Università degli Studi di Padova

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Department of Information Engineering
Master's Degree in Control Systems Engineering

Improving Driving Performance on a Scaled Autonomous Vehicle with Model Predictive Control

Supervisor:

Prof. Alessandro Beghi
University of Padua

Co-supervisor:

Dr. Mattia Bruschetta
University of Padua

Candidate:

Jona Petrovic
ID 2106875

Academic Year 2024/2025

Graduation date 27.11.2025

Contents

Abstract

This master thesis discusses the modeling and controller design of a 1/10 scaled ground vehicle. In particular, we explore how nonlinear predictive control influences performance in our automotive application. This control methodology is particularly effective when dealing with multiple-input multiple-output systems, as is the case in our thesis, and it allows systematic incorporation of physical and operational constraints in the control design. The vehicle model must balance accuracy and computational efficiency to ensure real-time feasibility, as the chosen model complexity directly influences different performance and goals that can be achieved such as trajectory tracking precision or lap time optimization. Different models are explored, starting from the simplest kinematic bicycle model to the double dynamic bicycle model. In the case of the dynamic model, the benefits of modeling longitudinal and lateral tire forces are considered in different ways. This makes space for better performance in the high speed scenarios and curvature track, but is not suitable for low-speed scenarios. Initially, the models in the time domain are studied, but later on focus is shifted to the spatial domain formulation. This allows easier description of the constraints, and to provide a velocity reference not dependent on time, but rather on the position on the track. The model and the controller were evaluated on a predefined track. This configuration allows the computation of reference trajectories offline, providing a reliable benchmark for trajectory tracking and control performance assessment.

All simulations are conducted in Python utilizing the ACADOS framework, which offers efficient tools for implementing real time Model Predictive Controllers. The tracks used in the simulation are elliptical and S-shaped configurations, selected for their ability to highlight different aspects of the controller's performance, such as cornering stability, trajectory tracking accuracy, and response to curvature variations. Later on, the control architecture is integrated into a ROS2-based system and validated in the Gazebo simulator on tracks from Bosch Future Mobility Challenge (BFMC) environment. The BFMC is an international competition organized by Bosch Engineering Center Cluj where teams of university students develop and implement autonomous driving and connectivity algorithms on 1/10 scale vehicles. Incorporating tracks inspired by this competition allows the simulation framework to better reflect real-world testing conditions in urban-like environments. A hardware-in-the-loop (HIL) configuration is additionally used, where a Raspberry Pi 5 runs the MPC controller. This configuration allows assessment of computational feasibility, actuator-rate limitations, communication delays, and integration effects that cannot be captured in purely offline simulations. Thus, the architecture loop bridges the gap between algorithmic design and physical implementation, ensuring that the controller's performance is validated under realistic constraints imposed by embedded hardware, middleware communication, and physics-based simulation.

Sommario

Questa tesi magistrale tratta la modellizzazione e la progettazione del controllore per un veicolo terrestre in scala \$1/10\$. In particolare, viene analizzato come il controllo predittivo non lineare influenzi le prestazioni nella nostra applicazione automobilistica. Questa metodologia di controllo è particolarmente efficace nella gestione di sistemi multi-input multi-output, come nel nostro caso, e consente di incorporare in modo sistematico i vincoli fisici e operativi nella progettazione del controllore. Il modello del veicolo deve bilanciare accuratezza ed efficienza computazionale per garantire real time feasibility, poiché la complessità del modello scelto influenza direttamente le prestazioni e gli obiettivi raggiungibili, come la precisione nel tracciamento della traiettoria o l'ottimizzazione del tempo sul giro. Vengono esplorati diversi modelli, partendo dal più semplice kinematic bicycle model fino al double dynamic bicycle model. Nel caso del dynamic model, i benefici della modellizzazione delle forze longitudinali e laterali degli pneumatici sono considerati in differenti modalità. Ciò permette migliori prestazioni negli scenari ad alta velocità e nelle piste con elevata curvatura, ma risulta meno adatto agli scenari a bassa velocità. Inizialmente i modelli vengono studiati nel dominio del tempo, ma successivamente l'attenzione si sposta sulla formulazione nel dominio spaziale, che consente una descrizione più semplice dei vincoli e la definizione di un riferimento di velocità non dipendente dal tempo, bensì dalla posizione lungo la pista. Il modello e il controllore sono stati valutati su una pista predefinita. Questa configurazione permette il calcolo offline delle traiettorie di riferimento, fornendo un benchmark affidabile per la valutazione del tracciamento e delle prestazioni del controllore.

Tutte le simulazioni sono condotte in Python utilizzando il framework ACADOS, che offre strumenti efficienti per l'implementazione di controllori predittivi in tempo reale. Le piste utilizzate nelle simulazioni sono configurazioni ellittiche e a forma di S, selezionate per evidenziare diversi aspetti delle prestazioni del controllore, come la stabilità in curva, l'accuratezza nel tracciamento della traiettoria e la risposta a variazioni di curvatura. Successivamente, l'architettura di controllo viene integrata in un sistema basato su ROS2 e validata nel simulatore Gazebo su piste provenienti dall'ambiente della Bosch Future Mobility Challenge (BFMC). La BFMC è una competizione internazionale organizzata dal Bosch Engineering Center Cluj in cui team di studenti universitari sviluppano e implementano algoritmi di guida autonoma e connettività su veicoli in scala 1/10. L'inclusione di piste ispirate a questa competizione permette al framework di simulazione di riflettere meglio condizioni di test reali in ambienti simili a quelli urbani. Viene inoltre utilizzata una configurazione hardware-in-the-loop (HIL), in cui un Raspberry Pi 5 esegue il controllore MPC. Questa configurazione consente di valutare la fattibilità computazionale, le limitazioni nel tasso di attuazione, i ritardi di comunicazione e gli effetti di integrazione che non possono essere catturati nelle simulazioni completamente offline. In questo modo, la struttura di

controllo colma il divario tra progettazione algoritmica e implementazione fisica, garantendo che le prestazioni del controllore siano validate sotto i vincoli realistici imposti dall'hardware embedded, dalla comunicazione middleware e dalla simulazione basata sulla fisica.

Introduction

Today, one of the main means of transportation are four-wheel ground vehicles. In order to improve safety, and ease the efforts of driving, autonomous car driving has been developing over the last decades. It is used to enhance road safety, improve convenience and speed, and optimize traffic flow and efficiency, leading to fewer congestion and emissions. There are number of research done on this topic, along with many techniques for self-driving for different scenarios. One of the most popular and prominent techniques in these applications is Model Predictive Control (MPC). Researchers have found it particularly useful for high-performance applications like racing, where it is proven to optimize vehicle performance and ensure safety in complex scenarios. Some of these applications will be discussed in this thesis, like real- time implementation, vehicle modeling and path tracking and stability.

But, this is not the only approach to solve the autonomous driving problem. Other solutions include state-transition controllers or rule based controllers that use deterministic logic to select actions from predefined tables or finite-state machines. In deep reinforcement learning approaches agents learn policies directly from high-dimensional perception inputs, handling continuous state/action spaces. There are also hybrid strategies that combine optimal control and deep reinforcement learning, where a high-level deep reinforcement learning decision maker selects routes while a low-level NMPC refines vehicle dynamics, reducing data requirements and respecting constraints. In multi agent scenarios, where the system interacts with other entities in a dynamic environment, game theoretic solutions for optimal decision making can be employed.

Although determining an optimal, safe, and robust strategy for successfully driving on roads and urban settings is demanding, executing the calculated strategy can be achieved using simple control methods. In contrast, consider the

racing problem, where a vehicle is driven around a pre-defined race track at high velocities to minimize lap time. The vehicle needs to be controlled at its limit of handling, where the dynamics are highly nonlinear.

One of the goals of the thesis is to improve performance of the scaled vehicle used for Bosch Future Mobility Challenge (BFMC) environment. The BFMC is an international competition organized by Bosch Engineering Center Cluj specifically for students to develop and implement autonomous driving and connectivity algorithms on 1/10 scale vehicles. Different tasks and segments of the road are given in order to test the performance of the car and replicate the conditions of a miniature smart city, such as intersections, roundabout, highway, tunnel. Therefore, some of the most desirable features for vehicles of this scale are to be accurate and swift in order to effectively move around in challenging conditions. In the previous competitions, the vehicle control was based on the lane-following functionality which is implemented using a state-machine-based framework that operates in parallel with continuous obstacle detection in the environment. The system employs a Raspberry Pi camera to capture images, which are processed by a convolutional neural network (CNN) trained in the Gazebo simulation environment. The CNN outputs a yaw angle error, which is used by the control system to maintain the vehicle centered within the lane. But this can raise a number of issues: if the camera moves or is not positioned well, or for any reason the lane falls outside of its view, performance decreases, tracking becomes unsatisfactory and unexpected behaviour may arise. For special cases when it is inevitable for the camera to lose sight either of one or both sides of the lane, such as at roundabout, strong curves, intersections and tunnel, different controllers are made, and depending on the situation, designated controllers are called. Since the track is predefined, each one of these situations is represented as a state or an event in the state-machine-based framework and the challenges that we have to deal with are known in advance. This is where the implementation of an MPC controller can be quite beneficial.

The objective is to use MPC as the main algorithm instead of lane following, and simplify and reduce the number of special events and different states, so we don't have to use multiple controllers just for the trajectory tracking. This simplifies the algorithm logic and is easier to work with. We can still keep the functionalities given by camera and CNN, and use it as feedback to our vehicle, but not completely rely on it. One advantage of MPC controller is its reliance on a vehicle model and its use of a prediction horizon to anticipate the system's future behavior based on the upcoming track. Incorporating a prediction horizon allows the controller to generate smoother and less aggressive control actions, resulting in more stable and predictable vehicle behavior. This characteristic can

also positively influence speed performance, which is a critical factor in competitive driving scenarios like this. BFMC 2025 consisted of two main tasks: the technical challenge and the arena fight. We will be focusing on the technical challenge, where in the given time budget the vehicle should be able to collect as many checkpoints as possible, while following standard regulations in traffic. The checkpoints are defined as the points on different parts of the track. In order to collect as many as possible in the given time, the vehicle should move as fast as possible, but with high precision.

Different vehicle models of varying levels of complexity and scale were implemented to evaluate the performance of the MPC controller. Each model and its corresponding advantages are discussed in detail. The objective is to demonstrate that the proposed technique is suitable for vehicles of different sizes and can be effectively applied across various driving environments and track conditions. By comparing the performance and specifications across differently scaled vehicle models, we aim to confirm the general applicability and robustness of this control approach. Later on, the focus is shifted solely on 1/10 scaled RC vehicles.

All tests were conducted within a simulated environment to ensure safe, repeatable, and controlled testing conditions. The evaluation began with simulations using the Acados framework, focusing on comparing the performance of different vehicle models on tracks with ellipsoidal and S-curve shapes. These initial tests were essential to understand the controller's behavior under varying curvature and trajectory complexities, allowing for the identification of each model's strengths and limitations. Following this, the testing framework was expanded to include track layouts inspired by those used in the competition. This extension aimed to better replicate urban-like driving conditions, incorporating tighter turns and intersections. By simulating these more complex scenarios, the testing environment provided a more realistic platform to assess the robustness, adaptability, and generalization of the proposed control strategies. Once the comparative analysis was completed, the most suitable model for the 1/10-scale vehicle was selected and integrated into the ROS2 environment. The model was then simulated in Gazebo on the BFMC track to further validate the controller's performance. This stage of testing aimed at verifying whether the proposed approach could achieve the desired improvements in trajectory tracking, stability, and overall driving behavior. Certain sections of the track were tested independently to obtain a more detailed understanding of the controller's performance in specific segments. This segmented analysis provided valuable insights into the dynamics of the control system and helped guide further refinement of the MPC tuning parameters.

Part I

Theoretical background

1 Modeling the Vehicle

The selection of a suitable prediction model is one of the most important steps in the design of Model Predictive Control (MPC). The complexity of the model directly affects both the accuracy of the predicted system behavior and the computational burden of solving the optimization, thus making the modeling problem very important. If a model is too simple, it may fail to capture important vehicle dynamics, leading to suboptimal or unsafe control actions. On the other hand, a highly detailed and computationally demanding model increases the time required to solve the optimization problem, which can make real-time implementation infeasible. For MPC to be effective in practice (feasible in real-time), the optimization must be solved within the system's sampling time, which needs a balance between model fidelity and computational efficiency. In practice, the choice of model depends on the operating scenario, as simpler formulations may be sufficient for basic trajectory following, whereas more detailed models are required for aggressive maneuvers or operation near the vehicle's handling limits. This trade-off is central to the choice of prediction models for vehicle applications and will be addressed in the following sections.

Four wheel ground-vehicles (that resemble cars) can be represented using different modeling approaches, depending on the level of detail required, the intended application, and size of the model (whether it is a true-size, 1/10 or 1/43). An important simplification that is needed for a model of any kind is that the car is considered a rigid body. This assumption is possible because the chassis of the car is considered stiff, and the dynamics is still sustained.

In cases that are focused on capturing only the key behavior of a vehicle, such as basic trajectory following or low-speed maneuvers, simpler models are often used. We reduce complexity while still preserving the key dynamics needed for analysis and control design. For the most part, this is achieved by simplifying the model to

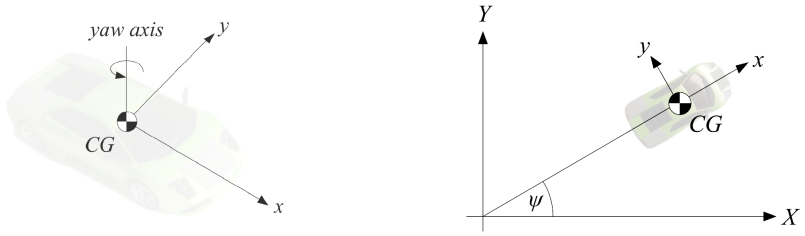
two wheel vehicle (bicycle) and using only the kinematic model. The kinematic bicycle model is the simplest one, obtained by neglecting tire slip and inertial effects, which significantly reduces the computational burden. This model is used for low-speed conditions, parking maneuvers, urban driving with large steering angles, or as a baseline for hybrid control schemes[1] [2]. While computationally efficient, its accuracy diminishes at higher speeds and in situations where lateral slip and tire dynamics play a significant role.

On the contrary, when the objective is to accurately represent more demanding scenarios, such as high-speed maneuvers, tire slip behavior, or vehicle responses under varying surface conditions like wet or icy roads, it becomes necessary to adopt more detailed models that account for nonlinear tire forces and wheel dynamics. A simplified dynamic bicycle model with three degrees of freedom (longitudinal, lateral, and yaw motions) is often chosen as a compromise between accuracy and computational effort, making it suitable for path tracking[3] [4]. More advanced models, such as two-track (four-wheel) formulations or high-degree-of-freedom (6–8 DOF) representations, account for load transfer, nonlinear tire forces (e.g., Pacejka's model), and even roll dynamics. These models are essential for high-speed driving, racing applications, and emergency scenarios such as collision avoidance, where accurate representation of tire-road interactions and combined longitudinal-lateral dynamics is critical for safety and stability[5] [6] [7].

In some of the research, learning-based and hybrid models are considered. They combine the robustness of mechanistic models with data-driven elements to enhance prediction accuracy in uncertain conditions [3]. Black-box models such as Gaussian Processes or gray-box residual models are used in Learning-based NMPC (LbNMPC) [5], when there is a need to compensate for model mismatch in nominal dynamics or when the car is operating near handling limits, thus improving control performance and safety when driving autonomously.

Each approach offers its own balance between accuracy and computational efficiency. In this work both simplified and more detailed vehicle models will be discussed to provide a clearer understanding of their respective advantages, limitations, and suitability for different applications. Note that there is a difference between the control and simulation model. The control model is less complex than the simulation or real-life one, due to the factors previously explained.

The important part of determining the position and movement of the car is to have coordinate systems aligned. The internal/vehicle coordinate system can be represented with (x, y) where the x-axis is aligned with the longitudinal axis of the car, and the world/global coordinate system with (X, Y, ψ), where ψ is the orientation of the car from positive X-axis.



(a) Internal Coordinate System **(b)** World Coordinate System

Figure 1.1: Coordinate Systems

1.1 Bicycle Model

The bicycle model is one of the most used representations of vehicle dynamics that reduces interactions of four wheels into a two-wheel system. Modeling a car as if it had one front and one rear wheel helps to more easily analyze some of the aspects of vehicle behavior, stability, control, and steering. Both longitudinal and lateral position are captured, maintaining accuracy without significant computational demand.

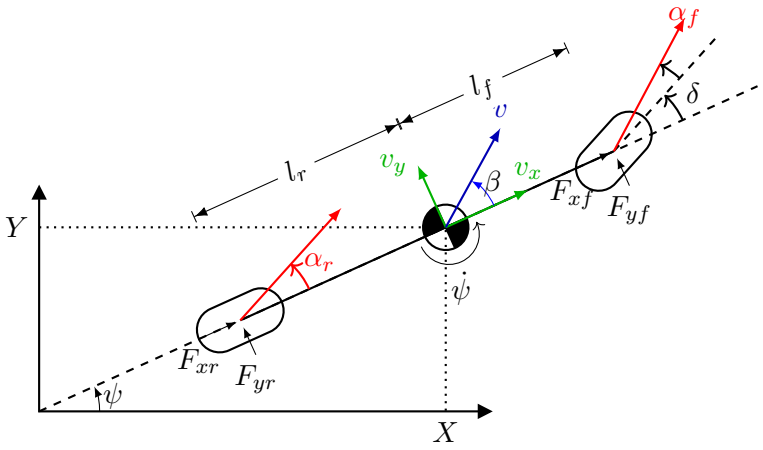


Figure 1.2: Represetation of the Bicycle model

1.1.1 Kinematic Bicycle Model

The simplest model that can be used is the Kinematic Bicycle Model. It describes the motion without considering the dynamics and is mostly used for the lateral vehicle control. The most important assumption of this model is that all slip angles on the wheels are equal to zero. The assumption that the slip on the wheels is negligible can reduce the performance characteristics, but that is not

the case when it comes to the scaled vehicles or small cars [1] and low speed conditions [8]. For more accurate control of a car, we will later introduce the dynamic bicycle model. However, instead of considering only the steering input, we can also take into account the vehicle side slip angle, which describes the difference between the vehicle's actual direction of motion and its orientation. In other words, while the steering angle is applied at the front wheels, the resulting side slip angle is observed at the vehicle's center of gravity (CoG). It can be thought of as the effective angle by which the whole vehicle is "shifted" relative to its intended path due to lateral dynamics.

$$\beta = \arctan\left(\frac{v_y}{v_x}\right) = \arctan\left(\frac{\ell_r}{\ell_f + \ell_r}\tan(\delta)\right) \tag{1.1}$$

It is important to note that we consider steering to be done solely by the front wheels, whereas the driving force is applied through the rear wheels. This configuration is also known as front-wheel steering with rear-wheel drive. It is widely used and serves as an important assumption for simplifying the analysis in vehicle dynamics modeling. Taking the acceleration a and the steering angle δ as the control inputs of the model $u = (a, \delta)$, the state vector can be represented as

$$\xi = \begin{Bmatrix} X \\ Y \\ \psi \\ v \end{Bmatrix} \tag{1.2}$$

Our model is described with the nonlinear dynamics:

$$\dot{X} = v\ cos(\psi + \beta) \tag{1.3}$$
$$\dot{Y} = v\ sin(\psi + \beta) \tag{1.4}$$
$$\dot{v} = a \tag{1.5}$$
$$\dot{\psi} = \frac{v}{\ell_r}\sin(\beta) \tag{1.6}$$
$$\theta = \psi + \beta \tag{1.7}$$

Where (X, Y) represent the position of the vehicle in the world frame, v velocity of the vehicle and ψ yaw/heading angle. We can also consider the angle of θ which is the direction angle in the Center of gravity (CoG).

1.1.2 Dynamic Bicycle Model

In addition to the kinematic bicycle model, we take into account the dynamics of the model, for more accurate representation. However, this makes our model much

more complex, which can be both good and bad. In terms of accuracy, this will make our modeling more precise, thus improving tracking capabilities, but will make control more computationally expensive. We use the Second Newton Law to form the set of equations that describe the model. Compared to the kinematic bicycle model, we are not only describing profiles of desired global position and angle, but also their respective derivatives - linear and angular velocities.

$$\dot{v}_x = v_y\,\dot{\psi} + \frac{1}{m}(F_{x,r} + F_{x,f} - F_x^d) \tag{1.8}$$

$$\dot{v}_y = -v_x\,\dot{\psi} + \frac{1}{m}(F_{y,f} + F_{y,r}) \tag{1.9}$$

$$\ddot{\psi} = \frac{1}{I_z}(\ell_f\,F_{y,f}\ -\ \ell_r\,F_{y,r}) \tag{1.10}$$

where we can define m as the vehicle mass, I_z the vehicle inertia around the z-axis, l_f and l_r as the distances from CoG to the rear and front wheel respectively. The force noted with $F_{x,i}$ represents the lateral force acting on the wheel, $F_{y,i}$ the longitudinal forces on the wheel, while $F_{i,r}$ means that the force is acting on the rear wheel and $F_{i,f}$ is acting on the front wheel. All forces are considered in the vehicle's reference frame. F_x^d is known as (longitudinal) air drag force, as a result of air resistance. Air drag force is modeled as:

$$F_x^d = \frac{1}{2}\,\rho\,C_d A\,\dot{x}^2 \tag{1.11}$$

where ρ is the air density, C_d is the drag coefficient and A is the frontal section area. Additionally, in some models for a small sized vehicles a constant roll resistances, zero order friction parameter, is included in the drag force [9] [1]:

$$F_x^d = C_{r0}\ + \frac{1}{2}\,\rho\,C_d A\,\dot{x}^2 \tag{1.12}$$

Similar to the dynamic bicycle model, a car can also be represented using a double bicycle model, also known as two track model [10]. If we assume that identical steering angles for both front wheels, the double bicycle model effectively doubles the forces acting on the wheels compared to the single bicycle model, thereby accounting for the contribution of all four wheels.

The world frame coordinates can be described as:

$$\dot{X} = \dot{x}\cos\psi - \dot{y}\sin\psi \tag{1.13}$$

$$\dot{Y} = \dot{x}\sin\psi + \dot{y}\cos\psi \tag{1.14}$$

This leads to a direct connection between the world frame (global coordinate system) and the vehicle's reference frame (local coordinate system), which can be useful in generating the proper reference trajectory.
The side slip angle of the vehicle can be defined as:

$$\beta = arctan\left(\frac{v_y}{v_x}\right) \tag{1.15}$$

Forces and slip angles acting on the wheels

The forces acting on the wheels can be expressed in terms of the forces acting on the tires in the vehicle frame, F_l and F_c, longitudinal and lateral forces, respectively. The lateral force is also known as the cornering force, thus F_c is used, in order not to confuse the two forces.

$$F_{x,f} = F_{l,f}\, cos(\delta) \; - \; F_{c,f}\, sin(\delta),\; F_{x,r} = F_{l,r} \tag{1.16}$$

$$F_{y,f} = F_{l,f}\, sin(\delta) \; + \; F_{c,f}\, cos(\delta),\; F_{y,r} = F_{c,r} \tag{1.17}$$

The main difference between the kinematic and dynamic bicycle model is, as previously mentioned, that the kinematic model neglects the effects of the slip angles on the wheels, thus forces are neglected as well. The slip angle is defined as the angle between the actual direction of travel of the wheel and the velocity direction at the contact patch (angular displacement). In particular, we denote by α_f the front wheel slip angle, and by α_r the rear wheel slip angle:

$$\alpha_f = -\arctan\left(\frac{\dot{\psi} l_f + v_y}{v_x}\right) + \delta \tag{1.18}$$

$$\alpha_r = \arctan\left(\frac{\dot{\psi} l_r - v_y}{v_x}\right) \tag{1.19}$$

Slip angles are used for modeling the forces F_l and F_c:

$$F_l = f(\alpha, \mu, k, F_z) \tag{1.20}$$

$$F_c = f(\alpha, \mu, k, F_z) \tag{1.21}$$

μ is the friction coefficient for the road, s is the slip ratio and F_z is the vertical load acting on the wheels of the vehicle. Different methods can be used to represent these forces which is discussed more in the next subsection.

Modeling of Forces

As discussed previously, the choice of a prediction model for MPC requires a balance between accuracy and computational efficiency. Similar considerations apply to the formulation of wheel-ground interaction models, since the representation of tire forces has a direct impact on vehicle stability and control performance. As with most real-world phenomena, tire-road interactions exhibit inherently nonlinear behavior., and their accurate modeling can be essential if we want to capture key effects such as force saturation and the onset of drift.

A wide range of modeling approaches exist: from simplified linear approximations to more detailed nonlinear formulations, such as the Pacejka "magic formula". Each of these models offers a different trade-off between accuracy and computational complexity, and therefore has different capability to describe realistically vehicle behavior while remaining suitable for real-time implementation.

Lateral forces and Pacejka's Magic Formula

Lateral forces play important role in vehicles ability to follow a desired path during cornering maneuvers. They arise primarily from the tire slip angle. At small slip angles, typical of everyday driving conditions, the lateral force can be approximated as being proportional to the slip angle through the cornering stiffness [3]. However, as the slip angle increases, nonlinear effects such as saturation and force reduction become dominant, requiring more advanced modeling approaches. Accurate representation of the lateral forces can also be good predictor of transient responses such as understeer, oversteer, and stability limits. Moreover, since the tire has a finite ability to generate force, the interaction between lateral and longitudinal forces must also be considered, particularly under combined slip conditions.

Among the most widely adopted representations is Pacejka's so-called Magic Formula, first introduced in the mid-1980s by Hans B.Pacejka together with Egbert Bakker, who published the original empirical formulation of the "Magic Formula" as a compact representation of measured tire forces [11] [12]. Since then the Pacejka formulation has become the standard steady-state tire model for high-performance vehicle-dynamics research and for the NMPC-based autonomous driving controllers that rely on accurate lateral-force predictions.

This model is widely used in both academia and industry due to its ability to capture tire forces under many different operating conditions and tire types, from scenarios such as obstacle avoidance to moving horizon scheme [10][6][7]. This formulation allows the tire force curve to be flexibly shaped to match the experimental data. Fundamentally, it is a semi-empirical steady-state tire friction law,

in which the tire forces are described using curve-fitted coefficients derived from experimental data. The Magic Formula is particularly valuable for estimating the lateral forces, which are expressed as function of the lateral slip angle α.

The general form of the Magic Formula is expressed as:

$$F = D \cdot \sin\left(C \cdot \arctan\left(B\alpha - E \cdot (B\alpha - \arctan(B\alpha))\right)\right) \tag{1.22}$$

where:

F : generated tire force (lateral or longitudinal),

α : slip angle in case of lateral forces,

B : stiffness factor, influencing the slope of the curve near the origin,

C : shape factor, controlling the general form of the curve,

D : peak factor, corresponding to the maximum force F_z,

E : curvature factor, adjusting the peak curvature and asymptote.

Note that, Pacejka's Magic Formula can be used to model also longitudinal forces. In this case, instead of the slip angle, the normalized longitudinal slip ratio k is used.

In cases that allows us to use reduced complexity, such as vehicles smaller in size (1/10 or 1/43) or scenarios where key behaviors are captured, a simplified Pacejka formula can be used, which neglects the curvature term E ($E = 0$) [4] [9]. This version is less accurate in capturing detailed tire behavior, but it preserves the main nonlinear characteristics while being computationally more efficient, making it attractive for real-time control and MPC implementations.

$$F = D \cdot \sin\left(C \cdot \arctan(B\alpha)\right) \tag{1.23}$$

Parameters B, C, D, and E can be identified through experimental measurements or by simply estimating them, where tire forces are recorded under varying slip angles, slip ratios, and normal loads or they can be estimated. The relation between the load forces and the slip angle is shown below.

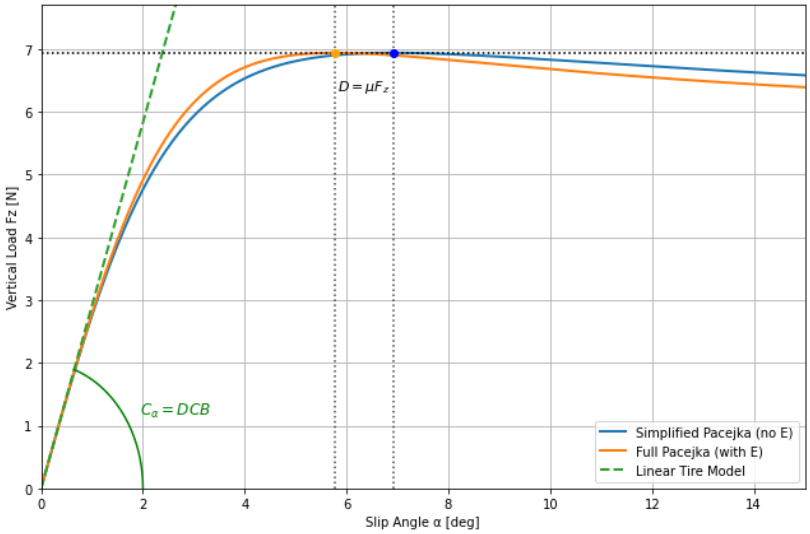


Figure 1.3: Pacejka's Magic Formula

The curve rises with the slip angle and peaks at a moderate angle of 6° (usually picked between 5° and 8°) before falling off. The peak factor D sets the maximum lateral force. It is calculated as $D \approx \mu F_z$ the product of the friction coefficient and the vertical load force in the ideal cases. The shape factor C controls the shape and height of the peak. Typical values for lateral force are in the range 1.3˘1.9, with larger value producing a taller, sharper peak. The stiffness B scales the low-slip response. The cornering stiffness can be calculated as $C_\alpha = BCD$ and it represents the angle of the initial slope. It can be defined as as the lateral force per degree of slip angle.

Longitudinal forces

In the study of vehicle dynamics, longitudinal tire forces represent the primary interaction between the tire and the road surface in the direction of travel. Accurate modeling of these forces can impact some of the key aspects of vehicle behaviour, such as acceleration, braking, and energy transfer. Moreover, they play a decisive role in the performance of advanced control systems including traction control, anti-lock braking systems (ABS), and electronic stability control. Therefore, the development and use of reliable longitudinal force models is fundamental for both theoretical analysis and practical implementation in modern automotive systems. Modeling them, either through simplified proportional models or more advanced formulations, allows us to predict how the vehicle will perform in scenarios ranging from everyday driving to critical maneuvers like emergency braking or launch on low-friction surfaces.

