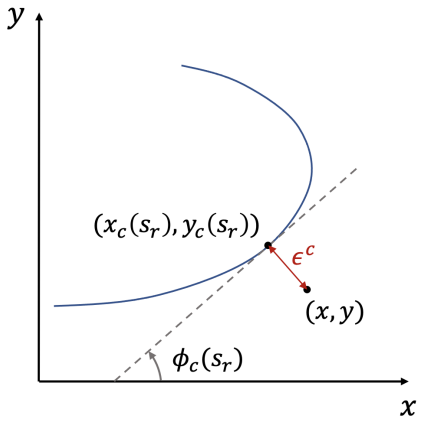

Figure 3.25: Contouring error by projection of the car's position into the reference path

The objective of the algorithm is therefore to compute the car's commands that minimize the contouring error while maximizing the speed along the path. However, this formulation requires to make the projection of a point into a curve, which can be computationally expensive depending on the degree of the curve, and does not allow to have a measure of the speed along the track. For these reasons, it is proposed to extend the system with a new state to define the progress along the path [51]. This is a virtual state, denoted as s, that indicates the distance traveled by the car along the path, and its evolution is governed by the speed v_s at which the car travels along the path,

$$\dot{s} = v_s, \tag{3.20}$$

and is called the virtual state model.

To ensure that the car is always progressing (not moving backward), v_s must be expressed as $v_s \in [0, v_{s,max}]$ with $v_{s,max} > 0$. This way, s is a purely increasing quantity. v_s is a virtual control input of the car determining at which speed the car must travel along the path.

The evolution of the virtual state s must be linked to the actual evolution of the car. If not, the car and the virtual state would evolve independently of each other. For this purpose, it is proposed to use s as an approximation to $s_r(x, y)$ and express the reference path in terms of s by arc-length parameterization.

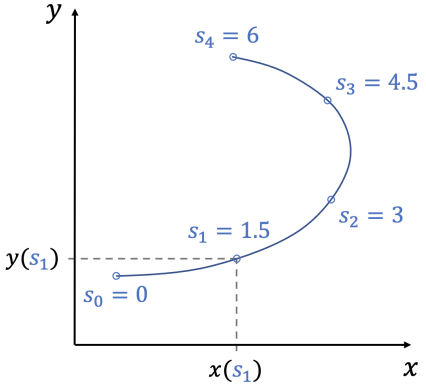

Figure 3.26: Example of curve parameterization by arc-length

Arc-length parameterization is a way of defining a parametric curve in terms of the distance along the curve. In other words, instead of defining a curve using its x and y coordinates as functions of a parameter such as time, the distance along the curve is used as the parameter. The distance along the curve is called the arc-length

and have the same definition as s. Therefore, for a given curve in a 2D frame, it is possible to determine the position on the curve $(x(s), y(s))$ based on the distance s traveled along that curve, as represented in Figure 3.26.

With the arc-length parameterization of the reference path, the virtual position of the car on the path $(x_c(s), y_c(s))$ can be found using the virtual state s and v_s is directly proportional to the real speed of the car v. Then, considering s as an approximation of s_r allows to link this virtual position with the projection of the car on the path $(x_c(s_r), y_c(s_r))$. This is represented in Figure 3.27, with ϵ^c the contouring error approximated by,

$$\begin{aligned}\hat{\epsilon}^c(\boldsymbol{x}, s) =& \sin(\phi_c(s))(x - x_c(s)) \\ &- \cos(\phi_c(s))(y - y_c(s))\end{aligned} \tag{3.21}$$

and ϵ^l the lag error, denoting the path distance that $(x_c(s_r), y_c(s_r))$ lags $(x_c(s), y_c(s))$, approximated by,

$$\begin{aligned}\hat{\epsilon}^l(\boldsymbol{x}, s) =& -\cos(\phi_c(s))(x - x_c(s)) \\ &- \sin(\phi_c(s))(y - y_c(s))\end{aligned} \tag{3.22}$$

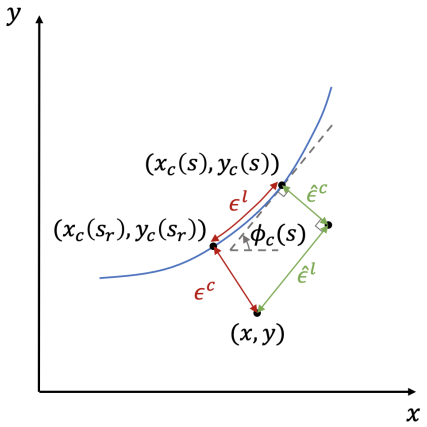

Figure 3.27: Contouring and lag errors and their approximations

For the virtual position on the path to be close to the actual car's position projection of the path, the lag error must be small, $\hat{\epsilon}^l(\boldsymbol{x}, s) \approx 0$. The virtual state s, as well as the virtual control command v_s, will need to be added to the state and commands of the system respectively. They become:

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\phi} \\ \dot{v} \\ \dot{s} \end{bmatrix} = \begin{bmatrix} v\cos(\phi + \beta) \\ v\sin(\phi + \beta) \\ \frac{v}{l_R}\sin(\beta) \\ \frac{1}{m}\left(f_m(D) - C_r - f_d(v)\right) \\ v_s \end{bmatrix}, \tag{3.23}$$

with,

$$\beta = \arctan\left(\frac{l_R}{l_R + l_F}\tan(\delta)\right),$$

and $\boldsymbol{u} = [\delta \ D \ v_s]^T$.

Reference path formulation

The reference path formulation is based on the needs of the contouring formulation and the output of the PP algorithm given to the PF algorithm. From the reference points computed at the center of the track, a curve parameterized by arc-length must be produced. In addition to the reference points, the virtual state s is known.

The objective of the reference path formulation is to give to the MPCC the tools needed to compute and update the approximated contouring and lag errors inside the optimization problem. During the prediction, s is updated by the computed sequence of control commands v_s, and the contouring and lag errors also need to be updated based on s. Like in the MPC theory Section 3.2, the subscript k will be used to define a quantity at prediction k, with $k = 1, 2, ..., N$.

During the optimization process, the evolution of successive s_k, described by the virtual state model discretized using order 1 Euler method with time step Δt_{MPCC},

$$s_{k+1} = s_k + \Delta t_{\text{MPCC}} v_{s,k},$$

is limited by the upper bound on v_s ($v_{s,max}$). Before each call to the MPCC, the initial state s is available and given to the solver as s_1. It is therefore possible to determine the maximum value of s at the end of the prediction s_N,

$$s_N \leq s_1 + N \Delta t_{\text{MPCC}} v_{s,max}$$

The reference points with which the reference path will be computed can therefore be selected based on the minimum arc-length needed and the current s. The reference point formulation thus requires two steps performed at each time step Δt: the selection of the reference points and the computation of the path based on these points.

1. Reference points selection:

   All the reference points can be utilized to form a linear piecewise curve, as shown in green split line in Figure 3.28.a). The reference points are denoted as r_i with $i = 0, 1, ..., n-1$, and n the number of reference points available. Each line between two successive points r_i, r_{i+1} has a length dl_{i+1} equal to the distance between them,

   $$dl_{i+1} = \sqrt{(x_{r_{i+1}} - x_{r_i})^2 + (y_{r_{i+1}} - y_{r_i})^2} \quad \text{for } i \in \{0, 1, ..., n-2\}$$

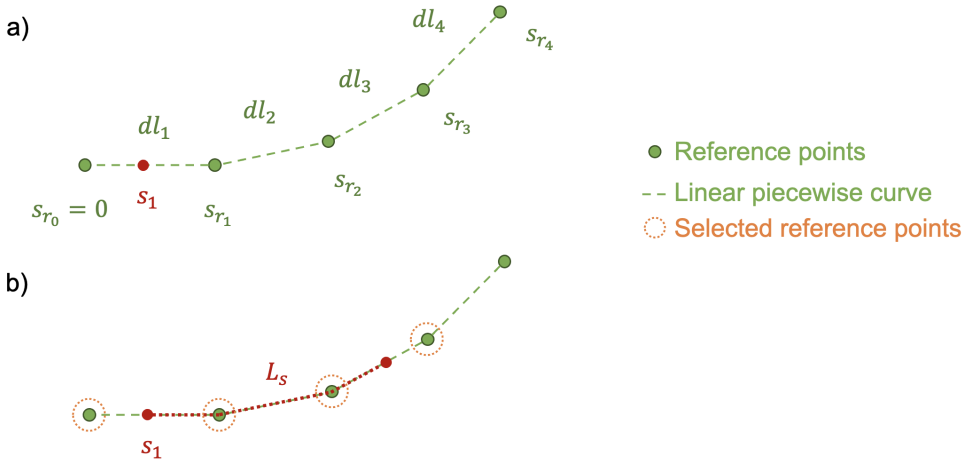

Figure 3.28: Reference points selection example with quantities used for the selection

Then, each point can be assigned an arc-length s_{r_i} on the linear piecewise curve, beginning with $s_{r_0} = 0$. The arc-length parameterized linear spline curve obtained can now be used to approximate the advancement of the car s on the reference path. The term approximation is used to indicate that the real reference path which would be a smooth curve is approximated by a linear piecewise curve, as represented in Figure 3.29. The reference points being relatively close to each other, this is a good approximation.

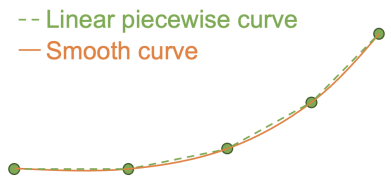

Figure 3.29: Smooth curve approximation by linear piecewise curve

The current advancement of the car s_1 can be positioned on the piecewise curve by looking at the arc-length of the reference points s_{r_i}. In the example of Figure 3.28.a), $s_{r_0} < s_1 < s_{r_1}$ thus the first reference point to be selected is r_0. Then the other reference points must be selected based on the required length $L_s = N\Delta t_{\text{MPCC}} v_{s,max}$ and the minimum number of reference points imposed n_{min}. This is important to impose a minimum number of reference points for the computation of the reference path later, depending on the type of curve that will be generated (a cubic curve will require minimum 4 points for example). The next reference points are added one at a time until the length formed by the sum of dl_i is greater than L_s plus the distance between the first selected reference point and s_1. For the example in Figure 3.28, the 4 first points are selected because,

$$dl_1 + dl_2 + dl_3 \geq L_s + (s_1 - s_{r_0}) \geq dl_1 + dl_2$$

If the number of selected reference points $n_{select} < n_{min}$ even when the required length is met, more reference points must be selected until $n_{select} = n_{min}$.

The procedure described above is the standard case. Special cases can occur when there is not enough reference points available, as illustrated in Figure 3.30.a). This can happened during the first lap, when the PP algorithm still generates the reference points in real-time with a limited view ahead. To handle this situation, virtual reference points are added after the last available reference point. This is illustrated in Figure 3.30.b), where a virtual reference point has been added to met the required length. The length dl_v to add is computed by,

$$dl_v = L_s - (dl_1 + dl_2 - (s_1 - s_{r_0})),$$

and the orientation of the virtual reference point with respect to the previous point r_2 is taken equal to the relative orientation ϕ_1 of the two previous points

r_1 and r_2, with,

$$\phi_1 = \arctan2(y_{r_2} - y_{r_1}, x_{r_2} - x_{r_1})$$

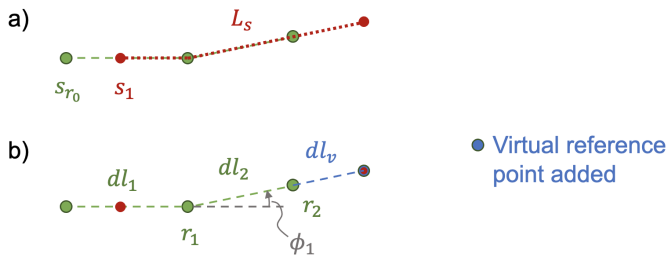

Figure 3.30: Special case of reference points selection - not enough reference points available

If more than one added point is required to respect the condition $n_{select} = n_{min}$, the length dl_v is split in the number of points that need to be added.

With this method, the added virtual reference points are not guarantied to follow the center of the track, which could deteriorate the quality of the prediction. the greater the length to be added, the greater the risk of deviation from the center of the track.

A last particular case occurs when the first lap has been closed, meaning that all the reference points have been computed, and points must be selected after the last computed ones. In this situation, new points do not have to be added since the track is complete. The points that come after the last ones at the end of the track are the first points of the track, as represented in Figure 3.31. The first selected reference points are the two last ones of the complete track, r_{n-2} and r_{n-1}. Then, to respect the required length L_s, the two first cones of the track r_0 and r_1 are selected.

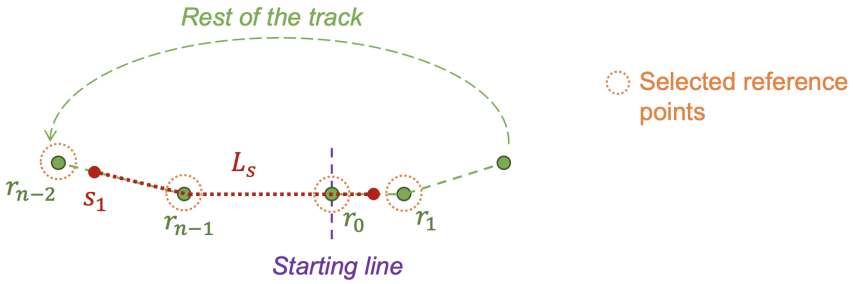

Figure 3.31: Special case of reference points selection - selection of last and first reference points when the track is completed

Since the arc-length of the reference points is defined on one lap, the arc-length of the first reference point of the track is equal to 0 when the starting line is crossed at the end of a lap. It means that between the last and first reference

point of the track, the arc-length suddenly drops from a positive value to 0. For the arc-length of the reference points and the advancement s of the car to be synchronized, s is also reinitialized to 0 when the starting line is crossed.

2. Reference path computation:

   The contouring formulation requires to give to the optimization solver the reference path in the form of parameters in order to compute the contouring and lag errors based on the updated s_k at each prediction. A solution that has been investigated is to compute the arc-length parameterized cubic spline curve with the selected reference points. This solution is the most precise since the curve is guarantied to pass exactly through the points and to be smooth thanks to its third order. However, the result is a piecewise curve, meaning that the number of parameters necessary to describe the full curve increases with the number of points (for n_{select} reference points, $n_{select} - 1$ sub-curves are necessary to form the full curve). The number of parameters to send to the optimization solver can therefore vary. In addition, finding the location of a s_k on the curve requires a condition statement: if s_k is between the arc-length of the first and second point, then the parameters of the first sub-curve must be used, and similarly for each pair of successive points. Since its impossible to make a condition statement inside the optimization solver, this solution has been discarded.

   In order to not have condition statements, the entire curve must be described by the same parameters. For this purpose, the solution that has been chosen is polynomial fitting by least squares regression. This method allows to find the best-fitting curve by minimizing the sum of squared differences between the data points and the predicted values. This technique does not guaranty that the curve passes through the points, but the higher the order of the curve, the higher the quality of the fitting.

   The reference points that must be fitted are in the (x, y) frame and the coordinates of the point on the curve must be found using the advancement (arc-length) s_k. In addition, the function (polynomial) returned by the least-square regression is well-defined, meaning that the data points must be strictly increasing on the x axis, which is not necessary the case for the reference points in the (x, y) frame. Therefore, the points are first expressed in terms of the arc-length s by splitting the (x, y) frame into two frames: (x, s) and (y, s), as shown in Figure 3.32. Moreover, the arc-length s is a strictly increasing quantity, meaning that the data sets of the two frames are well-defined.

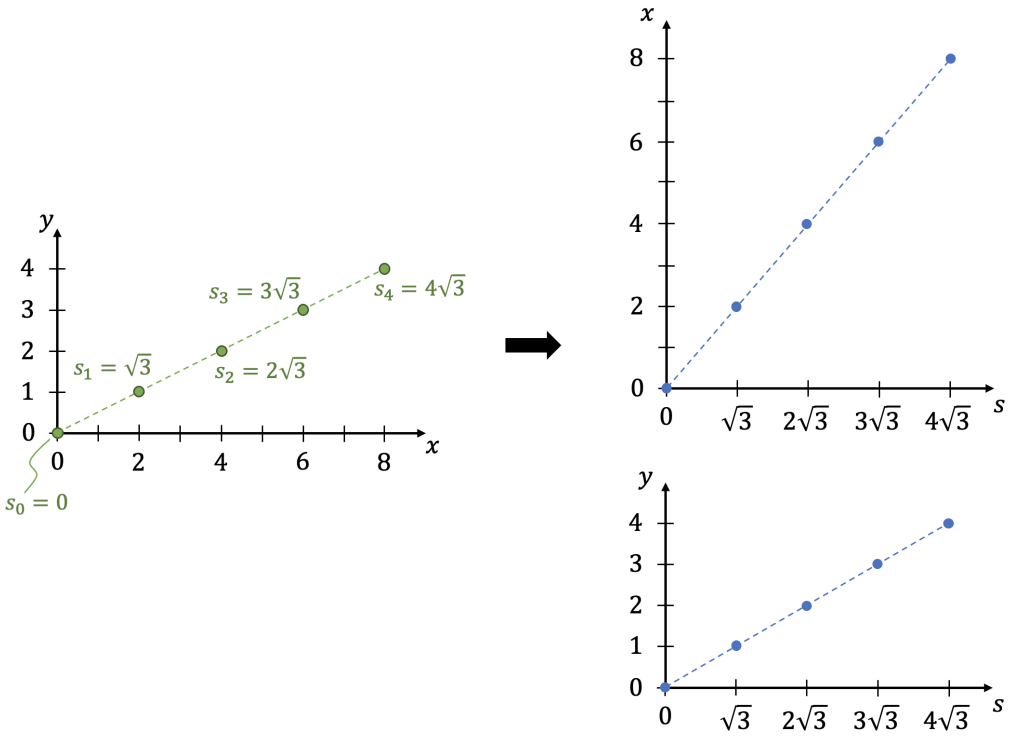

Figure 3.32: (x, y) frame split into (x, s) and (y, s) frames

Two polynomials can therefore be computed by least square regression on these two data sets to obtain the two following expressions,

$$\begin{cases} x(s) = X_0 + X_1 s + X_2 s^2 + ... + X_m s^m \\ y(s) = Y_0 + Y_1 s + Y_2 s^2 + ... + Y_m s^m \end{cases}$$

where m is the order of the polynomials and X_j, $Y_j \in \mathbb{R}$ for $j = 0, 1, ..., m$ the coefficients of the x and y polynomials respectively. Each polynomial thus produces $m + 1$ parameters that can be sent to the optimization solver to determine the coordinates on the reference path based on the advancement s.

To compute the coefficients, each data set (x_i, s_i), (y_i, s_i) for $i \in \{0, 1, ..., n_{select} - 1\}$ is transformed into a system $A\vec{x} = \vec{b}$ with $\vec{b} \in \mathbb{R}^{n_{select}}$ the vector of ordinates (x_i or y_i), $\vec{x} \in \mathbb{R}^{m+1}$ the vector of coefficients (X_j or Y_j) to find and $A \in \mathbb{R}^{n_{select} \times (m+1)}$ the matrix of abscissa (s_i) to which the exponents are applied. For the case of the (x_i, s_i) data set, it gives,

$$\underbrace{\begin{bmatrix} 1 & s_0 & s_0^2 & \cdots & s_0^m \\ 1 & s_1 & s_1^2 & \cdots & s_1^m \\ 1 & s_2 & s_2^2 & \cdots & s_2^m \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & s_{n_{select}-1} & s_{n_{select}-1}^2 & \cdots & s_{n_{select}-1}^m \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ \vdots \\ X_m \end{bmatrix}}_{\vec{x}} = \underbrace{\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{n_{select}-1} \end{bmatrix}}_{\vec{b}}$$

The committed error can be represented by the residue vector,

$$\vec{r} = \vec{b} - A\vec{x},$$

and the norm of the residue $\|\vec{r}\|_2 = \|\vec{b} - A\vec{x}\|_2$ is minimum if and only if $\vec{x}$ satisfies the normal equations,

$$(A^T A)\vec{x} = A^T\vec{b}$$

Therefore, the researched coefficients are found with,

$$\vec{x} = (A^T A)^{-1} A^T \vec{b}$$

For the generation of the polynomials, the arc-length of the first selected reference point is always set to 0 and the ones of the next selected points are determined taking the first point as reference. It means that a relative arc-length is used to generate the polynomials. The absolute arc-length of a reference point is the arc-length of the point in the track. This method allows to handle the particular case that happens when, at the end of a lap, the selected points include last and first reference points of the track.

To switch from the absolute arc-length to the relative arc-length of any point and vice-versa, only the absolute arc-length of the first selected reference point is needed. This is illustrated with an example in Figure 3.33. The index of the first selected reference point is $I = 2$, then the relative arc-length of the selected points can be computed by,

$$s_{rel_i} = s_{abs_i} - s_{abs_I} \qquad \text{for } i \in \{I, I+1, ..., I+n_{select}-1\}$$

The polynomials are then computed with the relative arc-lengths. The coefficients of the polynomials are sent as parameters to the optimization solver as well as the absolute arc-length of the first selected reference point s_{abs_I}. It allows the optimization solver to compute the coordinates $(x_c(s_k), y_c(s_k))$ and the tangent orientation $\phi_c(s_k)$ for each predicted s_k with,

$$\begin{cases} x_c(s_k) = X_0 + X_1(s_k - s_{abs_I}) + X_2(s_k - s_{abs_I})^2 + ... + X_m(s_k - s_{abs_I})^m \\ y_c(s_k) = Y_0 + Y_1(s_k - s_{abs_I}) + Y_2(s_k - s_{abs_I})^2 + ... + Y_m(s_k - s_{abs_I})^m \\ \phi_c(s_k) = \arctan2(\nabla y_c(s_k), \nabla x_c(s_k)) \end{cases}$$

where,

$$\begin{cases} \nabla x_c(s_k) = X_1 + 2X_2(s_k - s_{abs_I}) + ... + mX_m(s_k - s_{abs_I})^{m-1} \\ \nabla y_c(s_k) = Y_1 + 2Y_2(s_k - s_{abs_I}) + ... + mY_m(s_k - s_{abs_I})^{m-1} \end{cases}$$

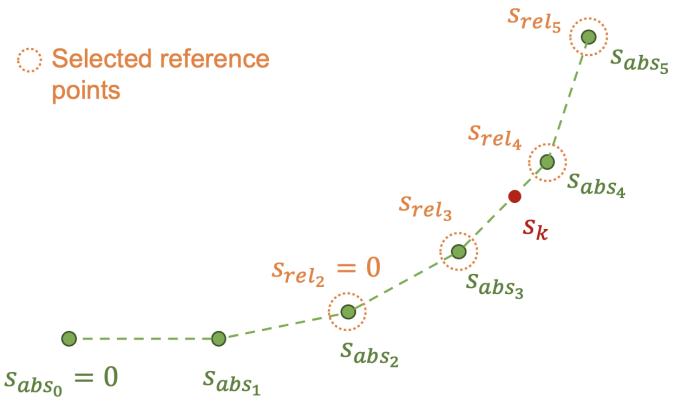

Figure 3.33: representation of the absolute and relative arc-lengths of a set of reference points

MPCC problem formulation

Now than the vehicle model, the contouring formulation and the reference path formulation have been developed, the MPCC problem can be formulated. The different constituents will first be developed then the overall problem will be formulated. As in the MPC theory Subsection 3.2, the prediction horizon is denoted as N with current prediction $k = 1, 2, ..., N$ and the time step inside the MPCC is Δt_{MPCC}.

- **State and control commands vectors**

  The state vector is the vehicle state extended with the contouring formulation. Its continuous model is the vehicle model (3.5) extended with the virtual state model (3.20), resulting in the model (3.23). For the MPCC formulation, the extended state model is discretized by order 1 Euler method for computational efficiency,

$$\boldsymbol{x}_{k+1} = \begin{bmatrix} x_{k+1} \\ y_{k+1} \\ \phi_{k+1} \\ v_{k+1} \\ s_{k+1} \end{bmatrix} = \begin{bmatrix} x_k \\ y_k \\ \phi_k \\ v_k \\ s_k \end{bmatrix} + \Delta t_{\text{MPCC}} \begin{bmatrix} v_k \cos(\phi_k + \beta_k) \\ v_k \sin(\phi_k + \beta_k) \\ \frac{v_k}{l_R} \sin(\beta_k) \\ \frac{1}{m} \left(f_m(D_k) - C_r - f_d(v_k) \right) \\ v_{s,k} \end{bmatrix}, \tag{3.24}$$

  with,

$$\beta_k = \arctan\left(\frac{l_R}{l_R + l_F} \tan(\delta_k) \right),$$

  and $\boldsymbol{u}_k = [\delta_k \ D_k \ v_{s,k}]^T$ the extended control commands vector.

  Therefore, the state vector $\boldsymbol{x}_k = [x_k \ y_k \ \phi_k \ v_k \ s_k]^T \in \mathbb{R}^{n_x}$ has size $n_x = 5$ and the control commands vector $\boldsymbol{u}_k = [\delta_k \ D_k \ v_{s,k}]^T \in \mathbb{R}^{n_u}$ has size $n_u = 3$. The evolution of the state vector is described by (3.24),

$$\boldsymbol{x}_{k+1} = f(\boldsymbol{x}_k, \boldsymbol{u}_k),$$

  with $f : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_x}$.

- **Inequality constraints**

  The MPCC problem accepts linear and nonlinear inequality constraints that must be respected when the optimization problem is solved. First of all, rectangle constraints are applied on the control commands. Their values are limited by lower and upper bounds to keep them within the desired range of values,

  $$\underline{\boldsymbol{u}}_k \leq \boldsymbol{u}_k \leq \overline{\boldsymbol{u}}_k \tag{3.25}$$

  In addition, rectangle constraints are applied on the control commands rate of change $\boldsymbol{\Delta u}_k = [\delta_k - \delta_{k-1} \quad D_k - D_{k-1} \quad v_{s,k} - v_{s,k-1}]^T$. This allows to limit the successive change of control command values in order to compute relatively smooth control commands and eliminate potential impossible changes of values for the real system,

  $$\underline{\boldsymbol{\Delta u}}_k \leq \boldsymbol{\Delta u}_k \leq \overline{\boldsymbol{\Delta u}}_k \tag{3.26}$$

  Finally, a nonlinear constraint is applied on the contouring error $\hat{\epsilon}_k^c$ to model the track limits. The car must not deviate too much from the center of the track to not touch the cones delimiting the track. The contouring error represents the lateral deviation from the reference path computed at the center of the track. It must not exceed a maximum value R_c on both sides of the center of the track,

  $$(\hat{\epsilon}_k^c)^2 \leq R_c^2 \tag{3.27}$$

- **Cost function**

  The cost function $J \in \mathbb{R}$ to minimize is expressed as the sum of cost functions J_k over the prediction horizon,

  $$J = \sum_{k=1}^{N} J_k,$$

  J_k is composed of several components. The first one is the objective function Q_k that expresses the main objectives that must be accomplished by the MPCC algorithm, which is maximizing the speed along the path while remaining within the track limits. Maximizing the speed along the track implies maximizing the virtual speed v_s and remaining within the track limits is formulated as minimizing the contouring and lag errors,

  $$Q_k = q_{l,k}(\hat{\epsilon}_k^l)^2 + q_{c,k}(\hat{\epsilon}_k^c)^2 + q_{v,k}(v_{s,max} - v_{s,k})^2, \tag{3.28}$$

  with $q_{l,k}$, $q_{c,k}$, $q_{v,k} \in \mathbb{R}^+$ the weights of the objective function. They are parameters of the problem and can be adjusted to obtain the desired performance. They can also vary depending on the prediction k. With this formulation, the optimization solver will try to minimize the errors and maximize the speed by minimizing the difference between the target speed $v_{s,max}$ and the speed along the path v_s.