Assuming small slip ratios, which is usually the case in everyday driving conditions (no racing and drifting scenarios), longitudinal forces acting on the wheels can be approximated as being proportional to the slip ratio. Similarly to the slip angle for lateral forces, the slip ratio is used to describe longitudinal forces. It is a measure of vehicle's wheel slipping relative to the road/sliding condition during acceleration or braking. In particular, it is defined as the relative difference between the wheel ground point velocity and the equivalent rotational velocity at the contact patch.

Under perfect rolling conditions, the slip ratio is zero, while positive slip occurs during acceleration (wheel spinning faster than the ground speed) and negative slip during braking (wheel rotating slower than the ground speed). Both braking and acceleration in this scenarios are considered to be mild, thus assuming that our longitudinal tire forces to be zero.

$$F_{lr} = 0, \qquad F_{lf} = 0 \tag{1.24}$$

In this scenario, however we introduce driving force that is proportional to the acceleration of the vehicle a, which impacts only longitudinal acceleration.

$$F_x = ma \tag{1.25}$$

$$\dot{v}_x = v_y \dot{\psi} + \frac{1}{m}(F_{x,r} + F_{x,f} - F_x^d + F_x) \tag{1.26}$$

Another way to represent longitudinal forces can be through the motor characteristics[1], where the longitudinal force of the rear tire $F_{l,r}$ is modeled using a motor model for the DC electric motor, while longitudinal force on the front tire $F_{l,f}$ is considered to be equal to zero. The duty cycle signal is the control input of the system instead of the acceleration, which was the case in the previous modeling:

$$F_{l,f} = 0 \tag{1.27}$$

$$F_{l,r} = F_{dc} = (-C_{m1} \ + \ C_{m2}\,\dot{x})D \tag{1.28}$$

where parameters C_{m1} and C_{m2} represent motor parameters, and $D \in [0, 1]$ is the Pulse Width Modulation (PWM) signal applied to motor. A challenge of this modeling approach is the problem of determining the parameters of the motor.

This definition of longitudinal forces is not applied for the real sized cars, but is mostly used for vehicles of smaller size and is referred to as drivetrain model [4] [13]. The drivetrain model maps an input duty cycle (PWM) through the electronic speed controller into an average motor voltage or current, which through the motor torque constant produces shaft torque and the longitudinal

tire force. Small RC vehicles usually employ an electronic speed controller that directly converts throttle commands to PWM applied to a DC or brushless motor. In contrast, full-scale vehicles typically expose higher-level torque or current commands via an ECU/inverter with fast inner control loops and much larger mechanical inertia, so vehicle-level models commonly use simplified torque-map abstractions and neglect raw PWM dynamics.

In cases of racing and drifting scenarios we need to model longitudinal forces more carefully. Longitudinal forces are generated at the tire-road contact patch, accelerating or decelerating the vehicle. These forces are directly proportional to the normal force (or vertical load) that pushes the tire against the road and the available friction coefficient. The greater the normal force, the greater the potential longitudinal force that can be generated before slip occurs. Thus, modeling them highly depends on vertical load, and limits of longitudinal forces are determined based on values of vertical load. If this is not the case, control of the car can be lost, leading to unwanted behaviour of the car, such as spinning. The simplest way to model the longitudinal forces is to use linear tire model:

$$F_l = \gamma \, \mu \, F_z \tag{1.29}$$

where γ is normalized acceleration/breaking coefficient, and F_z is the vertical load. The vertical load can be modeled as in [10], where it is represented as a combination of static and dynamic load.

$$F_{z,f} = F_{zf}^0 + \Delta F_z, \quad F_{z,r} = F_{zr}^0 - \Delta F_z \tag{1.30}$$

The static vertical load can be calculated from simple geometry:

$$F_{zf}^0 = mg \, \frac{l_r}{L}, \quad F_{zr}^0 = mg \, \frac{l_f}{L} \tag{1.31}$$

While dynamic vertical load is computed using differential equation:

$$\dot{\Delta} F_z = \frac{\tilde{\Delta} F_z - \Delta F_z}{\tau_\Delta}, \text{ with } \quad \tilde{\Delta} F_z = -\frac{h(F_{x,f} + F_{x,r})}{2L} \tag{1.32}$$

where h represents the height of the vehicle's CoG, and τ_Δ time constant.

In detailed dynamic models, longitudinal forces can be calculated using Pacejka's Magic Formula structure, which couples longitudinal slip k with vertical load, especially when considering tire saturation. On the other hand, it can be seen in [7] that longitudinal forces account for wheel dynamics by incorporating the effects of wheel inertia, accelerating torque, and braking torques to compute rotational velocities.

The influence of tire spin requires careful investigation. Spinning introduces a relative velocity between the tire and the ground, which directly affects the generation of lateral forces. To account for these effects, a combined slip model is necessary. Several combined slip formulations have been developed, many of which are based on extensions of the Magic Formula[11] [6]. These models describe how longitudinal and lateral slips are coupled, but are not going to be further discussed in this thesis.

This coupling reflects a physical limitation of the tire–road interface: the contact patch can only transmit a limited resultant force. A simple geometric way to represent this limitation is through the friction ellipse model. The friction ellipse model is used to show the limit of a tire's grip when both braking/traction and steering are applied at the same time. The admissible combinations of longitudinal and lateral forces must lie within an ellipse, indicating that an increase in longitudinal force reduces the maximum achievable lateral force, and vice versa. The admissible tire forces must therefore lie within the following ellipse:

$$\left(\frac{F_x}{F_{x,max}}\right) + \left(\frac{F_y}{F_{y,max}}\right) \leq 1 \tag{1.33}$$

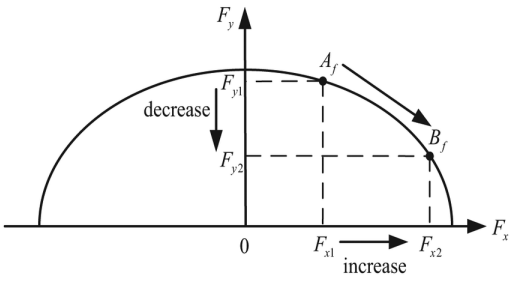


Figure 1.4: Coupling of tire forces, taken from [14]

1.1.3 Full Model

Combining the vehicle and tire models, the state vector of the complete mathematical dynamic bicycle model is:

$$\xi = \begin{Bmatrix} X \\ Y \\ \psi \\ v_x \\ v_y \\ \omega \end{Bmatrix} \tag{1.34}$$

The dynamics of the model is described by the following system of ODEs:

$$\begin{aligned}
\dot{X} &= v_x \cos\psi - v_y \sin\psi \\
\dot{Y} &= v_x \sin\psi + v_y \cos\psi \\
\dot{\psi} &= \omega \\
\dot{v}_x &= v_y\,\dot{\psi} + \frac{1}{m}(F_{x,r} + F_{x,f} - F_x^d) \\
\dot{v}_y &= -v_x\,\dot{\psi} + \frac{1}{m}(F_{y,f} + F_{y,r}) \\
\dot{\omega} &= \frac{1}{I_z}(\ell_f\,F_{y,f}\ -\ \ell_r\,F_{y,r})
\end{aligned} \tag{1.35}$$

where the forces acting on the wheels depend on the control inputs u. First control input varies depending on the specific model employed, while the second control input corresponds to the steering angle δ. If we assume that the slip ratio is neglected, we use as control inputs $u = (a, \delta)$, that correspond to the following modeling of the forces:

$$F_{x,f} + F_{x,r} = ma - F_{c,f}\,sin(\delta) \tag{1.36}$$

$$F_{y,f} = F_{c,f}\,cos(\delta), \quad F_{y,r} = F_{c,r} \tag{1.37}$$

If we consider the slip ratio impact is significant, and thus longitudinal forces are not negligible, we use normalized acceleration as first control input $u = (\gamma, \delta)$:

$$F_{x,f} = \gamma\,\mu\,F_{z,f}\,cos(\delta) - F_{c,f}\,sin(\delta), \quad F_{x,r} = \gamma\,\mu\,F_{z,r} \tag{1.38}$$

$$F_{y,f} = \gamma\,\mu\,F_{z,f}\,sin(\delta) + F_{c,f}\,cos(\delta), \quad F_{y,r} = F_{c,r} \tag{1.39}$$

This modeling requires introducing another state as part of the model:

$$\dot{\Delta}F_z = \frac{\tilde{\Delta}F_z - \Delta F_z}{\tau_\Delta} \tag{1.40}$$

When dealing with small sized vehicles and it is possible to model rear longitudinal force acting on the tire as a function od duty cycle, control inputs are considered to be $u = (D, \delta)$

$$F_{x,f} + F_{x,r} = (-C_{m1} + C_{m2}\,\dot{x})D - F_{c,f}\,sin(\delta) \tag{1.41}$$

$$F_{y,f} = F_{c,f}\,cos(\delta), \quad F_{y,r} = F_{c,r} \tag{1.42}$$

In all models $F_{c,r}$ and $F_{c,f}$ are approximated with simplified Pacejka Magic

formula:

$$F_{c,f} = D \cdot \sin\left(C \cdot \arctan(B\alpha_f)\right) \tag{1.43}$$

$$F_{c,r} = D \cdot \sin\left(C \cdot \arctan(B\alpha_r)\right) \tag{1.44}$$

Note that α_f is also depended of the second control input, the steering angle.

Symbol	Description
x	longitudinal position
y	lateral position
ψ	yaw angle
β	sideslip angle of the vehicle
δ	steering angle of the front wheels
m	vehicle mass
I_z	inertia around the vertical axis of the vehicle
l_f	front wheels to CoG longitudinal distance
l_r	rear wheels to CoG longitudinal distance
L	rear wheels to front wheels longitudinal distance
d	wheels to CoG lateral distance
h	height of the vehicle's CoG
$F_{t\{l,c\}\{i,j\}}$	wheels lateral/longitudinal forces in the tire frames[1]
$F_{x,y\{i,j\}}$	wheels lateral/longitudinal forces in the vehicle's frame
$F_{z\{i,j\}}$	normal forces on the wheels
F_x^d	longitudinal drag force in the vehicle frame
s	curvilinear abscissa or arch length
e_y	lateral tracking error
e_ψ	angular tracking error

Table 1.1: Main model quantities and parameters description.

1.1.4 Spatial reformulation

The traditional time-domain formulation of vehicle dynamics in some cases can be considered inefficient or impractical, due to the system's behaviour and constraints. Often, instead of representing the system as a function of time, we can describe it as a function of the traveled distance along the desired trajectory, also known as the arc of length. In [15], trajectory planning in Frenet frame is introduced in autonomous driving scenarios, which is now the basis of many spatial-domain planners. This reformulation is used to ease and simplify some of the problems that can be encountered in autonomous vehicle systems or advanced driver-assistance systems (ADAS). Some of the most important benefits include the fact that constraints can be formulated in terms of the path, and that the

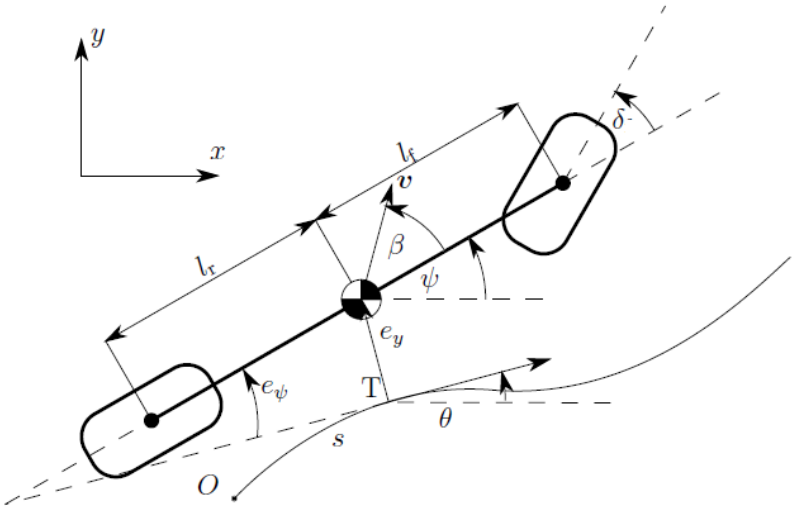


Figure 1.5: Kinematic bicycle model in the Frenet frame

reference velocity can be provided as a function of progress on the track instead of time [10] [7]. Most of the constraints and objective of vehicle motion such as road boundaries and features, lane markings and in some cases obstacles can be described more naturally in spatial than in time domain. Additionally, by using the spatial domain, the relationships between reference trajectory and map data become more intuitive to analyze and trajectory planning is not dependent on velocity based on time.

Spatial reformulation requires rewriting of the systems dynamics using spatial derivative instead of time derivative. Meaning that, instead of describing our states with respect to time, we describe it with respect to the arc of length along the track. The reference trajectory is then parameterized in s and the curvature of the trajectory $k = \frac{1}{\rho}$, where ρ is the instantaneous curvature radius, is locally defined as:

$$k = \frac{x'y'' - y'x''}{(x'^2 + y'^2)^{\frac{3}{2}}}, \tag{1.45}$$

where $f' = \frac{df(s)}{ds}$ and $f'' = \frac{d^2f(s)}{ds^2}$.

We introduce two additional states - angular and lateral error. They can be computed as:

$$e_\psi = \psi - \psi_s, \tag{1.46}$$

$$e_y = (Y_s - Y)\cos(\psi_s) + (X_s - X)\sin(\psi_s), \tag{1.47}$$

where ψ_s is the reference yaw angle and (X_s, Y_s) is the absolute reference position at the current spatial coordinate, while (X, Y) and ψ are the absolute current position and yaw angle. The tracking errors dynamics are:

$$\dot{e}_\psi = \dot{\psi} - k\dot{s}, \tag{1.48}$$

$$\dot{e}_y = v_x \sin(e_\psi) + v_y \cos(e_\psi). \tag{1.49}$$

The relationship with respect to time remains only through the arc of length itself, that can be computed as:

$$\dot{s} = \frac{1}{1 - k(s)e_y}(\dot{x}\cos(e_\psi) - \dot{y}\sin(e_\psi)) \tag{1.50}$$

In case of dynamic bicycle, the state vector becomes:

$$\xi = \begin{Bmatrix} e_\psi \\ e_y \\ X \\ Y \\ \psi \\ v_x \\ v_y \\ \omega \end{Bmatrix} \tag{1.51}$$

The state vector ξ is differentiated w.r.t. s using the chain rule as:

$$\xi' = \frac{d\xi}{ds} = \frac{d\xi}{dt} \cdot \frac{dt}{ds} = \frac{d\xi}{dt} \cdot \frac{1}{\dot{s}} = \frac{\dot{\xi}}{\dot{s}}. \tag{1.52}$$

On top of the spatial formulation, we can introduce half spatial reformulation[1]. In the spatial formulation, the arc length s along the path is chosen as the independent variable, and the dynamics are reformulated with respect to s using the chain rule. However, a limitation of the purely spatial formulation is that it cannot directly capture time-dependent objectives or constraints, which are sometimes needed. To address this, the half-spatial formulation combines both spatial and time representations. Both formulations come from the same underlying optimal control problem and share the ability to handle curvature, lateral and yaw errors, and both introduce the arc length s. The difference is that while certain dynamics are expressed with respect to the path coordinate in half spatial domain, time remains explicitly included, especially for objectives and constraints that inherently depend on time. This mixed approach can have

the advantages of spatial representation while enabling time-based considerations, though at the expense of increased model complexity and computational effort.

1.2 Model Predictive Control

When we are talking about autonomous vehicle scenarios, Model Predictive Control is considered a well-rounded approach that ensures accurate reference tracking of feasible trajectories by ground vehicles. It allows us to use accurate system models, which improves overall performance and reliability. On top of that, it provides flexibility in defining control objectives, inherently handles both equality and inequality constraints, and responds efficiently to unexpected disturbances. Some of the key applications of MPC in autonomous driving include:

- Time-Optimal Trajectory Planning - helps miniature race cars achieve significant lap time reductions by planning time-optimal trajectories and effectively avoiding obstacles [1] [4]
- Accurate trajectory planning and tracking - It is used for integrated lateral and longitudinal control, enabling high-performance autonomous cars to accurately follow trajectories at high speeds. MPC also enhances path-tracking performance and vehicle stability, addressing issues like slip rate fluctuation. [10]
- Learning-based approaches - To overcome challenges in costly model identification, learning-based NMPC (LbNMPC) and Learning Model Predictive Control (LMPC) utilize data-driven methods, such as Gaussian Processes or error dynamics regression, to infer vehicle dynamics from system data and learn unknown behaviors [5]
- Multi-vehicle scenarios - Game-theoretic MPC approaches are employed in head-to-head autonomous racing to predict competitor trajectories and plan strategic overtakes while maintaining track boundaries. MPC is also effective in avoiding static obstacles by modeling them as road boundary deformations.

Model predictive control is an optimization based control method that is based on receding horizon control (RHC). The main idea is to minimize an objective function, at each control step over the finite prediction horizon with respect to system's dynamics and set of constraints. This way, we predict the future states, generate optimal control signals over the horizon based on the current state, but apply only the first control signal in order to introduce feedback.

MPC today is considered to be one of the most effective solutions for the control of MIMO systems, but this is mostly limited to linear models with linear constraints. One of the biggest advantages of MPC is in its structured methodology

for managing constraints and its capability to achieve high control performance. However, its implementation can pose some challenges, as the optimization problem must be solved in real-time, thus within the system's sampling interval and with limited computational resources.

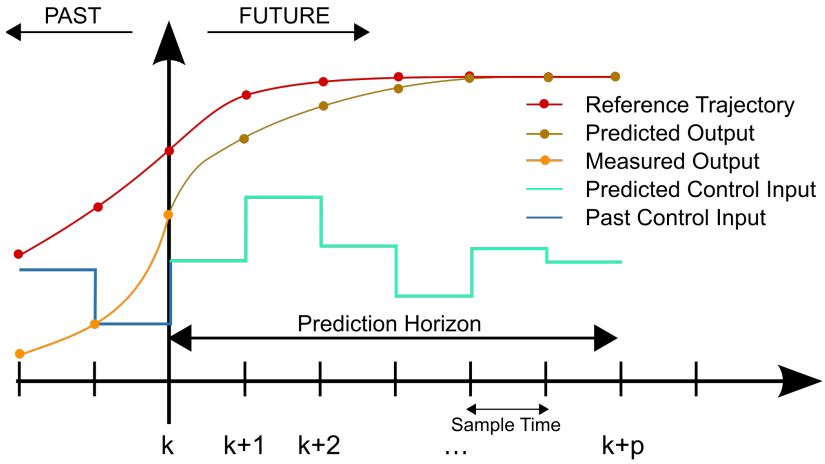


Figure 1.6: Prediction horizon and optimal control problem, taken from [16]

In order to formulate an MPC problem, we need to model the dynamics of the system. The model here is defined by ordinary differential equations (ODE):

$$\dot{x} = f(x(t), u(t), t) \quad \text{with} \quad \begin{cases} x(t_0) = x_0, \\ x(t) \in \mathbb{R}^n, \\ u(t) \in \mathbb{R}^m, \\ t \in \mathbb{R}. \end{cases} \tag{1.53}$$

where x is the state, u is the control input, and (t_0, x_0) are the initial time and state. In order to solve the optimal control problem, we want to minimize the cost function over the time horizon by optimizing over the input u. For a given initial state, the behaviors are parameterized by the control functions u and the cost function assigns a cost value to each possible control action. The cost function is denoted by J and is written in the form:

$$J = \int_{t_0}^{t_f} \Big(\|y(t) - y_r(t)\|_Q^2 + \|u(t) - u_r(t)\|_R^2 \Big)\, dt \; + \; \|x(t_f) - x_r(t_f)\|_P^2 \tag{1.54}$$

where $y(t) = h(x(t), u(t))$ and $x_f = x(t_f)$ is the final state, defined in t_f, the terminal time. Both x_f and t_f can be either free or fixed, with an addition that x_f can belong to some given target set, defined by the constraint $e(x_{t_f}) \leq 0$. Such

terminal constraint can play a significant role to guarantee stability of the closed loop system. At each sampling time, we wish to solve an optimization problem in the following form:

$$\begin{aligned} \min_{\mathbf{u}(\cdot)} \quad & J(x(\cdot), u(\cdot)) \\ \text{subject to} \quad & \dot{x} = f(x(t), u(t), t), \quad t \in [t_0, t_f], \\ & x(t_0) = x_0 \\ & g(u(t), x(t)) \leq 0, \quad t \in [t_0, t_f], \\ & e(x_{t_f}) \leq 0 \end{aligned} \tag{1.55}$$

where the constraints $g(u(t), x(t)) \leq 0$, $t \in [t_0, t_f]$ are generally referred to as path constraints, and encode operational and physical limitations that should not be violated for the whole duration of the task.

When any of f, g, h, e are nonlinear, the problem above is called a Nonlinear MPC (NMPC) problem, otherwise we have a linear MPC problem. NMPC problems arise quite naturally, as most physical systems are inherently nonlinear, and indeed for certain tasks modeling such nonlinearities can be of crucial importance. Some examples of effective deployments of NMPC in autonomous driving are [1] [10] [9].

However, directly using nonlinear models and constraints is not always desirable, because optimization problems are non-convex and computationally demanding, which can make real-time implementation infeasible. Thus, when a linearized model is accurate enough, we often prefer linear MPC, since it is less computationally expensive and less complex to implement. Often this allows us to provide better real time guarantees..

A linear MPC problem formulation consists of a prediction model based on linear system dynamics, affine input and state constraints, and a convex quadratic cost function. Under these assumptions, the corresponding Optimal Control Problem (OCP) can be reformulated as a finite-dimensional, convex Quadratic Programming (QP) problem. Such problems can be solved efficiently using well-

established numerical solvers with guaranteed convergence.

$$\begin{aligned}
\min_{x(\cdot),\, y(\cdot),\, u(\cdot)} \quad & J \;=\; \int_{t_0}^{t_f} \Big(\|y(t) - y_r(t)\|_Q^2 + \|u(t) - u_r(t)\|_R^2 \Big)\, dt \;+\; \|x(t_f) - x_r(t_f)\|_P^2 \\
\text{subject to} \quad & x(t_0) = x_0, \\
& \dot{x}(t) = Ax(t) + Bu(t) \\
& y(t) = Cx(t) + Du(t) \\
& F_x\, x(t) \le f_x \\
& F_u\, u(t) \le f_u
\end{aligned} \tag{1.56}$$

The weighted norm z_Q^2 is equivalent to $z^\top Q z$, with $Q, R, P \succeq 0$ defined as positive semidefinite weighting matrices. The matrices (A, B) are the continuous-time state-space matrices, and (C, D) define the output equation. Constraints can also be written more explicitly in box form. The original OCP for a linear MPC controller can be discretized and transformed into a QP.

1.2.1 Runge-Kutta methods for numerical integration

Numerical integration methods are tools for approximating the solution of initial value problems. The methods were developed and introduced around 1900 by the German mathematicians Carl Runge and Wilhelm Kutta. The idea is to transform a continuous-time problem into a discrete one by discretizing its domain. In control theory, numerical integration methods are used for simulation and control of dynamic systems, enabling accurate time-domain analysis when exact solutions are unavailable. One of the most used method is the explicit Runge-Kutta method of fourth order (RK4). Compared to other approaches that employ iterative schemes and require computation of the Jacobians of the dynamics, RK4 provides accuracy at a low computational cost, making it a standard choice among numerical integration methods. Higher order Runge Kutta methods tend to be more complicated, which makes them lass desirable to work with. For a fixed control signal $u(t)$, the model above can be represented in the form $\dot{x} = f(t, x)$, where x represents the system state, t denotes time, and Δt is the integration step size. The explicit RK4 procedure involves iterating the following update rule:

$$x_{i+1} = x_i + \frac{\Delta t}{6}(s_{1i} + 2s_{2i} + 2s_{3i} + s_{4i}) \tag{1.57}$$

$$\text{with} \quad \begin{cases} s_{1i} & = f(t_i, x_i), \\ s_{2i} & = f\left(t_i + \frac{\Delta t}{2}, x_i + \frac{\Delta t}{2} s_{1i}\right), \\ s_{3i} & = f\left(t_i + \frac{\Delta t}{2}, x_i + \frac{\Delta t}{2} s_{2i}\right), \\ s_{4i} & = f(t_i + \Delta t, x_i + \Delta t s_{3i}) \end{cases} \tag{1.58}$$

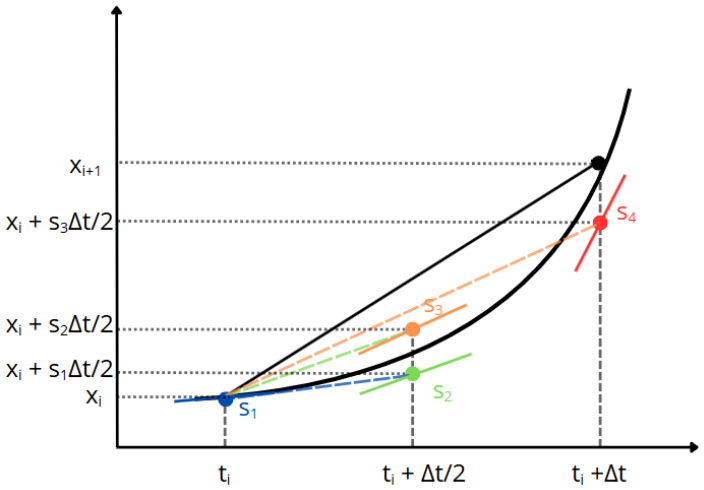


Figure 1.7: Operating principle of Runge Kutta method of fourth order

One possible limitation of explicit Runge Kutta is that it can display limited accuracy for stiff differential equations, which exhibit both fast and slow dynamics. In such cases, it is recommended to use the more complex but effective implicit methods.

In conclusion, the fourth-order Runge-Kutta method is one of the most widely used and versatile tools for solving ODEs in simulation, modeling, and control. It forms a core component in many numerical solvers because of its balance between simplicity, accuracy, and robustness.

1.2.2 Direct Multiple shooting

The idea behind direct methods for continuous optimal control is to transform the infinite dimensional decision variables, such as the control inputs $u(t)$, into a finite set of parameters. This way, the original optimal control problem is approximated by a finite dimensional nonlinear program (NLP). Firstly, the time

horizon is divided into multiple shooting intervals (N), and control inputs are set $u(t;q) = q_i$, if $t \in [t_i, t_{i+1}]$. Then, within each interval, the system dynamics are integrated numerically, starting from auxiliary state variables that serve as optimization parameters. Note that the continuity of the state trajectory across intervals is imposed by nonlinear equality constraints, as part of the optimization problem.

Compared to the single shooting, the underlying principle of multiple shooting is to limit the integration over short time intervals, in order to prevent the accumulating error of integration. By formulating our problem in this way, we have a few important advantages. The optimization variables are decoupled from the integration of the dynamics, leading to a reduced propagation of negative effects of nonlinearities along the integration interval in the state trajectory $x(t;q)$, thus improving numerical stability, especially when the open loop system is unstable. Secondly, it allows better choices of the initialization variables, since the intermediate states can be initialized independently before enforcing continuity.

Finally, the optimization problem obtained through the direct multiple shooting procedure has a specific sparsity pattern that can be exploited by dedicated solvers. This can constitute a significant computational advantage, especially for problems with a long horizon.

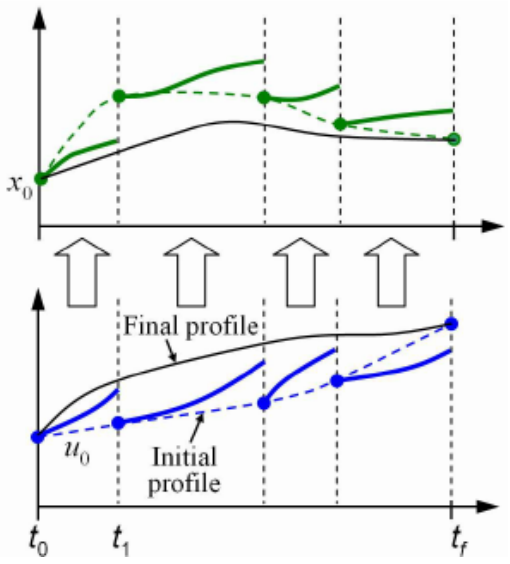


Figure 1.8: Direct Multiple Shooting, taken from [17]

1.2.3 Nonlinear Programming

Applying direct multiple shooting with uniform sampling interval to the continuous time Optimal Control Problem:

$$\begin{aligned} \min_{x(\cdot),u(\cdot)} \quad & J = \int_{t_0}^{t} \left(\|y(t) - y_r(t)\|_Q^2 + \|u(t) - u_t(t)\|_R^2 \right) dt + \|x(t_f) - x_r(t_f)\|_P^2 \\ \text{subject to} \quad & x(t_0) = \bar{x}_0 \\ & \dot{x}(t) = f(x(t), u(t)) \\ & g(u(t), x(t)) \leq 0 \\ & e(x_{t_f}) \leq 0 \end{aligned}$$

where $y(t) = h(x(t), u(t))$, we obtain a finite dimensional optimization problem which can be cast in the general form:

$$\begin{aligned} \min_{w} \quad & F(w) \\ \text{subject to} \quad & G(w) = 0 \\ & H(w) \leq 0 \end{aligned}$$

where $w \in \mathbb{R}^{n_w}$ and $F : \mathbb{R}^{n_w} \to \mathbb{R}$ is a scalar function, while $G : \mathbb{R}^{n_w} \to \mathbb{R}^q$ and $H : \mathbb{R}^{n_w} \to \mathbb{R}^p$ encode equality and inequality constraints. Note that here we assume all functions to be at least $\mathcal{C}^1$. In the case of NMPC, the above is in particular a Nonlinear Program (NLP).

Algorithms that compute (often local) solutions of general constrained NLP rely on the Karush-Kuhn-Tucker (KKT) necessary conditions for optimality. In order to state the KKT conditions, we first introduce the Lagrangian function $\mathcal{L} : \mathbb{R}^{n_w \times q \times p} \to \mathbb{R}$:

$$\mathcal{L}(w, \lambda, \mu) = F(w) + \lambda^\top G(w) + \mu^\top H(w) \; .$$

Theorem 1. *Let $w^* \in \mathbb{R}^{n_w}$ be a locally optimal solution of the Nonlinear Program above, and assume that the set $\{\nabla G_i(w^*)\}_{i=1}^p \cup \{\nabla H_i(w^*),\ i \text{ s.t. } H_i(w^*) = 0\}$ is linearly independent. Then there exist $\lambda^* \in \mathbb{R}^q$ and $\mu^* \in \mathbb{R}^p$ such that the triplet*

(w^*, λ^*, μ^*) *satisfies the following conditions:*

$$\begin{aligned}
&\textit{(Stationarity)} && \nabla_w \mathcal{L}(w^*, \mu^*, \lambda^*) \;=\; 0 && (1.59)\\
&\textit{(Primal feasibility)} && G(w^*) = 0 && (1.60)\\
& && H(w^*) \leq 0 && (1.61)\\
&\textit{(Dual feasibility)} && \mu^* \;\geq\; 0 && (1.62)\\
&\textit{(Complementary slackness)} && H(w^*)^\top \mu^* \;=\; 0 && (1.63)
\end{aligned}$$

Note that, as previously stated, the KKT conditions are only *necessary* for local optimality, and in general a w^* satisfying the KKT system is only a critical point (all candidate optimal solutions and thus possibly even a local maximum). Exploiting second-order information about the Lagrangian function we are sometimes able to assess wether a given critical point is actually a local minimizer, however such methods are not discussed in this thesis.

The KKT conditions can be considered the mathematical bridge between the abstract NLP and the numerical algorithms (SQP for example) that actually solve it. This is because numerical algorithms do not solve NLP directly, but they use approximations of KKT in order to solve it. Thus, in the next subsections we are going to talk about one of the methods that can be used to solve KKT conditions.

Sequential Quadratic Programming

Sequential Quadratic Programming (SQP), also known as Lagrange-Newton method, is an iterative scheme that aims at finding a critical point w^* starting from an initial guess $w^{(0)}$. At each step of the algorithm, we construct a quadratic approximation of the problem about the current iterate:

$$\begin{aligned}
\min_{\Delta w \in \mathbb{R}^{n_w}} \quad & \frac{1}{2}\Delta w^\top P^{(k)} \Delta w + \nabla F(w^{(k)})^\top \Delta w \\
\text{subject to} \quad & G(w^{(k)}) + \nabla G(w^{(k)})^\top \Delta w = 0 \\
& H(w^{(k)}) + \nabla H(w^{(k)})^\top \Delta w \leq 0
\end{aligned}$$

and solve to obtain a search direction $\Delta w^{(k+1)}$. Then the next iterate is found by means of some sort of line-search procedure:

$$w^{(k+1)} = w^{(k)} + \alpha^{(k+1)} \Delta w^{(k+1)} \,.$$

$P^{(k)}$ is typically an approximation of the Hessian of the Lagrangian at the current iterate. It is very common to choose $P^{(k)}$ according to the Gauss-Newton approximation of $\nabla^2 F(w)$, because in the context of MPC the objective $F(w)$

can often be rewritten as a Nonlinear Sum of Squares $F(w) = \frac{1}{2}\|r(w)\|^2$:

$$P^{(k)} = \nabla r(w^{(k)}) \nabla r(w^{(k)})^\top .$$

The Gauss-Newton method is particularly appealing in NMPC because it has two main advantages: firstly, the corresponding approximation $P^{(k)}$ can be computed very efficiently, as no second order information about r is required. In particular when $F(w)$ is quadratic (as, again, often happens in MPC) $P^{(k)}$ is obtained immediately, and in fact $P^{(k)} = \nabla^2 F(w^{(k)})$. Secondly, numerical stability is improved because it guarantees a positive semi-definite approximation.

The Real Time Iteration Scheme

The SQP algorithm has important limitations that make it difficult to meet the real time requirements of NMPC. Indeed, each SQP iteration is quite computationally demanding, as it requires the solution of a new Quadratic Program. Moreover, the total number of iterations to reach convergence is often unpredictable.

The Real Time Iteration scheme (RTI) [18] is a variant of SQP that addresses these key limitations by trading solution accuracy for increased control frequency (this tradeoff is known in the MPC community as the *Real Time Dilemma*). It is a strategy for implementing NMPC on systems with fast dynamics, enabling computation times in the millisecond or even microsecond range[18]. In RTI, at each sampling time we run a single SQP iteration, while linearizing the problem at the solution obtained at the previous sampling time and taking a full step ($\alpha = 1$).

An important characteristic of RTI is that we distinguish two phases of the algorithm: the *preparation* phase and the *feedback* phase. During the preparation phase computationally expensive tasks are done, as we linearize the problem and setup the QP solver, leveraging results from the previous sampling instant. After this step is completed we start the feedback phase where we gather a measurement (or estimate) of the current state of the system, provide this information to the QP solver and actually solve the problem to obtain the update direction. Compared to an approach where we simply perform one SQP iteration at each sampling time, separating the preparation phase from the feedback phase allows us to use more up-to-date state information to compute the control law. The RTI scheme provides real time performance by separating computation, while guaranteeing local convergence, ensuring that the performance will not degrade beyond what a linear MPC controller would achieve, provided the system state remains close enough to the reference solution.

1.2.4 Solving Quadratic Programs

When implementing a NMPC controller based on SQP or RTI, we need to repeatedly solve constrained Quadratic Programs. While to this purpose we could rely on generic QP solvers like OSQP **[osqp]**, in most cases using dedicated solvers is beneficial. Indeed, the QPs obtained through linearization preserve the sparsity pattern of OCPs. In particular (assuming for simplicity that the reference is equal to zero, F is quadratic and we are using the Gauss-Newton Hessian approximation) the QP can be rewritten in terms of the increments of the state and control variables as:

$$\begin{aligned}\min_{\{\Delta x_t\},\{\Delta u_t\}} \quad & \Delta x_t^\top Q_N \Delta x_t + \sum_{t=0}^{N-1} \Delta x_t^\top Q \Delta x_t + \Delta u_t^\top R \Delta u_t \\ & \Delta x_0 = 0 \\ & \Delta x_{t+1} = A_t \Delta x_t + B_t \Delta u_t + e_t \\ & C_t \Delta x_t + D_t \Delta u_t - b_t \leq 0\end{aligned}$$

Problems with this structure can be solved with specialized methods, which can provide a significant computational advantage over generic QP solvers.

qpOASES

qpOASES [19] is an active-set solver for dense QPs that was originally designed to overcome the limitations of Explicit MPC [20] [21]. An active set method is an algorithm that maintains a so called Active Set, which is the current guess of the set of active constraints at the solution. During execution, the Active Set is iteratively updated by adding and removing constraints.

In order to obtain a dense QP from the one showed above, we eliminate the state variables by iteratively substituting the update equation:

$$\Delta x_{t+1} = A_t \Delta x_t + B_t \Delta u_t + e_t$$

in the constraints and the objective. As a result, we obtain an equivalent dense QP, where the controls are the only optimization variables:

$$\begin{aligned}\min_{\bar{u}} \quad & \frac{1}{2} \bar{u}^\top \mathcal{H} \bar{u} + g^\top \bar{u} \\ \text{subject to} \quad & \mathcal{M} \bar{u} - d \leq 0\end{aligned}$$

Here $\bar{u}$ is obtained by stacking the control variables Δu_t over the horizon. This technique, called condensing, can be very beneficial when N is not too large and

when the dimensionality of the controls is significantly lower than that of the state variables. When qpOASES is employed alongside condensing techniques, it can be computationally competitive, particularly for NMPC problems with shorter prediction horizons, as the active-set method performs well when constraints are reformulated as general constraints [22].

However qpOASES, as most solution methods for dense QPs, requires performing operations whose runtime scales quadratically and even cubically with the dimension of $\bar{u}$, and therefore with the horizon length N. Moreover, if the system's dynamics is open-loop unstable, when condensing the problem over a long horizon numerical errors can be significantly magnified, thus compromising numerical conditioning. For these reasons, condensing should generally be avoided when N is large.

While the total runtime of active-set methods can be quite large and unpredictable when abrupt reference changes occur, in the context of MPC they can be effectively warm-started across sampling times or SQP iterations. In fact qpOASES expects each QP in the sequence (e.g. each MPC time-step) to be similar to the previous, so it reuses the previous solution's active set as a starting guess, reducing iteration counts. This often leads to very fast convergence if small changes occur, especially for small- to medium-scale QPs with dense Hessians and constraint Jacobians. Internally, it uses dense linear algebra to update factorizations as constraints are added or removed. qpOASES does not inherently exploit a multi-stage sparsity pattern, and it typically solves a dense QP.

HPIPM

HPIPM [23] is an established solver for MPC that relies on a primal-dual Interior Point Method (IPM). With HPIPM we can choose over various degrees of condensing:

- Full condensing, which consists in the technique previously described.
- No condensing, where all state variables are included as part of the optimization problem.
- Partial condensing. With this technique some state variables are aggregated together, and a QP with similar OCP structure is solved over a shorter horizon. It offers a trade-off by adjusting the block size, which can affect the balance between length of horizon and of control vector.

Partial condensing can be beneficial when state and control dimensions are rather small. In fact after partial condensing the core numerical routines run on larger matrices and vectors, and can therefore better utilize the parallel capabilities of the underlying hardware.

To conclude, full condensing (eliminate all x) yields a smaller QP in variables, but a dense Hessian $O(N^3)$ effort, whereas partial condensing yields a moderate-size QP that still has a banded block-structure. Full condensing is memory-intensive (dense matrix storage) while partial condensing retains sparsity and thus has a lower memory footprint. When no condensing is applied, one solves the original "sparse" QP (with all x and u) directly using structure-exploiting methods .

IPM computes a minimizer of a convex constrained QP by solving relaxed versions of the associated KKT system by means of the Newton method. IPMs move iteratively through the interior of the region defined by inequality constraints by replacing the non-smooth, L -shaped set resulting from the complementarity conditions with a smooth approximation. Starting from the large value of the smoothing constant $\tau > 0$ and decreasing the value, while using previous solution to initialize new iteration.

For an OCP-QP like the one above (where no condensing is applied), each Newton step requires solving a linear system with a specific sparsity pattern that allows the solution to be computed through a Riccati recursion. Even though the computational complexity of this approach is comparable to that of a generic sparse linear system solver, in practice it leads to consistently lower run-times [24].

Compared to the active-set solver qpOASES, HPIPM can provide consistently low solve times even on challenging problems or upon abrupt reference changes, as IPM is a very stable algorithm with predictable performance. qpOASES is known for its efficiency in solving smaller QPs and excellent performance when warm-started [19] [9], while HPIPM excels when the QP structure allows for exploitation of block-triangular matrices, particularly when many state bounds are present after partial condensing [5]. Active set solver is usually pared with full condensing forms, while IPM exploits partial-condensing structure. However, IPM cannot benefit from warm-starting to the same degree as active-set methods, and this is the main limitation of the approach.

Part II

Implementation

2 Implementation

2.1 Introduction to Acados

Acados is an open-source software package designed for solving OCPs in real time [25]. It's specifically built for applications that need fast and efficient NMPC and moving horizon estimation (MHE). It stands for *advanced control algorithms for dynamic optimization scenarios* and it is designed to be used for fast embedded platforms [26]. The framework is primarily implemented in the C programming language to ensure computational speed and determinism, while providing high-level interfaces for MATLAB, Python, and CasADi, facilitating user interaction and automatic code generation.

It converts a continuous time nonlinear OCP into a discrete time OCP by using direct multiple shooting, as described in the previous section. The problem is expressed as a nonlinear program (NLP) and it is solved repeatedly in real time [26] by means of SQP or RTI. The sequence of OCPs generated by these methods is solved using either HPIPM or qpOASES, the solvers described in the previous section. Empirical benchmarks comparing the two solvers have shown mixed results, sometimes favoring qpOASES slightly for average computation time on certain problems, but structural exploitation favors HPIPM in specific, heavily-constrained NMPC setups [13].

The linear algebra operations underlying these solvers are executed via the BLASFEO library (Basic Linear Algebra Subroutines For Embedded Optimization), which is specifically optimized for small- to medium-scale matrix operations on embedded processors, significantly improving cache efficiency and runtime performance [23].

One of the main applications of Acados is time-optimal trajectory planning

for nonlinear systems, showcasing the framework's capability to handle high-dimensional, nonlinear, and constrained control problems with real-time performance [27]. Additionally, the modular structure of Acados enables seamless integration with external optimization and simulation environments, making it suitable for both academic research and industrial control systems. Overall, the combination of modularity, computational efficiency, and code-generation capabilities positions Acados as one of the leading open-source frameworks for high-performance embedded optimal control.

2.2 Simulation solutions using Acados

In the experiments where we used Acados for simulation, the controller and the simulator employ the same plant model. In fact, while working exclusively with Acados our goal is to highlight characteristics of the specific models. In these cases, since simulation and control have the same model, whether it is in time or spatial domain, we expect perfect tracking.

Typically we simulate the system in time, while the controller may be implemented in the spatial domain. In these situations, because the quantities s, e_y, e_ψ are not part of the dynamic model that is used for simulation, they must be computed separately at each iteration step. Likewise, any controller, state not in the simulation (or vice versa) must be handled externally (for instance by integrating it or using a last-known value).

2.2.1 Projection - Time to Spatial Domain

A control model in the spatial domain is often advantageous for real-world applications. However, since the simulation is typically defined in the time domain, a connection between the two must be determined. This can be achieved through interpolation, by projecting the vehicle position onto the path segment, similarly as in [1] and in one of the Acados examples [25].

To determine the relative position of the vehicle with respect to a reference path (X, Y, ψ, v), the point P where the vehicle is at the current moment is projected onto the closest segment of the path defined by two consecutive reference points A and B. A represents the reference point closest to the real position of the vehicle P, while B is the second closest reference point. P is somewhere in between these two reference points. A and B are determined by searching through the array of sampled reference positions.

Denote the segment vector as $w = B - A$ and the vector from the point closer to the vehicle position to P, $v = P - A$. The projection on the path can be

expressed in terms of the parameter

$$p = \frac{w^T v}{w^T w} ,$$

and it consists of the point on the infinite line through A and B of minimum distance from P. Notice that we consider the road segment from A to B to be straight. This is a very common approximation, since our path is sampled densely and segment can be considered nearly straight between two reference points. The quantity p is a scalar that represents the position on the segment.

- $p = 0$ exactly at A
- $p = 1$ exactly at B
- $0 < p < 1$ somewhere in between
- $0 > p$ or $p > 1$ outside of AB

The projected point on the path can be written in the form $A + pw$, thus geometrically, the shortest distance between the vehicle and the path line is $\|A + pw - P\|$. The projected point $A + pw$ is used to interpolate reference quantities between the two points. This projection ensures that (e_y, e_ψ) are defined with respect to the nearest location on the path, allowing a consistent computation of tracking errors in the spatial frame.

However, this representation of the problem faces some problems when it comes to loop tracks, or in general if the path intersects itself at some point. In order to take care of this, we need to take into the account previous position of the car, or rather traveled distance of the car and assume that the next position will be close to it.

2.2.2 Sampling Time

In order to do experiments, it is necessary to choose appropriate sampling times (or distances, any update rates) for the controller and the simulation model. The sampling time represents the rate at which the controller or simulator recomputes the system values. It is important to distinguish *controller sampling time* (how often the MPC computes and sends a new command inputs) from the *simulation/plant step* (how finely the vehicle dynamics are numerically integrated). In realistic scenarios, simulation is updated at much finer rate than the controller.

For simplicity, in the initial tests, where Acados is used for both control and simulation, we choose the same simulation sampling time and controller sampling time. If we update the controller based on the time and not distance, the concept of the controller frequency is intoduced as $f_{ctrl} = \frac{1}{\Delta t_{ctrl}}$. Note that, the update of

the controller is not necessarily connected to the formulation of the MPC (spatial or time domain). It is common to use the spatial domain formulation of MPC and update the controller based on the frequency/time. However, the controller frequency has big impact on the control performance, and if it is too low it can potentially have larger tracking errors, oscillations or aliasing effect, and slow responsiveness.

If we are dealing with the real world system, many practical considerations influence the feasible the controller sampling time, such as update rates and precision of onboard sensors, actuator dynamics and communication delays, and the latency introduced by the vehicle's control hardware and data buses (such as CAN or Ethernet). Furthermore, the computational resources available on the embedded platform and the real-time operating constraints play a critical role in ensuring that the controller can consistently deliver commands within the required time frame. These aspects collectively determine the range of feasible sampling rates and to what extent the controller's theoretical performance can be achieved in real-world operation.

2.3 Generating references

2.3.1 Elliptical and S shaped trajectories

In order to test the performance of the controller, we used predefined elliptical and S-shaped references. The geometrical properties of these curves allowed us to asses controller capabilities such as cornering stability, response to variations in curvature and generally trajectory tracking accuracy.

The ellipse trajectory is analytically defined as $X(t) = a\ cos(\omega t)$, $Y(t) = b\ sin(\omega t)$, where a and b represent the ellipse's semi-major and semi-minor axes, respectively, and ω is the angular frequency that governs the rate at which the vehicle moves along the trajectory. The choice of ω directly influences the trajectory's time dynamics, thus making it an important variable. The trajectory is sampled over one period $T = \frac{2\pi}{\omega}$ using a uniform time step Δt, in this case equal to the controller's sample time. The derivatives of the reference are:

$$\dot{X}(t) = -a\ \omega\ sin(\omega t), \quad \dot{Y}(t) = b\ \omega\ cos(\omega t) \tag{2.1}$$

In the numerical implementation, the sampling time Δt must be chosen consistently with ω to provide good time resolution. The product between ω and Δt

determines the number of samples:

$$N = \frac{T}{\Delta t} = \frac{2\pi}{\omega \Delta t} \tag{2.2}$$

If Δt is too large relative to ω, the resulting trajectory becomes under-sampled, which leads to numerical inaccuracies in the computed derivatives of (X, Y). Conversely, if Δt is too small, it increases computational cost without significant benefit. By adjusting ω while maintaining a consistent sampling interval Δt, it is possible to explore the trade-off between control bandwidth, reference smoothness, and numerical stability, thus providing an evaluation of the controller's robustness and tracking capability of the time domain controller.

The S shaped trajectory, on the other hand, is constructed using cubic spline interpolation based on a predefined set of points. This method ensures C^2- continuity of the path, that is continuity of position, velocity, and acceleration, which is crucial for smooth control inputs and stable MPC performance.

To generate the reference trajectories used by the controller, both time-domain and spatial-domain formulations were implemented. Each trajectory provides reference values for position, orientation (sum of heading and slip angle), velocity, and in case of spatial formulation curvature κ and angular and lateral error (e_ψ, e_y). In the time domain, the trajectories were first defined parametrically as $(X(t), Y(t))$, from which the corresponding derivatives $\dot{X}(t)$ and $\dot{Y}(t)$ were obtained to compute the velocity $v(t) = \sqrt{\dot{X}^2 + \dot{Y}^2}$ and direction angle $\theta(t) = arctan2(\dot{Y}, \dot{X})$. Note that in this formulation, position of the vehicle CoG is defined with respect to the world frame coordinates, thus their derivatives are computed in the global frame rather than the vehicle frame. In case of the kinematic bicycle model, in order to compute the derivative of arc length $\dot{s}$ longitudinal and lateral velocities v_x and v_y (that is $\dot{x}$ and $\dot{y}$) are required, and since they are not the same as the derivatives $\dot{X}$ and $\dot{Y}$, they are computed as:

$$v_x = v\, cos(\beta) \neq v\ cos(\psi + \beta) = \dot{X} \tag{2.3}$$

$$v_y = v\, sin(\beta) \neq v\ sin(\psi + \beta) = \dot{Y} \tag{2.4}$$

$$\tag{2.5}$$

Additionally, it is important to distinguish the difference between the derivatives when discussing dynamic bicycle model, because it requires the velocities v_x, v_y and yaw rate $\dot{\psi}$ that describe motion in the inertial coordinate system. Based on the world frame coordinates, internal frame coordinates must be calculated. As previously stated, the relation between world and internal frame

coordinates is:

$$\dot{X} = \dot{x}\cos\psi - \dot{y}\sin\psi \tag{2.6}$$

$$\dot{Y} = \dot{x}\sin\psi + \dot{y}\cos\psi \tag{2.7}$$

The unknown variables are (x, y, ψ), while (X, Y, θ) are known. The relationship between the orientation and the heading angle is $\theta = \psi + \beta$. In order to be able to represent the internal frame velocities starting from the world frame ones, it is assumed that there is no slip in the reference trajectory. This means that $\beta = 0,\ \theta = \psi$, and we can calculate:

$$v_x = \dot{x} = \dot{X}\cos\theta + \dot{Y}\sin\theta \tag{2.8}$$

$$v_y = \dot{y} = -\dot{X}\sin\theta + \dot{Y}\cos\theta \tag{2.9}$$

This can be considered a reasonable assumption, because the reference trajectory should represent the "perfect" path the vehicle should follow, under ideal conditions, and a clean target for the controller to track. This simplification provides a smooth, computationally efficient, and numerically stable reference for the controller. Another reasonable assumption can be $v_x = v$, $v_y = 0$, where $v = \sqrt{\dot{X}^2 + \dot{Y}^2}$ since we assume there is no slip. These two definitions usually have the same or similar results. Modeling slip dynamics at the reference generation stage is not useful for our purposes. Instead, the MPC controller will compensate for deviations from this ideal behavior by optimizing the control inputs in the presence of lateral slip and dynamic effects. Furthermore, if we consider moderate speeds, the no-slip assumption remains valid, with minimal lateral slip expected. The reference defines where to go, while the controller determines how to get there considering slip dynamics.

The difference between spatial and time domain is in parametrization. In the time domain, trajectories were parameterized with respect to time t and sampled at a constant time step Δt, while in spatial domain each trajectory was discretized using a fixed spatial step Δs. To make the control problem independent of time, the references were then transformed into the spatial domain by computing the cumulative arc length:

$$s(t) = \int_0^t \sqrt{\dot{x}^2(\tau) + \dot{y}^2(\tau)}\, d\tau \tag{2.10}$$

and performing numerical interpolation to express all trajectory variables as functions of s. The path curvature was obtained from the equation 1.45 and

later interpolated using B-splines to ensure smoothness and compatibility with the MPC framework. Note that $\kappa(s)$ is the characteristic of the road, thus does not depend on the reference frame it is computed in. The final reference vectors include position, velocity components, orientation, curvature, and error states (e_ψ, e_y), which values are equal to zero since we do not want any tracking errors, providing consistent and continuous reference signals for both time- and space-based controller validation. The spatial formulation ensures a consistent path progression for the controller, independent of speed variations, and allows direct computation of curvature.

2.3.2 Trajectories based on the Bosch Future Mobility challenge 2025

The following step was to derive the reference trajectory from the tracks used in the Bosch Future Mobility challenge (BFMC). This environment is supposed to simulate the conditions of a miniature smart city, such as roundabouts, crossing sections and highway. Including these tracks allows the simulation framework to more accurately replicate real-world, urban-like driving conditions.

Only the spatial formulation of the reference is available, since the track is parameterized in space, and not analytically like the ellipsoidal or S shaped curve. Thus, the controller is implemented in spatial domain.

The map is represented as directed graph, where each node corresponds to a different position on the track, and each edge corresponds to the possible connections between the nodes. The desired nodes the car has to pass through are defined along with starting and ending node.

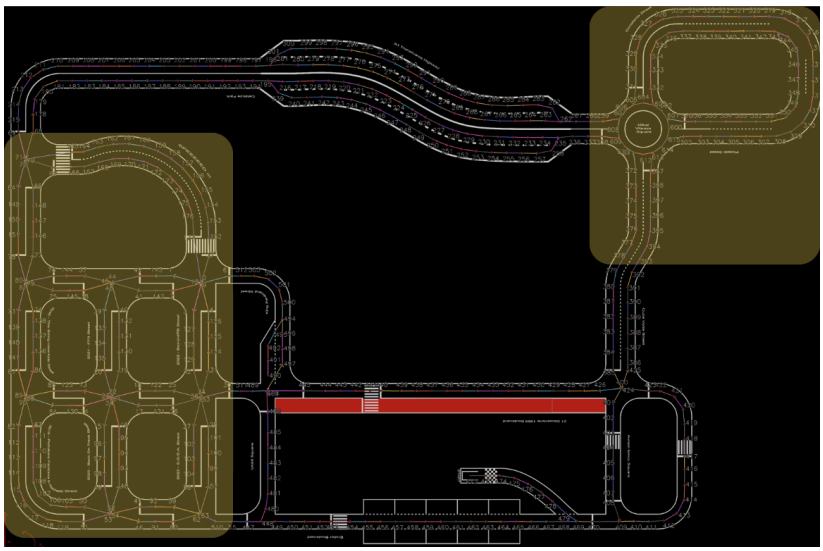


Figure 2.1: BFMC Competition Track

In the path planning step, a feasible reference trajectory is generated starting from the vehicle's initial position, and including the desired nodes in the path. This has been successfully implemented with Dijkstra algorithm. After path planning, we need to generate a trajectory for the MPC controller at equally spaced arc-length intervals. Firstly, we have to transform the discrete route formed from the list of nodes into the continuous and smooth reference path by connecting each segment between graph nodes. Instead of using the spline cubic interpolation that is used for formulating the S curves, the Clothoid Interpolation is used. With this method, the curvature $k(s)$ changes linearly with arc length thus making both curvature and steering transition smoother. This matches well with the physical turning of the vehicle and is very applicable in road-vehicle geometry. The curve is then discretized into finely spaced samples, providing a continuous sequence of reference states that include position (X, Y), yaw angle ψ, as well as the curvature $k(s)$ and arc length s.

The performance is tested on two important parts of the map as it is showed on the Picture 2.1: urban-like area and roundabout. The urban like area contains an irregular, mixed-curvatures, such as sharp 90° corner entries, long straights, nested S-bends, and a high-curvature elevated segment. This path introduces abrupt curvature changes and requires the controller to adapt continuously to non-periodic geometry. On the other hand, the roundabout trajectory includes a roundabout entry-exit manoeuvre and can be followed by a high-curvature oval segments with lane only marked on one side of the road.

Choice of the nodes for path planning

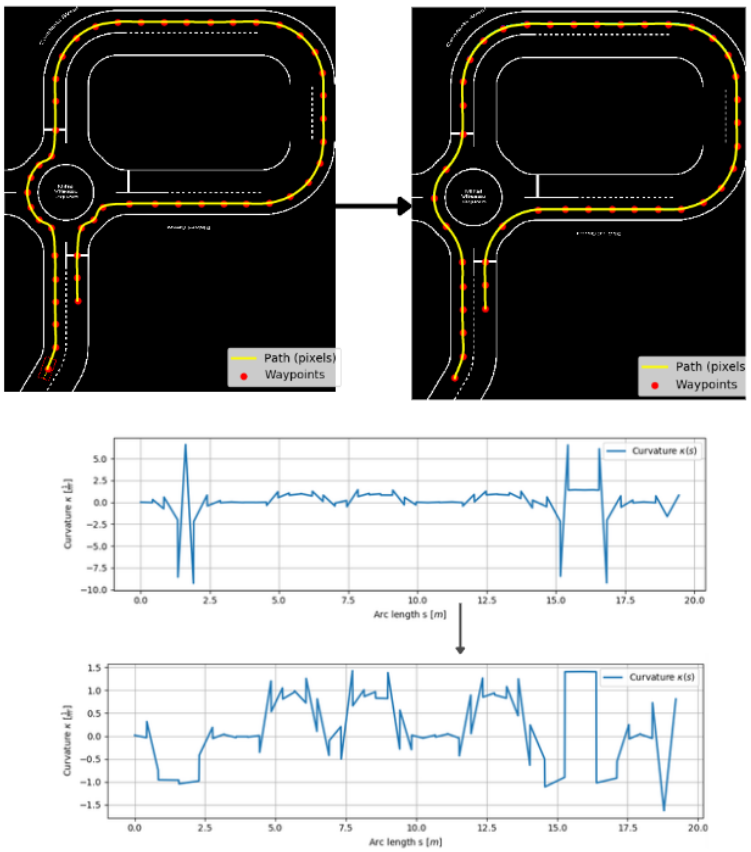


Figure 2.2: Before and After Removing Disruptive Nodes around the Roundabout

As previously discussed, the initial paths were created from the map by connecting segments between the graph nodes. While this representation follows the road geometry accurately, the resulting piecewise-linear path produces large curvature fluctuations when converted to the spatial representation used by the controller. This is visible in the curvature profiles of the original paths, where $\kappa(s)$ exhibits sharp spikes, abrupt sign changes, and short segments of very high curvature, even in geometrically smooth regions of the map. This behaviour do not originate from the road itself but from the excessive number of nodes.

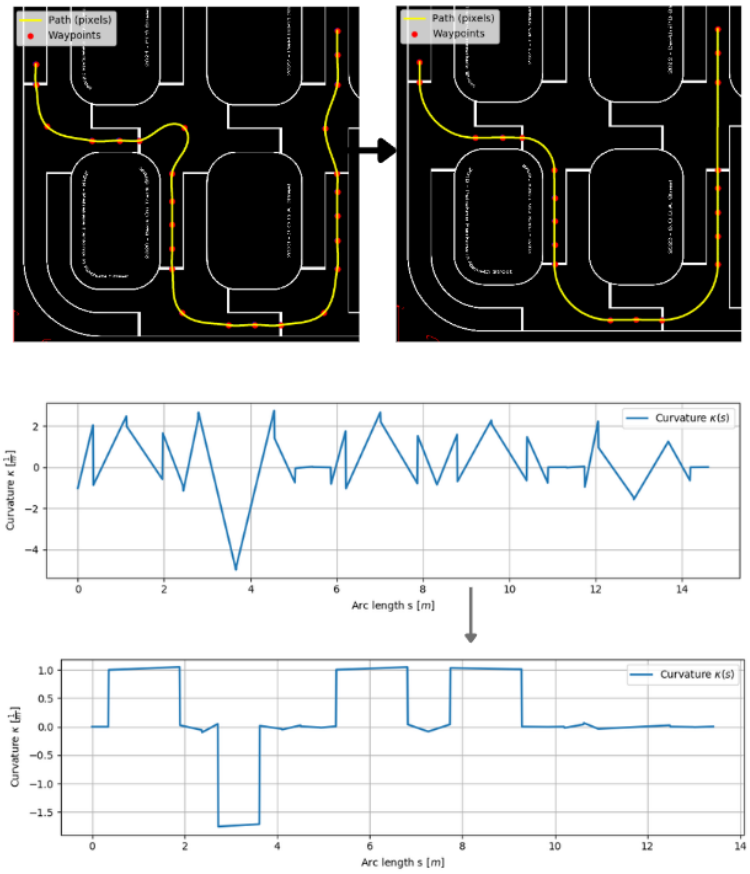


Figure 2.3: Before and After Removing Disruptive Nodes in the Urban-Like Area

In order to resolve this issue, the unnecessary intermediate graph nodes, the ones that were disrupting the natural flow of the curvatures, were removed from computing the path. This results in a smoother geometric representation of the road while remaining almost identical to the original lane centerline. The large spikes disappear entirely, curvature transitions become gradual, and the resulting $\kappa(s)$ aligns closely with the true structure of the road.