The errors are computed with the contouring formulation,

$$\boldsymbol{\epsilon}_k = \begin{bmatrix} \hat{\epsilon}_k^l(\boldsymbol{x}_k) \\ \hat{\epsilon}_k^c(\boldsymbol{x}_k) \end{bmatrix} = \begin{bmatrix} -\cos(\phi_c(s_k))(x_k - x_c(s_k)) - \sin(\phi_c(s_k))(y_k - y_c(s_k)) \\ \sin(\phi_c(s_k))(x_k - x_c(s_k)) - \cos(\phi_c(s_k))(y_k - y_c(s_k)) \end{bmatrix} \tag{3.29}$$

where,

$$\begin{cases} x_c(s_k) = X_0 + X_1(s_k - s_{abs_I}) + X_2(s_k - s_{abs_I})^2 + ... + X_m(s_k - s_{abs_I})^m \\ y_c(s_k) = Y_0 + Y_1(s_k - s_{abs_I}) + Y_2(s_k - s_{abs_I})^2 + ... + Y_m(s_k - s_{abs_I})^m \\ \phi_c(s_k) = \arctan2(\nabla y_c(s_k), \nabla x_c(s_k)) \end{cases}$$

and,

$$\begin{cases} \nabla x_c(s_k) = X_1 + 2X_2(s_k - s_{abs_I}) + ... + mX_m(s_k - s_{abs_I})^{m-1} \\ \nabla y_c(s_k) = Y_1 + 2Y_2(s_k - s_{abs_I}) + ... + mY_m(s_k - s_{abs_I})^{m-1} \end{cases}$$

are computed with the reference path formulation. s_{abs_I} and the coefficients X_j, Y_j for $j \in \{0, 1, ..., m\}$ are given parameters.

The second component is the control commands penalization function $R_{u,k}$. Its goal is to penalize the control commands $\boldsymbol{u}_k$ to adjust the behavior of the car. It is formulated as,

$$R_{u,k} = r_{\delta,k}\delta_k^2 + r_{D,k}D_k^2 + r_{v_s,k}v_{s,k}^2, \tag{3.30}$$

with the weights $r_{\delta,k}$, $r_{D,k}$, $r_{v_s,k} \in \mathbb{R}^+$.

The final component is the control commands rate penalization function $R_{\Delta u,k}$ utilized to also penalize the rate of change of the control commands and obtain more degrees of freedom to adjust the behavior of the car,

$$R_{\Delta u,k} = r_{\Delta\delta,k}(\delta_k - \delta_{k-1})^2 + r_{\Delta D,k}(D_k - D_{k-1})^2 + r_{\Delta v_s,k}(v_{s,k} - v_{s,k-1})^2, \tag{3.31}$$

with the weights $r_{\Delta\delta,k}$, $r_{\Delta D,k}$, $r_{\Delta v_s,k} \in \mathbb{R}^+$.

The cost function J_k is the sum of the three components (3.28), (3.30) and (3.31),

$$J_k = Q_k + R_{u,k} + R_{\Delta u,k} \tag{3.32}$$

- **Parameters vector**

  The parameters are the quantities used in the problem formulation that are not fixed but are not part of the state or control commands vectors. For example, the dimensions of the car l_F and l_R and its mass m used in the state model are fixed quantities, they are constants and not parameters of the problem.

  The parameters vector can be decomposed into two vectors:

1. Contouring parameters vector,

$$\boldsymbol{c}_k = [s_{abs_I}\ X_0\ X_1\ ...\ X_m\ Y_0\ Y_1\ ...\ Y_m\ R_c]^T \in \mathbb{R}^{n_c}, \tag{3.33}$$

with $n_c = 2 + 2(m+1)$. It is composed of the parameters linked to the contouring formulation, i.e. the parameters of the reference path and the parameter of the track constraint R_c. This vector does not vary as a function of prediction k as its parameters are defined for the all prediction.

2. Weights vector,

$$\boldsymbol{j}_k = [q_{l,k}\ q_{c,k}\ q_{v,k}\ r_{\delta,k}\ r_{D,k}\ r_{v_s,k}\ r_{\Delta\delta,k}\ r_{\Delta D,k}\ r_{\Delta v_s,k}]^T \in \mathbb{R}^{n_j}, \tag{3.34}$$

with $n_j = 9$. It is composed of the weights used in the cost function. This vector can be different depending on the prediction k.

The resulting parameters vector is the concatenation of the two above vectors,

$$\boldsymbol{p}_k = [\boldsymbol{c}_k^T\ \boldsymbol{j}_k^T]^T \in \mathbb{R}^{n_p}$$

with $n_p = n_c + n_j$. This vector can be different depending on the prediction k. The full sequence $\boldsymbol{p_{seq}} = [\boldsymbol{p}_1\ \boldsymbol{p}_2\ ...\ \boldsymbol{p}_N] \in \mathbb{R}^{n_p \times N}$ must be provided to the solver.

- **Initial conditions**

  Initial conditions must be provided to start the optimization. The initial state $\boldsymbol{x}_{init}$ is taken as first state $\boldsymbol{x}_1$ inside the optimization problem. Moreover, the control commands rate penalization function (3.31) and the control commands rate of change constraint (3.26) require the control commands $\boldsymbol{u}_0$ to compute the first rate of change $\boldsymbol{\Delta u}_1 = \boldsymbol{u}_1 - \boldsymbol{u}_0$. $\boldsymbol{u}_0$ is the control commands vector that has been applied by the previous optimization solving Δt time earlier, $\boldsymbol{u}_{1,prev}$.

The optimization problem can therefore be formulated as a minimization problem subject to constraints,

$$\begin{aligned}
\underset{\boldsymbol{u}\in\mathbb{R}^{n_u}}{\text{minimize}} \quad & \sum_{k=1}^{N} Q_k(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{p}_k) + R_{u,k}(\boldsymbol{u}_k, \boldsymbol{p}_k) + R_{\Delta u,k}(\boldsymbol{u}_k, \boldsymbol{u}_{k-1}, \boldsymbol{p}_k) \\
\text{subject to} \quad & \boldsymbol{x}_1 = \boldsymbol{x}_{init} \\
& \boldsymbol{u}_0 = \boldsymbol{u}_{1,prev} \\
& \boldsymbol{x}_{k+1} = f(\boldsymbol{x}_k, \boldsymbol{u}_k) \\
& \underline{\boldsymbol{u}}_k \leq \boldsymbol{u}_k \leq \overline{\boldsymbol{u}}_k \\
& \underline{\boldsymbol{\Delta u}}_k \leq \boldsymbol{\Delta u}_k \leq \overline{\boldsymbol{\Delta u}}_k \\
& (\hat{\epsilon}_k^c)^2 \leq R_c^2
\end{aligned}$$

The optimization problem is solved every time step Δt. In order to start the optimization, the initial state of the car $\boldsymbol{x}_{init} = \boldsymbol{x}_1$, the previous applied control

commands $\boldsymbol{u}_{1,prev} = \boldsymbol{u}_0$ and the parameters sequence $\boldsymbol{p_{seq}}$ are given to the MPCC. Then the sequence of optimal control commands $\boldsymbol{u_{seq}} = [\boldsymbol{u}_1\ \boldsymbol{u}_2\ ...\ \boldsymbol{u}_N] \in \mathbb{R}^{n_u \times N}$ is computed and the first control commands $\boldsymbol{u}_1 = [\delta_1\ D_1\ v_{s,1}]^T$ is kept. The two car control commands δ_1 and D_1 are applied to the car and the virtual control command $v_{s,1}$ is given to the discretized virtual state model to update the advancement s. The block diagram of the controller is shown in Figure 3.34.

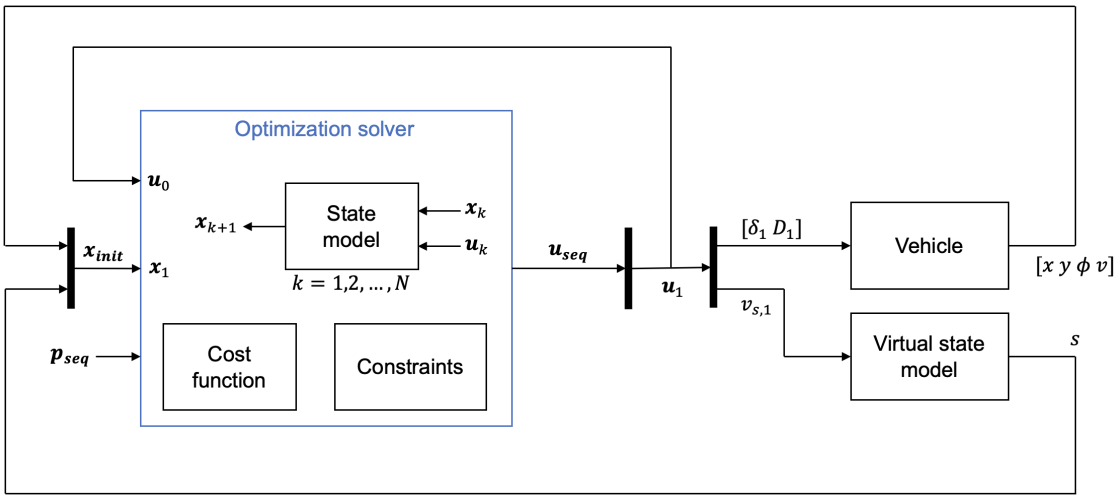

Figure 3.34: Model Predictive Contouring Controller (MPCC) block diagram

3.4.2 Implementation

Optimization solver

The solver used to solve the optimization problem is *Optimization Engine* (OpEn). It is free of utilization and benefits from an extensive documentation. The problem is formulated in Python, a library is generated which can be used in a C/C++ code. As mentioned in the introduction of this chapter, the tests revealed that the solver is limited in terms of performance. It was not able to solve the problem with prediction horizon $N > 20$. The analysis of this first solution's performance was therefore limited to $N = 20$. To extend the analysis of the improved solution, another solver was used and will be presented in the next chapter. The utilization of OpEn will therefore not be detailed in this section, but is available in Appendix D.

MPCC problem

The MPCC problem is implemented in the C++ code as a structure called `Mpcc`. It is used to store the different quantities of the problem and implement the methods necessary to solve the problem at each time step Δt.

The advancement s is stored and is updated at each time step with the control command v_s computed by the solver. Based on s and the reference points sent by the PP algorithm, a method is used to select the reference points and compute the

contouring parameters $\boldsymbol{c}$. Another structure called `PolyFit` is used to compute the coefficients of the fitting polynomial by least-square regression for a given data set. Also based on s and the current position of the car, the errors vector $\boldsymbol{\epsilon}$ is computed. The state of the car is stored in the structure `CarState` and updated each time the state of the car is measured by the sensors.

The parameters sequence and initial conditions are provided to the solver and the optimization problem is solved. A structure called `ControlCommand` is used to convert the first control commands δ and D of the computed sequence into throttle, brake, and steering inputs. The maximum steering of the car is 25° on the left and right, but the steering input must be sent between -1 and 1. δ being computed in radians, the conversion is,

$$steering = \frac{\delta \frac{180}{\pi}}{25}$$

For the throttle and brake, $(throttle, brake) = (D, 0)$ if $D \geq 0$ and $(throttle, brake) = (0, -D)$ if $D < 0$.

It is possible that the solver sometimes fails to computed the optimal control commands. To avoid applying bad control inputs to the car, the control commands computed by the solver are first checked before sending them to the car. If the three control commands rate of changes are between the lower and upper bounds (3.26) that have been defined, they are applied to the car. If not, the control commands previously computed are again applied to the car.

Path Following class and methods

The PF algorithm is implemented as a class called `PathFollowing` containing attributes and methods to perform the different tasks. This class is much lighter than the PP one because the major task is performed by the optimization solver.

The main attributes of the class are the `CarState`, `Mpcc` and `ControlCommand` structures, respectively used to store the measured state of the car, solve the MPCC problem and send the control inputs to the car. Additionally, the first method of the class, `polynomialsCallback()`, receives the PP algorithm's output at the same frequency as the SLAM algorithm's frequency f_{SLAM} and store it in the form of a `polynomials` structure. The car's position sent by the PP is updated in `CarState` at the same time. It also handles the reset of s at the starting line crossing at the start of a new lap.

A second method, `odometryCallback()`, receives the sensors data at a frequency f_{odo} and convert them into the desired car state. The velocity is received in the form of a vector $\boldsymbol{V} = (V_x, V_y)$ defined in the inertial frame, as represented in Figure 3.35. The velocity $\boldsymbol{v} = (v_x, v_y)$ defined in the frame relative to the car can be computed with,

$$\begin{cases} v_x = V_x \cos(\phi) + V_y \sin(\phi) \\ v_y = V_y \cos(\phi) - V_x \sin(\phi) \end{cases} \tag{3.35}$$

The speed of the car is then computed with,

$$v = \sqrt{v_x^2 + v_y^2},$$

which is equal to $V = \sqrt{V_x^2 + V_y^2}$.

The third method `commandPublisherTimer()` uses the structure `ControlCommand` to convert the control commands computed by the optimization solver into the car's control inputs and send them at a frequency $f_{commands}$.

Finally, the method `mpccTimer()` calls the optimization solver via the structure `Mpcc` at a frequency $f = \frac{1}{\Delta t}$.

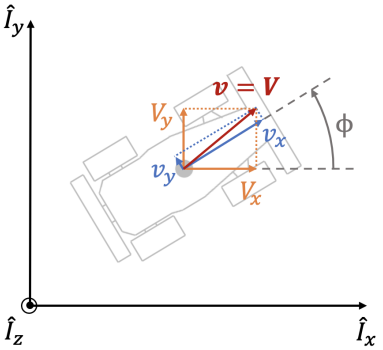

Figure 3.35: Velocities defined in the inertial frame (orange) versus velocities defined in the frame relative to the car (blue)

3.5 Tests and validation

The primary aim of this section is to validate the PF algorithm and determine its parameters and variables to obtain the best performance. This involves studying the influence of these factors on the algorithm's behavior. The initial analysis is conducted using a testbench, and subsequently applied and compared within the simulator environment. A benchmark is created to define criteria to respect in order to be considered valid.

3.5.1 Reference path validation

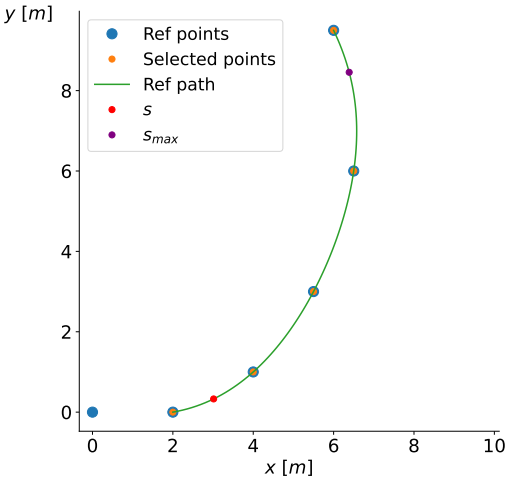

Figure 3.36: Reference path computation for the normal case (enough arc-length and enough points ahead of s with the available reference points)

To validate the reference path formulation, it is necessary to test different possible cases. Several factors should be considered when choosing the parameters m and n_{min}. First of all, the polynomials must be at least of order 3 to be able to fit complex shapes. The bigger the order, the better the fitting. However, since the polynomials are used inside the solver, the order should not be too large to reduce the computational complexity. To ensure safe operations of the solver, the maximum value $m = 4$ has been determined by testing. $n_{min} = m + 1 = 5$ points are thus necessary

To test the different cases, a list of reference points is generated, and different values for s and L_{min} are tested. The normal case occurs when there are enough points after the first selected to have enough arc-length on the reference path. It is illustrated in Figure 3.36, where $n_{select} = n_{min} = 5$ points have been selected. s and s_{max} can also be observed ($L_{min} = s_{max} - s$). In Figure 3.37, the case when there is not enough arc-length with the reference points ahead of s is illustrated. An additional point is created to have the minimal length and at the same time have 5 points. Finally, the case when there is enough arc-length but not enough points ahead of s is shown in Figure 3.38. In this case, instead of adding a point, a previous point is selected to have $n_{select} = n_{min}$.

These figures aim to illustrate the main cases that can occur but do not illustrate every possible situation. Basically, by adding points and/or selecting previous ones until the two conditions $L \geq L_{min}$ and $n_{select} \geq n_{min}$ are respected, every situation is handled. Of course, if necessary, more than n_{min} points can be selected.

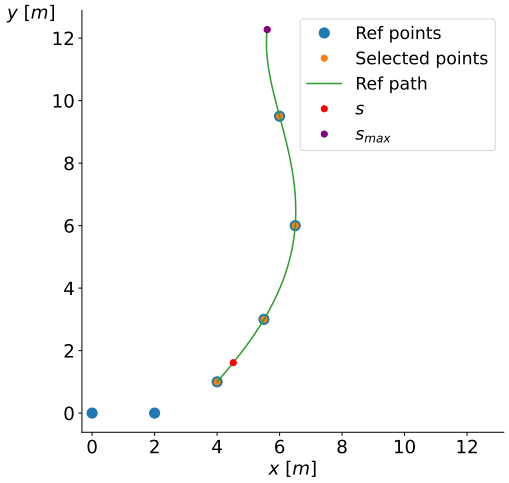

Figure 3.37: Reference path computation is the special case of not enough arc-length with the reference points available ahead of s

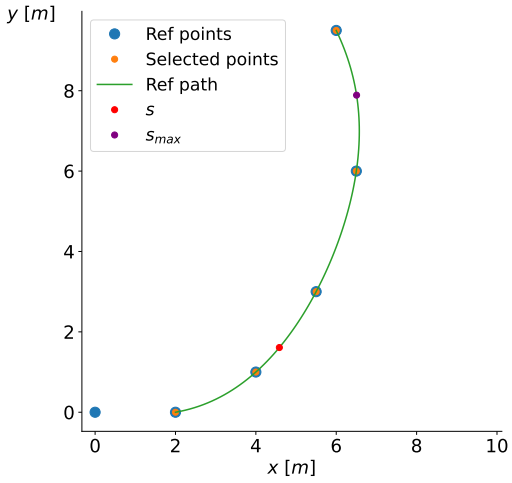

Figure 3.38: Reference path computation in the special case of enough arc-length but not enough points available ahead of s

3.5.2 Testbench and benchmark

Testbench

As for the PP algorithm, a testbench has been created to test and validate the algorithm independently of any external factor. It also has the objective of studying the influence of the different factors in a "perfect world". In this "perfect world", the model used for the optimization solver would be exactly the same as the real car. It enables as well a comparison with the performance on the simulator.

For these purposes, the testbench must provide the track as the form of reference points and a virtual car to compute its change of state. Instead of the cones, it is the limits that the car position must not cross in order to stay within the track that are indicated. Since the minimum distance between the center of the track and the cones on both sides is $R_{track} = 1.5\,m$, and the width of the car is $w = 1.13\,m$, the limits are positioned at a distance,

$$R_{limits} \leq 1.5 - \frac{1.13}{2}$$
$$\Leftrightarrow R_{limits} = 0.9\,m,$$

where a security margin of $3.5\,cm$ has been taken. An example of track generated with the testbench is shown in Figure 3.39.

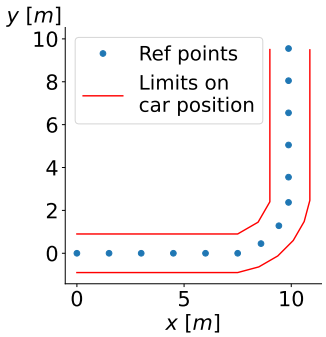

Figure 3.39: Example of track generated by the testbench

Then, to simulate the car's behavior, the same model used in the solver is implemented. This provides a non-real-time environment, enabling fast testing.

Benchmark

Main track characteristic	Specific track characteristic
1. Straight line	
2. Curve with large radius ($10\,m$) after reaching target speed in straight line	1. Curve angle of 45° 2. Curve angle of 180°
3. Curve with small radius ($4\,m$) after reaching target speed in straight line	1. Curve angle of 45° 2. Curve angle of 90° 3. Curve angle of 180°
4. Start with a horizontal offset in straight line ($0.5\,m$ to the left)	
5. Sequence of 2 curves of opposite directions (left then right) with small radius ($4\,m$) after reaching target speed in straight line	1. Curve angle of 90° 2. Curve angle of 180°

Table 3.1: MPCC algorithm's benchmark

To test and validate the algorithm, a benchmark is created. In order to be considered valid, the algorithm must succeed in all the tasks described in the benchmark. It also provides all the necessary tracks to perform the tests aiming at determining the performance parameters. The benchmark is represented in Table 3.1.

3.5.3 Tests and validation with testbench

The objective of the tests and validation with the testbench is to find the best performance parameters by studying their influence. The performance parameters for this MPCC problem formulation are the following:

1. Time steps Δt and Δt_{MPCC};
2. Prediction horizon N;
3. Control commands (rate) constraints $\underline{\boldsymbol{u}}_k$, $\overline{\boldsymbol{u}}_k$, $\underline{\boldsymbol{\Delta u}}_k$, $\overline{\boldsymbol{\Delta u}}_k$;
4. Weights $\boldsymbol{j}_k = [q_{l,k}\ q_{c,k}\ , q_{v,k}\ r_{\delta,k}\ r_{D,k}\ r_{v_s,k}\ r_{\Delta\delta,k}\ r_{\Delta D,k}\ r_{\Delta v_s,k}]^T$.

The time step Δt determines the frequency of the control commands refreshing. It is limited by the frequency at which the state measures are received. Since the position of the car is updated every 25 to 40 ms ($25 \leq f_{SLAM} \leq 40\,Hz$), the minimum time step is $\Delta t = 50\,ms$ to have a margin of error and be sure that the position has been updated.

The prediction horizon N has an influence on the complexity of the MPCC problem and therefore the computational time, and together with Δt_{MPCC}, it has an influence on the finite time horizon $T = N\Delta t_{\text{MPCC}}$. For this first solution, N is limited to 20 because of the solver, its influence will therefore not be studied.

The main performance parameters are the weights as they directly describe what behavior the car must have in order to meet the objectives. The constraints have also a role to play since they act on the control commands used to control the car.

To start the study of the different parameters, a starting point must be defined. Each parameter must be assigned an initial value used as reference value around which it will vary. The first tests will be conducted at low speed, $v_{s,max} = 3\,m/s$ is thus defined. $\Delta t_{\text{MPCC}} = 75\,ms$ is then chosen to have a time horizon $T = 1.5\,s$ which corresponds to maximum view ahead of $Tv_{s,max} = 4.5\,m$. Δt is chosen equal to Δt_{MPCC}, as it offers a descent refreshing frequency for this type of application (in the order of 100th of a millisecond).

For the initial guess of the constraints, reasonable values in the physical sens are chosen. First of all, the steering angle of the car can vary between -25 and $25°$, therefore,

$$\underline{\delta}_k = -25\frac{\pi}{180} \leq \delta_k \leq 25\frac{\pi}{180} = \overline{\delta}_k \tag{3.36}$$

Then, D is limited to restrain the acceleration of the car,

$$\underline{D}_k = -0.2 \leq D_k \leq 0.2 = \overline{D}_k, \tag{3.37}$$

and v_s must be between 0 and the target speed,

$$\underline{v}_{s,k} = 0 \leq v_{s,k} \leq v_{s,max} = \overline{v}_{s,k} \tag{3.38}$$

For the rates, the car must take at least $1\,s$ to turn the wheel from 0 to full steering,

$$\underline{\Delta\delta}_k = -25\frac{\pi}{180}\Delta t_{\text{MPCC}} \leq \Delta\delta_k \leq 25\frac{\pi}{180}\Delta t_{\text{MPCC}} = \overline{\Delta\delta}_k \tag{3.39}$$

Same approach for v_s,

$$\underline{\Delta v}_{s,k} = -v_{s,max}\Delta t_{\text{MPCC}} \leq \Delta v_{s,k} \leq v_{s,max}\Delta t_{\text{MPCC}} = \overline{\Delta v}_{s,k} \tag{3.40}$$

and the rate of D is also strongly limited as the slow target speed requires only very small D,

$$\underline{\Delta D}_k = -0.05\Delta t_{\text{MPCC}} \leq \Delta D_k \leq 0.05\Delta t_{\text{MPCC}} = \overline{\Delta D}_k, \tag{3.41}$$

For the track constraint (3.27), R_c is chosen equal to $R_{limits} = 0.9\,m$.

Finally, the initial values chosen for the weights are,

$$\boldsymbol{j}_k = [1\ 1\ 1\ 1\ 1\ 0\ 1\ 1\ 1]^T \tag{3.42}$$

where $r_{v_s} = 0$ because there is already a weight applied on v_s, q_v. The first parameters to be studied are the weights. For this purpose, a procedure using the benchmark is followed.

a. With track 1, the weights influencing the speed and acceleration of the car can be studied without being impacted by the parameters linked to the direction of the car. The weights that have the bigger impact are q_v and q_l, with q_v describing the importance of reaching the target virtual speed and q_l the importance of staying close to the virtual advancement of the car (which is done by having a real speed v close to v_s).

The influence on the speed is examined in Figure 3.40. In the top graph, only the weight q_v is varied while keeping q_l constant at 1. Increasing q_v does not have a significant impact, as a large overshoot followed by an undershoot is still observed. Next, in the middle and bottom graphs, q_l is varied while keeping q_v fixed at 10 and 1, respectively. When q_l is approximately 10 times larger than q_v in the bottom graph, the overshoot and undershoot are greatly reduced, resulting in faster convergence to a steady speed. In Figure 3.41, a comparison is made between the lag error $\hat{\epsilon}_l$ and the control commands D and v_s for weight combinations of $(q_l, q_v) = (20, 10)$ and $(20, 1)$. In addition to achieving a better speed profile, the weight combination of $(20, 1)$ significantly reduces the lag error and oscillations in the control commands D. The values,

$$\begin{cases} q_l = 20 \\ q_v = 1 \end{cases} \tag{3.43}$$

are therefore kept for the next analysis.