For both the roundabout and the urban-block trajectories, represented on Figures 2.2 and 2.3 , the raw curvature signal reaches values between $|\kappa| \approx 5-10\ m^{-1}$ in places where the underlying map geometry contains no such features. In the roundabout case, the standard deviation of $\kappa(s)$ is reduced by approximately $60-70\%$ after removing unnecessary nodes. Similar reductions are observed in the urban-block scenario, where the refined path remains within $|\kappa| \leq 1.5\ m^{-1}$ in both cases.

This refinement has a direct positive impact on the behaviour of the spatial

MPC, particularly in scenarios with tight curvature transitions or irregular road layouts. First, the smoother curvature profile reduces the artificial curvature gradients imposed on the MPC, preventing unnecessary steering actions and lowering the demand on the actuator limits. Second, by removing curvature discontinuities, the previewed reference becomes more predictable across the horizon, improving feasibility and reducing sensitivity to model mismatch. Both effects can lead to improvement of the tracking performance in the experiments, since steering peaks will be reduced, and the controller no longer reacts to non-physical fluctuations in curvature.

2.4 ROS2-based simulation solutions

2.4.1 ROS2

The Robot Operating System 2 (ROS 2) is a middleware software framework that enables modular development and real-time communication between software components in robotic systems. It is the successor to ROS 1, developed by Open Robotics (originally by OSRF -the Open Source Robotics Foundation) [28]. It provides standardized interfaces for sensor data, control commands, and interprocess communication, allowing different nodes, such as perception, planning, and control modules, to exchange information seamlessly. It is widely used in both industrial and academic environments. One of the main principles of ROS 2 is modularity, as the system is composed of many smaller modules, called nodes , each responsible for a specific function. Nodes can be combined and customized to create complex robotic behaviors, which allows for greater flexibility and code reusability. Another fundamental aspect of ROS 2 is its communication protocol, based on the Data Distribution Service (DDS) middleware, which ensures efficient, real-time, and reliable communication among distributed modules. This enables developers to build, test, and integrate components independently before combining them into a complete robotic system.

In contrast to ROS 1, ROS 2 does not rely on a centralized *Master* node. Instead, it adopts a fully distributed architecture, where all nodes can discover each other dynamically through the DDS discovery mechanism. This design improves robustness and eliminates the single point of failure present in ROS 1. Each node in ROS 2 is an independent process performing a specific task within the robotic system, such as sensor data processing, actuator control, or path planning. Nodes communicate with each other using *topics*, *services*, and *actions*, depending on whether the communication is continuous, request–response, or goal-oriented. Every node has a unique name and can be written in multiple programming languages, in this case in Python. The communication between nodes uses Quality

of Service (QoS) profiles, which allow fine control over message reliability, latency, and delivery guarantees, which is crucial for real-time applications. This modular and distributed architecture of ROS 2 enables scalable and efficient development of complex robotic systems.

The simulation environment for BFMC 2025 was developed in Gazebo 11 using ROS 1 and faithfully reproduces the behavior of the real vehicle. The car model incorporates custom Gazebo plugins that simulate the Ackermann steering mechanism, GPS, and IMU sensors. The interaction between the simulated vehicle and the control system was designed to closely mimic real-world operation: sensor readings, actuation delays, and the computation of the vehicle's geometric center are handled identically to the physical car.

Building upon the original BFMC competition framework, the DEI UNIPD team extended the existing codebase by replacing the ROS 1 communication layer with a ROS 2 implementation while maintaining the same functionality and message structure. This migration preserved full compatibility with the original measurement and communication logic, ensuring that control algorithms, delay characteristics, and data interpretation remain consistent across both simulated and physical environments. As a result, the transition from simulation to the real platform can be achieved seamlessly, without requiring modifications to the core control logic.

2.4.2 Gazebo

In order to validate the algorithms in a virtual setting before deployment on the physical vehicle, the Gazebo simulator is integrated with ROS 2. Gazebo offers a high-fidelity physics engine and realistic sensor modeling, allowing for accurate testing of vehicle dynamics, path planning, and control strategies under diverse environmental conditions. By linking Gazebo with ROS 2 topics and services, the simulation provides a closed-loop framework where the planner and controller can be evaluated in real time. This can give us the possibility of reproducing the same experiments and reduce the risk of hardware damage during early development phases.

The Gazebo simulator replicates the BFMC competition environment and is intended for integration and functional testing. Due to its lower level of environmental noise compared to the real-world setup, it is primarily used for testing.

Dynamic Vehicle Model in Gazebo

The vehicle model used for simulation in Gazebo is based on a 1/10 scale car-like robotic platform used in BFMC. Model is configured as a fully dynamic system,

where vehicle motion results from the interaction of applied forces, torques, and contact dynamics rather than from kinematic equations. This allows for the realistic reproduction of longitudinal and lateral behaviors such as acceleration, braking, steering, and wheel slip, which are essential for control system development and validation.

The simulation is governed by Gazebo's Open Dynamics Engine (ODE), which integrates the Newton-Euler equations of motion for all rigid bodies. The model consists of several links: the chassis, four wheels, and two steering connectors that are interconnected through revolute and fixed joints. The chassis serves as the main inertial body, while each wheel is represented as a dynamic cylindrical link interacting with the ground through frictional contact. Front wheels are attached via steering joints, allowing rotation about the vertical axis, whereas the rear wheels are fixed to the chassis through revolute joints.

Vehicle actuation is implemented through a control interface that converts high-level commands provided by the MPC controller into low-level wheel and steering actions. The interface receives linear velocity v and steering angle δ commands computed at the vehicle's center of gravity from ROS. Based on the vehicle's Ackermann geometry, these commands are converted into wheel angular velocities and front steering angles. The vehicle geometry used is Ackermann. Each wheel and steering joint is regulated by PID controller to ensure smooth actuation and dynamic stability. This configuration enables direct testing of control algorithms within a realistic closed-loop simulation, where commanded speeds and steering inputs produce physically consistent vehicle motion.

The model's motion evolves from the resultant forces generated by wheel torques, steering inputs, and frictional contact with ground. As simulation operates in a fully dynamic mode, lateral slip and nonzero v_y components may arise during sharp steering or low-friction conditions, replicating realistic vehicle dynamics. This property makes the model suitable for the validation of low-level control algorithms such as velocity, steering, and trajectory tracking controllers.

Characteristic	**Description**
Simulation type	Fully dynamic (ODE physics engine)
Control scheme	Ackermann steering
Actuation method	PID-based steering and velocity control
Contact model	Frictional ground interaction ($\mu = 0.9$)
Suspension	Rigid chassis (no suspension model)
Aerodynamics	Not modeled
Sensors	Simulated IMU and GPS

Table 2.1: Summary of key physical and control characteristics of the Gazebo vehicle model

Figure 2.4 illustrates the hierarchical organization of vehicle model, including the chassis, wheel, and steering subsystems, and the flow of control commands through the ROS interface.

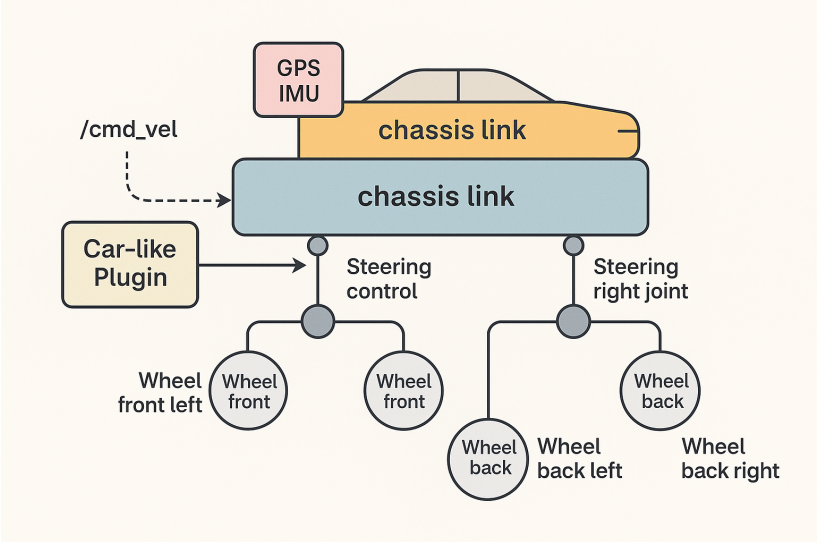


Figure 2.4: Schematic representation of the simulated vehicle and its control architecture.

Ackermann's steering angle model - relationship between bicycle model and four wheel vehicle

As previously discussed, when modeling four-wheel ground vehicles, the bicycle model is often used as a simplified representation. However, when translating the steering angle of the bicycle model to the actual wheels of a car, two approaches are possible: either both front wheels are steered by the same angle, or Ackermann steering geometry is applied. The key idea Ackermann geometry is that the inner wheel turns slightly sharper than the outer wheel. This ensures that both wheels follow circular paths that share a center of a turning circle, also known as instantaneous center of curvature (ICC) , which reduces tire slip when cornering. As a result, the single steering angle δ from the bicycle model is mapped to two different angles: δ_l for the left wheel and δ_r for the right wheel. Which wheel takes the role of the "inner" one depends on the turning direction. During a left turn, the inner wheel is the left wheel, whereas in a right turn, the inner wheel is the right one.

The Ackermann steering can be expressed geometrically using already introduced distances: the wheel base L and lateral distance from CoG to wheels d.

Additionally, inner and outer steering angle, δ_i and δ_o, depend on the turning radius r, which is defined as the distance between the ICC and the vehicle's center. From these, the inner and outer steering angles can be calculated relative to the vehicle's geometry:

$$tan(\delta) = \frac{L}{r}, \quad tan(\delta_i) = \frac{L}{r - \frac{d}{2}}, \quad tan(\delta_o) = \frac{L}{r + \frac{d}{2}} \tag{2.11}$$

Thus, it is possible, after geometrical transformations, to get:

$$\delta_i = tan^{-1}(\frac{2L\ sin\delta}{2L\ cos\delta - d\ sin\delta}), \quad \delta_o = tan^{-1}(\frac{2L\ sin\delta}{2L\ cos\delta + d\ sin\delta}) \tag{2.12}$$

Representing the steering angle in this form gives a solution even when $\delta = 0$, and it depends only on parameters of the car, not the curvature itself.

2.4.3 Raspberry Pi 5

The Raspberry Pi 5 (RPi5) is a compact single-board computer well-suited and often used for real-time control and robotics applications. In this thesis, the Raspberry Pi 5 is used as the real-time controller for the simulated vehicle, in addition to previously serving as a real time controller for physical car in the BFMC Challenge. Gazebo provides the virtual environment where the car and sensors are modeled, the RPi5 executes the control algorithms that compute steering, throttle, braking and velocity commands. These commands are sent to Gazebo through a communication interface such as ROS topics or sockets, effectively creating a hardware-in-the-loop (HIL) setup. This approach allows the control software to run on actual hardware under realistic timing conditions, while still testing the vehicle's behavior safely in simulation.

When running a control loop entirely within Gazebo, the simulation often runs significantly slower than real time because Gazebo must perform many computationally heavy tasks at each simulation step - such as physics calculations, collision detection, sensor emulation, and graphical rendering. These operations consume CPU and GPU resources, and if they cannot be completed within the allocated real-time interval, Gazebo's Real Time Factor (RTF) drops below 1. This means that for every one second of simulated time, several seconds of real-world time may pass. Consequently, the control loop, which typically executes at a fixed rate based on simulation time, appears to run very slowly in real time because the simulation clock itself is advancing slowly.

In contrast, when the same controller runs on real hardware like a Raspberry Pi 5, it is no longer tied to Gazebo's simulation clock but instead operates on the

hardware's own wall-clock time. This allows the control loop to execute exactly at its intended real-time frequency without being constrained by the computational load of a simulated environment. As a result, the controller running on the Raspberry Pi feels much faster and more responsive, even though the underlying control algorithm is the same. In summary, the observed difference arises not from the controller's efficiency, but from the fact that Gazebo's simulation time progresses slower than real time, while hardware execution is synchronized with real-world timing.

2.5 Parameters and values used for modeling vehicles

The vehicle parameters play a crucial role in the representation of the system dynamics and realistic behavior in both simulation and control design. The parameters and scaling values used for different vehicle configurations, are presented in this section, using three levels of abstraction: a 1/43 scaled vehicle, a 1/10 scaled vehicle, and a full-scale (real-sized) vehicle. The comparison between these models highlights the effects of scaling on vehicle dynamics and provides the foundation for evaluating control strategies across different scales.

2.5.1 1/43 Sized Vehicles

The model has been tested with different parameters and different size of vehicles, so that we can make sure the model is well posed. Starting from the smallest vehicle size 1/43 scaled vehicle, moving to the bigger ones 1/10 sized and full sized vehicle, each size has its own challenges. In case of the small 1/43 scaled vehicle, the values and parameters were taken from [9] for both kinematic and dynamic model. Note that $C_{r2} = \frac{1}{2}\rho\, C_d A$.

Parameter	**Symbol**	**Value**
Total vehicle mass	m	$0.0467kg$
Total inertia	I_z	$5.6919e^{-5}kgm^2$
Front axle to CoG	l_f	$0.0308m$
Rear axle to CoG	l_r	$0.0305m$
Motor parameter 1	C_{m1}	$12\frac{m}{s^2}$
Motor parameter 2	C_{m2}	$2.17\frac{1}{s}$
Zero order friction parameter	C_{r0}	$0.1\frac{1}{m}$
Second order friction parameter	C_{r2}	$0.6\frac{m}{s^2}$

Table 2.2: Geometry of the 1/43 sized vehicle and Additional Physical Parameters

Parameters	*front tire*	*rear tire*
B_c	3.47	3.173
C_c	0.1021	0.01921
D_c	5.003	19.01

Table 2.3: Parameters for estimating Lateral Tire Forces with Pacejka's Magic Formula for 1/43 sized vehicle

2.5.2 1/10 Sized Vehicles

When it comes to 1/10 scaled vehicles, parameters are based on the vehicle used in Gazebo simulator and the BFMC 2025.

Parameter	**Symbol**	**Value**
Total vehicle mass	m	$1.415kg$
Total inertia	I_z	$0.17423kgm^2$
Distance from front to rear wheels	L	$0.267m$
Tire friction coefficient	μ_{wheel}	0.9
Maximum steering angle	δ_{max}	$\pm 30^\circ$

Table 2.4: Vehicle Geometry of 1/10 sized vehicle

For calculating aerodynamic drag force, the frontal section area A can be approximately calculated based on the Gazebo model and equals to $A = 0.021\,m^2$. The drag coefficient and air density are equal to $C_d = 0.32$, $\rho = 1.225\,kg/m^3$,while the mass and the total inertia are approximated from the Gazebo model of the car.

Determination of Moment of Inertia, Center of Gravity and Vertical Load

Parameter	Symbol / Link	Value
Chassis mass	m_{chassis}	$0.26244kg$
Car body mass	m_{body}	$1.00000kg$
Wheel mass (each)	m_{wheel}	$0.038kg$
Steering link mass (each)	m_{steering}	$0.001kg$
Chassis inertia (about z)	$I_{zz,\text{chassis}}$	$0.00323293kgm^2$
Car body inertia (about z)	$I_{zz,\text{body}}$	$0.16666700kgm^2$
Wheel inertia (about z, each)	$I_{zz,\text{wheel}}$	$0.00001283kgm^2$
Steering link inertia (about z, each)	$I_{zz,\text{steering}}$	$0.00012500kgm^2$
Front wheel x-position	x_f	$0.1185m$
Rear wheel x-position	x_r	$-0.1485m$
Wheel lateral offset	y_{wheel}	$0.072m$
Car body CoG offset	$(x_{\text{body}}, y_{\text{body}})$	$(-0.0215, 0.0)m$

Table 2.5: Physical Parameters of the Vehicle based on the Gazebo Model

The yaw moment of inertia I_z of the vehicle was determined by applying the *parallel-axis theorem* to the individual inertial properties of each rigid body defined in the simulation model. Each link in the system possesses a known mass, CoG position relative to the chassis reference frame, and a local inertia tensor derived from its geometry and material density. The total moment of inertia about the vertical z-axis through the chassis origin was computed as the summation of the intrinsic yaw inertia of each link and the additional contribution due to its spatial offset. Mathematically, this is expressed as:

$$I_{z,total} = \sum_i \left(I_{zz,i} + m_i(x_i^2 + y_i^2) \right) \tag{2.13}$$

where $I_{zz,i}$,i is the local yaw moment of inertia of the i^{th} link about its own CoG, m_i is its mass, and (x_i, y_i) represent its CoG coordinates relative to the chassis reference frame. This approach captures both the *distributed mass effects* and the *spatial configuration* of each vehicle component, using the inertial and positional data extracted directly from the SDF model. The resulting total inertia is shown in **??**.

The longitudinal position of the vehicle's CoG plays a crucial role in accurately describing its dynamic behavior. The distances from the CoG to the front and rear axes, denoted as l_f and l_r respectively, are typically derived from the static weight

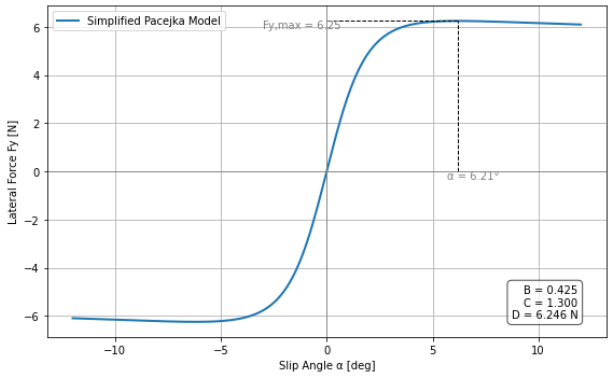


Figure 2.5: Pacejka Magic Formula- the relationship between the Tire Slip Angle and Lateral Force

distribution of the vehicle. The distribution can be approximated by analyzing the placement of heavy components such as the battery, sensors, and drive motors. For small ground vehicles, the CoG often lies slightly behind the midpoint of the wheelbase, typically between 45% and 55% of its total length, but for simplicity we will assume that $l_r = l_f = \frac{L}{2}$. Based on this assumption and a wheelbase of $L = 0.267$ m, the distances can be approximated as $l_f = l_r = 0.1335$ m.

Based on these values, we can determine a simplified Pacejka's magic Formula parameters for lateral tire forces as disscussed in the previous chapter. Firstly, we need to estimate the vertical load F_z on each tire using static weight distribution:

$$F_{zf} = \frac{l_r}{L}mg, \quad F_{zr} = \frac{l_f}{L}mg \tag{2.14}$$

The peak lateral force D was estimated as $D = \mu F_z$. The shape factor C was selected based on literature values for small-scale rubber tires, typically ranging between 1.2 and 1.4, a value of $C = 1.3$ was adopted to balance curve smoothness and realistic peak behavior. As previously mentioned, we want to ensure that the slip angle peaks between 5° and 8° before the force begins to decline due to tire saturation, $\alpha = 6.21°$. Finally, the stiffness factor B was calculated from the small-angle linearized cornering stiffness relationship $C_\alpha = BCD$, where C_α represents the slope of the linear tire model near $\alpha = 0$. The cornering stiffness C_α was approximated from empirical ratios found in scaled-vehicle studies. This parameterization captures both the linear and nonlinear regions of lateral tire force generation without requiring full experimental characterization. The figure below shows the shape of the curvature with chosen Pacejka parameters. Note that the values chosen are the same for both front and rear tires.

Parameter	*front and rear tire*
B_c	0.425
C_c	1.300
D_c	6.246

Table 2.6: Parameters for estimating Lateral Tire Forces with Pacejka's Magic Formula

2.5.3 Real Sized Vehicles

Finally, for the modeling of the real size vehicles, values are taken from the [29] for Jaguar X-Type, and are presented in the Table 2.7

Parameter	**Symbol**	**Value**
Total vehicle mass	m	$2200kg$
Total inertia	I_z	$3344kgm^2$
Front axle to CoG	l_f	$1.432m$
Rear axle to CoG	l_r	$1.472m$
Wheel base to CoG	h	$0.4m$
Half car width	$\frac{d}{2}$	$0.8125m$
Tire radius	r	$0.333m$

Table 2.7: Full Size Vehicle Geometry and Additional Physical Parameters

In this model, tire parameters for each tire on the vehicle has been determined, but since we will be using bicycle model the parameters used for calculating lateral forces with Pacejka's Magic formula are accordingly approximated. The road friction value is considered to be the input, but we are going to use the same value as for other models, $\mu = 0.9$.

Parameter	*front*	*rear*
B_c	0.201	0.174
C_c	1.3	1.3
D_c	4868.778	4753.954
E_c	-1.2293	-1.177

Table 2.8: Parameters for estimating Lateral Tire Forces with Pacejka's Magic Formula

All of these models are tested with both kinematic and dynamic bicycle models, to asses their performances.

2.6 Model Predictive Controller Design

Two different MPCs were implemented to control the vehicle: one based on the *spatial-domain model* and another based on the *time-domain model*. Controllers were formulated and solved using the Acados framework. Both kinematic and dynamic model have similar controller structure, constraints and solver configuration. Detailed specifications will be described with the results of the specific models.

One of the first and most important steps when formulating an MPC controller is the definition of the cost function which captures the control objectives through the stage and terminal cost terms. The choice of which states and inputs are penalized or weighted strongly influences the controller's behavior.

When the problem is described in time domain, all systems states are typically weighted in the cost function, since the objective is to regulate the full trajectory over time. On the other hand, in spatial domain reformulation, only the lateral deviation and heading error with respect to the reference are directly weighted, as our goal is to ensure accurate path following and this way other states are weighted at second hand. However, if there is a desire for a certain speed profile, it can be incorporated into the cost function, but it must be parametrized with respect to the spatial variable s.

In Acados, the cost function is implemented in a nonlinear least-squares (NLS) form:

$$J = \sum_{k=0}^{N-1} \|y_k - y_{\text{ref},k}\|_W^2 + \|y_N - y_{\text{ref},N}\|_{W_e}^2 , \tag{2.15}$$

where the first term represents the stage cost accumulated along the prediction horizon, and the second term represents the terminal cost at the end of the horizon. The weighting matrices W and W_e depend on the specific formulation (time or spatial domain), the desired controller performance, and the discretization parameters such as the sampling time or spatial step. Larger weights on certain states or inputs penalize their deviation more strongly, leading to tighter tracking but possibly more aggressive control actions.

Additionally, it is a common practice to include the control inputs in the stage cost by assigning them suitable weighting factors. This penalization prevents excessive or abrupt control actions, that results in smoother actuator commands and improved overall system stability. Relatively small weights were assigned to the control inputs, allowing the controller to maintain steering flexibility while avoiding overly aggressive acceleration or braking commands. The weighting

matrices can be expressed as:

$$W = \begin{bmatrix} Q & \mathbf{0}_{n_x \times n_u} \\ \mathbf{0}_{n_u \times n_x} & R \end{bmatrix}, \qquad W_e = Q_e \, \Delta s, \tag{2.16}$$

where $Q \in \mathbb{R}^{n_x \times n_x}$ represents the weighting matrix on the states, $R \in \mathbb{R}^{n_u \times n_u}$ represents the weighting matrix on the control inputs in the stage cost, and $Q_e \in \mathbb{R}^{n_x \times n_x}$ denotes the terminal cost matrix. Note that they are all diagonal.

Let us consider that the control input of the model is $u = \begin{bmatrix} a & \delta \end{bmatrix}^\top$ and kinematic bicycle model. If the MPC controller is based on the time-domain model, the states are defined as:

$$\xi_t = \begin{bmatrix} x & y & \psi & v \end{bmatrix}^\top \tag{2.17}$$

The cost function is expressed as:

$$y_k = \begin{bmatrix} x \\ y \\ \psi + \beta \\ v \\ u \end{bmatrix}, \quad y_N = \begin{bmatrix} x \\ y \\ v \end{bmatrix} \tag{2.18}$$

where $\theta = \psi + \beta$, and slipping angle is defined as $\beta = \arctan\left(\frac{l_r \tan(\delta)}{L}\right)$. This is the consequence of the definition of the reference, that provides the desired vehicle orientation (direction angle), rather than the vehicle's instantaneous direction of motion.

When the controller formulation based on the spatial-domain model, the state vector defined as:

$$\xi_s = \begin{bmatrix} e_\psi & e_y & x & y & \psi & v \end{bmatrix}^\top \tag{2.19}$$

The stage and terminal cost terms are defined by:

$$y_k = \begin{bmatrix} e_\psi + \beta \\ e_y \\ u \end{bmatrix}, \quad y_N = \begin{bmatrix} e_\psi \\ e_y \end{bmatrix}. \tag{2.20}$$

Similarly to the time domain formulation, the slip angle is considered in the cost definition of lateral tracking error. Including the slip angle term ensures its actual motion direction in the case of kinematic bicycle models, which is significant at higher steering angles or lower velocities.

In the case of the dynamic bicycle model, the effect of the slip angle already appears implicitly through the vehicle dynamics, since the model inherently captures tire slip and lateral forces. However, by introducing slipping angle in the heading error and thus formulating the heading error in terms of the direction angle, the controller is driven to align the actual motion direction with the desired trajectory, rather than constraining the chassis orientation alone. This approach allows the MPC to exploit the natural relationship between sideslip, lateral forces, and achievable curvature. For dynamic bicycle models, MPC controller based on the time-domain model is defined as:

$$\xi_t = \begin{bmatrix} x & y & v_x & v_y & \psi & \dot{\psi} \end{bmatrix}^\top \tag{2.21}$$

The cost function is expressed as:

$$y_k = \begin{bmatrix} \xi_t \\ u \end{bmatrix}, \quad y_N = \xi_t \tag{2.22}$$

When the controller formulation based on the spatial-domain model, the independent variable is the arc length s, the state vector defined as:

$$\xi_s = \begin{bmatrix} e_\psi & e_y & x & y & v_x & v_y & \psi & \dot{\psi} \end{bmatrix}^\top \tag{2.23}$$

The stage and terminal cost terms are defined by:

$$y_k = \begin{bmatrix} e_\psi + \beta \\ e_y \\ a \\ \delta \end{bmatrix}, \quad y_N = \begin{bmatrix} e_\psi + \beta \\ e_y \end{bmatrix}. \tag{2.24}$$

2.6.1 Constraints

In order to represent physical limits of the model and define the feasible operating region of the system, we define constraints. The constraints that are imposed on the control inputs are aligned with the behaviour and limits we would expect from physical scaled vehicle. These limits ensure that the control actions remain within feasible and safe operating ranges for the actuators. In cases where specific scenarios required adjustments to these parameters, such modifications are explicitly highlighted, along with the corresponding justification and their impact on the resulting system performance.

Type	Constraint Range
Longitudinal acceleration	$-1\frac{m}{s^2} \leq a \leq 1\frac{m}{s^2}$
Steering angle	$-30^\circ \leq \delta \leq 30^\circ$

Table 2.9: Hard input constraints imposed in the MPC formulations

The state constraints are introduced in the spatial domain reformulation, for the deviation from trajectory. They are considered hard constraints as well, since the violation would lead to unsatifsactory performance or violating the track geometry and limitations.

Type	Constraint Range
Heading error	$-20^\circ \leq e_\psi \leq 20^\circ$
Lateral error	$-0.25m \leq e_y \leq 0.25m$

Table 2.10: State constraints imposed in the spatial MPC formulation

2.6.2 Solver Configuration

The solver parameters used for the spatial MPC are shown in Table2.11.

Parameter	Value / Method
QP solver	FULL CONDENSING QPOASES PARTIAL CONDENSING HPIPM
Hessian approximation	Gauss-Newton
Integrator type	Explicit Runge-Kutta (ERK)
NLP solver type	Sequential Quadratic Programming (SQP) Real-Time Iteration (RTI)
Prediction horizon	$T_f = N_{\text{horizon}}\,\Delta t$

Table 2.11: Solver settings for the MPC

Most of the tests were conducted using the qpOASES solver. However, in cases where convergence issues or unexpected behavior occurred, the solver was temporarily switched to HPIPM to verify whether the problem originated from the solver itself. The Real-Time Iteration scheme was employed in the Gazebo simulation environment, as it provides faster solve times per iteration and enables a more realistic representation of the vehicle's motion during real-time execution.

Part III

Results

3 Results and Discussion

The initial experiments were conducted using identical models for both simulation and control to validate the correctness of the controller design. Tests were first performed in the time domain and subsequently extended to the half-spatial and full-spatial reformulation. No model mismatch or discretization differences were introduced, thus perfect tracking performance was expected. The key distinction between the formulations lies in the evolution of vehicle velocity along the trajectory. Model performances were compared and analyzed, after which separate models were employed: the spatial-domain formulation for control and the time-domain formulation for simulation, to better represent real-world operating conditions. The models were introduced from simpler ones to more complicated, starting from the kinematic bicycle model.

3.1 Elliptical and S shaped trajectories

Big part of the experiments were based on parameters corresponding to a 1/10-scale vehicle, with tracks scaled accordingly. When it comes to the elliptical reference trajectory, it is defined as mentioned in the previous section as $X(t) = a\ cos(\omega t), Y(t) = b\ sin(\omega t)$, where a and b denote the semi-major and semi-minor axes, respectively, and ω represents the angular frequency governing the vehicle motion. In case of the 1/10 scaled vehicle, the parameters were set to $a = 4\,m$, $b = 2.2\,m$, $\omega = 0.2\,Hz$. For the S-shaped trajectory, curvature was generated via cubic spline interpolation, with an $x : y$ progression ratio of 10:6 (i.e., for every 1 m along the x-axis, the vehicle advances 0.6 m along the y-axis). In all the simulations in this section SQP methods were used, along with Full Condensing qpOASES solver.