Next, the weights r_D, $r_{\Delta D}$ and $r_{\Delta v_s}$ are varied for values between 0.1 and 100. No real impacts have been observed and there are thus kept to 1. After this first analysis, the weights vector is therefore,

$$\boldsymbol{j}_k^a = [20\ 1\ 1\ 1\ 1\ 0\ 1\ 1\ 1]^T \tag{3.44}$$

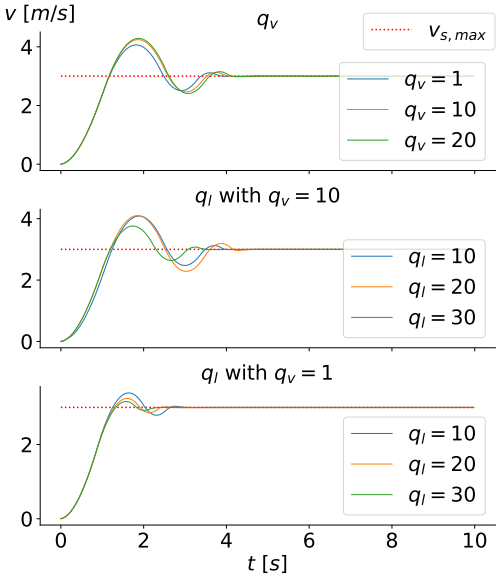

Figure 3.40: Influence of weights q_v and q_l on the speed on track 1

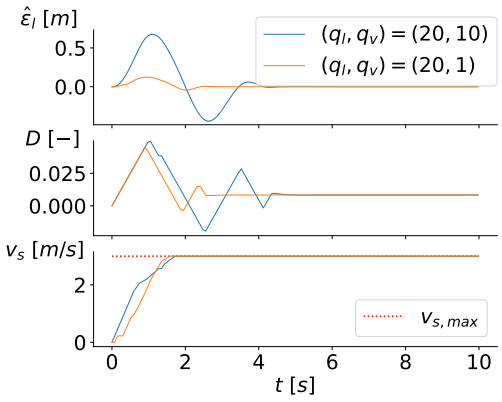

Figure 3.41: Influence of weights q_v and q_l on the lag error $\hat{\epsilon}_l$, control commands D and v_s on track 1

b. The influence of the three remaining weights q_c, r_δ and $r_{\Delta\delta}$ can now be studied.

Increasing q_c results in predictions, and therefore trajectories closer to the path. r_δ, on the other hand, acts on the steering by forcing it to be smaller as r_δ increases. The result is a better anticipation of the curve, as shown in Figure 3.42, but also a bigger deviation from the path at the end of the turn.

To obtain a better result, the effects of q_c and r_δ can be used together. The ideal behavior would be to have a better anticipation (take the racing line - interior of the curve) while keeping the prediction close to the path at the end to avoid big deviations. To have a better anticipation, r_δ must be on the order of 10 times big-

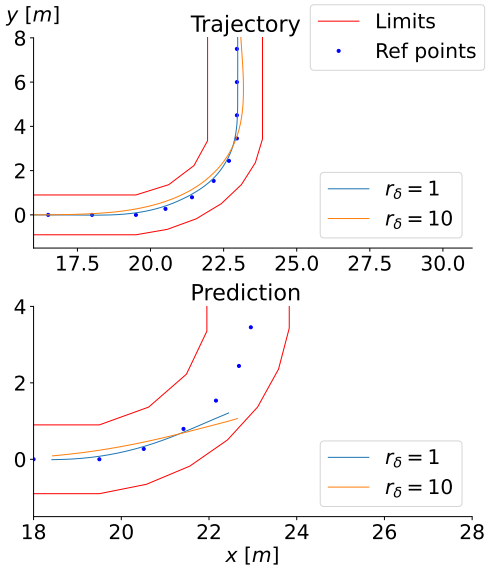

Figure 3.42: Influence of weight r_δ on the trajectory and prediction at the start of the curve on track 3.2 ($q_c = 1$)

ger than q_c. But to limit the deviation, q_c would need to be bigger than r_δ, at least at the end of the prediction. Based on this idea, a variable weight is introduced. The value of q_c will vary depending on the prediction k, where the values must be small at the start and bigger at the end to obtain the desired effects. The function chosen to implement this variation is,

$$q_c = f_{q_c}(k) = \begin{cases} q_{ci} & \text{if } k \leq N_i \\ q_{c,k-1} + \frac{q_{cf}}{N - N_i} & \text{if } k > N_i \end{cases} \qquad \text{for } k = 1, 2, ..., N \tag{3.45}$$

The profile is shown in Figure 3.43, where the values $q_{ci} = 1, q_{cf} = 50, N_i = \frac{3}{4}N$ have been chosen after testing several variations. Then the influence of r_δ is again studied with this new variable q_c. The trajectories and predictions on track 3.2 are shown in Figure 3.44. The desired effects are obtained. To confirm the positive effects of this method, it is tested on the double curve of track 5.1, with the results shown in Figure 3.45. The effects of the better anticipation are even more noticeable in this situation, where it allows to take a racing line.

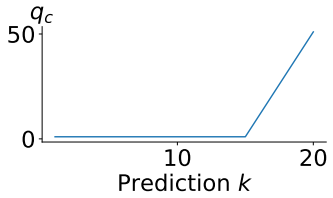

Figure 3.43: Variable profile of q_c (3.45) ($q_{ci} = 1, q_{cf} = 50, N_i = \frac{3}{4}N$)

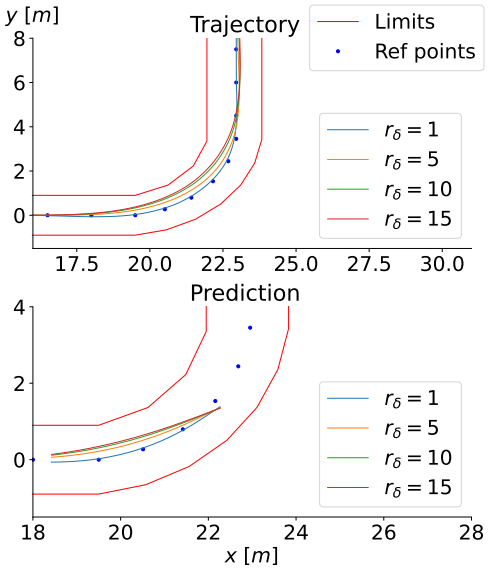

Figure 3.44: Influence of weight r_δ on the trajectory and prediction at the start of the curve on track 3.2 ($q_c = f_{q_c}(k)$)

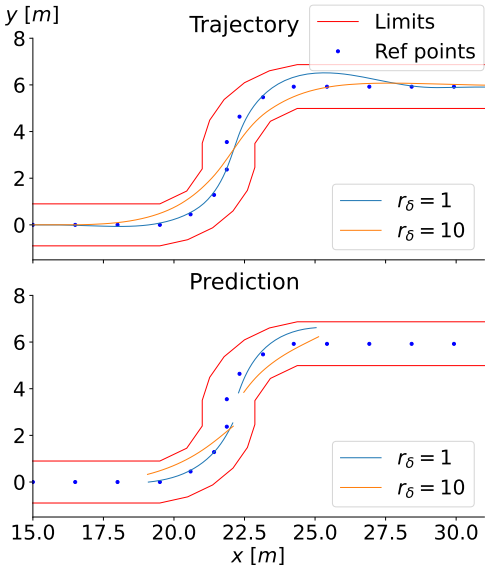

Figure 3.45: Influence of weight r_δ on the trajectory and predictions at the start of the curves on track 5.1 ($q_c = f_{q_c}(k)$)

The value $r_\delta = 10$ is therefore kept, with $q_c = f_{q_c}(k)$. The remaining tracks of the

benchmark have also been tested and approved. Then the influence of $r_{\Delta\delta}$ has been studied but no real impacts have been observed, and $r_{\Delta\delta} = 1$ has therefore been kept. The weights vector at the end of this second analysis, which complete the analysis at low speed $v_{s,max} = 3\,m/s$, is,

$$\boldsymbol{j}_k^b = [20\ f_{q_c}(k)\ 1\ 10\ 1\ 0\ 1\ 1\ 1]^T \tag{3.46}$$

The average computational time for the full analysis is,

$$t_{solve,avg} = 2.58\,ms \tag{3.47}$$

c. A first study has been made at low speed, now the speed can be increased to see if the found parameters are still efficient. The target speed is doubled, $v_{s,max} = 6\,m/s$ and the tests are again performed on the benchmark tracks.

The speed profile is still smooth but the car is not able to take the sharp turns of track 5.1 and 5.2 at this increased speed. This is caused by a too strict constraints on the steering command rate $\Delta\delta$, which is therefore relaxed,

$$\begin{cases} \underline{\Delta\delta}_k = -2 \times 25\frac{\pi}{180}\Delta t_{\text{MPCC}} \\ \overline{\Delta\delta}_k = 2 \times 25\frac{\pi}{180}\Delta t_{\text{MPCC}} \end{cases} \tag{3.48}$$

to obtain better results, as shown in Figure 3.46 and 3.47. δ is smoother and the trajectory better as a result.

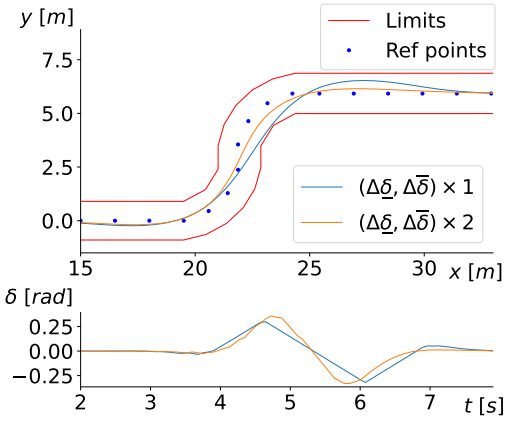

Figure 3.46: Comparison of the trajectories and δ profiles on track 5.1 obtained with the old and new constraints on $\Delta\delta$

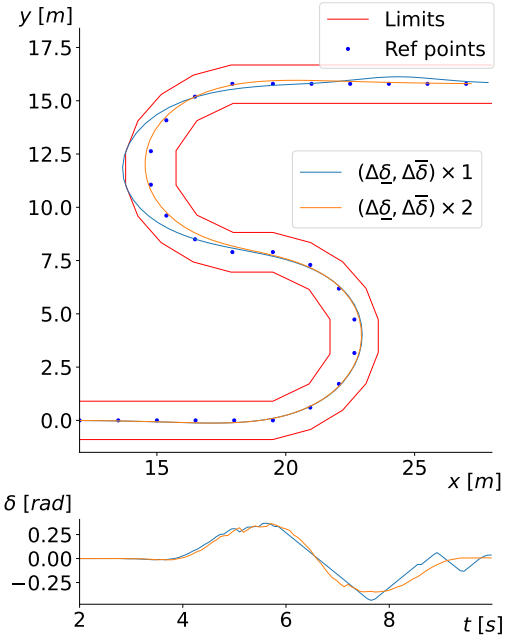

Figure 3.47: Comparison of the trajectories and δ profiles on track 5.2 obtained with the old and new constraints on $\Delta\delta$

d. The target speed is again increased to $9\,m/s$ and $12\,m/s$. For these higher speeds, the constraints on ΔD become too strict, resulting in a too slow increase of speed. To overcome this situation, the constraints are greatly relaxed to,

$$\begin{cases} \underline{\Delta D}_k = -0.025 \\ \overline{\Delta D}_k = 0.025 \end{cases} \tag{3.49}$$

Figure 3.48 shows the comparison of speeds and command D with the old and new constraints.

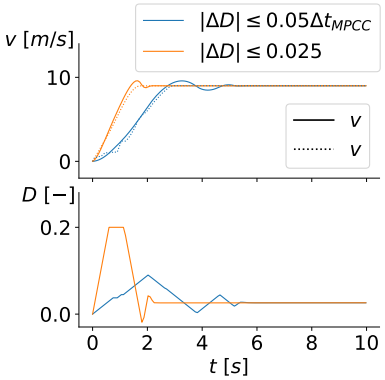

Figure 3.48: Comparison of the speeds and control command D at $v_{s,max} = 9\,m/s$ on track 1 for the old and new constraints on ΔD

Then the different tracks are tested at these higher speeds. The trajectories and speed profiles on track 3.2 and 5.1 are shown in Figure 3.49 and 3.50 for the target speeds $v_{s,max} = \{6, 9, 12\}\,m/s$. Since no slip is considered, the car is able to take almost the same trajectories by keeping its speed close to the target speed. The small difference in the trajectories are mostly due to the reference path. With the increase of the target speed, the maximum view ahead during the predictions is also increased, $Tv_{s,max} = \{9, 13.5, 18\}\,m$.

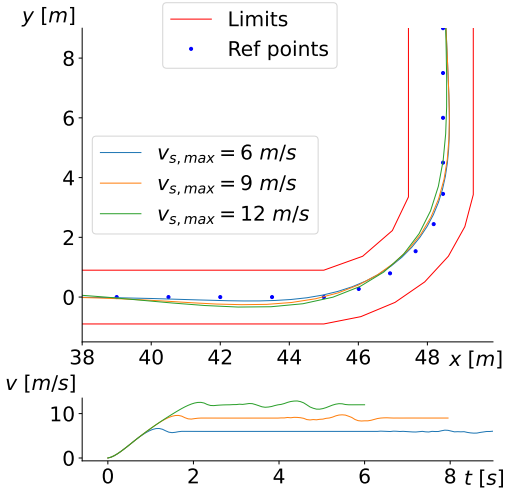

Figure 3.49: Comparison of the trajectories and speed profiles on track 3.2 for $v_{s,max} = \{6, 9, 12\}\,m/s$

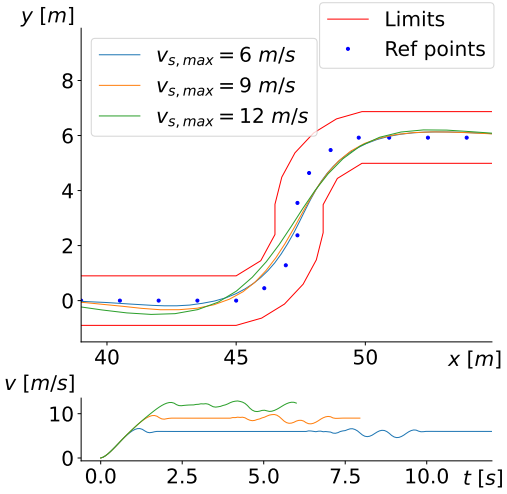

Figure 3.50: Comparison of the trajectories and speed profiles on track 5.1 for $v_{s,max} = \{6, 9, 12\}\,m/s$

The number of reference points to fit for the polynomials also increases, leading in differences for the reference path. It can be observed in Figure 3.51 that the computed reference path has more difficulties fitting the reference points as the speed increases. While it allows to have a reference path more close to a racing line, it also means that the track constraints (3.27) are not valid everywhere anymore. Since the contouring error $\hat{\epsilon}_c$ is computed based on the reference path, if the latter is not in

the center of the track the car could cross the track limits even if the constraints are technically respected.

The only way to improve the fitting in every case is to increase the order m of the polynomials, which cannot be done because of the complexity of the formulation.

This concludes the analysis with the testbench. The influence of the weights and constraints have been understood in the case of a model perfectly describing the behavior of the car. Now the algorithm must be tested on the simulator to validate the algorithm in the target environment and point the limitations of this first solution in order to develop an improved formulation.

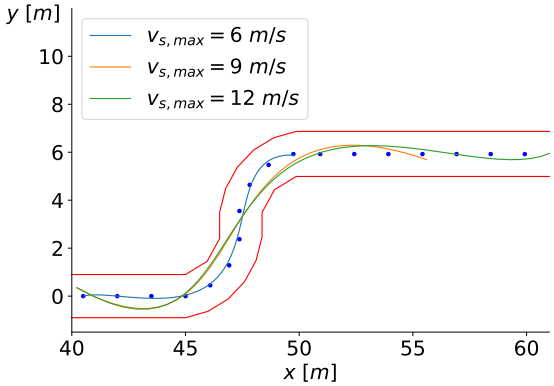

Figure 3.51: Comparison of the reference path computed at a certain time for $v_{s,max} = \{6, 9, 12\}\, m/s$

3.5.4 Tests and validation on Formula Student Driverless Simulator

The algorithm is first tested at low speed $v_{s,max} = 3\, m/s$ with the basic configuration of unit weights on track 1 to observe the difference with the testbench in terms of speed profile, control command D and lag error. The result is shown in Figure 3.52, where the constraints on v_s rate has been increase ((3.40)×10) to have a quicker speed response. With the approximations applied on the drivetrain model, the model does not perfectly describe the behavior of the car drivetrain. The solver does not manage to stabilize the command D in order to have a null lag error and constant speed. It oscillates because the variations on D are too abrupt. Increasing q_l reinforces this effect.

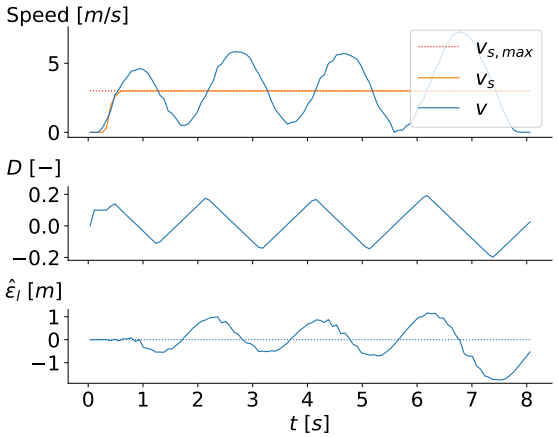

Figure 3.52: Speed, D and lag error profiles on track 1 at target speed $v_{s,max} = 3\, m/s$ with weights vector $\boldsymbol{j}_k = [1\ 1\ 1\ 1\ 1\ 0\ 1\ 1\ 1]^T$

To reduce the rate of change on D, the weight $r_{\Delta D}$ is increased. To have a real improvement, the increase must be important and the increase of weight r_D further improves the behavior of the car. The results are shown in Figure 3.53, where the values $r_{\Delta D} = 50$ and 100 are compared on the top graph, then the values $r_D = 50$ and 100 on the bottom graph, with $r_{\Delta D} = 100$ that has been kept. With the values $r_D = r_{\Delta D} = 100$, the speed is almost kept constant with an average lag error of $\hat{\epsilon}_l = 0.1\,m$. Compared to the analysis with the testbench, the weights r_D and $r_{\Delta D}$ play a crucial role to overcome the problems caused by the non exactitude of the model. The weight vector after this first analysis is,

$$\boldsymbol{j}_k = [1\ 1\ 1\ 1\ 100\ 0\ 1\ 100\ 1]^T$$

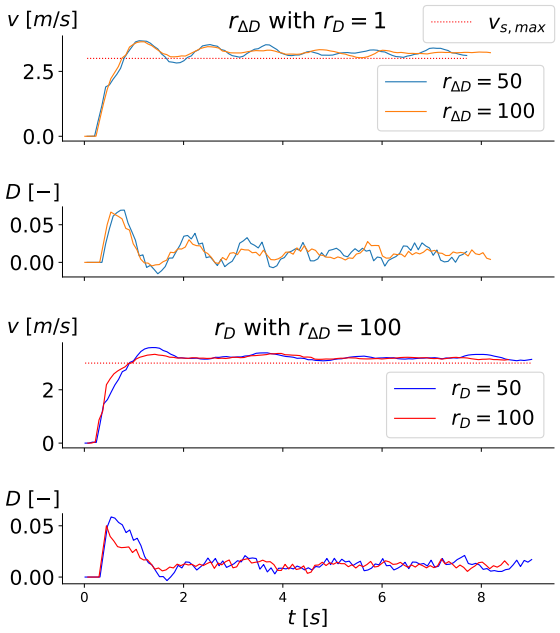

Figure 3.53: Comparison of speed and D profiles for different combinations of weights $r_{\Delta D}$ and r_D

Next, the behavior in curves is analyzed. This time, the influence of the weights q_c and r_δ is similar compared to the testbench. By defining a variable $q_c = f_{q_c}(k)$ (3.45) and changing r_δ, racing lines can be obtained on the different tracks, as shown in Figure 3.54. The values $r_\delta = 10$ offers the best results.

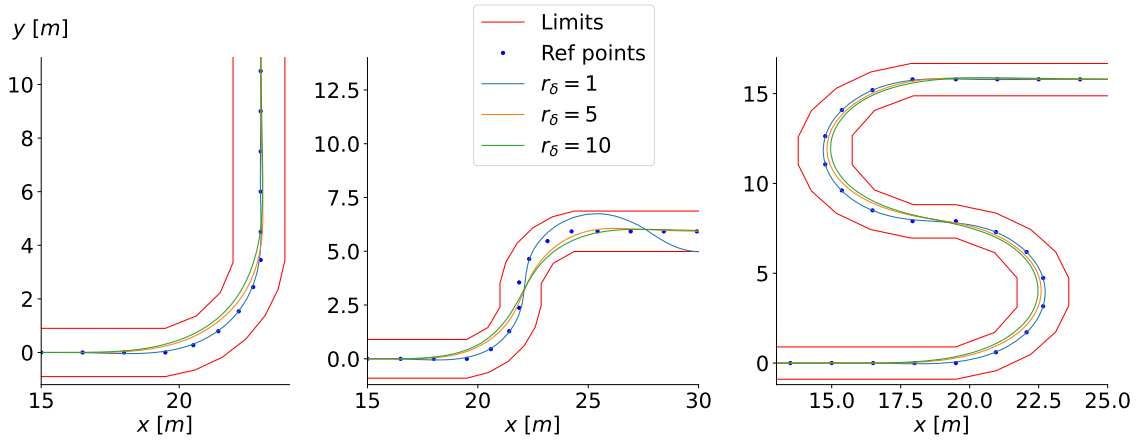

Figure 3.54: Influence of weight r_δ on the trajectory with $q_c = f_{q_c}(k)$ (3.45)

When the speed is increased to $v_{s,max} = 6\,m/s$, the car is able to complete the simpler tracks but not the ones with sharper turns. Because the kinematic model considers no slip, the behavior of the car further deviates from the model as the speed

increases. The algorithm therefore predicts trajectories that cannot be followed by the car because of the slip. As the car moves into the curve, the algorithm predicts that it can take it without reducing its speed. But the more the car advances in the curve, the more it deviates from the predictions until it is too late to regain control of the car. This too important difference between the model and the car cannot be compensated by changing the performance parameters, since whatever their values, the algorithm will never reduce the speed before its too late.

3.6 Limitations

The test and validation with the testbench and on the simulator have highlighted certain limitations of this first formulation. First of all, because of the need of the reference path inside the optimization problem and the complexity of the formulation, the order of the polynomials describing the reference path is limited. As a consequence, the quality of the fitting with the center of the track can be reduced in certain cases at higher speed, which could lead to exceeding the track limits. Secondly, the solver is not able to stabilize the throttle command in order to reach a steady speed, which results in an unstable behavior even in straight line. The performance of the solver is also limited, restricting the analysis and the choice of performance parameters.
The principal limitations come from the model of the car, especially the assumption of no slip which is of great importance at higher speed. In order to control the car at higher speed, the slip must be modeled.
It is with the aim of overcoming these limitations than an improved PF algorithm is developed.

4 | Improved Path Following algorithm

In this chapter, the first developed PF algorithm is improved to achieve the final solution of this work. In view of the first solution limitations mentioned in Section 3.6, the vehicle model is first upgraded to a dynamic model to integrate the modeling of the slip. Then the new optimization solver is briefly presented and a new MPCC problem formulation is designed to overcome the limitations of reference path and unstable speed profile. Finally, the new algorithm is tested and validated with the testbench and on the simulator.

4.1 Vehicle dynamic model

4.1.1 Design

The principal limitations of the first solution come from the vehicle model used to compute the predictive control commands. More precisely, the hypothesis of no slip is too strong, leading in bad prediction and therefore bad control of the car especially at higher speed. The main objective of this new vehicle model is thus to introduce tire slip.

For this purpose, the bicycle model is also used, but is improved by developing a dynamic model instead of a kinematic model [61]. The forces and torques are thus used to express the evolution of the car. The model, with all the involved quantities, is represented in Figure 4.1.

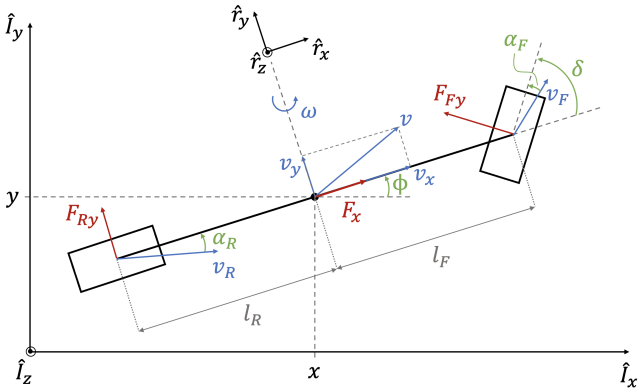

Figure 4.1: Dynamic bicycle model representation

The model fundamentals are kept from the kinematic bicycle model, with its mass being considered punctual at the center of mass and the distance of the front and rear wheels taken from the center of mass. $\hat{\boldsymbol{I}}$ is the inertial frame and $\hat{\boldsymbol{r}}$ the frame relative to the car. Are added to the formulation:

- The forces acting on the car. F_x is the longitudinal force acting in the relative x direction of the car, F_{Fy} and F_{Ry} are the lateral tire forces of the front and rear wheel respectively representing the force applied in the relative y direction of the car by the tire on the ground;
- The distinction of the velocities in the x and y directions of the center of mass of the car, v_x and v_y, instead of modeling the direction of the velocity v with the slip angle β;
- The rotational velocity ω of the center of mass of the car (yaw rate);
- The consideration of tire slip with the introduction of the angles between the velocity of the front wheel and its orientation, α_F, and the rear wheel and its orientation, α_R.

The car state is described by the vector $\boldsymbol{x} = [x\ y\ \phi\ v_x\ v_y\ \omega]^T$ and its model is (see Appendix E for details about the computation),

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\phi} \\ \dot{v}_x \\ \dot{v}_y \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} v_x \cos(\phi) - v_y \sin(\phi) \\ v_x \sin(\phi) + v_y \cos(\phi) \\ \omega \\ \frac{1}{m}(F_x - F_{F,y} \sin(\delta) + m v_y \omega) \\ \frac{1}{m}(F_{R,y} + F_{F,y} \cos(\delta) - m v_x \omega) \\ \frac{1}{I_z}(F_{F,y} l_F \cos(\delta) - F_{R,y} l_R) \end{bmatrix} \tag{4.1}$$

where I_z is the inertia at the center of mass and F_x, $F_{F,y}$ and $F_{R,y}$ must still be expressed in terms of the state quantities. By making the assumption that the drivetrain force acts on the center of mass, the longitudinal force F_x can be expressed by the reformulated drivetrain model,

$$F_x = \underbrace{C_{m1} \arctan(C_{m2} D)}_{f_m(D)} - \underbrace{C_d v_x}_{f_d(v_x)}, \tag{4.2}$$

where v_x is used instead of v.

For the lateral tire forces, a model for the tire must be developed. The lateral force applied by a tire is the result of the interaction between the tire and the ground. The force is therefore impacted by slip, which is modeled by the slip angles α_F and α_R. A slip angle different from 0 means that the velocity of the wheel is not aligned with its orientation, describing the fact that slip is occurring. When the car is moving forward, there is no steering applied and the tire do not create lateral forces. Then, when steering is applied, the tire apply the lateral forces necessary to make the car turn. Slip during the turn results in a reduced lateral force applied by the tire, making the car turn less efficiently. A model for the lateral forces, depending on the slip angles, must therefore be developed.

First of all, the slip angles are expressed in terms of the state quantities and control command of the car. The schemes used to find their expressions are represented in Figure 4.2. The velocities at the front and rear wheel v_F and v_R are formed by the x and y components of the velocity of each wheel. For the front wheel, the component $l_F\omega$ has the same direction as v_y, while for the rear wheel this is the opposite. The computed velocity forms an angle, ξ_F or ξ_R, with the absolute x axis that can be expressed in terms of the other quantities using the right-angled triangle rule,

$$\xi_F = \arctan\left(\frac{v_y + l_F\omega}{v_x}\right) \tag{4.3}$$

$$\xi_R = \arctan\left(\frac{v_y - l_R\omega}{v_x}\right) \tag{4.4}$$

The slip angle is the subtraction of the wheel orientation δ_F or δ_R with the ξ_F or ξ_R angles, with $\delta_F = \delta$ the car's steering and $\delta_R = 0$,

$$\begin{aligned}\alpha_F &= \delta_F - \xi_F \\ &= \delta - \arctan\left(\frac{v_y + l_F\omega}{v_x}\right)\end{aligned} \tag{4.5}$$

$$\begin{aligned}\alpha_R &= \delta_R - \xi_R \\ &= -\arctan\left(\frac{v_y - l_R\omega}{v_x}\right)\end{aligned} \tag{4.6}$$

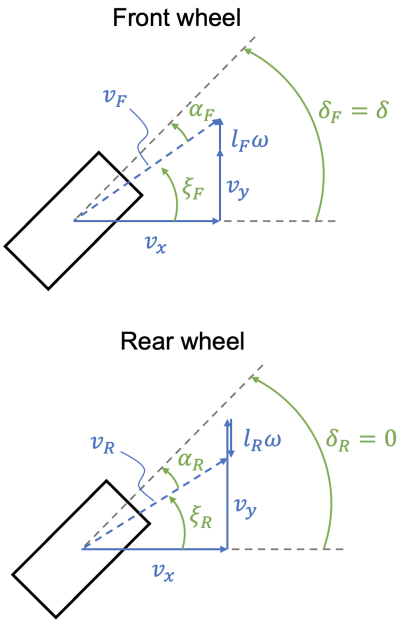

Figure 4.2: Schemes for the expression of the slip angles α_F and α_R

Then, to express the lateral forces in terms of slip angles, the Pacejka tire model [62] is used. The Pacejka tire model, named after its creator Hans B. Pacejka, is a widely used empirical tire model that describes the forces and moments generated by a tire under various operating conditions. It is used in vehicle dynamics simulations and plays a crucial role in understanding and predicting the behavior of tires. The model is based on experimental data and aims to capture the complex and nonlinear behavior of tires. It provides a mathematical representation of the tire's lateral force, longitudinal force, self-aligning torque, and cornering stiffness, among other parameters. It is a very complex model that must be simplified in order to be used in this application. The expression of the simplified Pacejka model for the lateral force is [63],

$$F_y = D\sin(C\arctan(B\alpha - E(B\alpha - \arctan(B\alpha)))) \tag{4.7}$$

This expression is further simplified by assuming that the slip angle is small, resulting in $\arctan(B\alpha) \approx B\alpha$ and therefore,

$$F_y = D\sin(C\arctan(B\alpha)) \tag{4.8}$$

The characterization and validation of this model is necessary to find the three coefficients, but typical values based on empirical tire data are known and are shown in Table 4.1. The coefficient D is decomposed in $D = F_z d$ where F_z is the vertical force (weight) acting on the tire.

Coefficient	Name	Typical range	Dry tarmac	Wet tarmac
d ($D = F_z d$)	Peak	0.1 .. 1.9	1	0.82
C	Shape	1 .. 2	1.9	2.3
B	Stiffness	4 .. 12	10	12

Table 4.1: Pacejka tire model typical values [62]

These typical values will be used as basis for the characterization and validation of the tire model. In addition to the coefficients, the weight F_z acting on each wheel (F_{Fz} and F_{Rz}) must be determined. The mass of the car is considered punctual at its center of mass. By making the assumption that the mass is equally distributed in the car's body, the weight on each wheel becomes,

$$F_{Fz} = \underbrace{\frac{l_R}{l_F + l_R} m}_{m_F} g, \tag{4.9}$$

$$F_{Rz} = \underbrace{\frac{l_F}{l_F + l_R} m}_{m_R} g, \tag{4.10}$$

where $g = 9.81\, m/s^2$ is the gravitational acceleration. The fractions of the length divide the total mass m and distribute it on the front and rear wheel. Since the front wheel is closer to the center of mass than the rear wheel, the front wheel supports more weight than the rear wheel.

The simplified model for the front and rear lateral tire forces is therefore,

$$\begin{cases} F_{Fy} = m_F g \cdot d \sin(C \arctan(B\alpha_F)) \\ F_{Ry} = m_R g \cdot d \sin(C \arctan(B\alpha_R)) \end{cases} \tag{4.11}$$

and is represented in Figure 4.3 for a dry and wet tarmac. For a wet tarmac, the slip is more important and as discussed, the lateral forces produced by the tire are smaller.

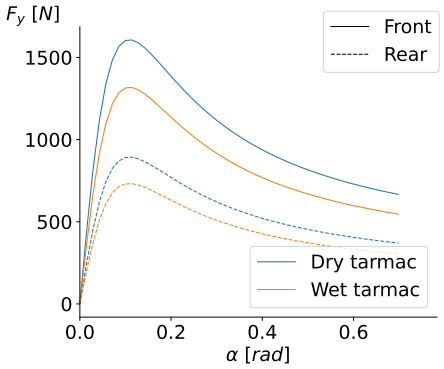

Figure 4.3: Simplified Pacejka model (4.11) of the front and rear wheel for dry and wet tarmac typical coefficients

The last value to determine is the inertia I_z. The shape of the car is approximated as a parallelepiped rectangle of width w, length l and height h. These dimensions are known

and the inertia can be expressed by the formula of the inertia of a parallelepiped rectangle of mass m,

$$I_z = \frac{m}{12}(l^2 + w^2), \tag{4.12}$$

with $l = 2.26\,m$ and $w = 1.13\,m$.

This concludes the design of the new dynamic model. The state of the car has changed compared to the previous kinematic model but the control command vector has not changed. The model must now be characterized and validated to find the missing coefficients.

4.1.2 Characterization and validation

The drivetrain model does not need to be characterized and validated again as the kinematic model (3.5) and dynamic model (4.1) are identical to the models used to characterize the drivetrain model in Subsection 3.3.2. Indeed, in straight line, $\delta = \omega = v_y = \phi = y = 0$ and the dynamic model becomes,

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\phi} \\ \dot{v}_x \\ \dot{v}_y \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} v_x \\ 0 \\ 0 \\ \frac{1}{m}F_x \\ 0 \\ 0 \end{bmatrix} \tag{4.13}$$

with $v_x = v$. The measured speed and acceleration during the characterization of the drivetrain model are thus identical for both kinematic and dynamic model. The value of the drivetrain model coefficients are kept the same,

$$\begin{cases} C_{m1} & = 1785 \\ C_{m2} & = 15 \\ C_b & = 4080 \\ C_d & = 74.01 \end{cases}$$

Now, to characterize and validate the tire model and thus the full model, the same approach used in Subsection 3.3.2 is used. The curves measured and computed with the model when the car is taking a turn are compared for different speeds. The results obtained with the dynamic model with dry and wet tarmac coefficients are compared to the measures and the kinematic model in Figure 4.4. At lower speeds, the results obtained with the different models are close from each other. There is no significant difference between the dynamic model using dry and wet tarmac coefficients. However, the curve taken is slightly wider compared to the kinematic model. At higher speeds, the dynamical model with dry tarmac coefficients fits the measurements well, whereas the wet coefficients result in a wider curve. Based on this information, it can be concluded that the dynamic model with dry tarmac coefficients performs very well at high speeds, but underestimates slip at lower speeds.

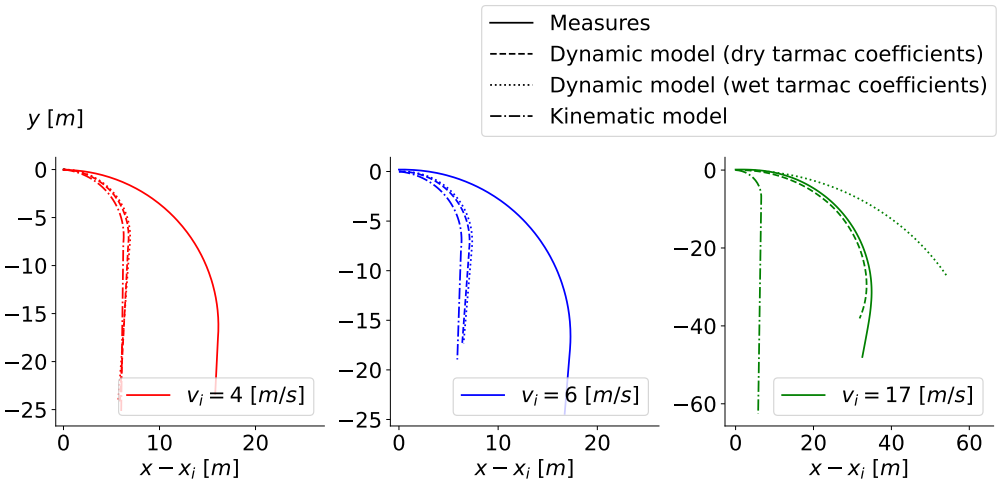

Figure 4.4: Validation of the tire model in a curve taken at different speeds but same steering angle (10°)

The dynamic model, therefore, exhibits a more realistic behavior compared to the kinematic model. As the speed increases, the accuracy of the dynamic model improves. With this improved model, the MPCC algorithm is expected to achieve higher performance and enable an increase in the target speed. The dry tarmac coefficients are kept as reference values,

$$\begin{cases} d & = 1 \\ C & = 1.9 \\ B & = 10 \end{cases} \tag{4.14}$$

The algorithm was tested with the new optimization solver and MPCC problem formulation described in the subsequent sections. As a result, it was concluded that the model is computationally complex primarily due to the tire model (4.11). The solver was unable to solve the problem, therefore it was necessary to simplify the tire model.

Tire model simplification

The more complex part of the car model is the tire model, as the expression of the lateral forces (4.11), although already simplified compared to the complete Pacejka model, are complex. Remembering that α_F (4.5) and α_R (4.6) also involve an arctan function, the equations are highly nonlinear. The curves were represented in Figure 4.1 for angles α between 0 and 40°. These full curves are hard to approximate with a simple function. The first step is therefore to determine the range of α_F and α_R than the model needs to handle in order to approximate the useful part of the curve.

Height experiments are performed to collect different data and have a complete analysis. For all theses experiments, the car is brought to a certain speed, then the throttle is cancelled and a certain steering angle is applied. To be able to compute the α angles, the velocities v_x and v_y, the rotational velocity ω and the steering command δ must be measured. These data were collected at both low and high

speeds, encompassing the entire range of steering angles. This was accomplished by conducting the following experiments:

- Experiments 1 to 4: speed of $5\,m/s$ and steering commands applied of $\{5\ 10\ 15\ 20\}°$;
- Experiments 5 to 8: speed of $15\,m/s$ and steering commands applied of $\{5\ 10\ 15\ 20\}°$.

The α angles are computed with the data of each experiment and the maximum angle computed are shown in Figure 4.5. For the front tire, the speed does not have a strong impact since the curves are almost identical for experiments 1 to 4 and 5 to 8. The factor that impacts α_F is the steering angle. For the rear tire, α_R increases with both speed and steering angle. The measured angles are around 10 times greater for the front tire. All experience combined, the maximum α angles are,

$$\begin{cases} \alpha_{F,max} = 0.300\,rad \\ \alpha_{R,max} = 0.036\,rad \end{cases} \tag{4.15}$$

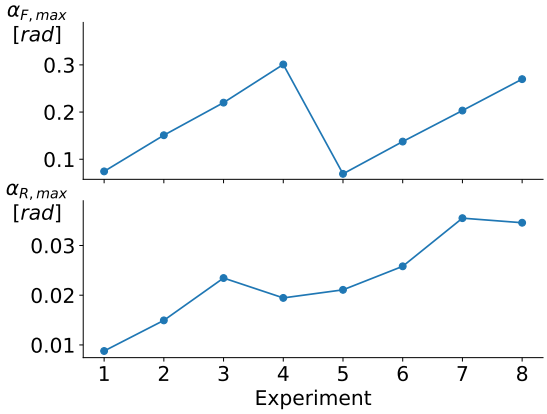

Figure 4.5: Maximum α angles computed with the data of the 8 experiments

The same experiments have been conducted with the model to verify the range of α computed and similar results have been observed. For the approximation, the range of α_F considered will be $[-0.3, 0.3]$ and the range of α_R will be $[-0.04, 0.04]$. The full tire model (4.11) in the defined range of α is compared to the curve computed by linear fitting in Figure 4.6. The linear tire model obtained is,

$$\begin{cases} F_{Fy} = 6522.30\alpha_F = C_F\alpha_F \\ F_{Ry} = 15552.60\alpha_R = C_R\alpha_R \end{cases} \tag{4.16}$$

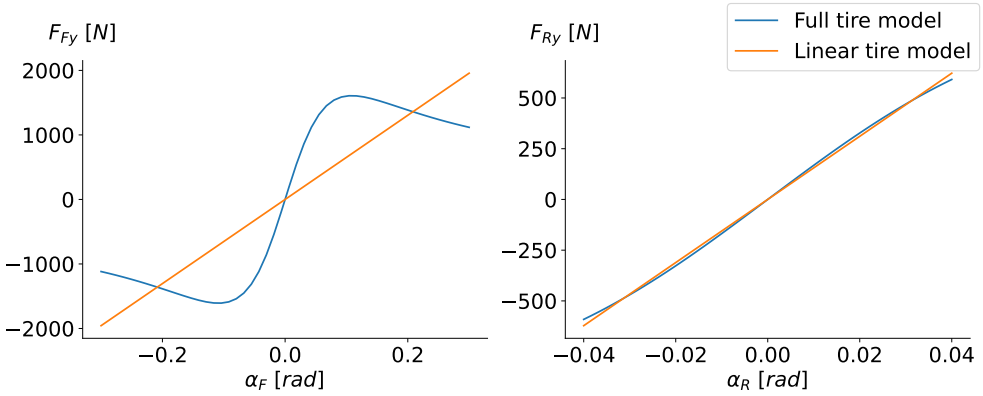

Figure 4.6: Comparison between full tire model (4.11) and linear tire model (4.16)

The small range of α_R allows a very good fitting by the linear model for the rear tire. For the front tire, because of the larger range of α_F, the fitting is less accurate. To validate the performance of the linear tire model, the bahavior of the car model is again compared to the measures when the car is taking a curve. In Figure 4.7, the linear tire model is compared to the measures and the full tire model for speeds of 6, 9 and 17 m/s. At high speed (right graph), where the tire model performs the better, a small loss of accuracy is observed compared to the full tire model, resulting in a small overestimation of the slip. At lower speeds, the difference between the linear and full tire model is larger, but the linear model is closer to the measures. The linear tire model therefore offers better performance at lower speeds while still remaining accurate at higher speeds. For the range of speeds under 10 m/s, the linear model has been manually adjusted to better fit the measures at these lower speeds, as it can be seen in Figure 4.7, at the cost of lesser performance at higher speed. The adjusted linear tire model (4.16) kept for the tests and validation of the algorithm is,

$$\begin{cases} F_{Fy} = \underbrace{4450}_{C_F} \alpha_F \\ F_{Ry} = \underbrace{13700}_{C_R} \alpha_R \end{cases} \tag{4.17}$$

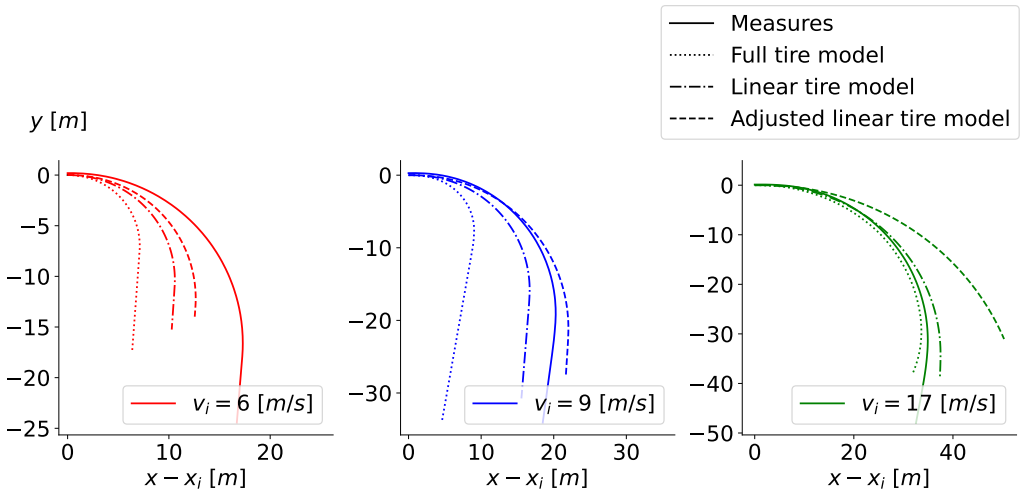

Figure 4.7: Validation of the linear tire model in a curve taken at different speeds but same steering angle (10°)

4.2 New optimization solver

The new solver is FORCESPRO from Embotech [64]. It is a commercial tool for generating highly customized optimization solvers that can be deployed on all embedded computers. FORCESPRO is intended to be used in situations were the same optimization problem has to be solved many times, possibly in real-time, as required for this application. Compared to OpEn, FORCESPRO offers a bigger panel of

options for the design of the problem and returns more detailed feedback.

FORCESPRO requires a license, which has been granted to us for a six month period as the form of a student license. This license offers all the necessary tools for this application.

The design and generation of the solver is similar to OpEn. The problem is designed in Python, then generated to create packages that must be included in the C++ code to call the necessary methods. However, the problem formulation is different. It is explained in this section to be able to relate the design of the new MPCC problem directly with the formulation of the solver in the next section.

The FORCESPRO NLP solver solves (potentially) non-convex, finite-time non-linear optimal control problems with prediction horizon N of the form [65]:

$$\begin{aligned}
\text{minimize} \quad & \sum_{k=1}^{N} f_k(\boldsymbol{z}_k, \boldsymbol{p}_k) \\
\text{subject to} \quad & \boldsymbol{z}_1(\mathcal{I}) = \boldsymbol{z}_{init} \\
& E_k \boldsymbol{z}_{k+1} = c_k(\boldsymbol{z}_k, \boldsymbol{p}_k) \\
& \boldsymbol{z}_N(\mathcal{N}) = \boldsymbol{z}_{final} \\
& \underline{\boldsymbol{z}}_k \leq \boldsymbol{z}_k \leq \overline{\boldsymbol{z}}_k \\
& F_k(\boldsymbol{z}_k) \in [\underline{\boldsymbol{z}}_k, \overline{\boldsymbol{z}}_k] \cap \mathbb{Z} \\
& \underline{h}_k \leq h_k(\boldsymbol{z}_k, \boldsymbol{p}_k) \leq \overline{h}_k
\end{aligned}$$

where $\boldsymbol{z}_k \in \mathbb{R}^{n_z}$ is the optimization variables vector, for example a collection of inputs and states; $\boldsymbol{p}_k \in \mathbb{R}^{n_p}$ are real-time parameters (not variable); $f_k : \mathbb{R}^{n_z} \times \mathbb{R}^{n_p} \to \mathbb{R}$ are the stage cost functions (the cost function can be changed depending on prediction k); $c_k : \mathbb{R}^{n_z} \times \mathbb{R}^{n_p} \to \mathbb{R}^{n_w}$ represents (potentially nonlinear) equality constraints, such as a state transition function; the matrices E_k are used to couple variables from the $(k+1)$-th stage to those of stage k through the function c_k; and the functions $h_k : \mathbb{R}^{n_z} \times \mathbb{R}^{n_p} \to \mathbb{R}^{n_h}$ are used to express potentially nonlinear, non-convex inequality constraints. The index sets $\mathcal{I}$ and $\mathcal{N}$ are used to determine which variables are fixed to initial and final values, respectively. At every stage k, the matrix F_k is a selection matrix that picks some coordinates in vector $\boldsymbol{z}_k$.

This formulation makes the design clearer and easier than OpEn. The variables of the problem are separated from the parameters, and the variable changes are expressed as equality constraints.

4.3 New MPCC problem formulation

The new formulation is designed in order to overcome the limitations of reference path and unstable speed profile while keeping the primary objective of the algorithm: maximizing the speed along the path while staying within the track limits. For this purpose, the virtual state formulation of the first solution is abandoned and a

formulation using targets is designed. Then the new algorithm is tested and validated with the testbench and on the simulator to achieve the final solution of this work.

4.3.1 Design

Targets formulation

The new formulation is called the targets formulation, because instead of relying on virtual state and control command to generate errors to minimize, the car is given targets to reach. These targets are in the form of positions on the reference path.

A target speed, v_{target}, is first defined to describe the desired speed along the reference path the car must reach. To link this target speed to the path, the car's position is linked to the path by projection. Based on the position on the path found by projection (x_i, y_i), the advancement of the car along the path s_i can be determined. Then, in order for the car to evolve at the target speed, the advancement of the car along the path Δt_{target} time in the future, s_{target}, must be,

$$s_{target} = s_i + \Delta t_{target} v_{target}$$

With the arc-length parameterization of the reference path, this target advancement can be transformed into a target position $(x_{target}(s_{target}), y_{target}(s_{target})$.

Now, for the MPCC problem, the prediction is done over the prediction horizon N with time step Δt_{MPCC}. Therefore, before calling the solver, a target advancement can be found for each prediction k based on the initial advancement s_i computed by projection of the car into the path,

$$s_{target,k} = s_{target,k-1} + \Delta t_{\text{MPCC}} v_{target} \quad \text{for } k = 1, 2, ..., N, \tag{4.18}$$

where $s_{target,0} = s_i$ (at $k = 1$). The resulting target positions are $(x_{target,k}(s_{target,k}), y_{target,k}(s_{target,k}))$ for $k = 1, 2, ..., N$.

This situation is represented in Figure 4.8 for $N = 10$. In order to set up the targets, the car at position (x, y) must first be projected into the reference path. But contrary to the first formulation, a virtual state s is not used to keep track of the car's advancement along the path. The car must therefore be located relative to the path. To perform this task, the only available components are the car's position and the position of the reference points. By finding the closest reference point in front of the car (called

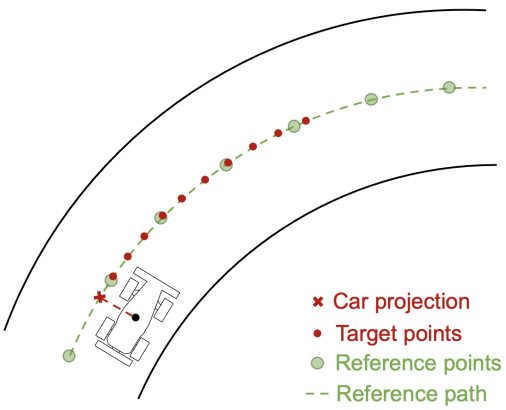

Figure 4.8: Representation of the targets formulation (for $N = 10$)

point B of coordinates (x_B, y_B)), the car can be known as located between this point and the point just before (called point A of coordinates (x_A, y_A)). This situation is represented in Figure 4.9 with X the position of the car and P the projection into the path.

The point B is found by computing the distance and orientation relative to the car of the reference points. As represented in Figure 4.10, the distance between the car and each available reference point is,

$$d = \sqrt{dx^2 + dy^2}, \tag{4.19}$$

as well as the relative orientation,

$$\phi_r = \underbrace{\arctan2(dy, dx)}_{d\phi} - \phi \tag{4.20}$$

A reference point is in front of the car if $\phi_r \in [-\frac{\pi}{2}, \frac{\pi}{2}]$ and the closest one (with the smaller d) is selected to be point B.

Once B has been found, A is also found by taking the previous reference point and the projection can be performed on the line joining these two points (see Appendix F for details about the computation of the projection).

Similarly to the previous formulation, once the first reference point (point A) has been selected, the selection of the next points depends on the required arc-length L_{min} and minimum number of points n_{min}. But for that, the arc-length of the projection $s_i = s_{target,0}$ must be determined. It is simply taken as the distance between the first selected reference point A and the projection P,

$$s_{target,0} = \sqrt{(x_P - x_A)^2 + (y_P - y_A)2} \tag{4.21}$$

Then the minimum required reference path's arc-length is computed by,

$$L_{min} = s_{target,0} + N\Delta t_{\text{MPCC}} v_{target} \tag{4.22}$$

The points are selected and the polynomials are computed to express the position on the path with the arc-length, similarly to the first formulation. The polynomials are used to compute the N targets with the N arc-length targets (4.18).

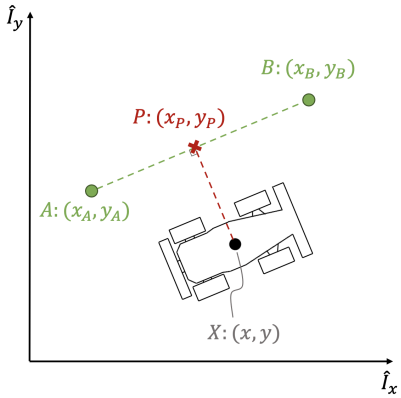

Figure 4.9: Representation of the four points A, B, X, P used for the computation of the projection into the reference path

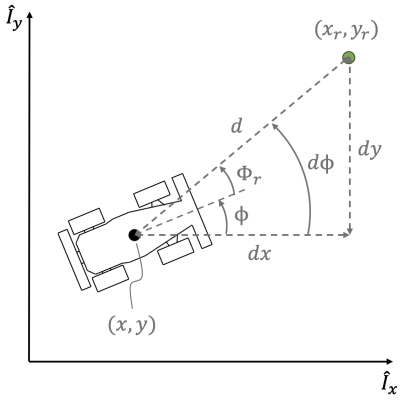

Figure 4.10: Distance and relative orientation between the car and a reference point of coordinates (x_r, y_r)

The major advantage of this targets formulation is that the reference path is not needed by the solver. The problem of computational complexity increasing with the order of the polynomials is no longer relevant. The order m can therefore be increase to have better fittings of the reference points even at higher speed. The value $m = 6$ (thus $n_{min} = 7$) has been chosen as it offers a fitting of quality in every situations.

MPCC problem formulation

The target formulation has the big advantages of not needing the virtual advancement s and to not requiring the reference path inside the optimization. The targets for the prediction horizon N are pre-computed based on the current car's position. The MPCC problem formulation must now be adapted to match the objective of staying close to the path while maximizing the speed using the targets.

- **Optimization variables vector**

  The first component of the optimization variables vector $\boldsymbol{z}_k$ is he control commands vector,

  $$\boldsymbol{u}_k = [\delta_k \ D_k]^T \in \mathbb{R}^{n_u}, \tag{4.23}$$

  with $n_u = 2$. The second component is the state vector, which is the state of the car since no virtual state is required,

  $$\boldsymbol{x}_k = [x_k \ y_k \ \phi_k \ v_{x,k} \ v_{y,k} \ \omega_k]^T \in \mathbb{R}^{n_x}, \tag{4.24}$$

  with $n_x = 6$. The third component is the previous control commands vector,

  $$\boldsymbol{u}_{prev,k} = [\delta_{prev,k} \ D_{prev,k}]^T \in \mathbb{R}^{n_u}, \tag{4.25}$$

  resulting in,

  $$\boldsymbol{z}_k = [\boldsymbol{u}_k^T \ \boldsymbol{x}_k^T \ \boldsymbol{u}_{prev,k}^T]^T \in \mathbb{R}^{n_z} \tag{4.26}$$

  with $n_z = n_u + n_x + n_u = 10$.