3.1.1 Kinematic Bicycle Model

As already discussed, the kinematic bicycle model serves as the fundamental representation of the vehicle dynamics, providing simplified yet effective description of vehicle motion by capturing the essential kinematic relationships between position, orientation, and velocity. The model assumes no lateral or longitudinal tire slip. Note that, in kinematic model the side slip of the whole vehicle β actually represents the consequence of steering, shifted to the CoG and is used in the modeling of the vehicle.

Time Domain

The first model that is going to be analyzed is the kinematic bicycle model with time domain formulation. The prediction horizon $N_{horizon} = 20$, and the tests were done with the sampling times $\Delta t = 0.01s$ and $\Delta t = 0.05s$, but there was no difference in reference tracking in these two cases. The parameters used correspond to a 1/10-scale vehicle, as this model is our main interest.

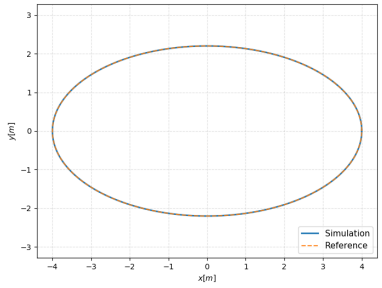
Stage Weights						**Terminal Weights**		
Q_x	Q_y	$Q_{\psi+\beta}$	Q_v	R_a	R_δ	Q_{x_N}	Q_{y_N}	Q_{v_N}
10	10	10	10	0.01	0.01	20	20	10

Table 3.1: Weights of Cost Function Time Domain MPC - Kinematic Bicycle Model

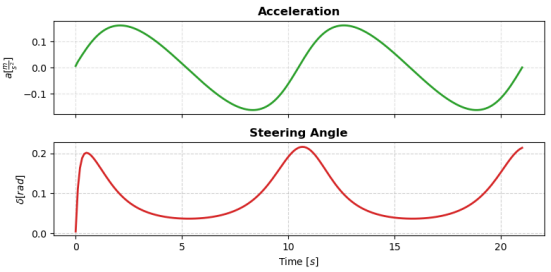
For both ellipsoidal and S shaped trajectory, the weights for the MPC cost function are reported in the Table 3.1. The weights are chosen to balance accurate trajectory tracking with smooth and feasible control inputs. Equal weighting was assigned to the position and orientation states, reflecting the need for uniform tracking accuracy across all geometric components of the reference path. The same weight was applied to the velocity term to ensure that the vehicle maintains the desired speed profile without allowing excessive acceleration or deceleration. The control-input penalties were kept small in order to avoid slow reactions and overly restrictive actuation limits while still preventing aggressive or oscillatory behaviour. Since this is the simplest modeling case, high magnitude weights were not required. Higher terminal weights were introduced for position to tighten convergence at the end of the horizon, while the terminal weight on velocity is the same as the stage one.

Metric	$\Delta t = 0.05\,\mathrm{s}$	$\Delta t = 0.01\,\mathrm{s}$
Average MPC solve time [ms]	4.856	2.951
Maximum MPC solve time [ms]	24.757	19.168
Minimum MPC solve time [ms]	0.336	0.224
Total simulation time [s]	2.498	8.430
Real-time ratio ($t_{\mathrm{loop}}/\Delta t_{\mathrm{OCP}}$)	0.12	0.4

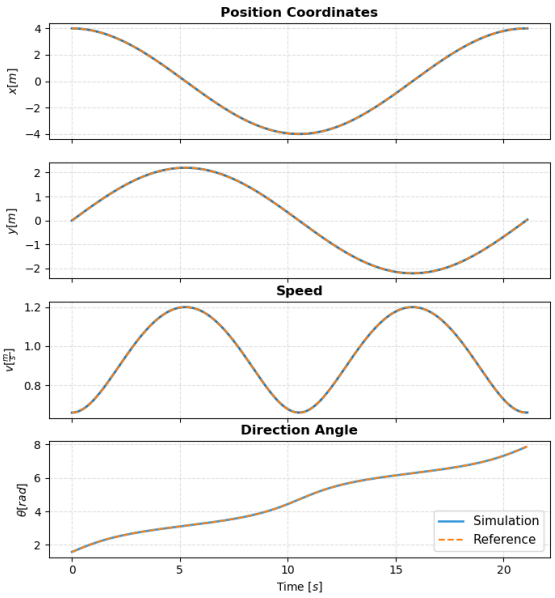
Table 3.2: Comparison of MPC Performance for Different Sampling Times on Ellipsoidal Track



(a) Trajectory



(b) Control inputs



(c) States

Figure 3.1: Kinematic Bicycle Model -Following of Ellipsoidal reference path, simulation and controller in Time Domain

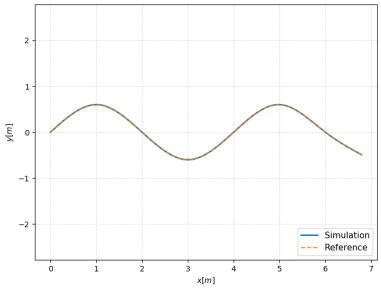
The Figures 3.1 and 3.2 show results of simulation of tracking elliptical and S shaped reference trajectories. As expected, the controller followed the reference path with no steady-state error. The velocity profile in both cases exhibits periodic acceleration and deceleration phases that are directly correlated with the curvature changes along the path. In the S-shaped trajectory, the average velocity is higher, but due to the shorter and alternating curvatures, the tracking of the ellipsoidal track produces a larger range of velocity. This is consistent with the curvature variation on the tracks, and is realistically representing vehicle behaivour. The acceleration behaviour ensures smooth transition avoiding sudden jerks, while the steering angle stays consistent with the curvature changes required for the desired motions. The steering angle ranged within ± 0.25 rad for the S–curve and ± 0.20 rad for the ellipsoidal case, following the curvature magnitude of each path. Control inputs remain continuous and with smooth transitions, which in a real-world implementation would be important for the actuators. Steering follows curvature: it oscillates about zero for the S–curve due to curvature sign changes, while it maintains a positive with two broad peaks on the ellipsoidal track, reflecting long arcs of nearly constant sign curvature.

Metric	$\Delta t = 0.05\,\mathrm{s}$	$\Delta t = 0.01\,\mathrm{s}$
Average MPC solve time [ms]	5.543	2.735
Maximum MPC solve time [ms]	22.474	18.815
Minimum MPC solve time [ms]	0.391	0.235
Total simulation time [s]	0.788	2.547
Real-time ratio ($t_{\mathrm{loop}}/\Delta t_{\mathrm{OCP}}$)	0.13	0.38

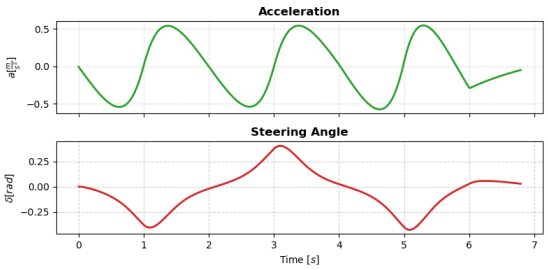
Table 3.3: Comparison of MPC Performance for Different Sampling Times on S shaped track

Tables 3.2 and 3.3 compare MPC solutions with different sampling times $\Delta t = 0.05s$ and $\Delta t = 0.01s$, across both the ellipsoidal and S-shaped tracks. By reducing the sampling time, the control updates become more frequent and consequently subsequent optimization problems are more similar and easier to solve when warm-started, thus leading to a lower average MPC solve time - by 39% on the ellipsoidal track and 51% on the S-shaped track. However, since the controller executes five times more often, the computational load relative to the control period increases, which is reflected in the higher real-time ratio. The real-time ratio reflects the overall computational load per unit of simulated time. Both setups usually remain within real-time limits, except in couple of iterations of $\Delta t = 0.01s$, where the solve time exceeds the real-time feasibility (seen in maximum MPC solve time). Additionally, the total simulation time increases due

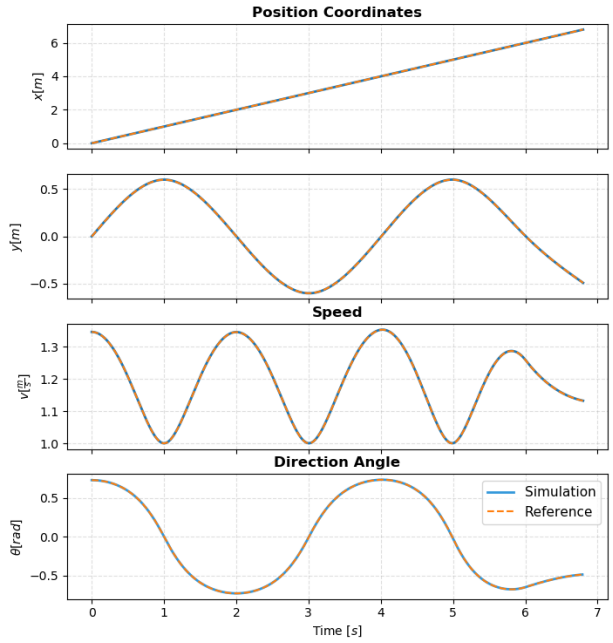
to the higher number of integration steps. Between tracks, the S-shaped trajectory exhibits slightly higher computation times, but the overall simulation time is much lower compared to the ellipsoidal track, since the track itself is shorter.



(a) Trajectory



(b) Control inputs



(c) States

Figure 3.2: Kinematic Bicycle Model -Following of S reference path, simulation and controller in Time Domain

Half-Spatial Domain

A half-spatial domain formulation is the bridge between time and spatial domain formulations of the problem. While we introduce all the same state variables as in the spatial domain approach, the structure relies heavily on the time based

discretization. Additionally to the error states, we introduced the the arc length s as a state itself, which allows the controller to reason the evolution of travelled distance.

The controller step size was chosen as $\Delta s_{\text{ctrl}} = 0.05\,m$, providing sufficiently fine spatial resolution along the path without imposing excessive computational burden. The simulation step size was kept at $\Delta t_{\text{sim}} = 0.01s$ to maintain smooth integration and preserve the accuracy of the time-evolving states. A prediction horizon of $N_{\text{horizon}} = 20$ points was selected as a compromise between tracking performance and computational efficiency, corresponding to a look-ahead distance that captures the curvature variations of both the ellipsoidal and S-shaped tracks.

Parameter	**Symbol**	**Value**
Controller step size	Δs_{ctrl}	$0.05m$
Simulation step size	Δt_{sim}	$0.01s$
Number of prediction horizon steps	$N_{horizon}$	20

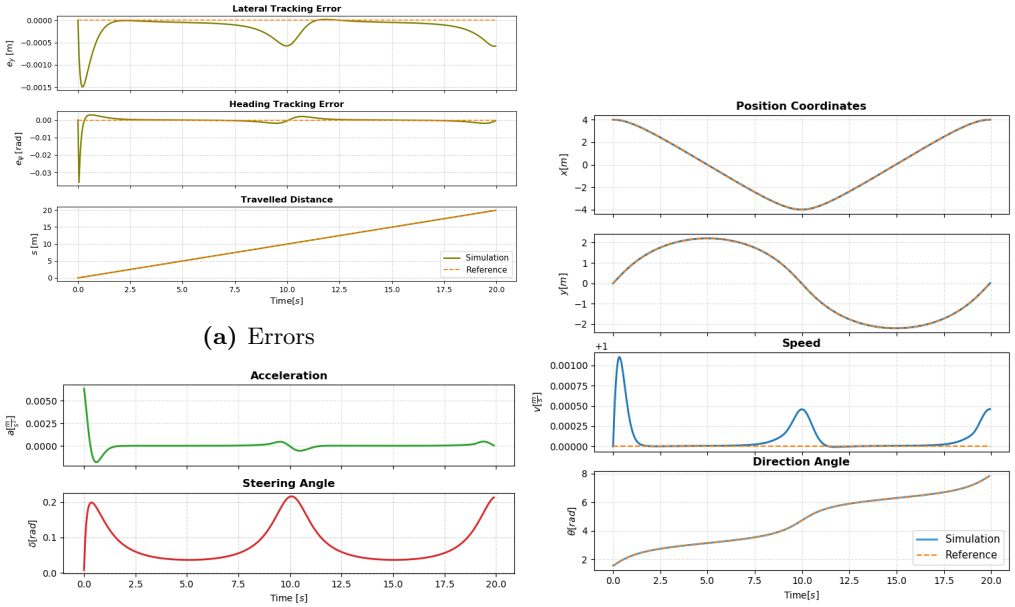
Table 3.4: Parameters of Half Spatial Domain MPC - Kinematic Bicycle Model

The cost-function weights for the half-spatial kinematic MPC shown in Table 3.5 were selected to reflect the specific structure of this formulation, where the model evolves in time but the reference is parameterized in arc length.

A relatively high weight was assigned to the spatial progress variable to ensure that the vehicle advances steadily along the reference trajectory. The lateral error e_y was penalized twice as much as the heading-sideslip term to keep the vehicle centered on the path. Moderate penalty on the heading tracking error allows flexibility in orientation during turning maneuvers. The control-input costs were selected to balance smoothness and responsiveness, with a stronger penalty on acceleration to avoid abrupt speed changes, and a smaller penalty on steering to preserve responsiveness in curved regions. The terminal weights are the same as the stage ones.

Stage Weights					**Terminal Weights**		
$Q_{e_{\psi+\beta}}$	Q_{e_y}	Q_s	R_a	R_δ	Q_{e_ψ}	Q_{e_y}	Q_s
10	20	100	0.1	0.01	10	20	100

Table 3.5: Weights of Cost Function Half Spatial Domain MPC - Kinematic Bicycle Model



(a) Errors

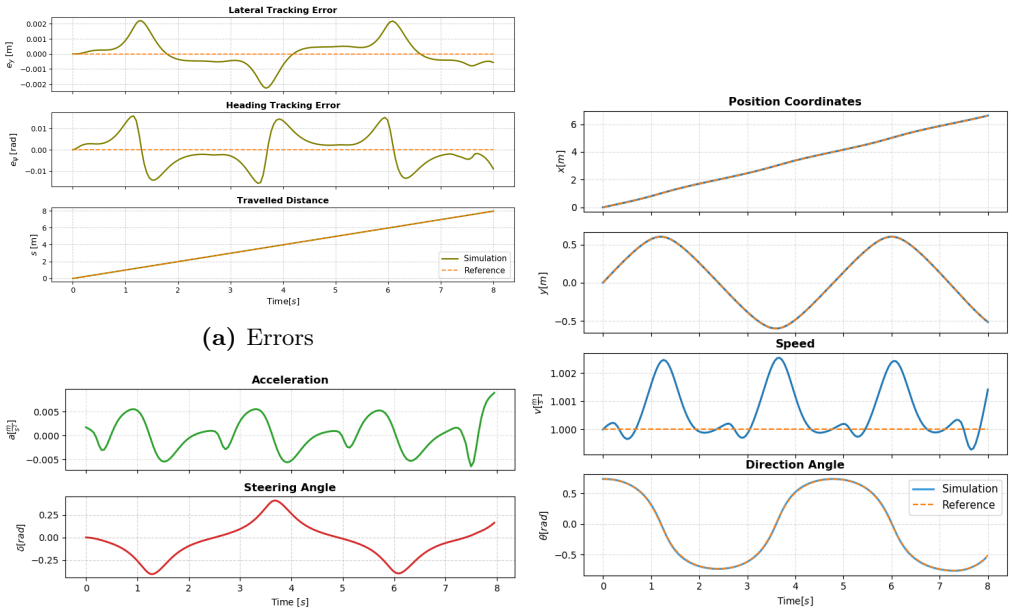
(b) Control inputs

(c) States

Figure 3.3: Kinematic Bicycle Model -Following of Ellipsoidal reference path, simulation and controller in Half Spatial Domain

The performance of the kinematic bicycle model formulated in the half-spatial domain is illustrated in figures 3.3 and 3.4 above. The evolution of the position and orientation shows that the simulated trajectory almost perfectly overlaps with the reference one. The travelled distance increases linearly with time, demonstrating a uniform progression along the spatial coordinate and validating the stability of the half-spatial domain formulation. The corresponding tracking errors remain extremely small, with lateral tracking error below 3 mm for the S-curve and 4 mm for the ellipsoidal trajectory, while the corresponding heading errors did not exceed 1.2×10^{-2} rad. The ellipsoidal case exhibits a slightly larger initial transient before exponentially settling near zero, which is consistent with the abrupt change from straight motion to sustained positive curvature at the lap start. By contrast, the S-curve features very small, alternating error lobes that mirror the curvature reversals.

Compared to the time-domain formulation, noticeable differences arise in the velocity and acceleration profiles. In the half-spatial domain, velocity is not explicitly included as a weighted state in the cost function. Instead, the controller is initialized with a starting velocity and is subsequently free to adjust it according to the optimal solution. Since the reference trajectory is parameterised by the arc length and the spacing between points is nearly uniform, the vehicle tends to maintain a velocity close to its initial value throughout the manoeuvre. The few



(a) Errors

(b) Control inputs

(c) States

Figure 3.4: Kinematic Bicycle Model -Following of S reference path, simulation and controller in Time Domain Half Spatial

observed velocity peaks correspond to regions of higher curvature, where the controller slightly modifies the longitudinal motion to preserve lateral stability and ensure accurate path tracking. This behaviour propagates to the acceleration profile, which exhibits small fluctuations near those curvature transitions, while remaining close to zero elsewhere. The half-spatial domain formulation produces smoother velocity and acceleration profiles than the time-domain counterpart, as time evolution is implicitly managed by spatial progress rather than by explicit timing constraints. The velocity deviation remains within $\pm 2\%$ of the starting value, while the acceleration amplitude does not exceed $6 \times 10^{-3}\,\mathrm{m/s^2}$. The low-magnitude corrections arise only around curvature transitions, while the steering angle resembles the behaviour in time domain.

Spatial Domain

When dealing with time domain formulation, the MPC controller directly accounts for time evolution of the systems states, which allows explicit control of the velocity profile at reference generation level. In contrast, the spatial domain reformulation focuses directly on control of the lateral and angular tracking errors. The controller thus has the freedom of choosing the best way to track the errors, meaning the freedom to determine the most suitable speed profile, as it

is not included in the cost function. Although, a predefined speed profile can be enforced if needed, the main advantage of the spatial domain reformulation is the implicit minimization the total travelling time. Note that, even though it is not needed to have a predefined speed profile, a lower bound on the vehicle velocity must be enforced. If the starting speed is set to zero, problem becomes infeasible due to the spatial formulation relying on the vehicle's motion to progress along the path.

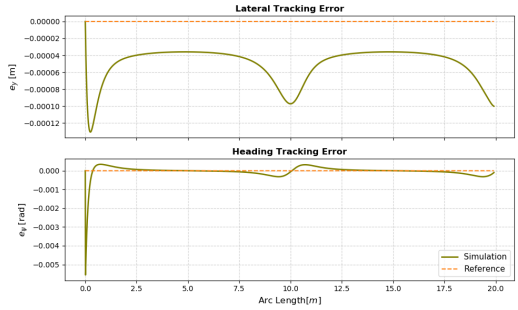
Parameter	**Symbol**	**Value**
Controller step size	Δs_{ocp}	$0.01m$
Simulation step size	Δs_{sim}	$0.01m$
Number of prediction horizon steps	$N_{horizon}$	50

Table 3.6: Parameters of Spatial Domain MPC - Kinematic Bicycle Model

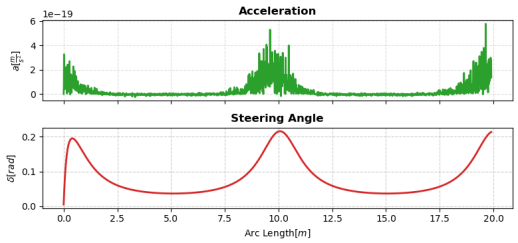
The stage weights chosen for spatial domain kinematic bicycle are presented in the Table 3.7. A high penalty was assigned to the lateral tracking error ($Q_{e_y} = 200$), ensuring that the vehicle remains close to the centreline of the reference path. This is particularly important in the spatial formulation, since any deviation in e_y directly influences the spatial mapping through the term $(1 - \kappa(s)e_y)$, thus may lead to numerical sensitivity if not properly controlled. The heading–sideslip term was penalised more moderately ($Q_{e_{\psi+\beta}} = 10$), allowing sufficient orientation flexibility to follow curved segments. The control-input penalties were set to $R_a = R_\delta = 0.1$, and terminal costs are the same as the stage ones, similarly to previous case.

Stage Weights				**Terminal Weights**	
$Q_{e_{\psi+\beta}}$	Q_{e_y}	R_a	R_δ	Q_{e_ψ}	Q_{e_y}
10	200	0.1	0.1	10	200

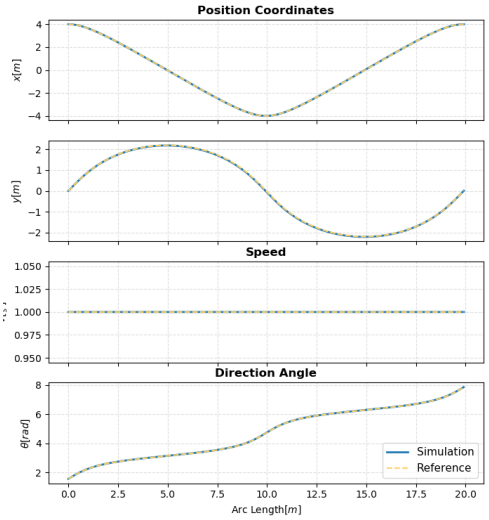
Table 3.7: Weights of Cost Function of Spatial Domain MPC - Kinematic Bicycle Model



(a) Errors



(b) Control inputs



(c) States

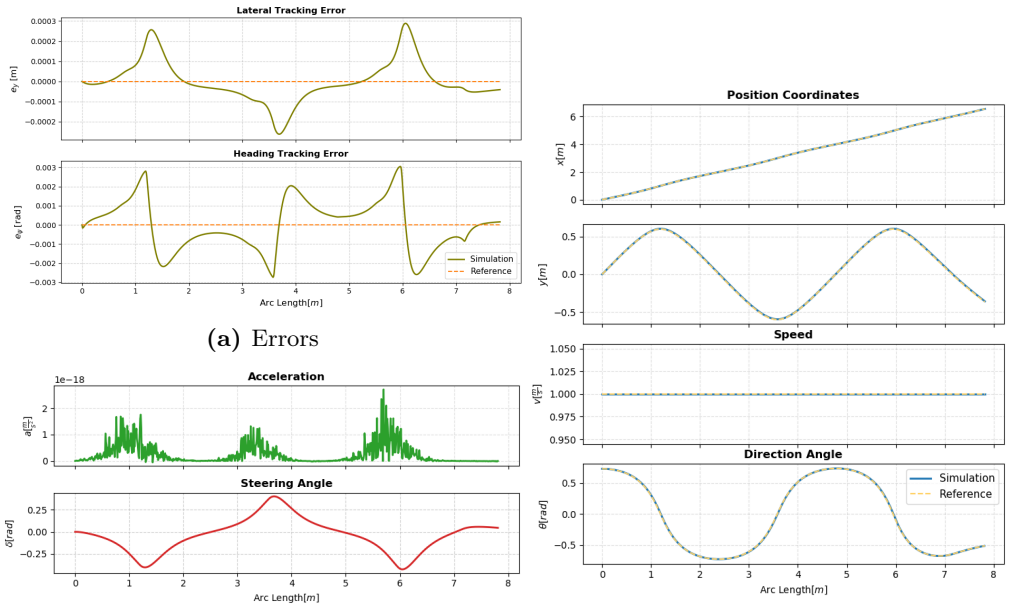
Figure 3.5: Kinematic Bicycle Model -Following of Ellipsoidal reference path, simulation and controller in Spatial Domain

Figures 3.6 and 3.5 report the performance of the kinematic bicycle model formulated in the spatial domain, where all signals are parameterized by arc length s. In both trajectories the simulated states closely overlap the references.

The tracking errors remain very small over the entire lap. On the ellipsoidal track a brief initial transient is observed as the vehicle enters the first curvature on the path, but later the error settles near zero. For the S-curve, the lateral and heading errors exhibit alternating sign with small amplitude, reflecting the curvature reversals.

Control inputs are smooth and consistent with the path curvature. Steering mirrors the curvature profile and did not change behaviour compared to the time and half spatial domain formulations. On the other hand, the longitudinal acceleration almost remains negligible during the whole simulation, with insignificant corrections that arise around curvature transitions. The velocity deviation remains below 1% of the starting value, and the acceleration amplitude is limited to $4 \times 10^{-3}\,\mathrm{m/s^2}$, confirming that the reference speed is preserved in the spatial formulation.

From a control perspective, the spatial-domain formulation produces the smoothest steering and acceleration profiles among the three models. This may be explained by the fact that the control problem is directly solved with respect to the spatial coordinate, avoiding time-scaling distortions.



(a) Errors

(b) Control inputs

(c) States

Figure 3.6: Kinematic Bicycle Model -Following of S reference path, simulation and controller in Spatial Domain

Hybrid spatial–time control - comparison between distance- and time-based updates

The following key step involves testing whether a simulation that employs time domain reformulation could be effectively controlled by a spatial domain controller. This represents very common scenario encountered in experimental setups and real-vehicle applications, where the system naturally progress in time, regardless how controller operates. In order to do so, it is essential to perform an accurate state mapping between the two systems. Since the simulation and control models system's representation varies, we need to extract the relevant states from the simulation to initialize the controller, then apply the control input generated by the spatial domain formulation to time domain formulation, and map controller-internal states back to the simulation. This has been successfully implemented with path projection to compute the spatial error states (e_ψ, e_y, s) at each control step. Note that, the horizon interpretation and discretization differ. A fixed spatial horizon covers different amounts of time depending on speed, so performance guarantees and terminal conditions must be rethought compared with MPC with a fixed time horizon.

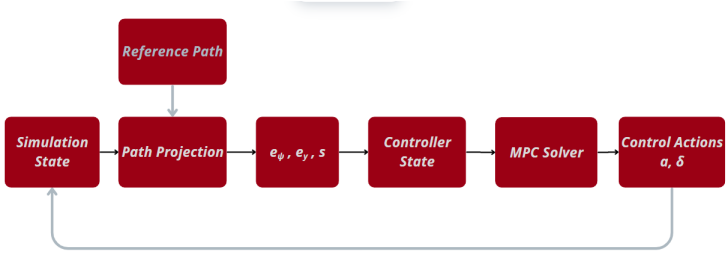


Figure 3.7: Strategy for state mapping

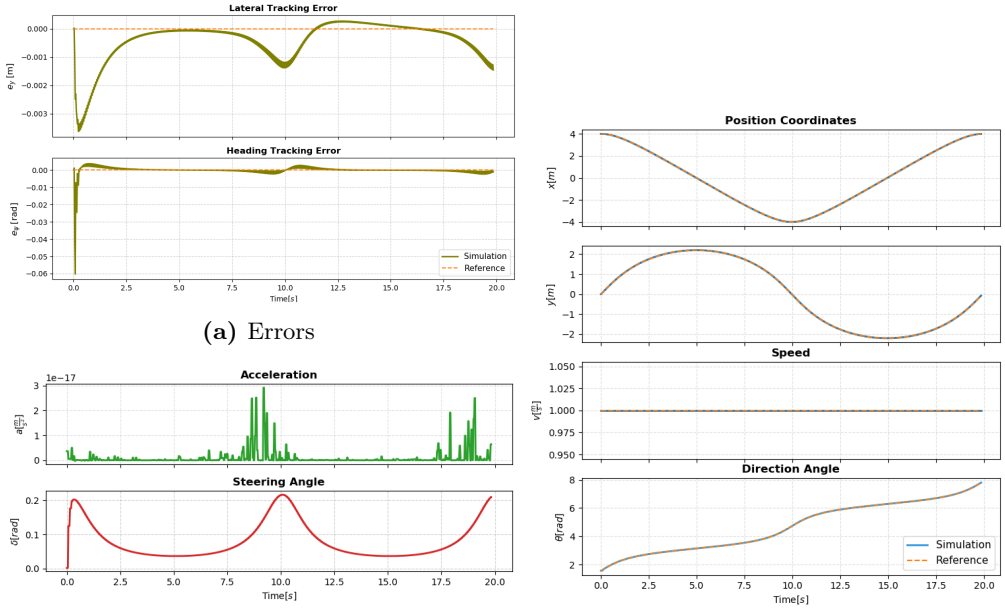
When employing models that operate in two different domains, the spatial and time domains, two main strategies can be considered for updating the control. In the first strategy, the controller is updated based on the travelled distance, meaning the control law is executed only when the vehicle has moved a predefined spatial step Δs_{ctrl}. This approach can be referred to as the distance-based, and it ensures that the control action is applied uniformly along the trajectory, which can be suitable for spatially defined references. it is used in the previous half-spatial and spatial reformulations of the problem. However, since the update rate depends on the vehicle's speed, it may vary significantly-leading to high-frequency updates at high speeds and sparse updates at low speeds, which can complicate real-time implementation. The second strategy consists of time-based updates, where the controller is executed at a constant frequency, i.e. every fixed time interval. This approach simplifies real-time implementation and ensures a consistent update rate regardless of the vehicle's motion. Nevertheless, it may lead to uneven spatial sampling of the control inputs, particularly when the vehicle speed changes significantly.

Parameter	**Symbol**	**Value**
Controller step size	Δs_{ctrl}	$0.05m$
Simulation step size	Δt_{sim}	$0.01s$
Number of prediction horizon steps	$N_{horizon}$	30

Table 3.8: Parameters of Hybrid Distance Based MPC - Kinematic Bicycle Model

The parameters listed in Table **??** were selected for kinematic bicycle model with distance based updates. The controller step size $\Delta s_{\text{ctrl}} = 0.05\,\text{m}$ provides a fine spatial resolution for evaluating curvature and geometric tracking errors. The simulation time step $\Delta t_{\text{sim}} = 0.01\,\text{s}$ maintains numerical stability of the dynamic model while keeping the integration sufficiently smooth. A prediction horizon of $N_{\text{horizon}} = 30$ steps was chosen to provide enough look-ahead distance for anticipating curvature changes on both the ellipsoidal and S-shaped trajectories. The

extended horizon also compensates for the fact that spatial progress and time evolution are decoupled in the hybrid formulation. The weights of the cost function of the spatial domain controller employs a stronger penalty on the combined heading-sideslip term compared to the purely spatial formulation. The heading error is penalized more heavily ($Q_{e_{\psi+\beta}} = 10^3$), as it directly influences the lateral deviation through the spatial mapping and, in the hybrid setup, inconsistencies between the spatial and time dynamics can amplify these effects. A higher weight on $e_{\psi+\beta}$ therefore ensures that the controller maintains proper vehicle orientation and stabilizes the spatial progress.