- **Equality constraints**

  The equality constraints describe the evolution of the optimization variables vector $\boldsymbol{z}_k$ except for the control commands. They are expressed as,

  $$E_k \boldsymbol{z}_{k+1} = c_k(\boldsymbol{z}_k, \boldsymbol{p}_k) \tag{4.27}$$

  where,

  $$E_k = \begin{bmatrix} 0 & & & & \\ & 0 & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \end{bmatrix} \in \mathbb{R}^{n_z \times n_z} \tag{4.28}$$

to only select the vectors $\boldsymbol{x}_k$ and $\boldsymbol{u}_{prev,k}$ that must be updated. The function c_k uses the discretized vehicle model to update the state,

$$\boldsymbol{x}_{k+1} = \begin{bmatrix} x_{k+1} \\ y_{k+1} \\ \phi_{k+1} \\ v_{x,k+1} \\ v_{y,k+1} \\ \omega_{k+1} \end{bmatrix} = \begin{bmatrix} x_k \\ y_k \\ \phi_k \\ v_{x,k} \\ v_{y,k} \\ \omega_k \end{bmatrix} + \Delta t_{\text{MPCC}} \begin{bmatrix} v_{x,k}\cos(\phi_k) - v_{y,k}\sin(\phi_k) \\ v_{x,k}\sin(\phi_k) + v_{y,k}\cos(\phi_k) \\ \omega_k \\ \frac{1}{m}(F_{x,k} - F_{Fy,k}\sin(\delta_k) + m v_{y,k}\omega_k) \\ \frac{1}{m}(F_{Ry,k} + F_{Fy,k}\cos(\delta_k) - m v_{x,k}\omega_k) \\ \frac{1}{I_z}(F_{Fy,k} l_F \cos(\delta_k) - F_{Ry,k} l_R) \end{bmatrix}, \tag{4.29}$$

with,

$$\begin{cases} F_{x,k} &= C_{m1}\arctan(C_{m2} D_k) - C_d v_{x,k} \\ F_{Fy,k} &= m_F g \cdot d \sin(C \arctan(B \alpha_{F,k})) \\ F_{Ry,k} &= m_R g \cdot d \sin(C \arctan(B \alpha_{R,k})) \end{cases} \tag{4.30}$$

and,

$$\begin{cases} \alpha_{F,k} = \delta_k - \arctan\left(\frac{v_{y,k} + l_F \omega_k}{v_{x,k}}\right) \\ \alpha_{R,k} = -\arctan\left(\frac{v_{y,k} - l_R \omega_k}{v_{x,k}}\right) \end{cases}$$

and updates the previous control commands by giving the current control commands.

- **Inequality constraints**

  The control commands and control commands rate constraints (3.25) and (3.26) are applied on the new control command vector (4.25).

  On the other hand, the track constraints (3.27) cannot be used since no contouring error is computed anymore. Because the reference path is not available anymore inside the optimization problem, the track constraints must be pre-computed like the targets. At each prediction k, the car must be within the track limits and has the objective to reach its target. The targets can thus be used as reference value of the projected car's position on the reference path at each prediction k. For each target, two lines tangent to the limits are computed, as represented in Figure 4.11. These two tangents have expression,

$$y = a_{l,k} x + b_{l,k} \quad \text{(left limit)} \tag{4.31}$$
$$y = a_{r,k} x + b_{r,k} \quad \text{(right limit)} \tag{4.32}$$

  In order to stay within the track limits at each prediction k, the position of the car (x_k, y_k) must be between the two tangents,

$$(a_{l,k} x_k + b_{l,k} - y_k)(a_{r,k} x_k + b_{r,k} - y_k) \leq 0 \tag{4.33}$$

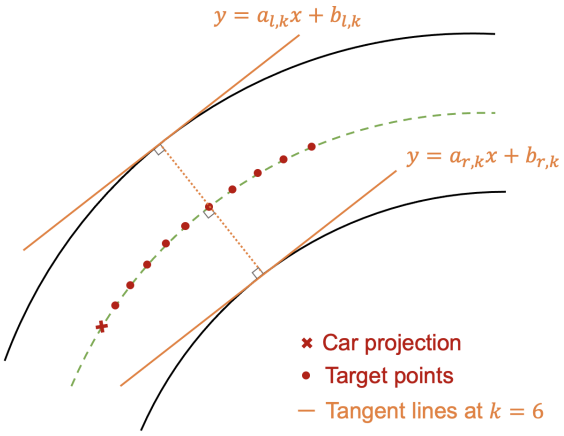

Figure 4.11: Representation of the tangent lines (at $k = 6$ in this example) used to express the track constraints

The four coefficients $\{a_{l,k}\ b_{l,k}\ a_{r,k}\ b_{r,k}\}$ for each target point must be computed to be given to the solver. To find the coefficients of each tangent line, two points per line must be computed. The first one, $(x_{l/r1,k}, y_{l/r1,k})$, will be the point at the intersection of the line perpendicular to the reference path and the limit. The second one, $(x_{l/r2,k}, y_{l/r2,k})$, just need to be further on the line, as shown in the left scheme of Figure 4.12. The orientation of the tangent at the reference path is,

$$\phi_{target,k} = \arctan2(\nabla y_c(s_{target,k}), \nabla x_c(s_{target,k})), \tag{4.34}$$

where $x_c(s)$ and $y_c(s)$ are the polynomials describing the reference path. The point $(x_{l1,k}, y_{l1,k})$ is oriented 90° to the left of $\phi_{target,k}$ and the point $(x_{r1,k}, y_{r1,k})$ 90° to the right,

$$\begin{cases} \phi_{l,k} = \phi_{target,k} + \frac{\pi}{2} \\ \phi_{r,k} = \phi_{target,k} - \frac{\pi}{2} \end{cases} \tag{4.35}$$

as represented in the right scheme of Figure 4.12. To handle every cases, these two angles are converted between $-\pi$ and π. The coordinates of the first points are given by,

$$\begin{cases} x_{l1,k} = x_{target,k} + R_c \cos(\phi_{l,k}) \\ y_{l1,k} = y_{target,k} + R_c \sin(\phi_{l,k}) \end{cases} \tag{4.36}$$

$$\begin{cases} x_{r1,k} = x_{target,k} + R_c \cos(\phi_{r,k}) \\ y_{r1,k} = y_{target,k} + R_c \sin(\phi_{r,k}) \end{cases} \tag{4.37}$$

The second points are then found based on the first point and the tangent orientation,

$$\begin{cases} x_{l2,k} = x_{l1,k} + d_{12} \cos(\phi_{target,k}) \\ y_{l2,k} = y_{l1,k} + d_{12} \sin(\phi_{target,k}) \end{cases} \tag{4.38}$$

$$\begin{cases} x_{r2,k} = x_{r1,k} + d_{12} \cos(\phi_{target,k}) \\ y_{r2,k} = y_{r1,k} + d_{12} \sin(\phi_{target,k}) \end{cases} \tag{4.39}$$

where $d_{12} = 2$ has been chosen. Finally, the coefficients of the tangents are,

$$\begin{cases} a_{l,k} = \frac{y_{l2,k} - y_{l1,k}}{x_{l2,k} - x_{l1,k}} \\ b_{l,k} = y_{l1,k} - a_{l,k} x_{l1,k} \end{cases} \tag{4.40}$$

$$\begin{cases} a_{r,k} = \frac{y_{r2,k} - y_{r1,k}}{x_{r2,k} - x_{r1,k}} \\ b_{r,k} = y_{r1,k} - a_{r,k} x_{r1,k} \end{cases} \tag{4.41}$$

When the tangents are vertical ($x_{l2,k} - x_{l1,k} = 0$ and $x_{r2,k} - x_{r1,k} = 0$), $a_{l,k}$ and $a_{r,k}$ are set up to 100 (a big value) to avoid the division by zero.

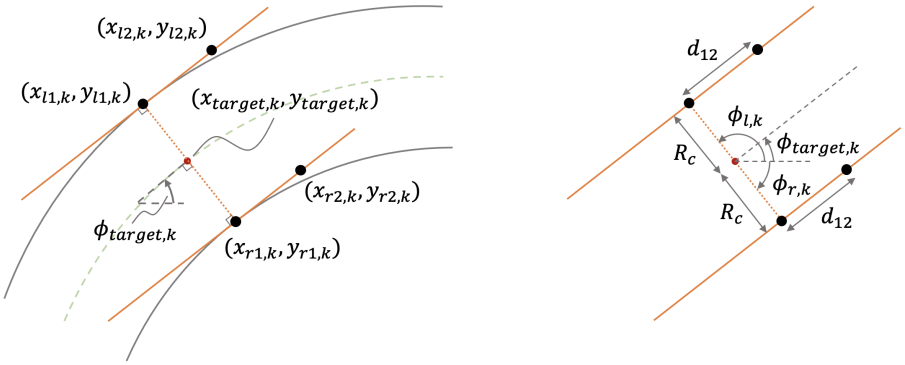

Figure 4.12: Schemes to represent the computation of the two points of each tangent line

- **Cost function**

  The cost function is still constituted of the control commands and control commands rate penalization functions,

$$R_{u,k} = r_{\delta,k}\delta_k^2 + r_{D,k}D_k^2, \tag{4.42}$$

$$R_{\Delta u,k} = r_{\Delta\delta,k}(\delta_k - \delta_{k-1})^2 + r_{\Delta D,k}(D_k - D_{k-1})^2 \tag{4.43}$$

  The objective function Q_k is changed to make use of the target points. The objective is to minimize the distance between the car and the target at each prediction k, resulting in the expression,

$$Q_k = q_{x,k}(x_{target,k} - x_k)^2 + q_{y,k}(y_{target,k} - y_k)^2 \tag{4.44}$$

- **Parameters vector**

  With the new FORCESPRO solver, the entire sequence of parameters $\boldsymbol{p_{seq}}$ must be given to the solver. The $\boldsymbol{p}_k$ vectors are filled in by the targets vector,

$$\boldsymbol{t}_k = [x_{target,k}\ y_{target,k}]^T \in \mathbb{R}^{n_t}, \tag{4.45}$$

  with $n_t = 2$; the track constraints coefficients vector,

$$\boldsymbol{c}_k = [a_{l,k}\ b_{l,k}\ a_{r,k}\ b_{r,k}]^T \in \mathbb{R}^{n_c}, \tag{4.46}$$

with $n_c = 4$; and the weights vector,

$$\boldsymbol{j}_k = [q_{x,k}\ q_{y,k}\ r_{\delta,k}\ r_{D,k}\ r_{\Delta\delta,k}\ r_{\Delta D,k}]^T \in \mathbb{R}^{n_j}, \tag{4.47}$$

with $n_j = 6$.

Therefore,

$$\boldsymbol{p}_k = [\boldsymbol{t}_k^T\ \boldsymbol{c}_k^T\ \boldsymbol{j}_k^T]^T \in \mathbb{R}^{n_p}, \tag{4.48}$$

with $n_p = n_t + n_c + n_j = 12$.

- **Initial conditions**

  Initial conditions must be provided to start the optimization. The initial optimization variables vector $\boldsymbol{z}_{init}$ is given, constituted of the initial guess for the control commands (last applied control commands), the measured car state and the previous control commands (last applied control commands).

The optimization problem can therefore be formulated as a minimization problem subject to constraints,

$$\begin{aligned}
\text{minimize} \quad & \sum_{k=1}^{N} Q_k(\boldsymbol{z}_k, \boldsymbol{p}_k) + R_{u,k}(\boldsymbol{z}_k, \boldsymbol{p}_k) + R_{\Delta u,k}(\boldsymbol{z}_k \boldsymbol{p}_k) \\
\text{subject to} \quad & \boldsymbol{z}_1 = \boldsymbol{z}_{init} \\
& E_k \boldsymbol{z}_{k+1} = c_k(\boldsymbol{z}_k, \boldsymbol{p}_k) \\
& \underline{\boldsymbol{u}}_k \leq \boldsymbol{u}_k \leq \overline{\boldsymbol{u}}_k \\
& \underline{\boldsymbol{\Delta u}}_k \leq \boldsymbol{\Delta u}_k \leq \overline{\boldsymbol{\Delta u}}_k \\
& (a_{l,k} x_k + b_{l,k} - y_k)(a_{r,k} x_k + b_{r,k} - y_k) \leq 0
\end{aligned}$$

4.3.2 Implementation

To implement the new formulation, new structures have been created.

- The structure `MpccPolyFits` computes the coefficients of the two polynomials describing the parameterized reference path. The determination of the reference points A and B, the projection and the selection of the other reference points are performed before applying the the least square regression to compute the coefficients;

- The structure `MpccTargets` computes the sequence of targets $(x_{target,k}, y_{target,k})$ and their derivatives $(\nabla x_{target,k}, \nabla y_{target,k})$ based on the defined target speed and the polynomials computed by the `MpccPolyFits` structure;

- The structure `MpccTrackConstraint` computes the sequence of track constraints coefficients $\{a_{l,k}\ b_{l,k}\ a_{r,k}\ b_{r,k}\}$ based on the targets and target derivatives computed by the `MpccTargets` structure and the limiting distance R_c defined;

- The structure `MpccWeights` generates the sequence of weights. A function is implemented to generate variable weights.

The first three structures are called at each call of the function `mpccTimer()` every Δt to generate the new polynomials, sequences of targets and track constraints. The fourth structure is only used at the start since the weights are not modified in real-time.

4.4 Tests and validation with testbench

The new MPCC algorithm is first tested and validated with the testbench. The objective is again to study the influence of the different performance parameters but also to point the difference between this new formulation and the first one. The benchmark in Table 3.1 is again used to perform the tests, and the model used for the car in the testbench is adapted to the one used in the optimization problem.

The analysis again starts with the unit weights vector,

$$\boldsymbol{j}_k = [1\ 1\ 1\ 1\ 1\ 1]^T, \tag{4.49}$$

where the two first weights q_x and q_y will always be taken equal since the car has no preferential direction. It will be noted $q = q_x = q_y$.

The lastly found constraints during the tests and validation with the testbench of the first formulation in Subsection 3.5.3 are used,

$$\begin{cases} \underline{\boldsymbol{u}}_k = [-25\frac{\pi}{180} \quad -0.2]^T \\ \overline{\boldsymbol{u}}_k = [25\frac{\pi}{180} \quad 0.2]^T \end{cases} \tag{4.50}$$

$$\begin{cases} \underline{\boldsymbol{\Delta u}}_k = [-2 \times 25\frac{\pi}{180}\Delta t_{\text{MPCC}} \quad -0.025]^T \\ \overline{\boldsymbol{\Delta u}}_k = [2 \times 25\frac{\pi}{180}\Delta t_{\text{MPCC}} \quad 0.025]^T \end{cases} \tag{4.51}$$

Since larger prediction horizons N are possible with the new solver, the time steps are reduced to $\Delta t_{\text{MPCC}} = \Delta t = 50\,m/s$ and the prediction horizon is increased to $N = 30$ to keep a finite time horizon of $T = 1.5\,s$ but with a faster control commands update.

The behavior of the car is first analyzed is straight line on track 1. The speed $v = \sqrt{v_x^2 + v_y^2}$ and control command D profiles for different target speeds v_{target} are represented in Figure 4.13. The steady speed is quickly reached with no overshoot, which is better than the first formulation. However, the steady speed reached, for the three different speeds, is approximately 20% higher than the target speed. The solver, instead of reducing the speed when it exceeds the target speed as it is done with the first formulation, finds the optimal control commands sequence that maintains a steady speed without reducing it. This is a consequence of the target formulation, but this is not a problem since the speed profile is smooth and reaches a steady value which is the wanted effect by imposing a target speed.

The behavior of the car in a curve can now be analyzed on track 3.2 at low speed, $v_{target} = 3\,m/s$. As the weight q_c in the first formulation, the weights q can play the role of determining the importance of the deviation from the reference path to take racing lines, but also the role of giving importance for reaching the last targets to maximize the speed. The function $f_{q_c}(k)$ describing the variation of the weight is again used on the weights q,

$$q = f_q(k) = \begin{cases} q_i & \text{if } k \leq N_i \\ q_{k-1} + \frac{q_f}{N - N_i} & \text{if } k > N_i \end{cases} \quad \text{for } k = 1, 2, ..., N \tag{4.52}$$

with $N_i = \frac{3}{4}N$, $q_i = 1$ and $q_f = 50$. With these variables weights, the focus is on the importance of reaching the last targets, allowing the car to deviate from the first targets depending on the value of the other weights. As it can be seen in Figure 4.14, changing $q = 1$ to $q = f_q(k)$ while keeping $r_\delta = 1$ has not a real impact on the trajectory taken (top graph) but on the speed (bottom graph). The speed profile is smoother and the speed is less decreased in the middle of the curve. Then, by increasing r_δ to 10, the car better anticipates the curve and takes a better racing line, like it was observed with the first formulation.

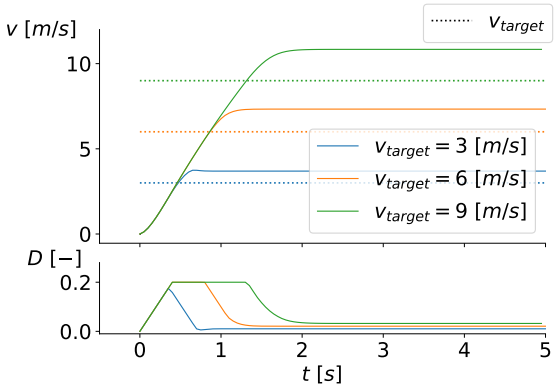

Figure 4.13: Speed and control command D profiles for different target speeds

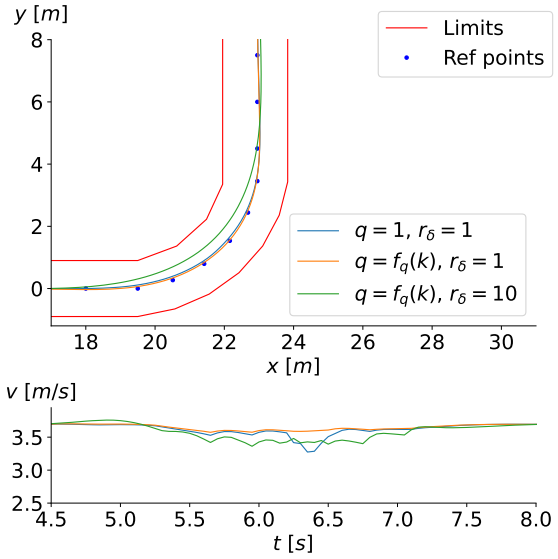

Figure 4.14: Influence of weights q and r_δ in a curve on track 3.2

Therefore, by playing only with q and r_δ, similar effect can be obtained than playing with q_l, q_c, q_v and r_δ for the first formulation. In a curve, the effects of the parameters can be compared to the first formulation but no the results, since the model of the car is not the same anymore.

Although the solver manages to control the car at low speed, its performance are bad in the curve, as it can be seen in Figure 4.15 for $v_{target} = 3\,m/s$ on track 3.2. The solve time almost reaches $50\,ms$, which is the limit and the solver returns some exitflag $= 0$, meaning than the maximum number of iterations has been reached. However, when the speed is increased ($v_{target} = 6\,m/s$), the solver performance are way better. Therefore, as it is observable for the validation of the tire model at low and high speeds, the dynamic model better performs at higher speeds.

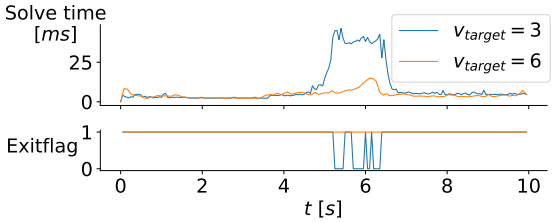

Figure 4.15: Solve time and exitflag of the solver on track 3.2 at different target speeds (exitflag = 1: success, exitflag = 0: maximum number of iterations reached)

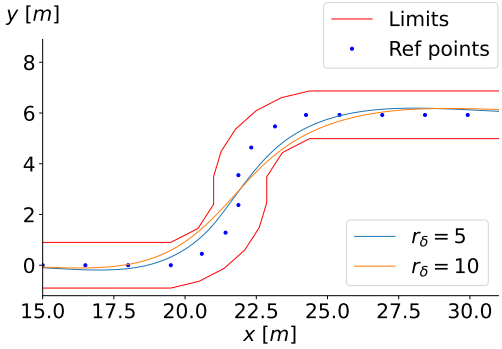

Figure 4.16: Influence of weight r_δ on the trajectory on track 5.1 - $q = f_q(k)$, $v_{target} = 6\,m/s$

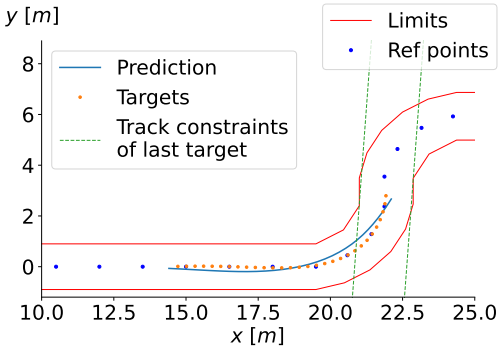

Figure 4.17: Prediction, targets and last target's track constraints at a certain time to illustrate the anticipation of the turn - $q = f_q(k)$, $r_\delta = 5$, $v_{target} = 6\,m/s$

The influence of the weight r_δ can be further analyzed at $v_{target} = 6\,m/s$ on track 5.1. The comparison of the trajectories at $r_\delta = 10$ and $r_\delta = 5$ is shown in Figure 4.16. As expected, reducing r_δ allows to take a trajectory closer to the reference path and thus less close to the limits. The value $r_\delta = 5$ is therefore kept.

In Figure 4.17, the anticipation mechanism is illustrated. The 30 ($= N$) targets follow the reference path at the center of the track and the prediction, thanks to the variable weights $q = f_q(k)$ and the weight $r_\delta > q_i$, anticipates the first curve by deviating the car from the targets to keep δ as low as possible. The track constraints of the last target are also represented to illustrate how they take form on the track.

The influence of the other weights, r_D, $r_{\Delta\delta}$ and $r_{\Delta D}$ has been analyzed on the different tracks but as for the first formulation, no real impact has been observed.

They should have a bigger impact on the simulator since the model will not be perfect anymore.

For the next analysis, the target speed is increased to $v_{target} = 9\,m/s$, thus a real speed of $v \approx v_{target}\left(1 + \frac{20}{100}\right) = 10.8\,m/s$. The algorithm performs well on all the tracks except track 5.2, where it fails to predict the trajectory to follow at the start of the second curve. The origin of the problem is illustrated in the left graph of Figure 4.18. In the second half of the first curve, the targets extend towards the middle of the second curve. Consequently, the track constraints of the last target do not align with the track's previous shape. As a result, it becomes impossible for the solver to adhere to these constraints while following the track. During the first curve, the car reduces its speed to navigate the sharp turn successfully. However, in order to stay on track, the car would need to change direction in the middle of the two curves, which is impractical at high speeds. Unfortunately, the last set of track constraints require the car to approach the last target closely, a maneuver that cannot be accomplished due to the car's inability to accelerate sufficiently while changing direction. Consequently, the solver adjusts the car's trajectory to deviate from the last target in order to respect the constraints. This specific situation arises solely in the case of the double hairpin turn.

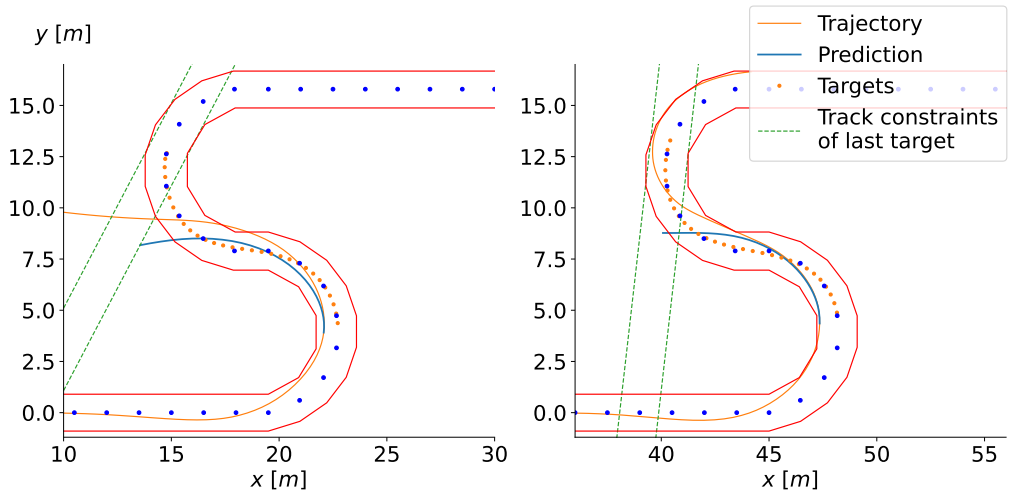

Figure 4.18: [Left graph]: illustration of the origin of the fail occurring on track 5.2 at $v_{target} = 9\,m/s$ - [Right graph]: improvement of the track constraints to enable the control of the car on track 5.2 at $v_{target} = 9\,m/s$

To improve the track constraints, they must better follow the direction of the portion of track that comes just before the considered target. For this purpose, the orientation of the track constraints at each prediction $\phi_{target,k}$ is not only computed with the target k but is computed as the average of the n_{avg} previous targets (if available: if $k < n_{avg}$ then all the previous targets are chosen), current target included. Also, to limit the acceleration between the two curves, the upper bound of D has been reduced to,

$$\overline{D}_k = 0.1, \tag{4.53}$$

instead of 0.2. The result for $n_{avg} = 5$ is shown on the right graph of Figure 4.18. Compared to the left graph, the track constraints of the last target follow much better the orientation of the track. With this improvement, the car is able to stay within the limits, but the trajectory is not yet optimal. n_{avg} must not be taken too large to avoid changing too much the orientation of the track constraints and enable the car to cross the limits in certain cases.

To try improving the trajectory, other weights have been tested, as well as other N but none of these changes improved the behavior of the car. The target speed has thus been reduced to $8\,m/s$ to observe the performance at a slightly lower speed. This time, without changing the track constraints ($n_{avg} = 1$), the car is able to complete the track with $N = 20$. The weight r_δ has also been reduced to 3 instead of 5 to have less anticipation on the first curve. The trajectory and velocity profile are shown in Figure 4.19 for $\overline{D} = 0.1$ and $\overline{D} = 0.2$. The trajectory taken is a little bit better with $\overline{D} = 0.1$ because the car less accelerates between the two curves, as it can be observed on the speed profile. With $N = 30$, the algorithm fails to keep the car within the track.

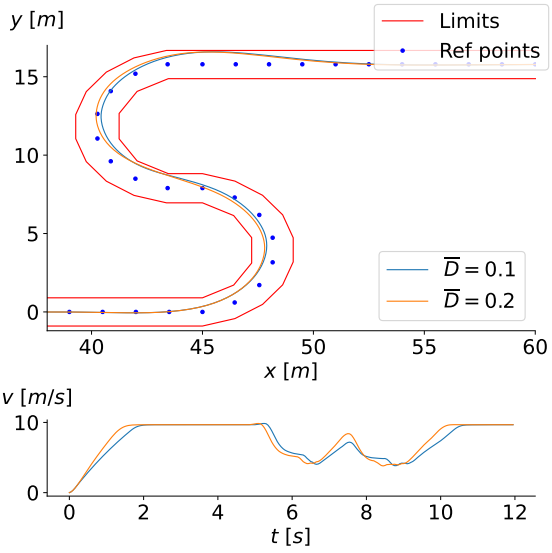

Figure 4.19: Trajectory and speed profile at $v_{target} = 8\,m/s$, $N = 20$, $r_\delta = 3$ and $n_{avg} = 1$ on track 5.2 - comparison between $\overline{D} = 0.1$ and $\overline{D} = 0.2$

Finally, the impact of N is analyzed at $v_{target} = 6\,m/s$ on the three difficult tracks. The trajectories, speed profiles and solve time are shown in Figure 4.20. For the three tracks, the results for $N = 30$ and $N = 40$ do not really differ. The difference is more evident with $N = 20$, where the smaller view ahead forces the car to more decelerate in the curves.

As expected, the solve time increases with N, both in average and in peak amplitudes. In every cases, the solve time does not exceed $25\,ms$, which leaves a good margin of error before reaching the maximum solve time of $50\,ms$ ($= \Delta t$).

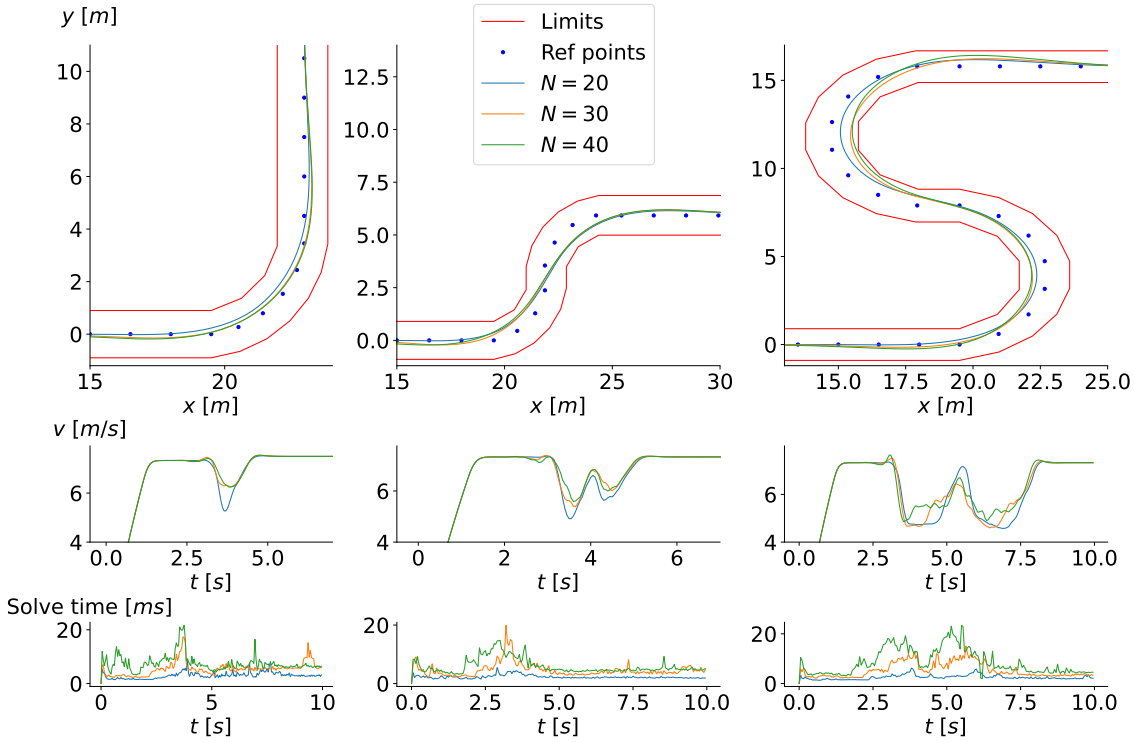

Figure 4.20: Influence of prediction horizon N on the trajectory, speed profile and solve time on tracks 3.2, 5.1 and 5.2 at $v_{target} = 6\,m/s$

To conclude this performance parameters study of the new formulation, a smoother speed profile has been obtained at the cost of a steady error. Results have not been shown but the increasing of the order of the polynomials at $m = 6$ allows to have a reference path fitting well the center of the track even at higher speed. Also, because of the dynamic model, the speed must be higher than $6\,m/s$ to enable the solver to efficiently solve the problem.

The analysis on the performance parameters is simpler thanks to the reduce number of weights and control commands. By defining the target weights q as variable weights like it was done for q_c in the first formulation, and by playing with r_δ to control the degree of anticipation taken by the car, the car is able to take racing lines at different speeds on difficult tracks. The bigger the target speed, the bigger the predicted view ahead and the smaller r_δ must be to not deviate too much from the path. Similarly to the first formulation, the other weights do not have a real impact on the behavior of the car with the testbench.

The only track that poses problems at higher speed is the double hairpin turns (track 5.2). In this particular case, the track constraints can lead to the loss of control of the car. Depending on the target speed, different parameter configurations will deliver the best performance. Reducing $\overline{D}_k$ to 0.1 has a positive effect but choosing $N = 30$ or $N = 20$ will depend on the target speed. The formulation of the track constraints can thus limit the performance of the algorithm in certain cases. No real differences were observed between $N = 30$ and $N = 40$, if not the solve time increase.

This analysis enabled to validate the formulation and provided a basis for the choice of parameters to apply in order to effectively control the car on the simulator.

4.5 Tests and validation on Formula Student Driverless Simulator

The initial configuration of the performance parameters takes into account the analyzed with the testbench in Subsection 4.4 for the constraints, with $\overline{D}_k = 0.1$ and control commands rates (4.51). $\Delta t = \Delta t_{\text{MPCC}} = 50\,ms$ is kept, N is set to 30, the weights vector is reinitialized,

$$\boldsymbol{j}_k = [1\ 1\ 1\ 1\ 1\ 1]^T, \qquad (4.54)$$

and the target speed is directly set at $v_{target} = 6\,m/s$, since the solver does not perform well at lower speed.

The result in terms of speed and D profiles are shown in blue in Figure 4.21. Small oscillations are observed but the velocity quickly stabilizes. This is a major improvement compared to the first formulation. This time also, the weights r_D and $r_{\Delta D}$ can improve the performance. By increasing r_D, the speed and D profiles becomes smoother, with no more overshoot and oscillations. The values $r_D = 10$ and $r_D = 40$ are compared in Figure 4.21. The value $r_D = 40$ has been kept as it offers a smooth velocity profile with a reduce gap compared to the target velocity of approximately 10%. Moreover, $r_{\Delta D}$ has been increased to 10 to further smooth the profiles.

Then the behavior in curves is analyzed. The influence of weights $q = q_x = q_y$ and r_δ are shown in

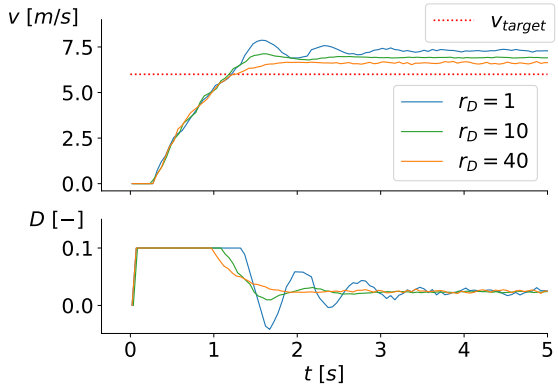

Figure 4.21: Comparison of speed and D profiles on track 1 at target speed $v_{target} = 6\,m/s$ for different values of r_D (with weights vector $\boldsymbol{j}_k = [1\ 1\ 1\ 1\ 1\ 1]^T$ as basis)

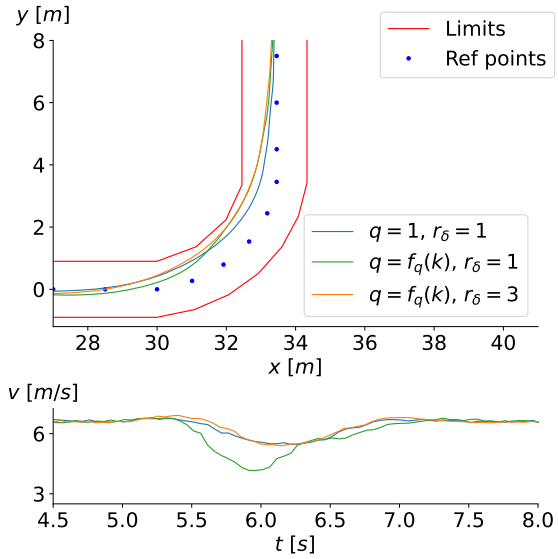

Figure 4.22: Comparison of trajectory and speed profile for different combinations of weights q and r_δ on track 3.2 at $v_{target} = 6\,m/s$

Figure 4.22. The result is already good with $q = r_\delta = 1$ but the best trajectory and velocity profile are obtained by setting $q = f_q(k)$ and increasing r_δ to 3. To obtain these results, the constraints on command δ rate has been increased to,

$$\begin{cases} \underline{\Delta\delta}_k = -5 \times 25\frac{\pi}{180}\Delta t_{\text{MPCC}} \\ \overline{\Delta\delta}_k = 5 \times 25\frac{\pi}{180}\Delta t_{\text{MPCC}} \end{cases} \tag{4.55}$$

to have more liberty on steering change in sharp turns taken at higher speed. In return, the weight $r_{\Delta\delta}$ is increased to 10 to eliminate the small oscillations on the steering appearing in straight line. Moreover, the lower bound on D has been extend to,

$$\underline{D}_k = -0.6 \tag{4.56}$$

to enable quicker braking.

The final algorithm design is able to efficiently and safely perform at $v_{target} = 8\,m/s$ ($v \approx 8.8\,m/s$) on all the benchmark tracks. The results are shown in Figure 4.23 for the three difficult tracks. Two changes have been done compared to the configuration at $v_{target} = 6\,m/s$:

- N has been decreased to 20 to improve the performance on track 5.2 while still being efficient on the other tracks. As explained in the previous section, having too much prediction on this particular track leads to lesser performance because of the track constraints;
- The initial value of q (q_i) is increased to 2 to reduce the curves anticipation that was causing too much braking on track 5.2.

The final performance parameters are therefore,

- $\Delta t = \Delta t_{\text{MPCC}} = 50\,ms$;
- $N = 20$;
- $v_{target} = 8\,m/s$;
- $\underline{\boldsymbol{u}}_k = [-25\frac{\pi}{180} \quad -0.6]^T$,
  $\overline{\boldsymbol{u}}_k = [25\frac{\pi}{180} \quad 0.1]^T$;
- $\underline{\boldsymbol{\Delta u}}_k = [-5 \times 25\frac{\pi}{180}\Delta t_{\text{MPCC}} \quad -0.025]^T$,
  $\overline{\boldsymbol{\Delta u}}_k = [5 \times 25\frac{\pi}{180}\Delta t_{\text{MPCC}} \quad 0.025]^T$;
- $\boldsymbol{j}_k = [f_q(k) \;\; f_q(k) \;\; 3 \;\; 40 \;\; 10 \;\; 10]^T$,
  with,

$$q_k = f_q(k) = \begin{cases} 2 & \text{if } k \leq \frac{3}{4}N \\ q_{k-1} + \frac{50}{N - \frac{3}{4}N} & \text{if } k > \frac{3}{4}N \end{cases} \qquad \text{for } k = 1, 2, ..., N \tag{4.57}$$

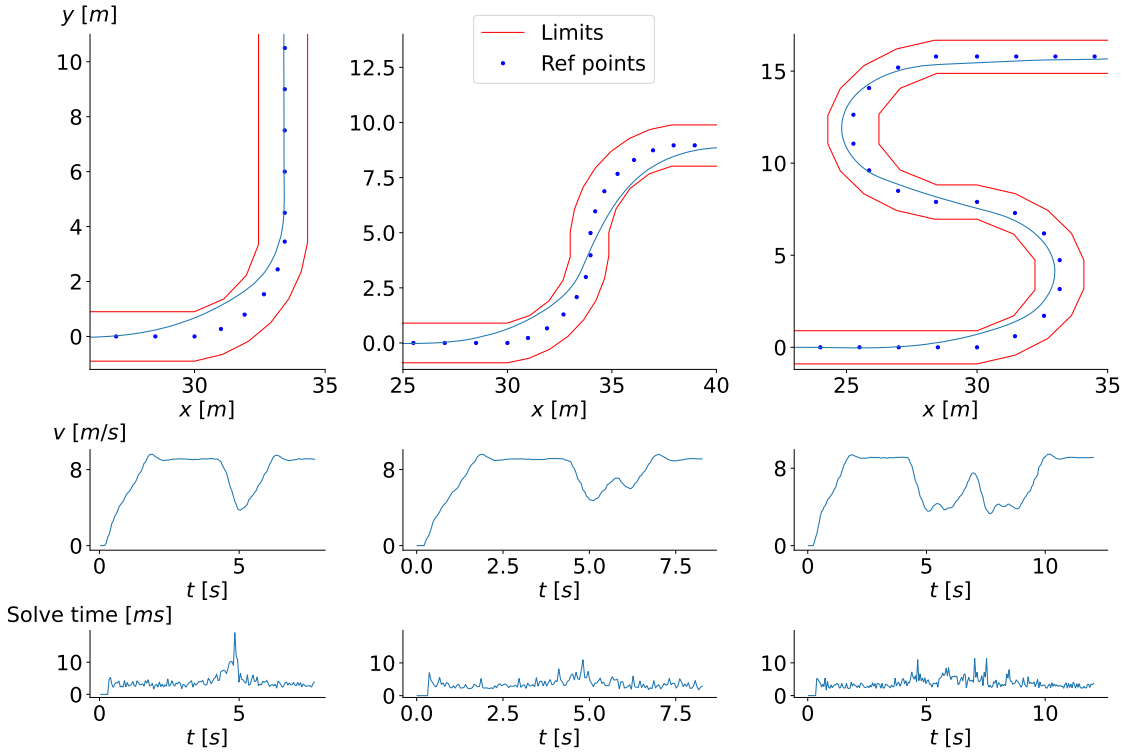

Figure 4.23: performance of the final design of the PF algorithm on tracks 3.2, 5.1 and 5.2 at $v_{target} = 8\,m/s$

The visualization of the performance of the algorithm in real-time on the tracks of Figure 4.23 are available at the following links: Track 3.2 [66], Track 5.1 [67], Track 5.2 [68]. In addition to the car position, track limits and center line, the polynomial formed by the targets is represented in green.

The car's performance was hindered at higher speeds in certain tracks due to the track constraints in this formulation. As the speed increases, more braking is required to navigate the sharp turns, resulting in a larger discrepancy between the predicted trajectory and the target points. Additionally, a broader lookahead is necessary to anticipate the curves more effectively and initiate braking in a timely manner. These factors contribute to a degradation in the quality of the track constraints, particularly evident in cases like track 5.2. To enhance the algorithm's performance further, it is imperative to improve the formulation of the track constraints. This improvement would enable the safe expansion of the prediction horizon N and facilitate better anticipation of upcoming difficulties.

4.6 Conclusion

The final formulation is able to overcome all the limitations of the first developed solution. The major improvement in terms of performance comes from the dynamic model and the modeling of the slip. The algorithm can better predict the behavior of the car and adapt the control commands to be able to negotiate sharp turns at

higher speed. The targets formulation allows to have a smooth speed profile with a quick response and a reference path following the center of the track in any situation.

On the downside, the new track constraints, which are precomputed based on the targets, restrict the algorithm's performance in specific scenarios. An attempt was made to enhance their calculation, but it did not yield significant improvements. As a result, the maximum target speed attainable with this formulation could not be further enhanced. However, the reduce complexity of the formulation, resulting in computational times below $25\,ms$ by the solver, offers room for improvement.

Determining the best performance parameters to achieve robust and efficient control in every situation is a real challenge given the large number of factors influencing the MPCC algorithm's performance. Through deeper analysis and research to improve the formulation, the performance of the algorithm could be further improved.

5 | Code architecture

The control algorithm code is organized as a Robot Operating System (ROS) project. ROS is an open-source framework that provides a collection of software libraries and tools to help developers build robot applications. It offers a flexible and modular architecture that simplifies the development process by providing standardized communication mechanisms, hardware abstraction, and various libraries for common robotics tasks.

ROS facilitates the creation of robotic systems by enabling components to work together seamlessly. It offers a distributed computing environment, allowing different nodes (software modules) to communicate with each other using a publish-subscribe messaging system.

A publisher is a node that generates and sends messages on a specific topic. It collects data or information from sensors, algorithms, or other sources and publishes it on a designated topic. The publisher is responsible for transmitting the data to any interested nodes within the ROS network.

On the other hand, a subscriber is a node that receives messages from a specific topic. It expresses its interest in a particular topic by subscribing to it. Once subscribed, the subscriber node will receive any messages published on that topic.

The publisher and subscriber nodes operate in an asynchronous manner, meaning that they do not need to be active at the same time or run at the same frequency. Publishers can publish messages at their own rate, and subscribers can receive messages whenever they are available.

In this work, the ROS network consists of four nodes (see Figure 5.1):

- SLAM
- PP
- PF
- Car/Simulator

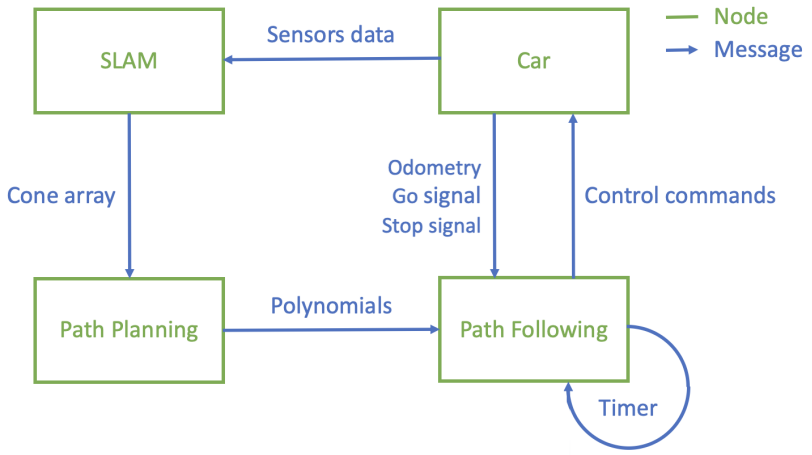

Figure 5.1: ROS network

As described in Table 5.1, the nodes in the ROS network establish communication by exchanging messages on specific topics at designated frequencies.

Topic	Publisher	Subscriber	Message	Frequency [Hz]
`polynomials`	PP	PF	• Reference points • Lap closure • Car position	$25 \leq f_{SLAM} \leq 40$
`odom`	Car	PF	• Odometry data	$100 \leq f_{odo} \leq 200$
`go`	Car	PF	• Signal to go	-
`stop`	Car	PF	• Signal to stop	-
`control_command`	PF	Car	• Throttle • Steering • Brake	$f_{commands} = 200$
`cone_array`	SLAM	PP	• Cone position • Cone color	$25 \leq f_{SLAM} \leq 40$

Table 5.1: ROS communication

Furthermore, in ROS, a Timer API is available to facilitate periodic task execution. Timers play a crucial role in coordinating and synchronizing actions within a node. Specifically, in the context of PF as depicted in Figure 5.1, two Timers are utilized:

- `command_pub_timer` calls `commandPublisherTimer()` at frequency $f_{commands}$
- `mpcc_timer` calls `mpccTimer()` at frequency $f = \frac{1}{\Delta t}$

6 | Control algorithm performance and results

This chapter presents the performance and results of the full control algorithm on the simulator. In the previous chapters, both PP and PF algorithms have been designed and validated independently. Now both of them will be used simultaneously to be able to control the car on the full track starting from an unknown environment.

For the PP, using it simultaneously with the PF does not change its behavior as it does not receive any information from the PF. However, the PF relies on the PP to be provided with the reference points. During the first lap, the PP algorithm computes the reference points in real-time with an ahead vision of $6\,m$, meaning that reference points are available at maximum $6\,m$ ahead of the car. Then, fore subsequent laps, the full track has been computed by the PP and all the reference points are available. Therefore, the target speed during the first lap must be limited in order to keep the control of the car while the track appears in front of it. The reference path formulation enables to compute a path even when not enough reference points are available in the prediction horizon, but the portion added does not necessary follow the track. This is represented in Figure 6.1, where the blue/yellow points are the detected cones, the red lines the computed triangles, the small green dots the computed reference points, the big green point the car and the green line the polynomial formed by the targets computed on the reference path. With a too long portion of guessed reference path, the PF algorithm could wrongly predict the path to the point that it could not regain control of the car.

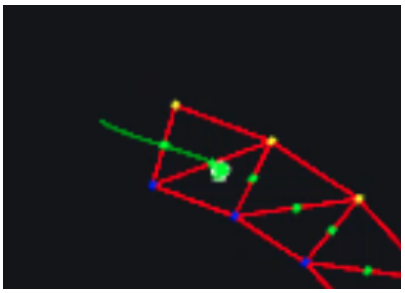

Figure 6.1: Illustration of reference path computation ahead of detected track

The target speed is thus set up at $v_{target} = 6\,m/s$ for the first lap, then increased to $v_{target} = 8\,m/s$ for subsequent laps. $6\,m/s$ is the maximum speed at which the algorithm is able to keep a safe control of the car during the first lap. The algorithm is tested on two tracks, a simpler one with more large curves and a technical one with more sharp turns. The real-time visualizations are available at the following links: Track 1 [69], Track 2 [70]. In Figure 6.2, the fully computed track 2 is represented.

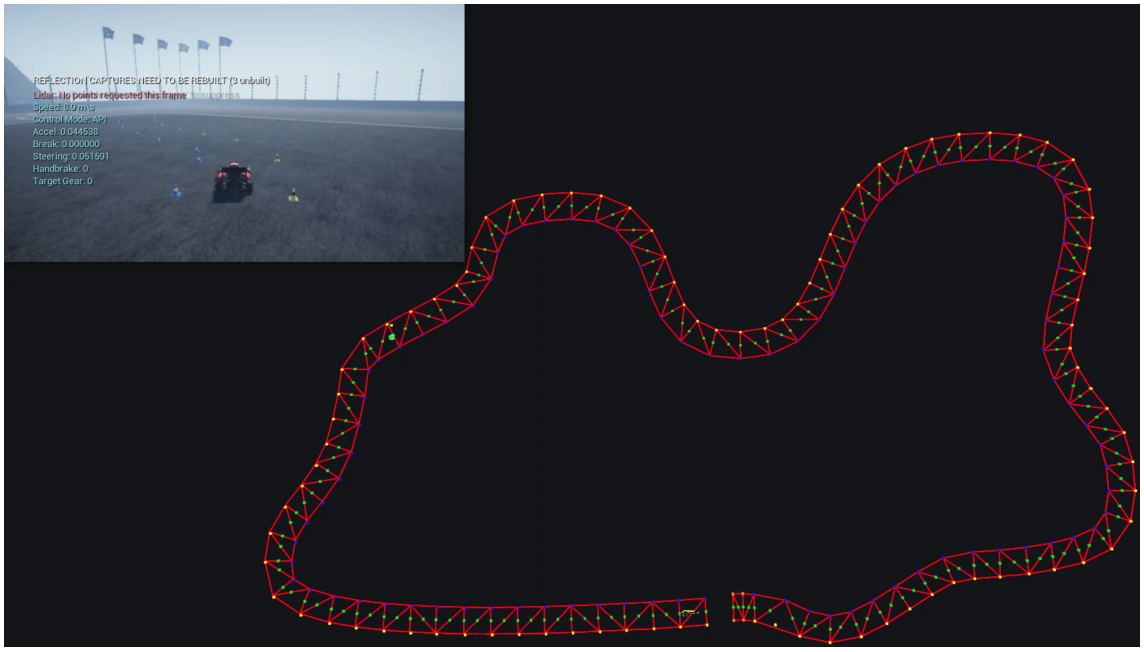

Figure 6.2: Full track computed when the first lap has been completed

During the initial lap, the car's trajectory exhibits less smoothness due to the inability to fully anticipate upcoming curves. However, the algorithm displays swift responsiveness to changes in direction, ensuring safe control. Subsequent laps benefit from having the complete path available for the prediction horizon, resulting in smoother and optimized trajectories.

When assessing the overall performance on both tracks, the car demonstrates the ability to navigate most curves without significant speed reduction, thanks to the effectiveness of the chosen trajectories. However, in more challenging curves, there is still room for improvement in maintaining speed without compromising safety. This highlights the difficulty of determining optimal performance parameters for every possible situation. Ultimately, the algorithm delivers a safe and effective control strategy at a reasonable speed (approximately $8.8\,m/s$), laying a solid foundation for future enhancements.

7 | Conclusion and future work

In conclusion, this paper has presented a comprehensive exploration of optimized control solutions, with a specific focus on path following and the implementation of a Model Predictive Contouring Control (MPCC) algorithm for an autonomous electric race car developed by Formula Electric Belgium. The research addressed the unique challenges and requirements of the Formula Student Driverless competition, where autonomous electric race cars are designed and constructed for various static and dynamic events.

The paper has provided a detailed explanation of the path planning algorithm, which plays a crucial role in determining the race car's trajectory. It has delved into the complexities of path following, considering factors such as track geometry, constraints, and real-time adjustments to optimize steering, throttle, and brake inputs in order to maximize the progress along the path while staying within the track boundaries.

Multiple solutions have been explored, encompassing different optimization problem formulations, car models, and optimization solvers, with a particular emphasis on the MPCC algorithm. Tests and evaluations were conducted using the Formula Student Driverless Simulator, a realistic virtual environment that replicates real-world race tracks. Through meticulous analysis and comparison conducted within this simulator, effective strategies and approaches for achieving superior control performance in the competitive setting have been identified.This rigorous evaluation process ensured that the proposed solutions were thoroughly tested and validated.

The study has also discussed the benefits and challenges of MPCC and presented the results and performance achieved through various strategies, providing valuable insights for implementing control systems in autonomous race cars. Ultimately, the algorithm has delivered a safe and effective control strategy at a reasonable speed of approximately $9\,m/s$, capable of dealing with any track configuration. The formulation of the proposed solution has enabled achieving high performance in terms of computational time, as the optimization solver is capable of solving the problem in under $20\,ms$, providing ample room for improvement.

Future work could explore the promising application of machine learning techniques to optimize performance parameters and enhance the tire model. By leveraging machine learning algorithms, performance parameters could be dynamically optimized for each corner, enabling fine-tuned control inputs and maximizing performance dur-

ing cornering. Furthermore, by incorporating machine learning algorithms that adapt the tire model in real-time, more precise control inputs could be achieved, leading to optimized vehicle performance and improved the efficiency of the MPCC algorithm.