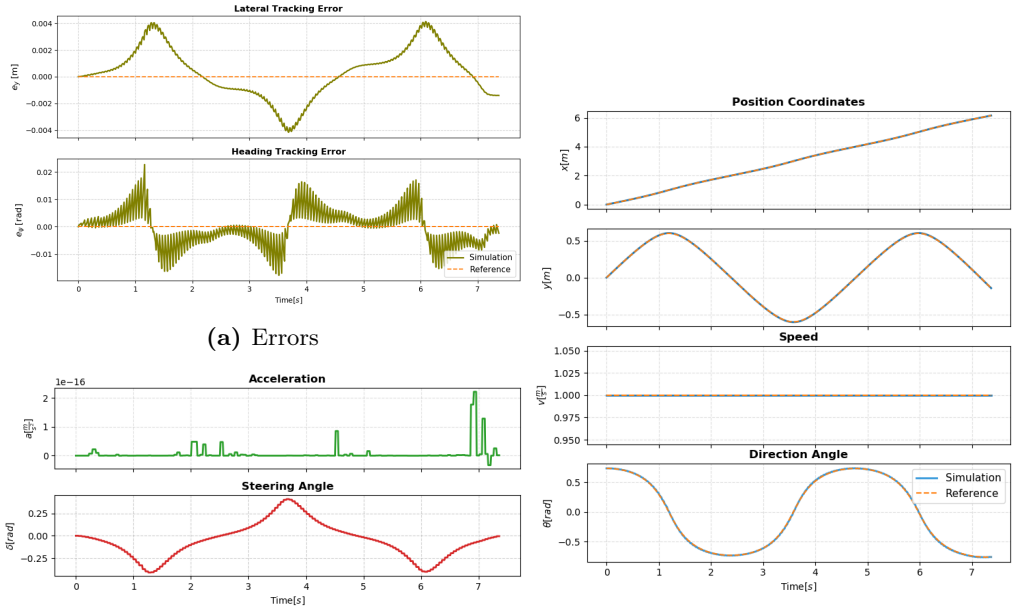
(a) Errors

(b) Control inputs

(c) States

Figure 3.8: Kinematic Bicycle Model-Following of Ellipsoidal reference path, in Hybrid Spatial-Time Domain Distance Based

The results, shown in Figures 3.8 and 3.9, demonstrate that distance-based approach with hybrid spatial-time control maintains excellent tracking accuracy. The velocity remains nearly constant throughout both the ellipsoidal and S-shaped trajectories, with only small fluctuations occurring around high-curvature regions. The lateral and heading tracking errors remain minimal, comparable to those achieved in the fully spatial formulation. Still, the maximum lateral deviation does not exceed 3×10^{-3} m, and the heading error remains below 6×10^{-2} rad, demonstrating precise path following despite the time–space coupling. This strategy provides smooth tracking, particularly in curved sections, as each update corresponds to an equal portion of the path. However, it results in a variable



(a) Errors

(b) Control inputs

(c) States

Figure 3.9: Kinematic Bicycle Model -Following of S reference path, in Hybrid Spatial-Time Domain Distance Based

computational rate, since the update frequency depends on the instantaneous velocity.

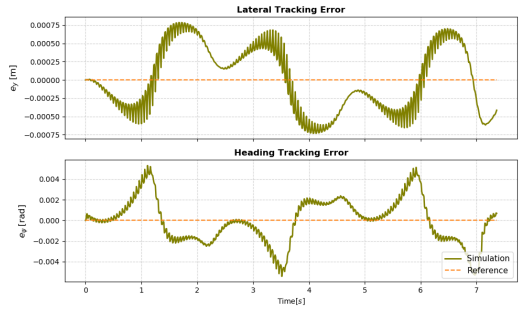
Next simulations show the performance of the kinematic bicycle model on a 1/10 scaled vehicle with updated based on frequency. The controller frequency was set to $f_{\text{ctrl}} = 100\,\text{Hz}$, while other parameters are the same when controller is updated based on distance.

Parameter	Symbol	Value
Controller step size	Δs_{ctrl}	$0.05m$
Controller frequency	f_{ctrl}	$100Hz$
Simulation step size	Δt_{sim}	$0.01s$
Number of prediction horizon steps	$N_{horizon}$	30

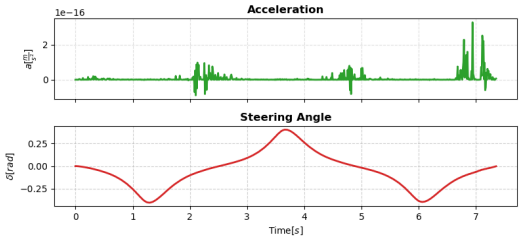
Table 3.9: Parameters of Hybrid Time Based MPC - Kinematic Bicycle Model

The results, shown in Figures 3.11–3.10, indicate that the controller maintains accurate trajectory tracking. The vehicle follows both the ellipsoidal and S-shaped trajectories with minimal deviation, and the overall system remains stable throughout the maneuver. The lateral and heading tracking errors remain small, although slight oscillations can be observed in regions of rapidly changing

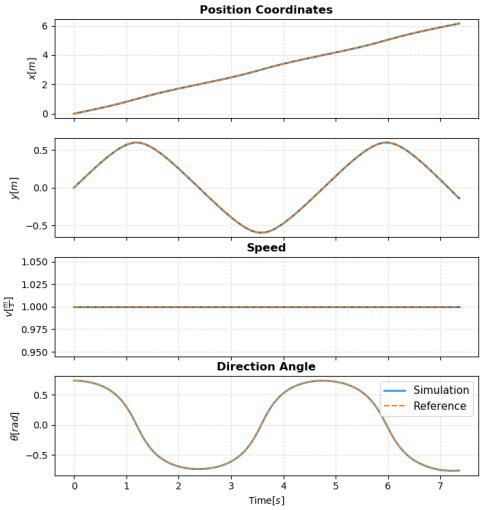
curvature. These fluctuations arise because the controller updates are performed at constant time intervals, which can cause small mismatches between the controller's prediction horizon and the vehicle's actual spatial progress, particularly when the velocity varies. The acceleration and steering profiles exhibit smooth behaviours that correlate with the curvature of the path, confirming that the controller effectively adapts to trajectory geometry despite the fixed update rate. However, small, high-frequency peaks are noticeable in the acceleration signal, especially near curvature transitions.



(a) Errors



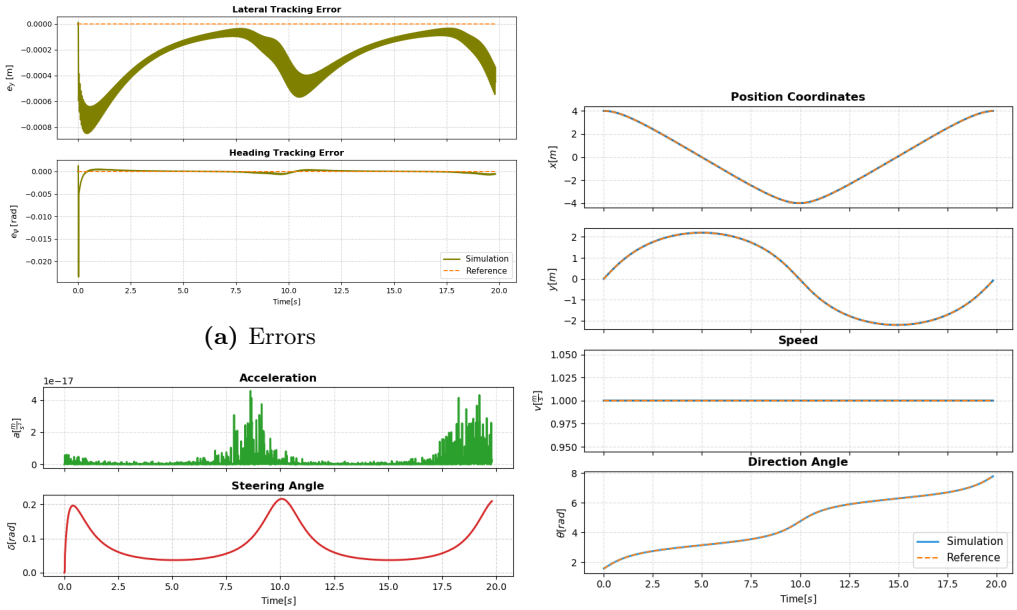
(b) Control inputs



(c) States

Figure 3.10: Kinematic Bicycle Model -Following of S reference path, in Hybrid Spatial-Time Domain Time Based

In contrast, the frequency-based approach updates the controller at fixed time intervals Δt, ensuring a constant computation rate that is easier to implement in real-time systems. The main trade-off is that the spatial distance between updates varies with the vehicle's speed, leading to minor deviations in curvature adaptation when velocity changes rapidly. Both strategies achieve accurate and stable tracking performance, with only marginal differences in error magnitude. The distance-based update method achieved a maximum lateral error of 0.8×10^{-3} m and a heading error below 4×10^{-3} rad, while the frequency-based update exhibited slightly higher but comparable values of 1×10^{-3} m and 5×10^{-3} rad, respectively. These results demonstrate that both approaches maintain high tracking precision, and the choice between them mainly depends on the



(a) Errors

(b) Control inputs

(c) States

Figure 3.11: Kinematic Bicycle Model -Following of S reference path, in Hybrid Spatial-Time Domain Time Based

desired balance between spatial accuracy and computational predictability. The distance-based update strategy ensures better spatial consistency and smoother tracking, since each update is tied to the travelled path rather than the elapsed time. On the other hand, the time-based strategy is easier to implement in real-time systems, as it guarantees a fixed computational rate and the setup better reflects the update rate typically limited by computational resources or hardware constraints. Since the final goal is to replicate real-time dynamics system, from now on, the second strategy will be considered. With this, the analysis of the kinematic bicycle model on ellipsoidal and S shaped reference paths has been completed.

3.1.2 Dynamic Bicycle Model

In this subsection, the performance and the behaviour of the dynamic bicycle model are discussed. The model was tested with different parameters and different size of vehicles, in order to fully understand it's behaviour and validate if the problem is well posed. The first evaluation was conducted on the 1/43 scaled vehicle, where the longitudinal forces were modeled as a function of the motor duty cycle. For the full-size vehicle, a more realistic representation was employed by incorporating load-dependent longitudinal tire forces, allowing the controller to

account for vertical load redistribution during acceleration and cornering. Finally, two different longitudinal force formulations were examined for the 1/10 scaled vehicle: one in which the driving force is directly proportional to the commanded longitudinal acceleration, and another in which the available longitudinal force depends on the instantaneous vertical load. The objective of these variations is to observe how each formulation influences the dynamic response, sideslip generation, and tracking accuracy, thus providing a comprehensive assessment of the dynamic model's robustness across different scales and force representations.

Spatial model with longitudinal forces modeled as a function of duty cycle

This modeling can be applied to small sized vehicles, where the relationship with the duty cycle and the throttle and breaking can be easily exploited. Control inputs are considered to be $u = (D, \delta)$ and derivatives of velocities are described as:

$$\begin{aligned} \dot{v}_x &= v_y\,\dot{\psi} + \frac{1}{m}((-C_{m1} \;+\; C_{m2}\,\dot{x})D - \;F_{c,f}\,sin(\delta) - F_x^d) \\ \dot{v}_y &= -v_x\,\dot{\psi} + \frac{1}{m}(F_{c,f}\,cos(\delta) + F_{y,r}) \end{aligned} \tag{3.1}$$

For this simulation, 1/43 scaled vehicle parameters are used, and the reference trajectories accordingly adapted. The values for ellipsoidal reference path were set to $a = 1.2m$, $b = 0.7m$, $\omega = 0.2$ Hz. For the S-shaped trajectory, curvature was generated via cubic spline interpolation, as it advances 0.3 m along the x-axis, the vehicle advances 0.07 m along the y-axis). The tests were carried out using a purely spatial formulation for both the controller and the simulation model. This setup serves as an introductory evaluation of the dynamic bicycle model, ensuring that no model-plant mismatch or sampling inconsistency is present.

Parameter	**Symbol**	**Value**
Controller step size	Δs_{ctrl}	$1 \times 10^{-3}\,m$
Simulation step size	Δs_{sim}	$1 \times 10^{-3}\,m$
Number of prediction horizon steps	$N_{horizon}$	50

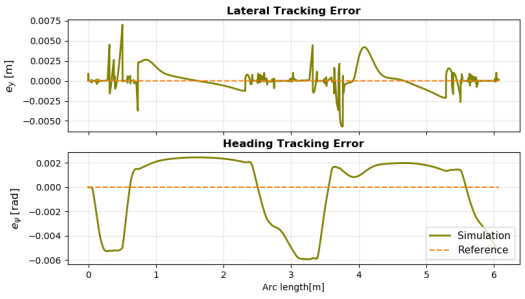
Table 3.10: Parameters of Spatial Domain MPC - Dynamic Bicycle Model, 1/43 Scaled Vehicle

The Tables 3.17 and 3.12 show the weights of cost function for spatial domain MPC when this model is employed. For the S-shaped trajectory, equal weights of $Q_{e_y} = Q_{e_{\psi+\beta}} = 30$ were chosen for both the stage and terminal costs. In this case, higher heading error penalties are unnecessary. However, for the ellipsoidal reference track, the heading-related stage and terminal weights were

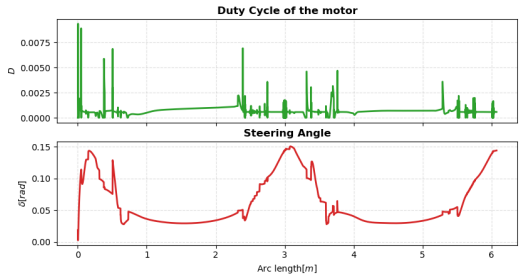
increased to $Q_{e_{\psi+\beta}} = 60$. The ellipsoidal trajectory contains longer segments of sustained curvature, during which heading alignment plays a more dominant role in maintaining accurate tracking. The stronger heading penalty helps reduce steady-state orientation drift and ensures that the vehicle remains properly aligned over curved sections. The lateral error weight was kept identical across both tracks, as e_y exhibits similar behaviour in both scenarios. Since the model is applied to a lightweight 1/43 platform with low speeds and small cornering forces, very large or aggressive weights were unnecessary. The control penalties ($R_a = R_\delta = 0.1$) remain the same for both trajectories, providing a balance between responsiveness and smoothness, thus preventing abrupt throttle or steering actions.

Stage Weights				**Terminal Weights**	
$Q_{e_{\psi+\beta}}$	Q_{e_y}	R_a	R_δ	$Q_{e_{\psi+\beta}}$	Q_{e_y}
60	30	0.1	0.1	60	30

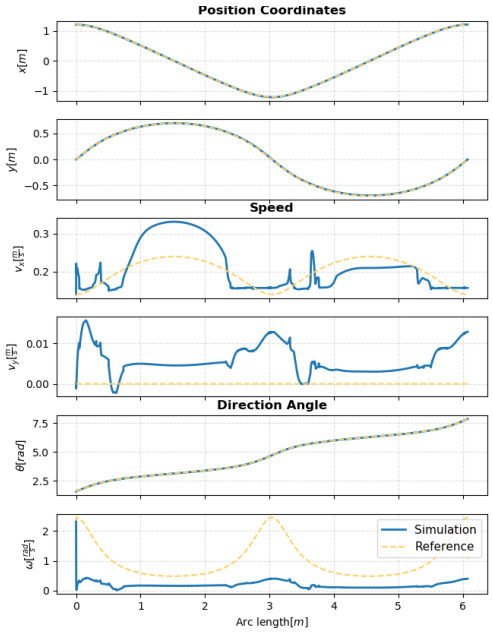
Table 3.11: Weights of Cost Function Spatial Domain MPC - Dynamic Bicycle Model for 1/43 Scaled Vehicle for Ellipsoidal reference track



(a) Errors

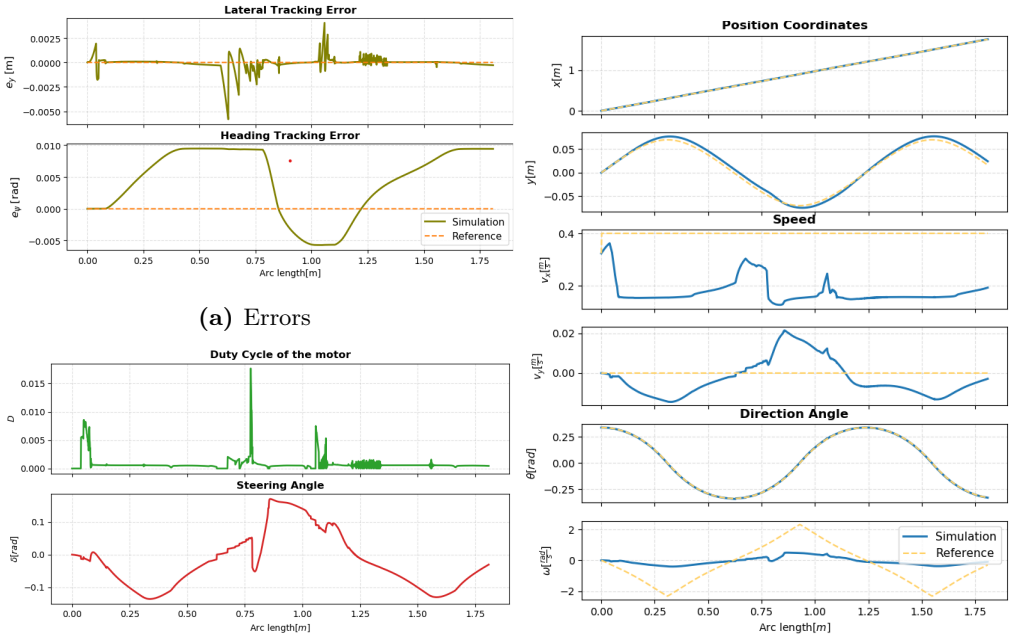


(b) Control inputs



(c) States

Figure 3.12: Dynamic Bicycle Model -Following of Ellipsoidal reference path, in Spatial-Domain for 1/43 scaled vehicle



(a) Errors

(b) Control inputs

(c) States

Figure 3.13: Dynamic Bicycle Model -Following of S shaped reference path, in Spatial Domain for 1/43 scaled vehicle

Stage Weights				**Terminal Weights**	
Q_{e_ψ}	Q_{e_y}	R_a	R_δ	Q_{e_ψ}	Q_{e_y}
30	30	0.1	0.1	30	30

Table 3.12: Weights of Cost Function Spatial Domain MPC - Dynamic Bicycle Model for 1/43 Scaled Vehicle for S shaped reference track

Figures 3.12 and 3.13 show accurate tracking on both reference path. The lateral tracking error e_y remains close to zero with small ripples and remains within $\pm$(2-6)$\times 10^{-3}$ m, with larger spikes during initial transients, and the heading error within $\pm$(1-2) $\times\ 10^{-2}$ rad. The visible errors on the x and y are the consequence of the small vehicle size. The longitudinal velocity v_x follows its reference with modest deviations and lateral velocity v_y remains below 2×10^{-2} m/s (consistent with stable handling at this scale), around 10% of the longitudinal velocity v_x. Note that, the velocity profile is not weighted in the cost function, and the velocity is let to adapt to the track and controller's needs. The starting longitudinal velocity is given, but it stays lower than in other simulations when 1/10 scaled vehicle is employed, which is logical since the vehicle is smaller size. This

influences ω behaviour, which is different then the expected one. Steering $\delta(s)$ mirrors the curvature, stays within $|\delta| < 0.15\,\text{rad}$ for both reference paths, less smooth and with noticeable stair steps caused by PWM signal. The duty cycle $D(s)$ stays near a low mean value with sporadic spikes $D < 3 \times 10^{-2}$, reflecting the low operating speed and PWM quantization.

This formulation's strengths lie in the fact that tracking accuracy is high despite the small scale and duty cycle actuation that captures RC powertrain characteristics. However, there are some difficulties that have to be overcome in order for this model to work more efficiently. With larger $|e_y|$ or sharp curvature, the denominator $1 - \kappa(s)\, e_y$ of the spatial derivative approaches zero, which can lead to small high-frequency ripples visible in $e_y(s)$ and $D(s)$. Additionally, The force map $F_x(D, v_x)$ requires good identification of $\{C_{m1}, C_{m2}, c_r\}$; any bias or sign mismatch directly shows up as duty spikes or steady-state error.

These results confirm precise tracking with physically plausible dynamics for a 1/43 vehicle, while highlighting the need for careful handling of the spatial denominator, PWM mapping, and low-speed operation. However, since our focus is to test the model with 1/10 scaled vehicle used for BFMC Competiton, advancing with this model requires determining the motor characteristics, as well as directly controlling the PWM signal of the motor. In simulated environment like Gazebo, it is not possible to access to this signal, since the vehicle movement is only simulated as it is described in the previous chapter. Thus, further tests with this model are not executed.

Hybrid spatial-time dynamic bicycle model with driving force proportional to the longitudinal acceleration

In this model we assume that the driving force is directly proportional to the commanded longitudinal acceleration and the 1/10 scaled vehicle is employed. The control inputs are $u = (a, \delta)$, and derivatives of velocities are described as:

$$\begin{aligned} \dot{v}_x &= v_y\,\dot{\psi} + \frac{1}{m}(ma - F_{c,f}\,sin(\delta) - F_x^d) \\ \dot{v}_y &= -v_x\,\dot{\psi} + \frac{1}{m}(F_{c,f}\,cos(\delta) + F_{y,r}) \end{aligned} \tag{3.2}$$

The simulation was carried out in a hybrid configuration, where the vehicle dynamics evolve in the time domain and the controller operates in the spatial domain with time-based updates.

Parameter	Symbol	Value
Controller step size	Δs_{ctrl}	$0.01m$
Controller frequency	f_{ctrl}	$50\ Hz$
Simulation step size	Δt_{sim}	$0.005s$
Number of prediction horizon steps	$N_{horizon}$	50

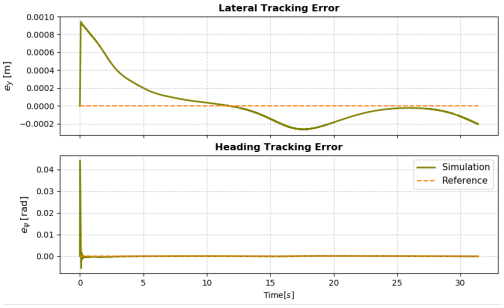
Table 3.13: Parameters used for Dynamic Bicycle model for 1/10 scaled vehicle in Hybrid Spatial-Time Domain

The controller step size was set to $\Delta s_{\text{ctrl}} = 0.01\,\text{m}$ to ensure fine spatial resolution, while the simulation time step $\Delta t_{\text{sim}} = 0.005\,\text{s}$ offers stable numerical integration of the dynamic states. The controller frequency of 50 Hz provides sufficiently frequent updates for the highly responsive 1/10 platform. A prediction horizon of $N_{\text{horizon}} = 50$ steps was selected to allow the controller to anticipate curvature changes.

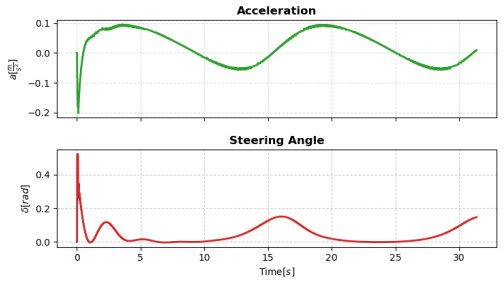
Stage Weights					Terminal Weights	
$Q_{e_{\psi+\beta}}$	Q_{e_y}	Q_{v_x}	R_a	R_δ	$Q_{e_{\psi+\beta}}$	Q_{e_y}
50	100	10	0.1	0.1	50	50

Table 3.14: Weights of Cost Function Time Domain MPC Controller - Dynamic Bicycle Model, 1/10 Scaled Vehicle

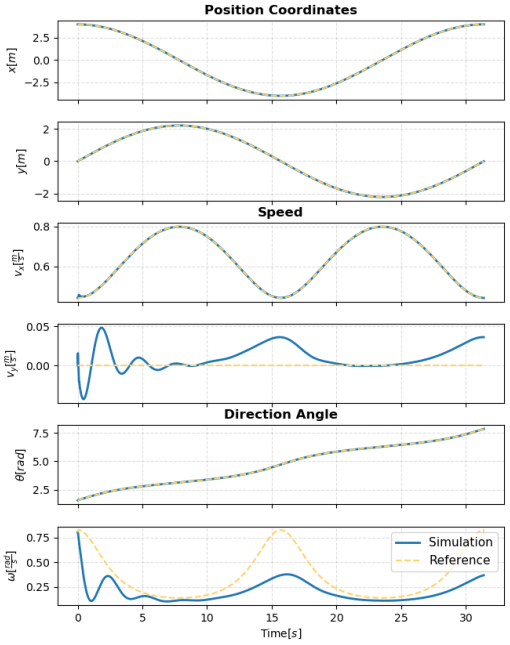
The cost-function weights were chosen as reported in the Table 3.14 to reflect the stronger dynamic effects characteristic of the 1/10 scaled vehicle, particularly the higher sensitivity to sideslip, yaw-rate changes, and lateral disturbances. A substantial penalty was placed on the combined heading-sideslip term ($Q_{e_{\psi+\beta}} = 50$) to limit excessive orientation error and try to prevent large slip angles during turning. The lateral deviation was penalized even more strongly ($Q_{e_y} = 100$), ensuring that the vehicle remains close to the reference centerline. The longitudinal velocity was included as an additional cost term, and assigned a moderate weight ($Q_{v_x} = 10$) to stabilize the speed evolution. The control-input penalties ($R_a = R_\delta = 0.1$) ensure smooth actuator behaviour while providing rapid dynamic changes. The terminal weights magnitudes ($Q_{e_{\psi+\beta}} = Q_{e_y} = 50$), enforce final alignment with the reference path.



(a) Errors



(b) Control inputs



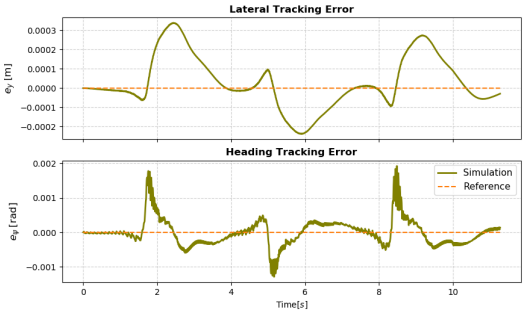
(c) States

Figure 3.14: Dynamic Bicycle Model -Following of Ellipsoidal reference path, in Hybrid Spatial-Time Domain

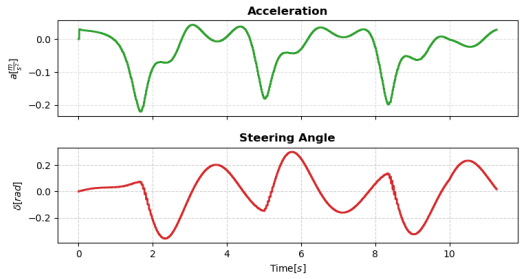
The results, presented in figures. 3.14 and 3.15, demonstrate that the controller achieves high tracking accuracy when governing the dynamic model for both ellipsoidal and S shaped reference path. On ellipsoidal track, the lateral error exhibits a short initial transient, but remains below 1.0×10^{-3} m, before quickly converging towards zero. The heading error remains practically negligible throughout the entire maneuver, with magnitude below 4.5×10^{-2} rad . This indicates that the controller effectively handles the additional coupling between longitudinal and lateral dynamics introduced by the dynamic bicycle model.

Note that the longitudinal velocity profile is different then in case of the spatial kinematic bicycle model, resembling the one of the time kinematic case. This is more aligned with the natural behaviour of the vehicle to slow down when it curves, and is adapted to the size of the vehicle. The longitudinal velocity v_x follows the reference profile smoothly, while the lateral velocity v_y remains small, within 5×10^{-2} m/s , proving that the vehicle maintains good lateral stability. Although the dynamic model yields a non-zero lateral velocity, the resulting sideslip remains within $\approx 3°$, which is consistent with stable, nominal handling. The yaw rate ω also tracks its reference trend with deviation, which highlights the controller's capability to manage dynamic responses. The acceleration and

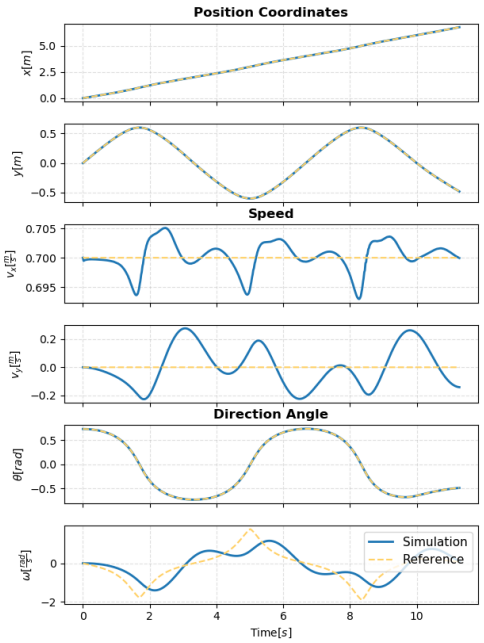
steering signals remain smooth, with mild variations corresponding to curvature transitions, confirming stable control actions without excessive oscillations.



(a) Errors



(b) Control inputs



(c) States

Figure 3.15: Dynamic Bicycle Model -Following of S shaped reference path, in Hybrid Spatial-Time Domain

The results on the S shaped reference path provide additional insight on the behaviour of dynamic bicycle model. The lateral error e_y stays within $\pm 0.35\,\text{mm}$ and the heading error e_ψ within $\pm 2 \times 10^{-3}\,\text{rad}$ with short, well-damped spikes. The steering input oscillates smoothly in phase with the curvature of the S-curve, and the commanded longitudinal acceleration exhibits brief negative dips. Note that the nominal velocity is lower then the one in case of kinematic bicycle model. Due to the introduction of dynamics, the controller is not able to follow such high velocity profile, thus the lower value is introduced to test the behaviour on this track. The longitudinal velocity v_x follows its reference closely with small fluctuations of about $\pm 0.7\%$, while the lateral velocity v_y shows alternating lobes driven by the curvature reversals.