In addition to these advancements, future research could also focus on refining the accuracy and fidelity of the car model itself. Considering factors such as vehicle dynamics, tire characteristics, and aerodynamics, enhancing the car model would contribute to improved predictions and overall performance of the control algorithm.

References

[1] FEB Team, "Formula Electric Belgium," *Formula Electric Belgium*. [Online]. Available: https://formulaelectric.be/ [Accessed: May 13, 2023].

[2] Sturm, "Formula Student Germany 2017," *Formula Student Germany*, Aug. 9, 2017. [Online]. Available: https://media.formulastudent.de/2017/Hockenheim/20170809-Wednesday/i-gFw2z3R [Accessed: May 17, 2023].

[3] FEB Team, "The Project," *Formula Electric Belgium*. [Online]. Available: https://formulaelectric.be/about-us/ [Accessed: May 17, 2023].

[4] @christian_hsa, "FPGA accelerated object detection for the Formula Student Driverless," *Imaginghub*, [Online]. Available: https://imaginghub.com/projects/393-fpga-accelerated-object-detection-for-the-formula-student-driverless [Accessed: May 17, 2023].

[5] FSG Team "FSG: Formula Student Germany," *Formula Student Germany*. [Online]. Available: https://www.formulastudent.de/fsg/ [Accessed: May 17, 2023].

[6] Institution of Mechanical Engineers, "Formula Student," *Institution of Mechanical Engineers*. [Online]. Available: https://www.imeche.org/events/formula-student/team-information/rules [Accessed: May 17, 2023].

[7] FSE Team, "Formula Student East," *Formula Student East*. [Online]. Available: https://fseast.eu/ [Accessed: May 17, 2023].

[8] Formula Student Team Delft, MIT Driverless, and FSEast, "Formula Student Driverless Simulator," *Formula Student Driverless Simulator*, 2020. [Online]. Available: https://fs-driverless.github.io/Formula-Student-Driverless-Simulator/v2.2.0/ [Accessed: May 17, 2023].

[9] W. Farag, "Track Maneuvering using PID Control for Self-Driving Cars," *Recent Advances in Electrical & Electronic Engineering*, vol. 12, Jan. 2019. [Online]. Available: http://dx.doi.org/10.2174/2352096512666190118161122 [Accessed: Sep. 21, 2022].

[10] C. V. Samak, T. V. Samak, and S. Kandhasamy, "Control Strategies for Autonomous Vehicles," Autonomous Systems Laboratory, Department of Mechatronics Engineering, SRM Institute of Science and Technology, 2021. [Online].

Available: http://dx.doi.org/10.1201/9781003048381 [Accessed: Sep. 25, 2022].

[11] Y. Kebbati, N. Ait-Oufroukh, V. Vigneron, D. Ichalal, and D. Gruyer, "Optimized self-adaptive PID speed control for autonomous vehicles," in 2021 26th International Conference on Automation and Computing (ICAC), Portsmouth, United Kingdom: IEEE, Sep. 2021, pp. 1–6. [Online]. Available: http://dx.doi.org/10.23919/ICAC50006.2021.9594131 [Accessed: Sep. 21, 2022].

[12] S. Tippannavar, S. Jain, R. Harshith, and Y. S D, "Stanley Controller based Autonomous Path planning and Tracking in Self-Driving Cars," *International Journal of Innovative Research in Advanced Engineering*, vol. 10, pp. 40–48, Mar. 2023. [Online]. Available: http://dx.doi.org/10.26562/ijirae.2023.v1003.01 [Accessed: May 17, 2023].

[13] A. AbdElmoniem, A. Osama, M. Abdelaziz, and S. A. Maged, "A path-tracking algorithm using predictive Stanley lateral controller," *International Journal of Advanced Robotic Systems*, vol. 17, no. 6, Nov. 2020. [Online]. Available: http://dx.doi.org/10.1177/1729881420974852 [Accessed: Sep. 28, 2022].

[14] S. F. Campbell, "Steering control of an autonomous ground vehicle with application to the DARPA Urban Challenge," Massachusetts Institute of Technology, 2007. Accessed: May 24, 2023. [Online]. Available: https://dspace.mit.edu/handle/1721.1/42301 [Accessed: Sep. 26, 2022].

[15] D. S. Lal, A. Vivek, and G. Selvaraj, "Lateral control of an autonomous vehicle based on Pure Pursuit algorithm," in 2017 International Conference on Technological Advancements in Power and Energy (TAP Energy), Dec. 2017, pp. 1–8. [Online]. Available: http://dx.doi.org/10.1109/TAPENERGY.2017.8397361 [Accessed: Sep. 26, 2022].

[16] R. Wang, Y. Li, J. Fan, T. Wang, and X. Chen, "A Novel Pure Pursuit Algorithm for Autonomous Vehicles Based on Salp Swarm Algorithm and Velocity Controller," IEEE Access, vol. 8, pp. 166525–166540, Jan. 2020. [Online]. Available: http://dx.doi.org/10.1109/ACCESS.2020.3023071 [Accessed: Sep. 26, 2022].

[17] T. Yang, Z. Bai, Z. Li, N. Feng, and L. Chen, "Intelligent Vehicle Lateral Control Method Based on Feedforward + Predictive LQR Algorithm," *Actuators*, vol. 10, no. 9, Sep. 2021. [Online]. Available: http://dx.doi.org/10.3390/act10090228 [Accessed: Sep. 21, 2022].

[18] J. Chen, W. Zhan, and M. Tomizuka, "Constrained iterative LQR for on-road autonomous driving motion planning," in 2017 IEEE 20th International Conference on Intelligent Transportation, Oct. 2017, pp. 1–7. [Online]. Available: http://dx.doi.org/10.1109/ITSC.2017.8317745 [Accessed: Sep. 21, 2022].

[19] J. Ma, Z. Cheng, X. Zhang, M. Tomizuka, and T. H. Lee, “Alternating Direction Method of Multipliers for Constrained Iterative LQR in Autonomous Driving,” *arXiv*, Jul. 27, 2022. [Online]. Available: http://dx.doi.org/10.48550/arXiv.2011.00462 [Accessed: Sep. 21, 2022].

[20] H. Friji, H. Ghazzai, H. Besbes, and Y. Massoud, “A DQN-Based Autonomous Car-Following Framework Using RGB-D Frames,” in 2020 IEEE Global Conference on Artificial Intelligence and Internet of Things (GCAIoT), Dec. 2020, pp. 1–6. [Online]. Available: https://doi.org/10.1109/GCAIoT51063.2020.9345899 [Accessed: Sep. 26, 2022].

[21] Y. Savid, R. Mahmoudi, R. Maskeliūnas, and R. Damaševicius, "Simulated Autonomous Driving Using Reinforcement Learning: A Comparative Study on Unity's ML-Agents Framework," *informations*, 2023. [Online]. Available: https://doi.org/10.3390/info14050290 [Accessed: May 17, 2023].

[22] G. Wu et al., “Dyna-PPO reinforcement learning with Gaussian process for the continuous action decision-making in autonomous driving,” *Appl Intell*, Dec. 2022. [Online]. Available: https://doi.org/10.1007/s10489-022-04354-x [Accessed: May 14, 2023].

[23] A. J. M. Muzahid, S. F. Kamarulzaman, and M. A. Rahman, “Comparison of PPO and SAC Algorithms Towards Decision Making Strategies for Collision Avoidance Among Multiple Autonomous Vehicles,” in 2021 International Conference on Software Engineering Computer Systems and 4th International Conference on Computational Science and Information Management (ICSECS-ICOCSIM), Aug. 2021, pp. 200–205. [Online]. Available: https://doi.org/10.1109/ICSECS52883.2021.00043 [Accessed: Sep. 28, 2022].

[24] A. Reda and J. Vásárhelyi, “Design and Implementation of Reinforcement Learning for Automated Driving Compared to Classical MPC Control,” *Designs*, vol. 7, no. 1, Feb. 2023. [Online]. Available: http://dx.doi.org/10.3390/designs7010018 [Accessed: May 14, 2023].

[25] F. Curinga, “Autonomous racing using model predictive control,” KTH Royal Institue of Technology, School of Electrical Engineering, 2018. [Online]. Available: http://www.diva-portal.org/smash/get/diva2:1182133/FULLTEXT01.pdf [Accessed: Sep. 21, 2022].

[26] S. Nekkah et al., “The Autonomous Racing Software Stack of the KIT19d.” *arXiv*, Oct. 06, 2020. [Online]. Available: http://arxiv.org/abs/2010.02828 [Accessed: Sep. 21, 2022].

[27] C. Jung, S. Lee, H. Seong, A. Finazzi, and D. H. Shim, “Game-Theoretic Model Predictive Control with Data-Driven Identification of Vehicle Model for Head-to-Head Autonomous Racing,” *arXiv*, Jun. 08, 2021. [Online]. Available: 10.48550/arXiv.2106.04094 [Accessed: Sep. 21, 2022].

[28] AMZ Team, “Home | AMZ Racing,” AMZ Racing. [Online]. Available: https://www.amzracing.ch/en/node/1255 [Accessed: May 14, 2023].

[29] AMZ Team, “gotthard,” AMZ Racing. [Online]. Available: https://www.amzracing.ch/en/team/2016 [Accessed: May 14, 2023].

[30] J. Kabzan, L. Hewing, A. Liniger and M. N. Zeilinger, "Learning-Based Model Predictive Control for Autonomous Racing,", *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3363-3370, Oct. 2019. [Online]. Available: http://dx.doi.org/10.1109/LRA.2019.2926677 [Accessed: Oct. 9, 2022].

[31] U. Rosolia, A. Carvalho, and F. Borrelli, “Autonomous Racing using Learning Model Predictive Control.” arXiv, Nov. 08, 2017. [Online]. Available: http://dx.doi.org/10.48550/arXiv.1610.06534 [Accessed: Oct. 9, 2022].

[32] M. L. C. Vianna, E. Goubault, and S. Putot, “Neural Network Based Model Predictive Control for an Autonomous Vehicle,” *arXiv*, Jul. 30, 2021. [Online]. Available: http://dx.doi.org/10.48550/arXiv.2107.14573 [Accessed: Oct. 15, 2022].

[33] A. Aswani, H. Gonzalez, S. S. Sastry, and C. Tomlin, “Provably Safe and Robust Learning-Based Model Predictive Control,” *arXiv*, Aug. 03, 2012. [Online]. Available: http://dx.doi.org/10.48550/arXiv.1107.2487 [Accessed: Sep. 22, 2022].

[34] Y. Gao, “Model Predictive Control for Autonomous and Semiautonomous Vehicles,” University of California, Berkeley, 2014. [Online]. Available: https://escholarship.org/content/qt8xd0b56h [Accessed: Oct. 9, 2022].

[35] K. hu and K. Cheng, “Robust Tube-Based Model Predictive Control for Autonomous Vehicle Path Tracking,” *IEEE Access*, vol. PP, pp. 1–1, Jan. 2022. [Online]. Available: http://dx.doi.org/10.1109/ACCESS.2022.3231443 [Accessed: Sep. 23, 2022].

[36] A. Wischnewski, T. Herrmann, F. Werner, and B. Lohmann, “A Tube-MPC Approach to Autonomous Multi-Vehicle Racing on High-Speed Ovals,” IEEE Transactions on Intelligent Vehicles, vol. 8, no. 1, pp. 368–378, Jan. 2023. [Online]. Available: http://dx.doi.org/10.1109/TIV.2022.3169986 [Accessed: May 14, 2023].

[37] B. A. Hernandez Vicente, P. A. Trodden, and S. R. Anderson, “Fast Tube Model Predictive Control for Driverless Cars Using Linear Data-Driven Models,” *IEEE Transactions on Control Systems Technology*, vol. 31, no. 3, pp. 1395–1410, May 2023. [Online]. Available: http://dx.doi.org/10.1109/TCST.2022.3224089 [Accessed: May 14, 2023].

[38] Y. Dai and D. Wang, “A Tube Model Predictive Control Method for Autonomous Lateral Vehicle Control Based on Sliding Mode Control,” Sensors, vol. 23, no. 8, Art. no. 8, Jan. 2023. [Online]. Available: http://dx.doi.org/10.3390/s23083844 [Accessed: May 14, 2023].

[39] N. Chowdhri, L. Ferranti, F. S. Iribarren, and B. Shyrokau, “Integrated nonlinear model predictive control for automated driving,” Control Engineering Practice, vol. 106, p. 104654, Jan. 2021. [Online]. Available: http://dx.doi.org/10.1016/j.conengprac.2020.104654 [Accessed: Sep. 25, 2022].

[40] J. P. Allamaa, P. Listov, H. Van der Auweraer, C. Jones, and T. D. Son, “Real-time Nonlinear MPC Strategy with Full Vehicle Validation for Autonomous Driving,” in 2022 American Control Conference (ACC), Jun. 2022, pp. 1982–1987. [Online]. Available: http://dx.doi.org/10.23919/ACC53348.2022.9867514 [Accessed: Sep. 25, 2022].

[41] T. M. Vu, R. Moezzi, J. Cyrus, and J. Hlava, “Model Predictive Control for Autonomous Driving Vehicles,” *Electronics*, vol. 10, no. 21, p. 2593, Oct. 2021. [Online]. Available: http://dx.doi.org/10.3390/electronics10212593 [Accessed: Sept. 17, 2022].

[42] V. Cataffo, G. Silano, L. Iannelli, V. Puig, and L. Glielmo, “A Nonlinear Model Predictive Control Strategy for Autonomous Racing of Scale Vehicles,” in 2022 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Oct. 2022, pp. 100–105. [Online]. Available: http://dx.doi.org/10.1109/SMC53654.2022.9945279 [Accessed: May 17, 2023].

[43] A. Zanelli, R. Quirynen, J. Jerez, and M. Diehl, “A Homotopy-based Nonlinear Interior-Point Method for NMPC,” *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 13188–13193, Jul. 2017. [Online]. Available: http://dx.doi.org/10.1016/j.ifacol.2017.08.2175 [Accessed: Sep. 22, 2022].

[44] E. Alcalá, V. Puig, and J. Quevedo, “LPV-MPC Control for Autonomous Vehicles,” *IFAC-PapersOnLine*, vol. 52, no. 28, pp. 106–113, 2019. [Online]. Available: http://dx.doi.org/10.1016/j.ifacol.2019.12.356 [Accessed: Sep. 19, 2022].

[45] M. Misin and V. Puig, “LPV MPC Control of an Autonomous Aerial Vehicle,” in 2020 28th Mediterranean Conference on Control and Automation (MED), Sep. 2020, pp. 109–114. [Online]. Available: http://dx.doi.org/10.1109/MED48518.2020.9183041 [Accessed: Oct. 9, 2022].

[46] F. K. Pour, V. Puig, and C. Ocampo-Martinez, "Model predictive control based on LPV models with parameter-varying delays," *scic.es*, Universitat Politècnica de Catalunya, Institut de Robòtica i Informàtica Industrial (CSIC-UPC), Barcelona, 2022. [Online]. Available: https://www.google.com/2ahUKEwjX9taL6JD [Accessed: Sep. 28, 2022].

[47] J. Cao, C. Song, S. Peng, S. Song, X. Zhang and F. Xiao, "Trajectory Tracking Control Algorithm for Autonomous Vehicle Considering Cornering Characteristics," *IEEE Access*, vol. 8, pp. 59470-59484, 2020. [Online]. Available: http://dx.doi.org/10.1109/ACCESS.2020.2982963 [Accessed: Oct. 9, 2022].

[48] L. Hewing, A. Liniger, and M. N. Zeilinger, “Cautious NMPC with Gaussian Process Dynamics for Autonomous Miniature Race Cars,” in 2018 European Control Conference (ECC), Limassol: IEEE, Jun. 2018, pp. 1341–1348. [Online]. Available: http://dx.doi.org/10.23919/ECC.2018.8550162 [Accessed: Oct. 9, 2022].

[49] R. R. Arany, "Gaussian Process Model Predictive Control for Autonomous Driving in Safety-Critical Scenarios," Department of Electrical Engineering, Linköping University, 2019. [Online]. Available: https://liu.diva-portal.org/smash/get/diva2:1372269/FULLTEXT01.pdf [Accessed: Sep. 25, 2022].

[50] W. Liu, C. Liu, G. Chen, and A. Knoll, “Gaussian Process Based Model Predictive Control for Overtaking in Autonomous Driving,” *Frontiers in Neurorobotics*, vol. 15, 2021. [Online]. Available: http://dx.doi.org/10.3389/fnbot.2021.723049 [Accessed: Sep. 22, 2022].

[51] D. Lam, C. Manzie and M. Good, "Model predictive contouring control," in 49th IEEE Conference on Decision and Control (CDC), Atlanta, GA, USA, 2010, pp. 6137-6142. [Online]. Available: http://dx.doi.org/10.1109/CDC.2010.5717042 [Accessed: Sept. 17, 2022].

[52] B. Brito, B. Floor, L. Ferranti, and J. Alonso-Mora, “Model Predictive Contouring Control for Collision Avoidance in Unstructured Dynamic Environments,” *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4459-4466, Oct. 2019. [Online]. Available: http://dx.doi.org/10.1109/LRA.2019.2929976 [Accessed: Sept. 17, 2022].

[53] J. K. Fredlund and K. S. Sulejmanovic, “Autonomous driving using Model Predictive Control methods," Department of Automatic Control, Lund University, 2017. [Online]. Available: http://lup.lub.lu.se/student-papers/record/8906188 [Accessed: May 23, 2022].

[54] A. Liniger, A. Domahidi, and M. Morari, “Optimization-Based Autonomous Racing of 1:43 Scale RC Cars,” Optim. Control Appl. Meth., vol. 36, no. 5, pp. 628–647, Sep. 2015. [Online]. Available: http://dx.doi.org/10.1002/oca.2123 [Accessed: Sep. 28, 2022].

[55] D. Abaffyová and A. Gordon, "Boundary Estimation and Model Predictive Contouring Control," Faculty of Engineering Technology, KU Leuven, 2020.

[56] J. Kabzan et al., “AMZ Driverless: The Full Autonomous Racing System.” *arXiv*, May 13, 2019. [Online]. Available: http://dx.doi.org/10.48550/arXiv.1905.05150 [Accessed: Sep. 25, 2022].

[57] “Delaunay triangulation,” Wikipedia. May 11, 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Delaunay_triangulation&oldid=1154286516 [Accessed: May 21, 2023].

[58] Mapbox, *Delaunator*. 2021. [Online]. Available: Available:https://github.com/mapbox/delaunator [Accessed: May 30, 2022].

[59] Michael, "Vehicle Dynamics: The Kinematic Bicycle Model," Sep. 21, 2020. [Online]. Available: https://thef1clan.com/2020/09/21/vehicle-dynamics-the-kinematic-bicycle-model/ [Accessed: Dec. 23, 2022].

[60] Driverless Race Car Control, "Path Planning Performance," *Youtube*, May 16, 2023. [Online video]. Available: https://youtu.be/OK0PyTViO88[Accessed: June 3, 2023].

[61] Michael, "Vehicle Dynamics: The Dynamic Bicycle Model," Dec. 23, 2020. [Online]. Available: https://thef1clan.com/2020/12/23/vehicle-dynamics-the-dynamic-bicycle-model/ [Accessed: Dec. 23, 2022].

[62] A. Garcia, "Pacejka '94 parameters explained – a comprehensive guide," Edy's Projects, Dec. 24, 2011. [Online]. Available: https://www.edy.es/dev/docs/pacejka-94-parameters-explained-a-comprehensive-guide/ [Accessed: Jan 18, 2023].

[63] M. Blundell and D. Harty, "Chapter 5 - Tyre Characteristics and Modelling," in *The Multibody Systems Approach to Vehicle Dynamics* (Second Edition), 2015, pp. 335–450. [Online]. Available: http://dx.doi.org/10.1016/B978-0-08-099425-3.00005-4 [Accessed: Jan 18, 2023].

[64] A. Domahidi and J. Jerez, "FORCES Professional," *Embotech AG*, 2014. [Online]. Available: https://embotech.com/FORCES-Pro [Accessed: Mar. 13, 2023].

[65] A. Domahidi and J. Jerez, "9.1. Supported Problems — FORCESPRO 6.1.0 documentation," *Embotech AG*, 2023. [Online]. Available: https://forces.embotech.com/Documentation/high_level_interface/index.html#supported-problems [Accessed: Mar. 13, 2023].

[66] Driverless Race Car Control, "Path Following Benchmark: Track 3.2," *Youtube*, Jun. 2, 2023. [Online video]. Available: https://youtu.be/ksDTstk4Ky4 [Accessed: June 3, 2023].

[67] Driverless Race Car Control, "Path Following Benchmark: Track 5.1," *Youtube*, Jun. 2,, 2023. [Online video]. Available: https://youtu.be/5EkK2OWQcwA [Accessed: June 3, 2023].

[68] Driverless Race Car Control, "Path Following Benchmark: Track 5.2," *Youtube*, Jun. 2, 2023. [Online video]. Available: https://youtu.be/dpxRf0J0NbM [Accessed: June 3, 2023].

[69] Driverless Race Car Control, "Control Algorithm Performance - Track 1," *Youtube*, Jun. 2,, 2023. [Online video]. Available: https://youtu.be/Kd7FYdpAWsc [Accessed: June 3, 2023].

[70] Driverless Race Car Control, "Control Algorithm Performance - Track 2," *Youtube*, May 16, 2023. [Online video]. Available: https://youtu.be/60PzyAsCCHE [Accessed: June 3, 2023].

[71] OpenAI, "ChatGPT (Version Used: GPT-3.5)," *AI language model*. [Online]. Available: https://chat.openai.com. [Accessed: Jun. 5, 2023].

[72] DeepL GmbH, "DeepL Translator," [Online]. Available: https://www.deepl.com. [Accessed: Jun. 5, 2023].

[73] Zotero, "Zotero | Your personal research assistant," [Online]. Available: https://www.zotero.org/. [Accessed: Jun. 5, 2023].

Appendix A | Path Planning testbench design and implementation

In this appendix, the design and implementation of the PP algorithm's testbench, constituted of the track builder and SLAM output replicator, is described.

Track builder

The first function of the testbench is a track builder. By giving some parameters, a track respecting the specifications listed in the problem formulation must be created.

The track can be divided into two types of sequence: a straight line and a curve. A straight line is defined by a length L and an orientation ϕ, while a curve is defined by a curvature radius r, an initial orientation ϕ_i and a final orientation ϕ_f, as illustrated in Figure A.1. In addition, the space between the points formed by the midpoints of the pairs of cones dL is given. For a straight line, this space directly represents the length between two cones on the same side of the track. Whereas for a curve, this dL will be use along side with the orientation of the currently considered midpoint to determine the positions of the interior and exterior cones of the curve.

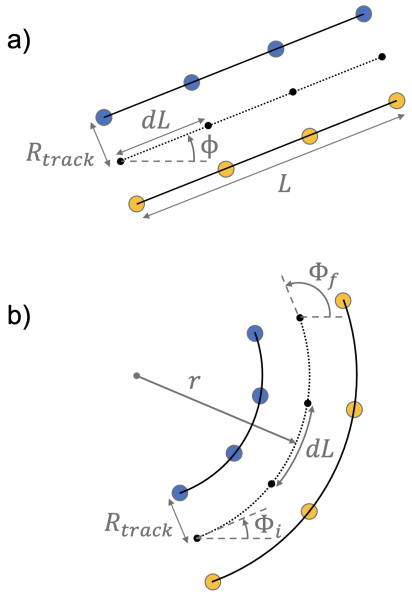

Figure A.1: Definition of the 2 sequences of the track - a) straight line, b) curve

Therefore, the input of the track builder is a list of sequences, with each sequence defined by 5 parameters:

1. indicator of the type of sequence: 0 for a straight line, 1 for a curve;
2. initial orientation: ϕ for a straight line: ϕ_i for a curve;

3. final orientation: 0 for a straight line (same as initial orientation, thus no interest), ϕ_f for a curve;

4. length L for a straight line, curvature radius r for a curve;

5. space between midpoints dL.

The initial midpoint position of the first sequence is provided to determine where to begin constructing the track. Additionally, the half-width of the track, denoted as R_{track} and treated as a global parameter for both sequences, is used to establish the gap between the side cones and the centerline. Subsequently, each sequence is constructed, starting from the provided initial position for the first sequence, or from the last midpoint position of the previous sequence for subsequent sequences. From the initial midpoint, the sequences can be constructed.

a) Straight line:

The number of cones on each side necessary to build the sequence is $n = \frac{L}{dL} + 1$. L must thus be a multiple of dL. If this is the first sequence, the positions of the first left cone and the first right cone is computed using the following equation, with (x_0, y_0) the position of the first midpoint:

$$\begin{cases} x_{left,0} &= x_0 + R_{track} \cos\left(\phi + \frac{\pi}{2}\right) \\ y_{left,0} &= y_0 + R_{track} \sin\left(\phi + \frac{\pi}{2}\right) \end{cases}$$

$$\begin{cases} x_{right,0} &= x_0 - R_{track} \cos\left(\phi + \frac{\pi}{2}\right) \\ y_{right,0} &= y_0 - R_{track} \sin\left(\phi + \frac{\pi}{2}\right) \end{cases}$$

If this is not the first sequence, the positions of the first left cone and right cone are the positions of the previous sequence's last left and right cones. Then the next cones positions are computed based on the previous one's position,

$$\begin{cases} x_{left,k} &= x_{left,k-1} + dL \cos(\phi) \\ y_{left,k} &= y_{left,k-1} + dL \sin(\phi) \end{cases} \quad \text{for } k = 1, ..., n-1$$

$$\begin{cases} x_{right,k} &= x_{right,k-1} + dL \cos(\phi) \\ y_{right,k} &= y_{right,k-1} + dL \sin(\phi) \end{cases} \quad \text{for } k = 1, ..., n-1$$

b) Curve:

The chord in the center of the curve, the interior and exterior chord are first computed,

$$\begin{cases} c & = |\phi_f - \phi_i| r \\ c_{int} & = |\phi_f - \phi_i| (r - R_{track}) \\ c_{ext} & = |\phi_f - \phi_i| (r + R_{track}) \end{cases}$$

The number of cones on each side necessary to build the sequence is $n = \frac{c}{dL} + 1$ converted in integer. The new dL adapted to the conversion is $dL = \frac{c}{n}$ and the interior and exterior dL are $dL_{int} = \frac{c_{int}}{n}$ and $dL_{ext} = \frac{c_{ext}}{n}$ respectively.

The difference in orientation between each cone is $d\phi = \frac{(\phi_f - \phi_i)}{n}$. If $d\phi > 0$, this is a left curve and the left cones form the interior curve ($dL_{left} = dL_{int}$ and $dL_{right} = dL_{ext}$). If $d\phi < 0$, this is a right curve and the right cones form the interior curve ($dL_{right} = dL_{int}$ and $dL_{left} = dL_{ext}$). The position of the successive cones on each side can then be computed,

$$\begin{cases} \phi_k & = \phi_{k-1} + d\phi \\ x_{left,k} & = x_{left,k-1} + dL_{left} \cos(\phi_k) \\ y_{left,k} & = y_{left,k-1} + dL_{left} \sin(\phi_k) \\ x_{right,k} & = x_{right,k-1} + dL_{right} \cos(\phi_k) \\ y_{right,k} & = y_{right,k-1} + dL_{right} \sin(\phi_k) \end{cases} \quad fork = 1, ..., n-1$$

Finally, the four orange cones must be positioned among the other cones. Their respective positions are given to be added to the lists of left and right cones. The sequence following the orange cones takes the midpoint of the two first orange cones as starting point.

Illustration of a track created with the track builder is represented in Figure A.2. The oranges cones have been positioned $6\,m$ in front of the starting position of the car $(0, 0)$. Four sequences are necessary since the orange cones divides the first straight line,

1. sequence $= 0, 0, 0, 3, 3$;
2. sequence $= 0, 0, 0, 3, 3$;
3. sequence $= 1, 0, \frac{\pi}{2}, 6, 3$;
4. sequence $= 0, \frac{\pi}{2}, 0, 9, 3$.

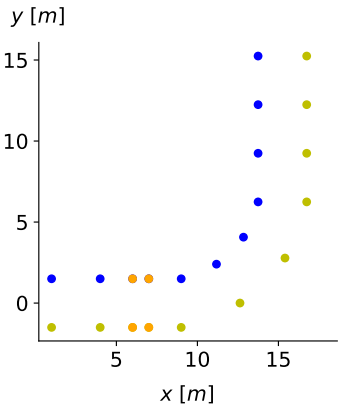

Figure A.2: Example of track created by the track builder

SLAM output replicator

The second function of the testbench is to replicate the output of the SLAM algorithm with the difference that the position and orientation of the car is given and not computed. Therefore, based on the cones of the track generated by the track builder and the position and orientation of the car, a list of cones in the detection range in front of the car and a list of cones in the detection range behind the car must be returned.

A possible situation is illustrated in Figure A.3. The car has a position (x, y) and an orientation ϕ and the orange dotted line splits the detection range circle into a rear detection area and a front detection area. The cones in the front detection area are considered as cones in front of the car and cones in the rear detection area are considered as cones behind the car.

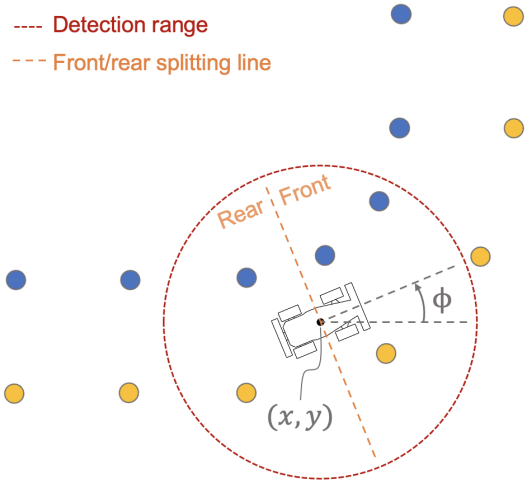

Figure A.3: Definition of the front/rear detection areas and car position and orientation for the SLAM output replicator

The first step is to determine if a cone is in the detection range by computing the distance d between the cone and the car. Then some distances and angles are used to determine the detection zone in which the cone is positioned. They are represented in Figure A.4 considering a cone at position (x_{cone}, y_{cone}).

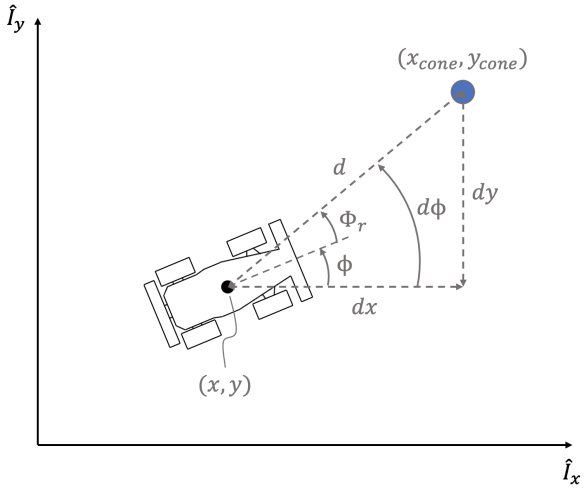

Figure A.4: Distances and angles used to determine the detection zone in which a cone is positioned

The distance d is first computed, and is compared to the defined detection range ($6\,m$) to determine if the cone is detected,

$$d = \sqrt{(x - x_{cone})^2 + (y - y_{cone})^2}$$

If this is the case, the angle formed by the car and cone positions $d\phi$ is determined by,

$$d\phi = \text{arctan2}(dy, dx)$$

and the relative orientation of the cone with respect to the car can be found,

$$\phi_r = d\phi - \phi$$

This angle is then used to determine the detection area of the cone. If $\phi_r \in [-\frac{\pi}{2}; \frac{\pi}{2}]$, the cone is in front of the car (front detection area). Otherwise, the cone is behind the car (rear detection area).

At the end, giving a car's position and orientation and when every cones have been checked, the lists of the cones in front of the car and behind the car are returned in the form of a `ConeArray` structure. By giving successive car positions to simulate a displacement of the car, the different replicated SLAM outputs are generated and the PP algorithm can be validated.

Appendix B | Path Planning algorithm benchmark

Validation of designed standard and special actions		
Main case	Sub case	Expected results
1. Car at starting position (validation of starting line operation)	1. With 0 orange cones detected, only normal cones detected before	• Normal cones not used and not added to detected cones • Initial reference point + virtual reference point at $8\,m$
	2. With 2 first orange cones detected and normal cones detected before	• 2 orange cones registered but starting line operation not finished • Normal cones not used and not added to detected cones • Initial reference point + virtual reference point at $8\,m$
2. After starting line operation executed and only 4 orange cones detected	1. 1 normal cone detected	• 1 triangle generated (validation of triangles generation – special case 2) • 1 reference point added
	2. 1 pair of cone detected	• 2 triangles generated (validation of triangles generation – special case 1) • 2 reference points added
	3. 2 pairs of cones detected in curve	• 4 triangles generated and 1 triangle with an orange cone in it discarded (validation of triangles filtering – special case) • 4 reference points added

3. Starting line operation executed and car just after the starting line for the first time		• Initial reference point removed
4. After starting line operation executed and 1 pair of normal cones detected	1. 1 normal cone detected	• 1 triangle generated (validation of normal addition of 1 cone) • 1 reference point added
	2. 1 pair of cone detected	• 2 triangles generated (validation of normal addition of multiple cones) • 2 reference points added
	3. 2 pairs of cones detected in curve	• 4 triangles generated and 1 triangle discarded (validation of triangles filtering – normal case) • 4 reference points added
5. Starting line operation executed and car at the end of the lap (lap closure validation)	1. Orange cone detected in front	• Lap closure setup

Validation of the global functioning for different tracks
6. Long straight line
7. Large curve (radius $r = 10\,m$)
8. Sharp curve (radius $r = 2\,m$)
9. Complete track

Table B.1: PP algorithm's benchmark

Appendix C | Vehicle kinematic model state equations derivation

To derive the state equations of the kinematic bicycle model, the instantaneous center of rotation is used. The perpendicular lines meeting in the instantaneous center of rotation form two right-angled triangles of small angles β and δ, as shown in Figure C.1, that can be used to express certain quantities.

First of all, the time derivative of the position can easily be determined by,

$$\dot{x} = v\cos(\phi + \beta),$$
$$\dot{y} = v\sin(\phi + \beta)$$

Then, the orientation's evolution is given by the rotational velocity around the z axis ω that can be expressed in terms of the velocity v and radius R,

$$\dot{\phi} = \omega = \frac{v}{R}$$

Using the right-angled triangles, some useful expressions can be developed:

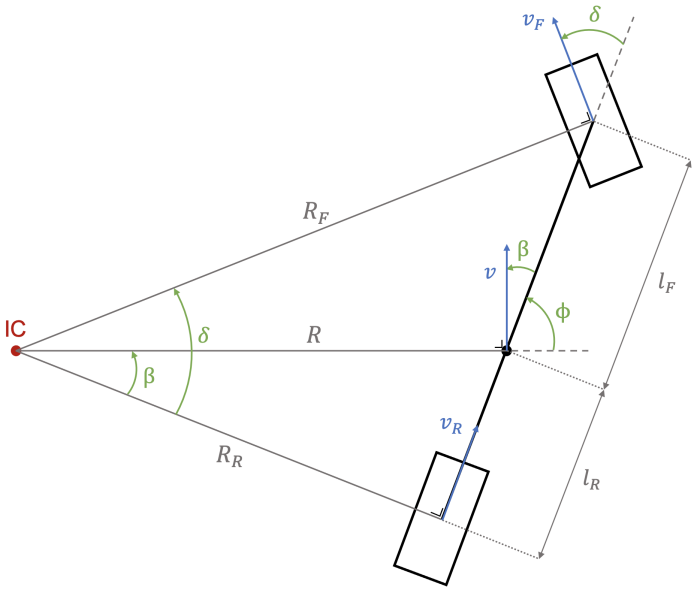

Figure C.1: kinematic bicycle model representation with instantaneous center of rotation (IC)

$$R_R = \frac{l_R + l_F}{\tan(\delta)},$$

$$R = \frac{R_R}{\cos(\beta)} = \frac{l_R + l_F}{\tan(\delta)\cos(\beta)}, \tag{C.1}$$

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

The expression of $\tan(\delta) = \frac{l_R+l_F}{l_R}\tan(\beta)$ coming from equation (C.2) can be put in equation (C.1) to obtain,

$$\dot{\phi} = \frac{v}{\frac{l_R}{\tan(\beta)\cos(\beta)}} = \frac{v}{l_R}\sin(\beta)$$

Finally, β can be expressed in terms of the control command δ with equation (C.2),

$$\beta = \arctan\left(\frac{l_R}{l_R + l_F}\tan(\delta)\right)$$

Appendix D | Optimization Engine solver utilization

This appendix describes the utilization of the *Optimization Engine* (OpEn) solver used for the first developed PF algorithm.

The design and deployment of the embedded optimizer is made easy and rapid thanks to the use of multiple tools for the different steps of the solver generation. The OpEn utilization scheme is represented in Figure D.1. The problem is formulated in Python or MATLAB, then the solver is generated with RUST and it can be utilized either on a TCP Socket Server or in a C/C++ code. In this work, the Python interface is used to formulate the problem and launch the solver generation. This creates the necessary packages and libraries that are then included in the C++ code to use the solver in real-time.

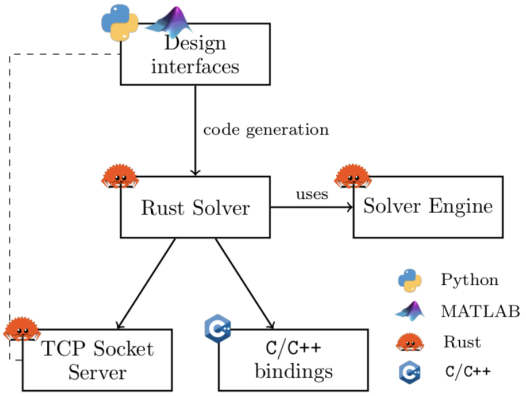

Figure D.1: OpEn utilization scheme

OpEn can solve parametric non-convex optimization problems of the form,

$$\begin{aligned} \underset{\boldsymbol{u}\in\mathbb{R}^{N_u}}{\text{minimize}} \quad & f(\boldsymbol{u},\boldsymbol{p}) \\ \text{subject to} \quad & \boldsymbol{u}\in U \\ & F_1(\boldsymbol{u},\boldsymbol{p})\in C \\ & F_2(\boldsymbol{u},\boldsymbol{p})=0 \end{aligned}$$

where $\boldsymbol{u}\in\mathbb{R}^{N_u}$ is the vector decision variables (control commands vector) of the problem and $\boldsymbol{p}\in\mathbb{R}^{n_p}$ is a vector of parameters. This is a very flexible problem formulation that allows the user to model a very broad class of optimization problems.

Problems that involve a smooth cost function, $f(\boldsymbol{u},\boldsymbol{p})$ and simple constraints of the form $\boldsymbol{u}\in U$ are solved using Proximal Averaged Newton-type method for Optimal

Control (PANOC). In order to solve problems with more complex constraints such as $F_1(\boldsymbol{u}, \boldsymbol{p}) \in C$ and $F_2(\boldsymbol{u}, \boldsymbol{p}) = 0$, the augmented Lagrangian and penalty methods are respectively used.

A very wide range of constraints can be modeled with the $F_1(\boldsymbol{u}, \boldsymbol{p}) \in C$ constraints by choosing different sets C.

- Equality constraints, $F_1(\boldsymbol{u}, \boldsymbol{p}) = 0$ can be described by taking $C = \{0\}$.
- Element-wise constraints of the form,

$$f_{min} \leq F_1(\boldsymbol{u}, \boldsymbol{p}) \leq f_{max},$$

  can be described by taking C to be a rectangle.

The second constraint $F_2(\boldsymbol{u}, \boldsymbol{p}) = 0$ can be used to encode either equality or inequality constraints of the form $h(\boldsymbol{u}, \boldsymbol{p}) \leq 0$ using,

$$F_2(\boldsymbol{u}, \boldsymbol{p}) = max\{0, h(\boldsymbol{u}, \boldsymbol{p})\}$$

The designed MPCC problem formulation can now be applied to the OpEn solver formulation. The problem is formulated in Python using symbolic equations with CasADi library. In the solver formulation, the only variables differentiation is between the decision variables $\boldsymbol{u}$ and the other variables gathered in the parameter vector $\boldsymbol{p}$. No distinction is made between state variables and parameters contrary to what was defined in Subsection **??**.

- $\boldsymbol{u} \in \mathbb{R}^{N_u}$ is the whole sequence of control commands symbolically generated as a vector of size $N_u = N \cdot n_u$, with N the prediction horizon and $n_u = 3$ the number of control commands as defined in Subsection **??**. Its this vector that will be the output of the optimization.
- $\boldsymbol{p} \in \mathbb{R}^{n_p}$ regroups all the other variables and parameters of the problem in the form of a vector of size n_p. Its initial value must be given to the solver when it is called, then the vector is updated at each prediction k. It is the only input of the solver.

  It is composed of:

  - The state vector $\boldsymbol{x}$ of size $n_x = 5$;
  - The contouring parameters vector $\boldsymbol{c}$ of size $n_c = 2 + 2(m + 1)$;
  - The errors vector $\boldsymbol{\epsilon} = [\hat{\epsilon}^l \ \hat{\epsilon}^c]^T \in \mathbb{R}^{n_\epsilon}$ with $n_\epsilon = 2$. This vector has been added to the parameters because the contouring error $\hat{\epsilon}^c$ is used at two different places: the objective function (3.28) and the track constraint (3.27). By adding it in $\boldsymbol{p}$, it must only be computed one time when $\boldsymbol{p}$ is updated;
  - The weights vector $\boldsymbol{j}$ of size $n_j = 9$;

- The previous control commands vector $\boldsymbol{u}_{prev} = [\delta_{prev}\ D_{prev}\ v_{s,prev}]^T \in \mathbb{R}^{n_u}$ with $n_u = 3$. This vector has been added to the parameters to be able to provide the initial control commands $\boldsymbol{u}_0$ to the solver.

The parameters vector $\boldsymbol{p}$ is therefore of size $n_p = n_x + n_c + n_\epsilon + n_j + n_u$.

The simple constraints $\boldsymbol{u} \in U$ are used to model the control commands constraints (3.25), where $U = [\underline{\boldsymbol{u}}, \overline{\boldsymbol{u}}]$ is defined as a rectangle. No constraints are applied on the state.

The constraints $F_1(\boldsymbol{u}, \boldsymbol{p}) \in C$ are used to model the control commands rate of change constraints (3.26), where $C = [f_{min}, f_{max}]$ is defined as a rectangle, with $f_{min} = \underline{\boldsymbol{\Delta u}}$ and $f_{max} = \overline{\boldsymbol{\Delta u}}$, and $F_1(\boldsymbol{u}, \boldsymbol{p}) = \boldsymbol{\Delta u}$.

Finally, the constraints $F_2(\boldsymbol{u}, \boldsymbol{p}) = max\{0, h(\boldsymbol{u}, \boldsymbol{p})\}$ are used to model the track constraints (3.27), by defining $h(\boldsymbol{u}, \boldsymbol{p}) = (\hat{\epsilon}_k^c)^2 - R_c^2$.

To symbolically create the cost function $J = f(\boldsymbol{u}, \boldsymbol{p})$ and the constraints, by using the control commands sequence $\boldsymbol{u}$ and parameters vector $\boldsymbol{p}$ that must be updated at each prediction k, the procedure described in algorithm 2 is followed.

Algorithm 2 OpEn solver - symbolic equation generation procedure

$J \leftarrow 0$
$F_2 \leftarrow [\ ]$
for k in range(0, N) **do**
    J += `stage_cost(`$\boldsymbol{p}, \boldsymbol{u}[k * n_u : k * n_u + n_u]$`)`
    $F_2 \leftarrow$ `F2_update(`$F_2, \boldsymbol{p}$`)`
    $\boldsymbol{p} \leftarrow$ `p_model(`$\boldsymbol{p}, \boldsymbol{u}[k * n_u : k * n_u + n_u]$`)`
end for

$F_1 \leftarrow$ `F1_init(`$\boldsymbol{p}, \boldsymbol{u}[0, n_u]$`)`
for k in range(0, $N - 1$) **do**
    $F_1 \leftarrow$ `F1_update(`$F_1, \boldsymbol{u}[k * n_u : (k + 1) * n_u + n_u]$`)`
end for

The `stage_cost()` function generates the stage cost function J_k (3.32) with the $\boldsymbol{p}$ and $\boldsymbol{u}$ at prediction k.
The constraints F_2 must be applied to the whole sequence. The `F2_update()` function add the constraints of the current prediction k to F_2.
Then at the end of the first loop, $\boldsymbol{p}$ is updated by the `p_model()` function. The new state is computed with the current control commands at prediction k, as well as the new errors and the weights are updated is weights that vary with the prediction k are wanted.

Finally, the `F1_init()` function initializes the F_1 constraints with the first control commands and the previous control commands stored in $\boldsymbol{p}$, and the `F1_update()`

function is used in a second loop to add the constraints of the next predictions to F_1.

Appendix E | Vehicle dynamic model computation

This appendix details the computation of the dynamic bicycle model represented in Figure E.1.

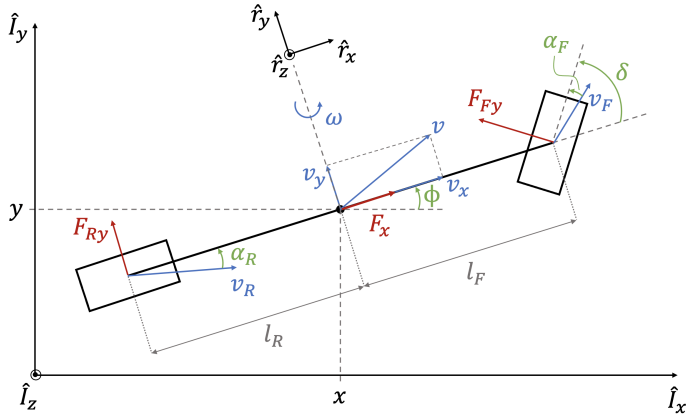

Figure E.1: Dynamic bicycle model representation

The change of absolute position of the car is described by the v_x, v_y velocities and the car's orientation ϕ,

$$\dot{x} = v_x \cos(\phi) - v_y \sin(\phi), \tag{E.1}$$
$$\dot{y} = v_x \sin(\phi) + v_y \cos(\phi), \tag{E.2}$$

and the change of orientation by the rotational velocity ω,

$$\dot{\phi} = \omega \tag{E.3}$$

The rate of change of the tree velocities $\dot{v}_x$, $\dot{v}_y$ and $\dot{\omega}$ must now be expressed. The equations of motion in the inertial frame, involving the forces and moments acting on the car, are used,

$$\sum \boldsymbol{F} = m\boldsymbol{a} \tag{E.4}$$
$$\sum \boldsymbol{M} = \boldsymbol{I}\dot{\omega} \tag{E.5}$$

where $\boldsymbol{F}$ is the force vector, $\boldsymbol{a}$ the acceleration vector, $\boldsymbol{M}$ the vector of moments and $\boldsymbol{I}$ the matrix of the moment of inertia at the center of mass.

(E.4): Since the model is in two dimensions, the equation can be re-written,

$$\sum \begin{bmatrix} F_x \\ F_y \end{bmatrix} = m \begin{bmatrix} a_x \\ a_y \end{bmatrix} \tag{E.6}$$

The accelerations are first developed. Unit vectors $\boldsymbol{i}$ and $\boldsymbol{j}$ are defined along the x and y in the relative frame to express the velocity at the center of mass,

$$\boldsymbol{v} = v_x \boldsymbol{i} + v_y \boldsymbol{j}, \tag{E.7}$$

and unit vector $\boldsymbol{k}$ defined along the z axis to expressed the instantaneous rotational velocity vector of frame $\hat{\boldsymbol{r}}$ in $\hat{\boldsymbol{I}}$ at the center of mass,

$$\boldsymbol{\Omega} = \omega \boldsymbol{k} \tag{E.8}$$

The variation as a function of time of the velocity at the center of mass $\boldsymbol{v}$ in the inertial frame is the sum of the relative acceleration and the cross product between the instantaneous rotational velocity $\boldsymbol{\Omega}$ and the velocity $\boldsymbol{v}$,

$$\left(\frac{d\boldsymbol{v}}{dt}\right)_{\hat{\boldsymbol{I}}} = \left(\frac{d\boldsymbol{v}}{dt}\right)_{\hat{\boldsymbol{r}}} + \boldsymbol{\Omega} \times \boldsymbol{v}, \tag{E.9}$$

where,

$$\left(\frac{d\boldsymbol{v}}{dt}\right)_{\hat{\boldsymbol{r}}} = \dot{v}_x \boldsymbol{i} + \dot{v}_y \boldsymbol{j}, \tag{E.10}$$

and,

$$\begin{aligned} \boldsymbol{\Omega} \times \boldsymbol{v} &= \omega \boldsymbol{k} \times (v_x \boldsymbol{i} + v_y \boldsymbol{j}) \\ &= v_x \omega \boldsymbol{j} - v_y \omega \boldsymbol{i} \end{aligned} \tag{E.11}$$

Therefore,

$$\left(\frac{d\boldsymbol{v}}{dt}\right)_{\hat{\boldsymbol{I}}} = \underbrace{(\dot{v}_x - v_y \omega)}_{a_x} \boldsymbol{i} + \underbrace{(\dot{v}_y + v_x \omega)}_{a_y} \boldsymbol{j} \tag{E.12}$$

(E.5): The car is only subject to moment in the z direction applied by the lateral forces $(F_F)_{\hat{r}_y}$ and $(F_R)_{\hat{r}_y}$ at the front and rear wheel respectively,

$$(F_R)_{\hat{r}_y} l_R + (F_F)_{\hat{r}_y} l_F = I_z \dot{\omega}, \tag{E.13}$$

where I_z is the inertia at the center of mass in the z direction.

Equation (E.4) thus becomes,

$$\begin{aligned} \begin{bmatrix} \dot{v}_x - v_y \omega \\ \dot{v}_y + v_x \omega \end{bmatrix} &= \frac{1}{m} \begin{bmatrix} \sum F_x \\ \sum F_y \end{bmatrix} \\ &= \frac{1}{m} \begin{bmatrix} F_x - F_{Fy} \sin(\delta) \\ F_{Ry} + F_{Fy} \cos(\delta) \end{bmatrix} \\ \Leftrightarrow \begin{bmatrix} \dot{v}_x \\ \dot{v}_y \end{bmatrix} &= \begin{bmatrix} \frac{1}{m}(F_x - F_{Fy} \sin(\delta) + m v_y \omega) \\ \frac{1}{m}(F_{Ry} + F_{Fy} \cos(\delta) - m v_x \omega) \end{bmatrix}, \end{aligned} \tag{E.14}$$

and equation (E.5),

$$\dot{\omega} = \frac{1}{I_z}(F_{Fy} l_F \cos(\delta) - F_{Ry} l_R) \tag{E.15}$$

Appendix F | Car projection computation

This appendix details the computation of the car projection into the reference path as illustrated in Figure F.1.

The direction vector of the line $\overrightarrow{AB}$ joining the two points is expressed as,

$$\overrightarrow{AB} = \begin{bmatrix} x_B - x_A \\ y_B - y_A \end{bmatrix} \tag{F.1}$$

$\overrightarrow{AB}$ is parameterized by a parameter $t \in \mathbb{R}$,

$$\overrightarrow{AB} = \begin{bmatrix} x_A + (x_B - x_A)t \\ y_A + (y_B - y_A)t \end{bmatrix}, \tag{F.2}$$

and the projection P is expressed as a point belonging to the line $\overrightarrow{AB}$,

$$P = \begin{bmatrix} x_A + (x_B - x_A)t_P \\ y_A + (y_B - y_A)t_P \end{bmatrix}, \tag{F.3}$$

where t_P is the value of the parameter t describing the position of P into $\overrightarrow{AB}$. Then the point P is the point on the line $\overrightarrow{AB}$ for which the cross product between $\overrightarrow{AB}$ and $\overrightarrow{XP}$ is null,

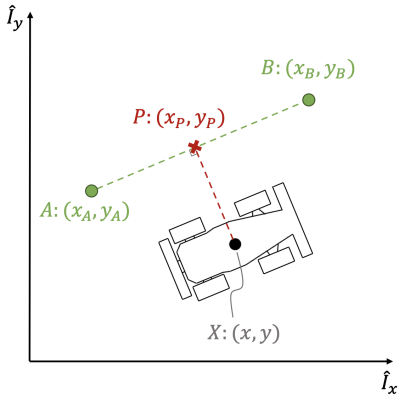

Figure F.1: Representation of the four points A, B, X, P used for the computation of the projection into the reference path

$$\begin{aligned} 0 &= \overrightarrow{XP} \cdot \overrightarrow{AB} \\ 0 &= (x_A + (x_B - x_A)t_P - x)(x_B - x_A) + (y_A + (y_B - y_A)t_P - y)(y_B - y_A) \\ \Leftrightarrow t_P &= -\frac{(x_A - x)(x_B - x_A) + (y_A - y)(y_B - y_A)}{(x_B - x_A)^2 + (y_B - y_A)^2}, \end{aligned} \tag{F.4}$$

and the position of the projection of the car into the reference path is computed by injecting (F.4) into (F.3).

Appendix G | Digital tools

ChatGPT

ChatGPT [71], an AI language model developed by OpenAI based on the GPT-3.5 architecture, was instrumental in helping with the formulation of ideas and improving the overall quality of the text. It provided valuable assistance in generating well-structured and coherent content, contributing to the clarity and effectiveness of the written work.

DeepL

DeepL [72], a language tool, played a valuable role in assisting with the formulation of ideas and improving the quality of the written content. By leveraging its advanced language processing capabilities, DeepL provided valuable support in refining and enhancing the expression of ideas, contributing to the overall clarity and effectiveness of the text.

Zotero

Zotero [73], a powerful reference management tool, played a pivotal role in organizing and enhancing the quality of references in the written work. By leveraging its comprehensive features and intuitive interface, Zotero provided valuable support in efficiently managing and formatting references, contributing to the accuracy and credibility of the research. Its seamless integration with various platforms and easy-to-use functionalities made it an indispensable tool for effective reference management throughout the project.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**