The high value of the lateral velocity compared to the longitudinal one results in the high slip angle $\beta \approx 15°$ when following S shaped curvature and dealing with abrupt changes of the curvature. This is considered to be undesirable, since the β directly contributes to the heading error term in the cost function, and thus big part of angle produced is from slipping and not steering. No such effects are

noticeable on ellipsoidal path, since no high lateral velocities are noticeable. The yaw rate ω reproduces the reference trend with a mild phase lag. The coexistence of nonzero v_y and ω explains the intervals in which the total speed increases even when the longitudinal command a is negative, as the positive coupling term $v_y\omega$ contributes to $\dot{v}_x$.

Relative to the kinematic model, the dynamic formulation shows slightly larger and more oscillatory transients in e_y and e_ψ during curvature changes, arising from the additional tire and yaw dynamics absent in the kinematic representation. The yaw rate and steering response exhibit a small phase delay, reflecting the system's inertial behaviour, whereas the kinematic model reacts instantaneously to curvature variations. Nevertheless, both achieve similar steady-state accuracy once transients decay.

Spatial model with load-dependent longitudinal tire forces

We introduce the modeling of the longitudinal tire forces as a linear function of the vertical load. This modeling is primarily used for real size vehicles, particularly the model for Jaguar X . The control inputs are $u = (a, \delta)$, and derivatives of velocities are described as:

$$\begin{aligned} \dot{v}_x &= v_y\,\dot{\psi} + \frac{1}{m}(\gamma\,\mu\,F_{z,f}\,cos(\delta)\;-\;F_{c,f}\,sin(\delta) + \gamma\,\mu\,F_{z,r} - F_x^d) \\ \dot{v}_y &= -v_x\,\dot{\psi} + \frac{1}{m}(\gamma\,\mu\,F_{z,f}\,sin(\delta)\;+\;F_{c,f}\,cos(\delta) + F_{c,r}) \end{aligned} \tag{3.3}$$

where $\gamma = a/a_{max}$. The acceleration is normalized by dividing with $a_{max} = 3\,\mathrm{m/s^2}$, appropriately for model this size. The tests were carried out using a purely spatial formulation for both the controller and the simulation model. The model was implemented entirely in the spatial domain, as no hybrid spatial–time configuration is introduced for this case.

Parameter	**Symbol**	**Value**
Controller step size	Δs_{ctrl}	$0.2\,m$
Simulation step size	Δs_{sim}	$0.2\,m$
Number of prediction horizon steps	$N_{horizon}$	50

Table 3.15: Parameters of Spatial Domain MPC - Dynamic Bicycle Model, 1/43 Scaled Vehicle

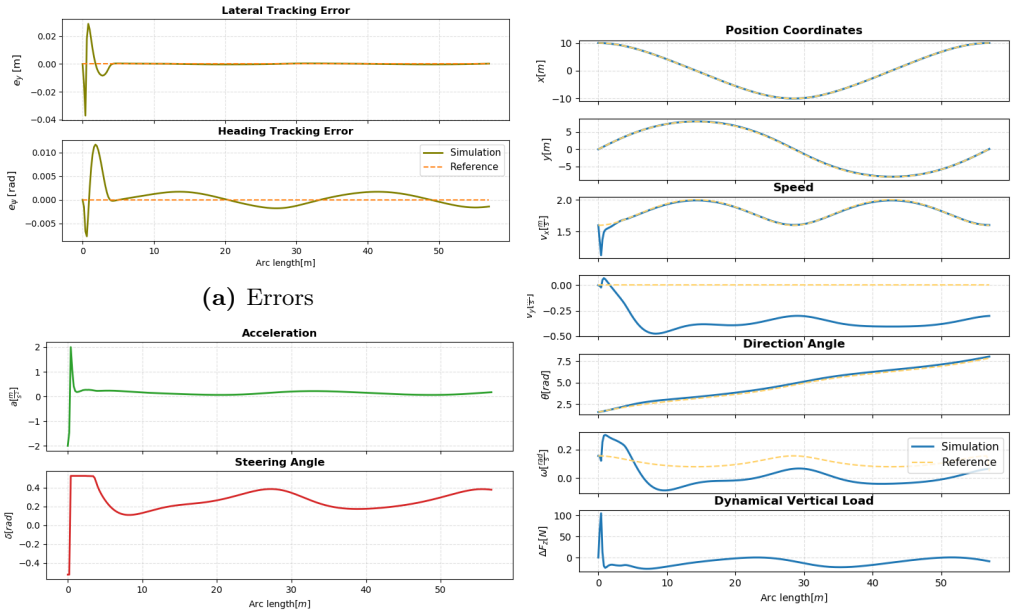
The full size vehicle parameters are used, and the reference trajectories are accordingly adapted. The values for ellipsoidal reference path were set to $a = 10$ m, $b = 8$ m, $\omega = 0.2$ Hz. For the S-shaped trajectory, curvature was generated via

cubic spline interpolation, with the ratio 10 : 3. Both the controller and simulation step sizes were set to $\Delta s_{\text{ctrl}} = \Delta s_{\text{sim}} = 0.2\,\text{m}$, which provides an fine spatial resolution relative to the size of the vehicle. A smaller step size would increase the computational burden without providing significant improvement in tracking accuracy. A prediction horizon was extended to $N_{\text{horizon}} = 50$ to provide a sufficiently long look-ahead distance, enabling the controller to anticipate upcoming curvature segments on both the ellipsoidal and S-shaped trajectories.

Stage Weights					**Terminal Weights**	
$Q_{e_{\psi+\beta}}$	Q_{e_y}	Q_{v_x}	R_a	R_δ	$Q_{e_{\psi+\beta}}$	Q_{e_y}
100	50	10	0.1	0.1	100	50

Table 3.16: Weights of of Cost Function Spatial Domain MPC - Dynamic Bicycle Model, Full Size Vehicle

In the Table 3.16 report the weights of cost function of controller used for this case. For the full-sized vehicle, the spatial-domain dynamic MPC requires a stronger weighting, due to the presence of load-dependent longitudinal forces and size. In this setting, the controller must account not only for geometric path-following accuracy but also for the interaction between longitudinal dynamics, lateral tire forces, and the redistribution of normal loads. A high penalty was placed on the combined heading-sideslip term ($Q_{e_{\psi+\beta}} = 100$) to ensure stable orientation and to prevent large slip angles during curved segments, particularly in the S-shaped trajectory where curvature variations can induce significant sideslip. The lateral error was penalized with $Q_{e_y} = 50$, striking a balance between accuracy and excessive stiffness. The longitudinal velocity term ($Q_{v_x} = 10$) was included to stabilize the speed evolution, as the load-dependent force model introduces nonlinearity that can cause fluctuations in v_x if left unpenalized. The control-input penalties ($R_a = R_\delta = 0.1$) were selected, as usual. The terminal weights mirror the structure of the stage costs, enforcing accurate final alignment in both heading and lateral position. The weights reflect the increased dynamic complexity and the physical scaling of the full size vehicle model.



(a) Errors

(b) Control inputs

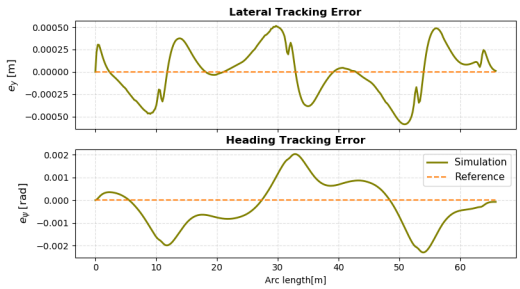
(c) States

Figure 3.16: Dynamic Bicycle Model -Following of Ellipsoidal reference path, in Spatial Domain for real size vehicle

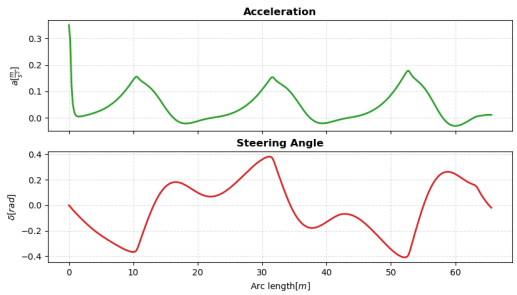
Figures 3.16 and 3.17 show accurate trajectory tracking with small steady errors on both the ellipsoidal and the S-shaped tracks. The steering profile $\delta(s)$ tries to mirror the curvature distribution, not as successfully as in previous cases, while the commanded longitudinal acceleration $a(s)$ exhibits brief peaks at curvature transitions, but generally stays around small values. When following the ellipsoidal reference path, lateral and heading errors settle rapidly after the initial transient and remain within insignificantly small ranges compared to the size of the vehicle, while on the S shaped track, errors have smaller value and without initial peak. The lateral velocity $v_y(s)$ is nonzero but bounded, and in case of the S curvature, the value reaches the $5 \times 10^{-1}\,\mathrm{m/s}$, while the longitudinal velocity is around $2\,\mathrm{m/s}$. This can considered to be undesirable behaviour, but due to the size of the vehicle can be tolerated. Since the tests were done on a full size vehicle model, the model can reach higher velocity without any difficulty. Even though, if we imagine the car going this speed in real life scenarios, it would be considered quite slow. This however results in high precision.

The results obtained show a clear distinction between the ellipsoidal and the S-shaped trajectories in terms of vertical load transfer. On the ellipsoidal track, the vehicle experiences a long and continuous acceleration phase at the beginning of the lap. As shown in the results, the longitudinal speed increases from approximately $1.5\,\mathrm{m/s}$ to almost $2.0\,\mathrm{m/s}$ within the first $10\,\mathrm{m}$ of arc length. This

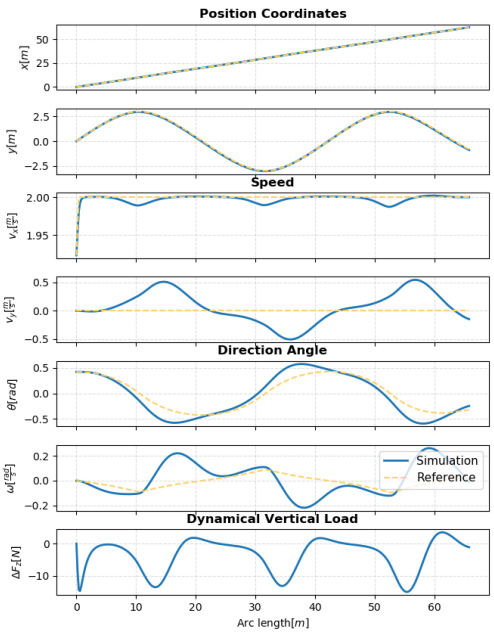
sustained acceleration generates a large forward load transfer, resulting in a peak vertical load variation of about $\Delta F_z \approx 120$N. After the initial acceleration phase, the vehicle settles at a nearly constant speed and the longitudinal acceleration drops to near zero. Because the ellipsoidal track does not introduce noticeable lateral acceleration peaks, the total load transfer gradually diminishes, and ΔF_z stabilizes at low values for the rest of the trajectory. In contrast, the S-shaped track contains frequent curvature reversals, forcing the vehicle to alternate between mild acceleration and mild braking while continuously adjusting its lateral direction. As a consequence, the accumulated longitudinal force remains small and highly oscillatory. This leads to much lower vertical load transfer, with ΔF_z typically remaining within ± 10–20 N and changing sign several times along the path. The large radius and smooth curvature of the ellipsoidal track reduce the lateral acceleration peaks compared to the S-curve, meaning that the dominant contributor to load transfer is longitudinal acceleration rather than lateral cornering. On the S-track, however, the continuous left right steering cause moderate lateral accelerations but prevents the buildup of sustained longitudinal load transfer. This explains the substantially lower amplitude of ΔF_z observed in the S-shaped curvature case.



(a) Errors



(b) Control inputs



(c) States

Figure 3.17: Dynamic Bicycle Model -Following of S shaped reference path, in Spatial Domain for real size vehicle

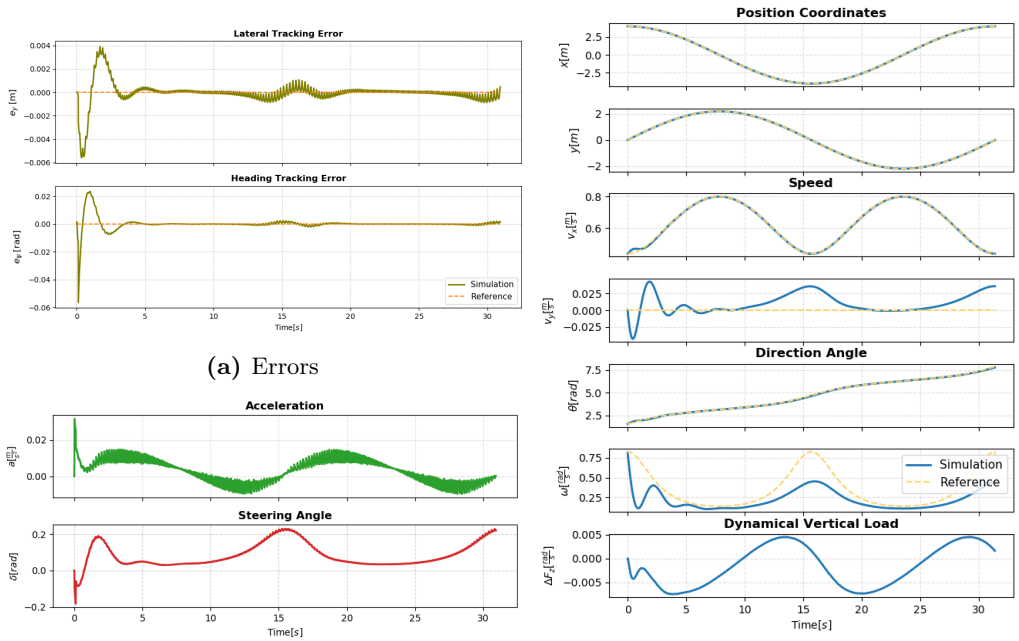
Hybrid spatial-time dynamic bicycle model with load-dependent longitudinal tire forces

Next tests consist of implementation of the dynamic bicycle where longitudinal tire forces are dependent on the vertical load on the 1/10 scaled vehicle. Since we successfully implemented the model on the full sized vehicle, the idea is to do it on the smaller scaled vehicle, and introduce hybrid spatial-time domain formulation. Parameter used are the same as in case of the hybrid spatial-time dynamic bicycle model with driving force proportional to the longitudinal acceleration **??**.

Stage Weights					**Terminal Weights**	
$Q_{e_{\psi+\beta}}$	Q_{e_y}	Q_{v_x}	R_a	R_δ	Q_{e_ψ}	Q_{e_y}
20	10	1	0.1	0.1	20	10

Table 3.17: Weights of Cost Function Hybrid MPC Controller- Dynamic Bicycle Model, 1/10 Scaled Vehicle

For the 1/10 scaled vehicle with load-dependent longitudinal forces, the hybrid MPC controller requires a slightly different weighting strategy compared to the other dynamic formulations and smaller penalties to the full sized vehicle. In this model, the available longitudinal force depends on the normal loads, which vary dynamically as a function of acceleration, cornering forces, and the vehicle's geometry. This coupling introduces stronger nonlinearities and naturally limits the magnitude of both longitudinal and lateral accelerations, reducing the need for larger penalties to regulate slip and heading behaviour. The heading–sideslip term was penalized with a weight of $Q_{e_{\psi+\beta}} = 20$, while the lateral deviation was assigned a slightly smaller weight $Q_{e_y} = 10$, as the load dependent longitudinal model inherently produces smoother lateral behaviour and less abrupt curvature induced disturbance than the simpler acceleration-based model. The longitudinal velocity was given a smaller weight $Q_{v_x} = 1$ to stabilize the speed evolution without interfering with the natural behaviour of the load transfer mechanism. The control penalties were kept at $R_a = R_\delta = 0.1$. The terminal weights follow the same structure as the stage costs, ensuring proper convergence of heading and lateral alignment by the end of the prediction horizon.



(a) Errors

(b) Control inputs

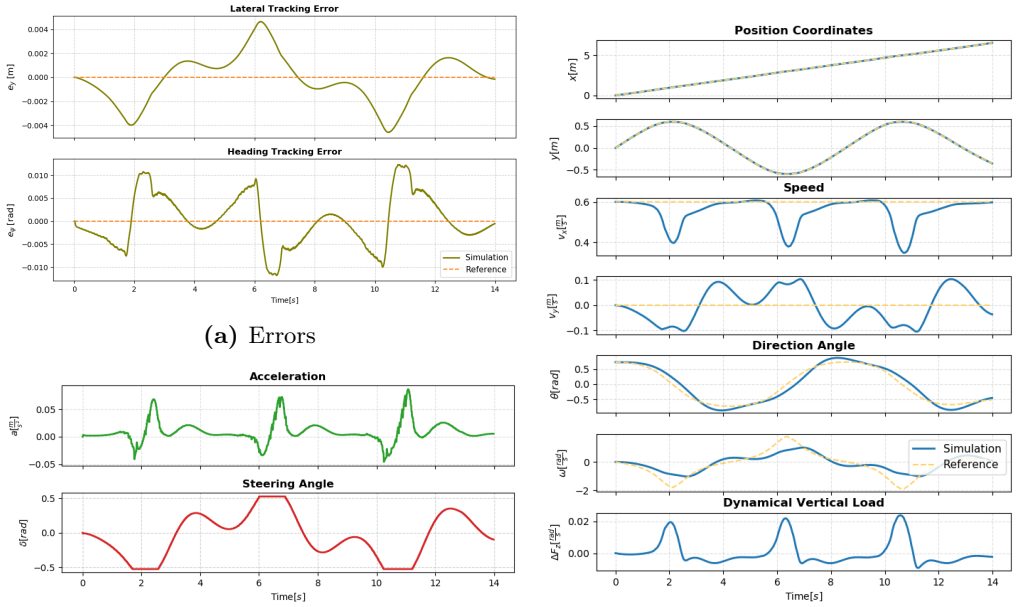
(c) States

Figure 3.18: Dynamic Bicycle Model -Following of Ellipsoidal reference path, in Hybrid Spatial-Time Domain

The results in Figure 3.18 show the results with the inclusion of load- dependent forces in hybrid domain on $1/10$ sized vehicle on ellipsoidal trajectory. The simulated trajectory overlaps closely with the reference, and the lateral and heading tracking errors remain small and well bounded. The lateral error e_y stays within $\pm 6 \times 10^{-4}$ m with minor oscillation in the beginning, and the heading error e_ψ remains below 2×10^{-2} rad. The longitudinal velocity v_x follows its reference profile closely, but with slightly damped peaks compared to the case without load variation. The lateral velocity v_y remains small throughout the manoeuvre, confirming that the controller maintains good stability and compensates effectively for the induced load transfer. The longitudinal velocity deviation stays within $\pm 1.5\%$ of the nominal value, while the peak acceleration amplitude reaches $0.025\,\mathrm{m/s^2}$.

The yaw rate ω has expected small delay compared to the expected behaviour, and the dynamic vertical load ΔF_z oscillates in phase with curvature, capturing the longitudinal load transfer pattern: positive during acceleration and negative during braking or steering into curvature. The dynamic vertical load variation ΔF_z remains within $\pm 5 \times 10^{-3}\,\mathrm{N}$, consistent with light load transfer for small-scale vehicle dynamics. The steering angle δ and commanded acceleration a remain smooth and correlated with curvature. The oscillatory structure in the acceleration curve aligns with the curvature and the load transfer profile, showing

that the extended model captures this coupling realistically.



(a) Errors

(b) Control inputs

(c) States

Figure 3.19: Dynamic Bicycle Model -Following of S shaped reference path, in Hybrid Spatial-Time Domain

The figure. 3.19 shows performance of this model on the S shaped trajectory that further confirm that the controller can effectively handle the additional coupling between load transfer and traction capacity while maintaining high tracking precision and smooth control behaviour. The lateral and heading tracking errors remain very small and well bounded. The lateral deviation e_y stays within ± 4–$5\,\text{mm}$, and the heading error e_ψ within $\pm 1.2 \times 10^{-2}\,\text{rad}$.

The longitudinal velocity v_x remains close to its nominal value of $0.6\,\text{m/s}$, with brief drops to 0.36–$0.40\,\text{m/s}$ at the tightest curvature points. The lateral velocity v_y exhibits alternating lobes of approximately 0.08–$0.10\,\text{m/s}$, corresponding to physically realistic sideslip levels ($|\beta| \approx \arctan(v_y/v_x) \approx 9°$ at the peaks). The dynamic vertical load ΔF_z oscillates in phase with the curvature, showing positive peaks under acceleration and negative ones during braking or steering reversals.

The acceleration magnitude stays under $0.06\,\text{m/s}^2$, but exhibits short positive bursts near each curvature transition. These pulses correspond to instances where the controller compensates for reduced longitudinal traction when lateral load transfer peaks. The steering action remains smooth and symmetric, but it does get saturated when the curves are strong, which was not the case previously.

Two minor drawbacks can be observed. Firstly, transient sideslip peaks up to

$|\beta| \approx 9^\circ$ occur at curvature reversals, increasing yaw coupling and lateral load transfer. Secondly, small high-frequency oscillations appear in a around the same transitions.

Compared with the previous dynamic hybrid model without load dependency, this formulation introduces slightly higher control activity in the longitudinal forces but results in a more physically accurate representation of traction limitations. Despite this added complexity, the overall path-tracking accuracy and stability remain at the same high level as before. The controller can be further improved by introducing sideslip penalties, friction-aware velocity profiling, and an adaptive update strategy.

However, the introduction of full vehicle dynamics also reveals a small but important side effect: an increase in the lateral velocity component v_y, particularly during curvature sign reversals that take part in S shaped reference path. These higher v_y peaks reflect the natural sideslip of a dynamic vehicle, but can become undesirable when precision or stability is prioritised over physical fidelity.

A nonzero v_y is physically expected during cornering, as the velocity vector is not perfectly aligned with the vehicle longitudinal axis. The corresponding sideslip angle $\beta = \arctan(v_y/v_x)$ should remain small, but as we saw that is not always the case especially when dealing with S shaped curvature. Excessive sideslip compromises the controller's ability to distinguish between geometric tracking deviations and physical tire slip, thereby reducing the interpretability and effectiveness of the heading-error minimisation. Ideally, β should remain within a small range ($|\beta| < 5^\circ$) to ensure that most of the yaw dynamics originate from controlled steering rather than uncontrolled lateral motion. In the hybrid models, transient peaks in v_y arise mainly from:

1. the yaw dynamics coupling term $v_y\omega$ in $\dot{v}_x$,
2. load transfer and varying lateral stiffness across the tires, and
3. spatial updates executed at constant time intervals that do not always coincide with curvature changes, causing temporary prediction mismatch.

To mitigate or suppress high v_y, several strategies can be applied, among which the most common ones are the *sideslip penalisation* and *angular velocity damping*. The sideslip penalisation introduces a cost term on slipping angle β in the MPC objective to limit the lateral motion. Alternatively, we can enforce a soft constraint $|\beta| \leq \beta_{\max}$ (e.g. 5°) to prevent excessive sideslip. On the other hand, angular velocity damping required increasing the weights on ω and its rate-of-change to suppress oscillations and improve directional stability.

In summary, moderate nonzero v_y is a natural characteristic of dynamic models, representing physically realistic sideslip and load coupling. Excessive v_y may how-

ever be reduced through proper control weighting, curvature-based speed adaptation.

3.2 Trajectories based on the Bosch Future Mobility Challenge

The final evaluation of the model has been tested in Gazebo environment on the BFMC track with the kinematic bicycle model. The kinematic representation offers significantly lower computational complexity and numerically stable behaviour over the dynamic bicycle model, ensuring that the real-time constraints can be satisfied on embedded hardware. Moreover, the operational regime of the 1/10 scale vehicle is characterized by moderate speeds and limited lateral slip. This aligns well with the assumptions of the kinematic model, making it a suitable compromise between computational efficiency and predictive fidelity. In the simulation in this section RTI methods were used, along with Partial Condensing HPIPM solver.

Since the trajectory is defined in space, the spatial domain reformulation is used for the MPC controller. The parameters used are represented in the Tables 3.18 and 3.19. The horizon of $N_{horizon} = 100$ points extended compared to previous simulations, and is chosen to to provide a sufficiently long look-ahead distance. This allows the controller to anticipate upcoming curvature segments, which in the BFMC map contain much sharper and more irregular turns than the elliptical or S-shaped paths. A longer prediction window enables the MPC to incorporate the geometric information of these segments early enough to generate smooth control actions. An important aspect of this test is the inherent model mismatch. As already commented, Gazebo simulates vehicle using dynamic model, while the controller relies on a simplified kinematic bicycle model. By increasing the horizon length, the MPC can compensate for part of this mismatch by planning over a longer horizon. The longer horizon can stabilize the behaviour in tight turns and reduces sensitivity to instantaneous modeling inaccuracies.

The control loop is executed in a hardware-in-the-loop (HIL) configuration on a Raspberry Pi 5, achieving an average control loop time of $4.1ms$ without overruns. This performance represents a substantial improvement compared to running the controller directly within the simulation environment, where additional overhead from Gazebo and ROS2 components limits timing consistency.

Parameter	**Symbol**	**Value**
Controller step size	Δs_{ctrl}	$0.01\,m$
Controller frequency	f_{ctrl}	$50Hz$
Number of prediction horizon steps	$N_{horizon}$	100

Table 3.18: Parameters of Spatial Based MPC used in Gazebo on BFMC track

The weighting matrices were tuned specifically for the Gazebo–ROS2 environment, where unmodeled dynamics, actuator delay, and sensor noise require a stronger emphasis on the tracking compared to offline simulations. The large weight on the heading error $Q_{e_{\psi+\beta}} = 1000$, forces the optimizer to closely follow the reference orientation, since in the BFMC environment heading deviations can later propagate into larger lateral offsets due to narrow lanes and tight curvature transitions. The lateral error weight $Q_{e_y} = 200$ provides an additional constraint on path position, ensuring centimeter-level tracking when the vehicle is subject to the more complex friction and contact interactions of the Gazebo physics engine. The small velocity-tracking weight $Q_{v_x} = 1$ reflects the fact that maintaining the exact commanded speed is less crucial than preserving the geometric alignment. This also prevents the controller from generating aggressive acceleration or braking commands solely to enforce a specific velocity. The input regularization terms $R_a = R_\delta = 0.05$ were chosen to be smaller than in previous cases, allowing adaptiveness and responsiveness to abrupt changes. Still, the values are not negligible, since we still require promote smoother control actions to maintain compatibility with the steering-rate limitations . The terminal weights mirror the stage weights for heading and lateral error, reinforcing convergence toward the final reference pose.

Stage Weights					**Terminal Weights**	
$Q_{e_{\psi+\beta}}$	Q_{e_y}	Q_{v_x}	R_a	R_δ	$Q_{e_{\psi+\beta}}$	Q_{e_y}
1000	200	1	0.05	0.05	1000	200

Table 3.19: Weights of Cost Function Spatial Domain MPC - BFMC track

Type	**Constraint Range**
Heading error	$-45° \leq e_\psi \leq 45°$
Lateral error	$-0.15m \leq e_y \leq 0.15m$
Longitudinal acceleration	$-2\frac{m}{s^2} \leq a \leq 2\frac{m}{s^2}$
Steering angle	$-30° \leq \delta \leq 30°$

Table 3.20: Constraints imposed on the MPC Controller used in Gazebo Simulator

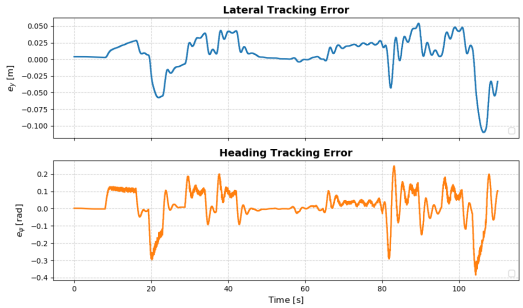
The constraints imposed on the heading and lateral tracking error are reported in the Table 3.20. The bound on the lateral tracking error is tightened to match half of the track width, ensuring that the vehicle remains safely within lane limits during all manoeuvres. On the contrary, the constraint on the heading error is relaxed to allow additional freedom during turning, where larger yaw deviations are required to initiate and complete curved segments of the trajectory.

The constraints on the steering input were kept identical to those introduced in previous chapter and used in previous section, while the longitudinal acceleration is twice as big. The steering angle bound represents the actual limitation of the physical system and model in simulation. The acceleration constraint, on the other hand, is primarily imposed for safety and comfort reasons. The actual physical limit of the motor is not precisely known, as it depends on factors such as battery voltage, internal resistance, and the effective torque–speed characteristics under load, none of which are directly measured in the current setup. Consequently, a conservative bound on the acceleration is used to guarantee stable behaviour and avoid actuator saturation during aggressive manoeuvres.

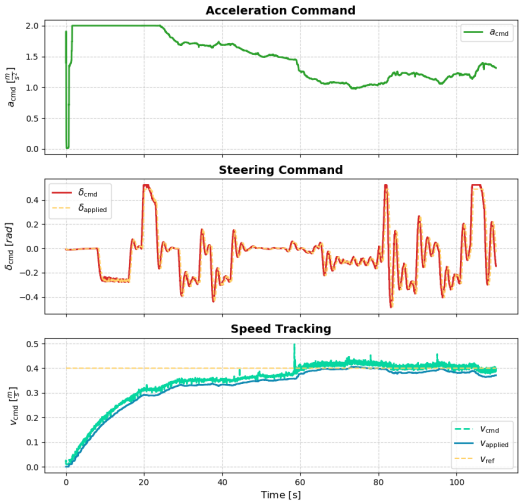
As previously discussed, the actuator command that is given to the actuators is the vehicle's longitudinal velocity. Since the acceleration a_{cmd} is treated as the control input, the commanded velocity is obtained through the discrete Forward Euler integration $v_{cmd}(t+1) = v(t) + a_{cmd}(t)\, t_{ctrl}$, where $t_{crtl} = \frac{1}{f_{ctrl}}$. Given the v_{cmd} the motor's PID controller computes the applied velocity v_{app}, that represents the effective longitudinal speed delivered by the actuator. An additional aspect of velocity generation in spatial domain reformulation should be considered . It is known that models expressed in the spatial domain cannot initialize the motion at zero velocity, since the spatial derivative $\frac{d}{ds} = \frac{1}{v}\frac{d}{dt}$ becomes singular at $v = 0$. As the purpose of the Gazebo simulation is to describe as accurately as possible the real life scenario, the starting velocity is considered to be negligibly small for the movement of the vehicle, but enough to keep the controller from singularity. Thus, to ensure convergence toward the desired operating speed, the longitudinal velocity is included in the optimal control problem's cost function.

During the runs that took place at the BFMC Competition 2025, Team DEI

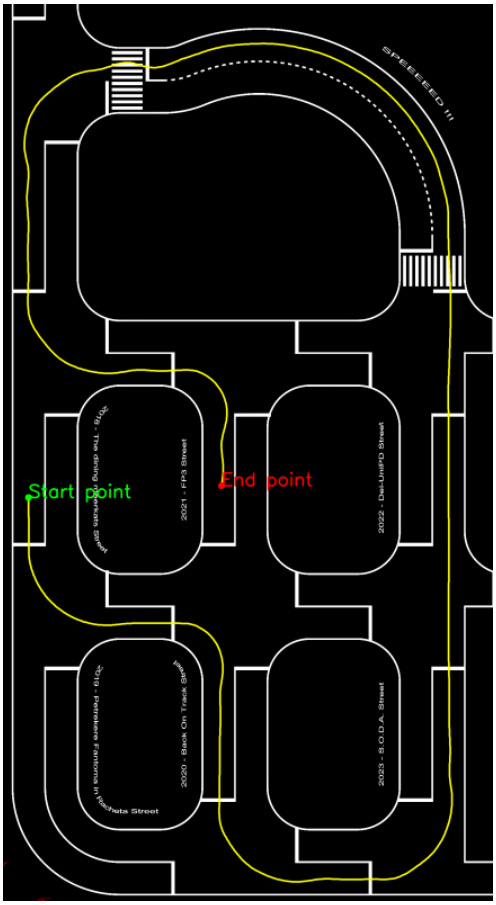
UNIPD's vehicle operated at a nominal speed of $0.35\,m/s$. Here we increase the nominal velocity to $0.4\,m/s$, in order to assess wether MPC allows us to improve performance by better exploiting the capabilities of the scaled vehicle. Note that, during the competition, Team DEI UNIPD's vehicle was among the fastest competitors in terms of the speed profile.



(a) Errors



(b) Control inputs



(c) Reference

Figure 3.20: Performance in Gazebo with Spatial Domain MPC Controller in Urban-like Area

Figure 3.20 shows the performance of the controller testes in urban-like area of the track. The (x, y) coordinates follow the reference closely on both straight and curved segments, and the only visible deviations occur around the sharp left–right transition and the S-shaped transition. In both cases the deviation remains small enough, and the vehicle stays inside the intended lane corridor. The heading error remains smooth and bounded. For most of the run, the lateral error stays within ± 2–3 cm, increasing to approximately 7–10 cm at the two previously mentioned

high-curvature transitions. These peaks coincide with near-saturated steering commands and with the highest heading-error of roughly 0.25–0.35 rad, or 20°.

The control inputs reflect the irregularity of the path. Steering commands reach ±0.45–0.5 rad several times, particularly during tight curvature transitions. Commanded and applied steering remain nearly identical, with expected slight delay, confirming fast actuation relative to the control rate. Acceleration saturates during the first second, after which it starts gradually decreasing in magnitude as speed approaches the reference. The applied velocity lies consistently 0.03–0.05 m/s below the command one, that can be a consequence of the low-level speed controller or possibly of other aspects that weren't modeled rather than by the MPC formulation.

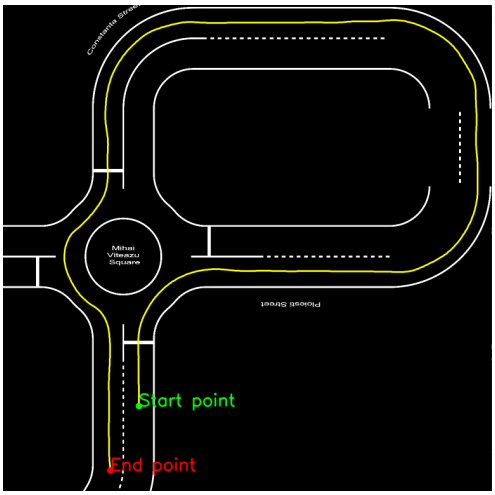
Overall, MPC handles irregular, non-repetitive trajectories reliably. Tracking remains accurate on regular geometry, whereas abrupt curvature changes cause short deviations linked to actuator saturation and model mismatch around tight turns.

The spatial-domain MPC was further evaluated on a longer closed-loop trajectory that includes a roundabout entry-exit maneuver followed by a high-curvature oval segment shown on the Figure 3.21. The position states (x, y) remain aligned with the reference throughout the run, with deviations visible only in the most curved transitions of the oval. On the other side, there is a small, but noticeable delay between the reference heading and the actual one, but the heading error remains below 0.15 rad for most of the trajectory and increases to 0.30–0.35 rad that arise from a combination of high curvature, rapid curvature gradients, and actuator saturation. Lateral tracking error remains within ±3–4 cm over most of the oval, with larger peaks of approximately 6–8 cm appearing while transitioning out of the roundabout and during the inner-to-outer lane curvature change on the top right of the track. These peaks coincide with strong steering corrections and increased heading error, as a consequence of high-demand segments. However, the car is able to stay within the track and give satisfactory performance.

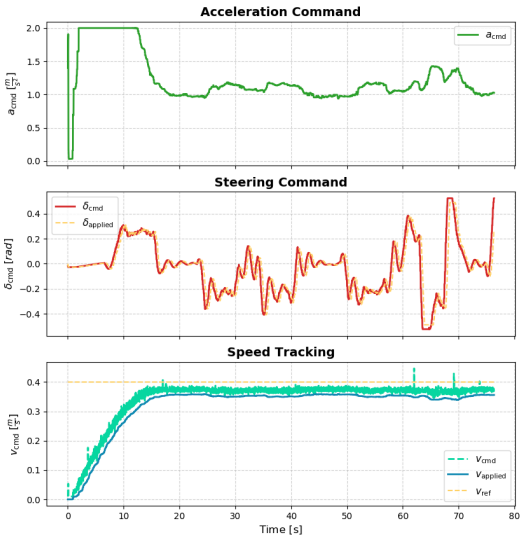
Steering commands reach approximately ±0.45–0.5 rad in several places, indicating that the controller consistently operates near the actuator limits in the tightest curves. Commanded and applied steering remain nearly identical. The initial acceleration saturation for speed build-up followed by a slow decay. Speed tracking shows the same bias as before, the applied velocity stabilizes 0.03–0.05 m/s below the nominal reference.

Overall, the controller maintains consistent tracking across the entire extended circuit, but the roundabout exit and the tight interior segment of the oval expose its main limitations: sensitivity to curvature gradients, limited preview horizon in the spatial formulation, and reliance on steering commands that approach the

(a) Errors



(c) Reference



(b) Control inputs

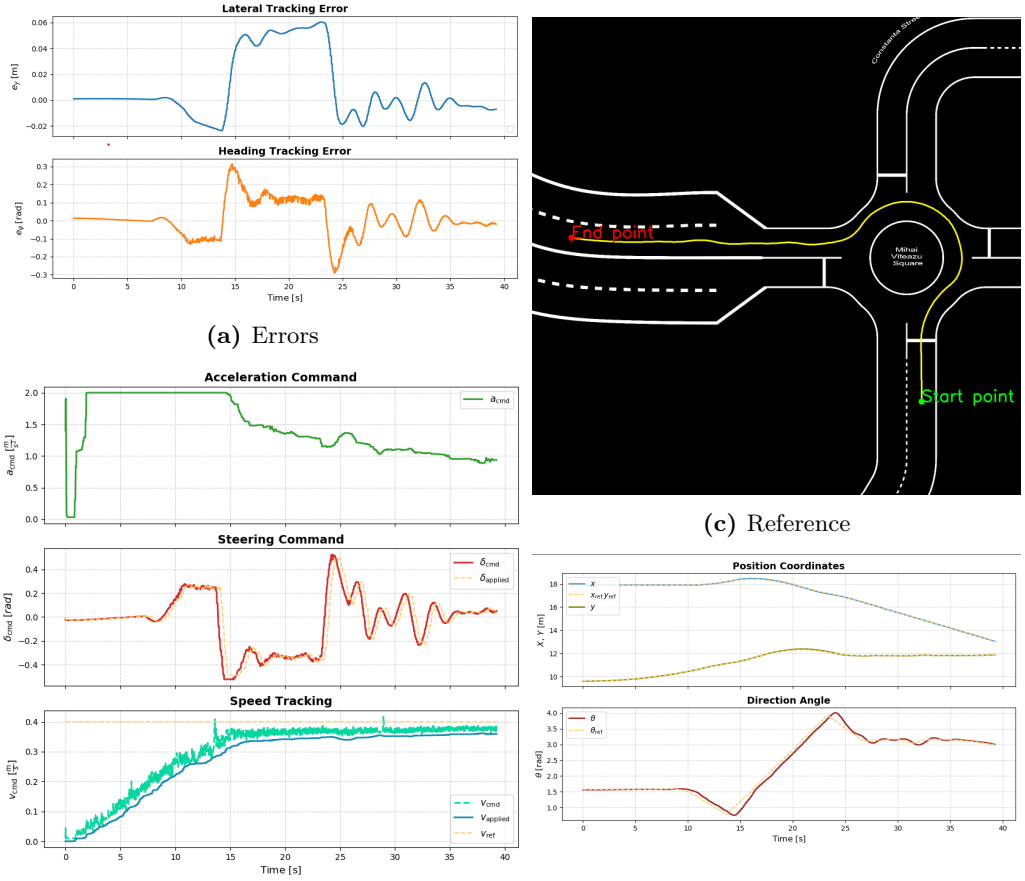
(d) Trajectory

Figure 3.21: Performance in Gazebo with Spatial Domain MPC Controller on a Track with First and Second Exit from Roundabout and Missing Lane Marker

physical limits of the actuator.

In the Figure 3.22 the focus is on the third exit maneuver from the roundabout. The executed position trajectory closely follows the reference during the entrance and exit, with only minor deviations visible in the transition between the circular arc and the straight line. The heading profile confirms that the steering maneuver is well aligned with the reference curvature: the evolution of θ matches the reference trajectory except for a short mismatch occurring at the peak curvatures of the entrance and the exit.

The lateral error remains within $\pm 2-3\,\text{cm}$ during most of the exit and increases to approximately 6cm at the curvature inflection point, where the controller corrects a small under-steer response. After the transition to the straight road, e_y quickly returns to within a few millimetres of zero and remains bounded for the



(a) Errors

(c) Reference

(b) Control inputs

(d) Trajectory

Figure 3.22: Performance in Gazebo with Spatial Domain MPC Controller on a Track with Third Exit from Roundabout

rest of the trajectory. The heading error behaves similarly, it remains near zero inside the roundabout, exhibits a short excursion of up to ≈ 0.3 rad during the rapid curvature change, and then settles below 0.05 rad on the straight segment.

The acceleration command saturates longer than in other two tests in the beginning to reach the target speed, after which it settles into a monotonically decreasing trend as the vehicle approaches the constant-velocity, similarly as in previous simulations. The steering command exhibits two dominant features: a negative steering peak during the final part inside the roundabout and a subsequent positive peak required to stabilise the transition onto the straight segment. These peaks remain within ± 0.45 rad and applied steering closely follows the commanded input. The speed-tracking plot shows the same behaviour observed in other experiments, where the applied velocity converges to a value slightly

below the reference.

In these experiments the difference between commanded and applied steering is negligible, indicating that the steering actuator dynamics introduce no measurable delay relative to the control-loop frequency. If this wasn't the case, there is a possibility of introducing the prediction of the states based on the delay of the actuatuors. The prediction can be done by simple Euler integration, and based on the values in the predicted moment, the steering angle is calculated and then applied.

Conclusions

This thesis explored the application of MPC in autonomous driving scenarios and analysed the advantages and benefits of various modelling approaches and domain formulations. The primary motivation was to improve trajectory-tracking solution for small-scale autonomous vehicles, motivated by the challenges and limitations observed in the BFMC competition environment. Earlier approaches relied heavily on lane-following strategies supported by a CNN-based perception system and a state-machine logic with numerous special cases. In contrast, the aim of this work was to simplify the control architecture and improve performance by replacing these multiple controllers with a single MPC-based strategy.

To achieve this, several vehicle models of different complexity and scale were implemented and evaluated within the Acados simulation framework. The models were initially tested along the trajectories with varying curvature, including an ellipsoidal and S shaped paths, allowing systematic comparison under both smooth and rapidly changing curvature conditions. The study investigated how model complexity influences prediction accuracy, controller smoothness, and overall driving performance while exploiting single track definition. Both kinematic and dynamic bicycle models were examined, together with three domain formulations - time, half-spatial, and spatial - each contributing distinct advantages.

In the time-based MPC, the vehicle dynamics are explicitly propagated in time, which allows direct control of velocity and acceleration. This formulation provides accurate transient behaviour and ease of actuator constraint handling. However, it is less suited for path-following tasks with variable speed, since the spatial progression depends on instantaneous velocity, which introduces non-uniform path discretisation and makes curvature-based control more difficult.

The purely spatial model reformulates the system along the path arc length, decoupling the motion from time and providing a uniform geometric represen-

tation of the trajectory. It achieves highly accurate path tracking and allows curvature-based control. Nonetheless, the absence of explicit time evolution limits its ability to handle transient acceleration effects. Furthermore, a predefined or lower-bounded velocity profile is required to ensure model feasibility, as zero or very low speeds make the spatial dynamics singular.

The half-spatial formulation was employed exclusively with the kinematic bicycle model, primarily to investigate the conceptual differences between the two domains and to provide a smooth transition from a fully time-based representation to a fully spatial one. This intermediate formulation integrates elements of both domains by maintaining time-dependent dynamics while parametrising the reference in the spatial domain. It enabled clearer understanding of how path parameterisation influences velocity evolution, tracking behaviour, and controller structure, before extending the spatial approach to more complex dynamic models.

The most used formulation that was considered and used in later scenarios consists of the MPC that operates in the spatial domain but updates at fixed time intervals. It retains spatial uniformity of the reference and curvature tracking while enabling realistic dynamic evolution in time.

Throughout this work differences between kinematic and dynamic vehicle models were highlighted, both in terms of prediction quality and control performance. The kinematic bicycle model proved effective for scenarios with ellipsoidal and S shaped paths. Its simplicity enables fast computation and smooth control actions, and in all formulations it consistently achieved near-perfect path tracking with minimal lateral and heading error. However, the kinematic model systematically underestimates the coupling between longitudinal and lateral motion, leading to optimistic predictions during aggressive manoeuvres or rapid curvature changes.

In contrast, the dynamic bicycle model captures phenomena that the kinematic approximation cannot, including sideslip, yaw-rate transients, and longitudinal force variation due to load transfer. These effects become particularly relevant in the S-shaped trajectory and in simulations with higher vehicle speed, where the dynamic model provides more accurate and physically consistent representation of the vehicle behaviour. The dynamic model successfully reproduced velocity dips at high curvature, realistic variations in v_y and ω.

Several variants of the dynamic model were examined, differing primarily in the computation of the longitudinal forces: a version driven by commanded longitudinal acceleration, a motor-parameter-based formulation, and a load-dependent version. All of these formulations, including the load-sensitive one, achieved high tracking accuracy, with lateral and heading errors remaining small even under combined-slip conditions. Transient oscillations in acceleration and yaw rate ap-

pear at curvature sign reversals, but these remain bounded and do not compromise stability.

A notable characteristic of the dynamic model is the increase in lateral velocity and slip angle during the S-shaped manoeuvres. In these parts of the track, v_y grows, which leads to an increase of slip angle, that considered undesirable for path tracking. This behaviour highlights the need to explicitly regulate sideslip through additional penalties, constraints, or curvature-based speed adaptation when deploying dynamic models in practice.

Despite the higher computational complexity and the slightly larger transient tracking errors, the dynamic formulation offers better physical fidelity. It captures the limitations imposed by the friction ellipse, actuator bounds, and tire nonlinearities much more effectively than the kinematic model, making it the preferred choice for high-performance. Continued development should focus on controlling the slip angle and lateral velocity to ensure stable and predictable behaviour under more aggressive manoeuvres.

After the comparative analysis, the kinematic spatial domain formulation for 1/10-scale vehicles was adopted and fully integrated into the ROS2 environment, followed by validation in Gazebo on BFMC track. The kinematic bicycle model was chosen over the dynamic formulation for several practical and methodological reasons. First, the kinematic model offers significantly lower computational complexity, which is essential for achieving real-time performance when operating in a simulation environment where sensor emulation, visualisation pipelines, and ROS2 communication already impose substantial computational load on the CPU, or possibly on a resource constrained device in a real setting. Because of greater state-dimension and more complex dynamics when implementing dynamic bicycle model,the optimization problems solved by MPC become more challenging and harder to execute reliably within Gazebo's real-time loop. Second, the kinematic model accurately represents the operational regime of the small-scale vehicle used here: at moderate velocities, dynamic effects such as load transfer, combined-slip behaviour, and nonlinear tire forces are limited and do not dominate the motion. Third, using a kinematic model in Gazebo ensures a clean separation between the high-level controller and the physics engine. This avoids the oscillatory or unstable interactions that can arise when dynamic models are imperfectly aligned with Gazebo's internal contact, friction, and integration mechanisms.

In order to successfully implement the MPC controller in the simulator,a smooth and dynamically feasible reference path must be generated. This was achieved using clothoid-based interpolation between graph nodes and by removing unnecessary intermediate nodes that produced non-physical curvature discontinuities. This geometric refinement significantly improved the curvature profile of the path:

curvature spikes were removed, curvature gradient was reduced, and the reference became more consistent with the assumptions of the spatial model. This smoothing step also improved numerical conditioning inside the optimiser, reducing solver iterations and improving feasibility in regions with tight curvature changes.

Using arc length as the independent variable eliminates the strong speed dependence present in time-domain models and allows the controller to reason explicitly about path geometry. Across all roundabout-exit scenarios, the controller demonstrated high lateral accuracy and heading errors typically below 0.1–0.15 rad, with deviations localised primarily around regions of abrupt curvature change. In the most demanding sections - such as the curvature break during the roundabout exit or the tight S-bend in the urban scenario-lateral errors increased temporarily but remained bounded, typically below 6–8 cm when using a constant speed profile and around 3- 4 cm when using curvature-adaptive speed planning. The behaviour remained stable throughout the simulation, with no drift or cumulative deformation of the trajectory over long horizons. These results were obtained with full dynamics simulated in Gazebo, confirming that the controller remains effective despite mismatch between the simplified kinematic model and the more complex simulated physics.

The controller consistently handled roundabout exits, long straights, narrow passages, and irregular urban geometries without drift, divergence, or loss of feasibility, even under actuator saturation and the state uncertainty inherent to simulation-based sensors. The results demonstrate that the combination of spatial-domain modeling and path-smoothing planning results in robust and computationally efficient control architecture for small-scale autonomous vehicles.

Despite the benefits, some limitations remain. Although MPC provides satisfactory performance, it still depends on accurate models and real-time computational efficiency. Several directions for future research arise from the findings and limitations of this work.

- **Dynamic vehicle modelling** Incorporating a reduced-order dynamic model, or a hybrid kinematic-dynamic formulation, would improve fidelity in scenarios where lateral slip and load transfer become significant.
- **Actuator regulation** The experiments revealed mismatch between applied and commanded velocity and steering inputs. The applied velocity consistently remained below the commanded value and exhibited additional transient mismatch in the adaptive-speed scenario. Similar effects were observed in the steering signal, where high-frequency curvature transitions produced oscillations and small phase lags. Integrating explicit actuator dynamics or low-level control constraints into the MPC formulation would increase robustness and reduce the influence of unmodelled actuation delays.

- **Extended prediction strategies** Performance in tight or rapidly changing curvature segments can be limited by the spatial prediction horizon. Adaptive horizon methods or multi-rate prediction could provides longer preview ranges when needed without sacrificing real-time feasibility.
- **Dynamic obstacle handling and extended prediction strategies** Extending the controller to consider moving obstacles and vehicles on the track represents an important step toward real-world deployment. Approaches such as collision-avoidance constraints or online path reparameterisation could be integrated into the MPC.
- **Integration with real-time perception and hardware validation** The present framework relies on a pre-defined, pixel-based map. Incorporating onboard perception, already used in BFMC competitions, such as vision-based lane detection, would allow the system to adapt to map inaccuracies, dynamic changes, and imperfections in the environment. Finally, transferring the proposed method to a physical small-scale platform would provide insight into real-world effects such as friction variation, sensor noise, mechanical backlash, and communication delays, offering a complete experimental validation of the approach.

Nonetheless, within the scope of this thesis, the developed methods provide a complete and experimentally validated control pipeline capable of robust and precise trajectory tracking in realistic miniature-vehicle scenarios. In summary, this thesis demonstrates that MPC is a robust and generalizable control strategy for scaled autonomous vehicles, capable of improving performance while simplifying the overall architecture.

Bibliography

[1] Daniel Kloeser et al. "NMPC for Racing Using a Singularity-Free Path-Parametric Model with Obstacle Avoidance". en. In: *IFAC-PapersOnLine* 53.2 (2020), pp. 14324–14329. ISSN: 24058963. DOI: 10.1016/j.ifacol.2020.12.1376. URL: https://linkinghub.elsevier.com/retrieve/pii/S2405896320317845.

[2] Jinrui Nan, Xucheng Ye, and Wanke Cao. "Nonlinear Model Predictive Control with Terminal Cost for Autonomous Vehicles Trajectory Follow". en. In: *Applied Sciences* 12.22 (Jan. 2022). Publisher: Multidisciplinary Digital Publishing Institute, p. 11359. ISSN: 2076-3417. DOI: 10.3390/app122211359. URL: https://www.mdpi.com/2076-3417/12/22/11359 (visited on 10/05/2025).

[3] Jianhua Guo et al. "Data-Driven Enhancements for MPC-Based Path Tracking Controller in Autonomous Vehicles". en. In: *Sensors* 24.23 (Jan. 2024). Publisher: Multidisciplinary Digital Publishing Institute, p. 7657. ISSN: 1424-8220. DOI: 10.3390/s24237657. URL: https://www.mdpi.com/1424-8220/24/23/7657.

[4] Vittorio Cataffo et al. "A Nonlinear Model Predictive Control Strategy for Autonomous Racing of Scale Vehicles". In: *2022 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. arXiv:2302.04722 [cs]. Oct. 2022, pp. 100–105. DOI: 10.1109/SMC53654.2022.9945279. URL: http://arxiv.org/abs/2302.04722.

[5] Enrico Picotti et al. "A Learning-based Nonlinear Model Predictive Controller for a Real Go-Kart based on Black-box Dynamics Modeling through Gaussian Processes". In: *IEEE Transactions on Control Systems Technology* 31.5 (Sept. 2023). arXiv:2305.17949 [eess], pp. 2055–2065. ISSN: 1063-6536,

1558-0865, 2374-0159. DOI: 10.1109/TCST.2023.3291532. URL: http://arxiv.org/abs/2305.17949.

[6] Mario Zanon, Janick V. Frasch, and Moritz Diehl. "Nonlinear Moving Horizon Estimation for combined state and friction coefficient estimation in autonomous driving". In: *2013 European Control Conference (ECC)*. July 2013, pp. 4130–4135. DOI: 10.23919/ECC.2013.6669832. URL: https://ieeexplore.ieee.org/document/6669832/.

[7] Janick V. Frasch et al. "An auto-generated nonlinear MPC algorithm for real-time obstacle avoidance of ground vehicles". en. In: *2013 European Control Conference (ECC)*. Zurich: IEEE, July 2013, pp. 4136–4141. ISBN: 978-3-033-03962-9. DOI: 10.23919/ECC.2013.6669836. URL: https://ieeexplore.ieee.org/document/6669836/.

[8] Meng Wang et al. "Path Tracking Method Based on Model Predictive Control and Genetic Algorithm for Autonomous Vehicle". en. In: *Mathematical Problems in Engineering* 2022.1 (2022), p. 4661401. ISSN: 1563-5147. DOI: 10.1155/2022/4661401. URL: https://onlinelibrary.wiley.com/doi/abs/10.1155/2022/4661401.

[9] Giorgio Fontana. "Autonomous Driving of Miniature Race Cars". it. PhD thesis. POLITECNICO DI MILANO, 2013. URL: https://www.politesi.polimi.it/retrieve/a81cb05b-02b5-616b-e053-1605fe0a889a/2014_10_Fontana.pdf.

[10] Mattia Bruschetta et al. "A Nonlinear Model Predictive Control based Virtual Driver for high performance driving". In: *2019 IEEE Conference on Control Technology and Applications (CCTA)*. Aug. 2019, pp. 9–14. DOI: 10.1109/CCTA.2019.8920504. URL: https://ieeexplore.ieee.org/document/8920504.

[11] Egbert Bakker, Hans B. Pacejka, and Lars Lidner. "A New Tire Model with an Application in Vehicle Dynamics Studies". English. In: ISSN: 0148-7191, 2688-3627. SAE International, Apr. 1989. DOI: 10.4271/890087. URL: https://saemobilus.sae.org/papers/a-new-tire-model-application-vehicle-dynamics-studies-890087.

[12] Hans B. Pacejka. "Semi-Empirical Tire Models". en. In: *Tire and Vehicle Dynamics*. Elsevier, 2012, pp. 149–209. ISBN: 978-0-08-097016-5. DOI: 10.1016/B978-0-08-097016-5.00004-8. URL: https://linkinghub.elsevier.com/retrieve/pii/B9780080970165000048.

[13] Alexander Liniger, Alexander Domahidi, and Manfred Morari. "Optimization-Based Autonomous Racing of 1:43 Scale RC Cars". In: *Optimal Control Applications and Methods* 36.5 (Sept. 2015). arXiv:1711.07300 [math], pp. 628–647. ISSN: 0143-2087, 1099-1514. DOI: 10.1002/oca.2123. URL: http://arxiv.org/abs/1711.07300.

[14] Lisheng Jin et al. "Research on the control and coordination of four-wheel independent driving/steering electric vehicle". en. In: *ResearchGate* (). DOI: 10.1177/1687814017698877. URL: https://www.researchgate.net/publication/316208176_Research_on_the_control_and_coordination_of_four-wheel_independent_drivingsteering_electric_vehicle.

[15] Moritz Werling et al. "Optimal trajectory generation for dynamic street scenarios in a Frenet Frame". en. In: *2010 IEEE International Conference on Robotics and Automation*. Anchorage, AK: IEEE, May 2010, pp. 987–993. ISBN: 978-1-4244-5038-1. DOI: 10.1109/ROBOT.2010.5509799. URL: http://ieeexplore.ieee.org/document/5509799/.

[16] Mitesh Agrawal. *What is Model Predictive Control (MPC)? - Technical Articles*. en. URL: https://control.com/technical-articles/what-is-model-predictive-control-mpc/.

[17] Zoltan K Nagy, Rüdiger Franke, and Frank Allgöwer. *NONLINEAR MODEL PREDICTIVE CONTROL OF BATCH PROCESSES: AN INDUSTRIAL CASE STUDY*. en. URL: https://www.researchgate.net/publication/242219107_NONLINEAR_MODEL_PREDICTIVE_CONTROL_OF_BATCH_PROCESSES_AN_INDUSTRIAL_CASE_STUDY.

[18] Sébastien Gros et al. "From linear to nonlinear MPC: bridging the gap via the real-time iteration". en. In: *International Journal of Control* 93.1 (Jan. 2020), pp. 62–80. ISSN: 0020-7179, 1366-5820. DOI: 10.1080/00207179.2016.1222553. URL: https://www.tandfonline.com/doi/full/10.1080/00207179.2016.1222553.

[19] Hans Joachim Ferreau et al. "qpOASES: a parametric active-set algorithm for quadratic programming". en. In: *Mathematical Programming Computation* 6.4 (Dec. 2014), pp. 327–363. ISSN: 1867-2957. DOI: 10.1007/s12532-014-0071-1. URL: https://doi.org/10.1007/s12532-014-0071-1.

[20] H. J. Ferreau, H. G. Bock, and M. Diehl. "An online active set strategy to overcome the limitations of explicit MPC". en. In: *International Journal of Robust and Nonlinear Control* 18.8 (2008), pp. 816–830. ISSN: 1099-1239. DOI: 10.1002/rnc.1251. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/rnc.1251.

[21] Alberto Bemporad et al. "The explicit linear quadratic regulator for constrained systems". In: *Automatica* 38.1 (Jan. 2002), pp. 3–20. ISSN: 0005-1098. DOI: 10.1016/S0005-1098(01)00174-1. URL: https://www.sciencedirect.com/science/article/pii/S0005109801001741.

[22] Barbara Barros Carlos et al. *An Efficient Real-Time NMPC for Quadrotor Position Control under Communication Time-Delay*. arXiv:2010.11264 [cs]. Oct. 2020. DOI: 10.48550/arXiv.2010.11264. URL: http://arxiv.org/abs/2010.11264.

[23] Gianluca Frison and Moritz Diehl. "HPIPM: a high-performance quadratic programming framework for model predictive control∗". In: *IFAC-PapersOnLine*. 21st IFAC World Congress 53.2 (Jan. 2020), pp. 6563–6569. ISSN: 2405-8963. DOI: 10.1016/j.ifacol.2020.12.073. URL: https://www.sciencedirect.com/science/article/pii/S2405896320303293.

[24] Gianluca Frison and John Bagterp Jørgensen. "Efficient implementation of the Riccati recursion for solving linear-quadratic control problems". In: *2013 IEEE International Conference on Control Applications (CCA)*. ISSN: 1085-1992. Aug. 2013, pp. 1117–1122. DOI: 10.1109/CCA.2013.6662901. URL: https://ieeexplore.ieee.org/document/6662901.

[25] *acados — acados documentation*. URL: https://docs.acados.org/index.html.

[26] Robin Verschueren et al. *acados: a modular open-source framework for fast embedded optimal control*. arXiv:1910.13753 [math]. Nov. 2020. DOI: 10.48550/arXiv.1910.13753. URL: http://arxiv.org/abs/1910.13753.

[27] Amon Lahr et al. *L4acados: Learning-based models for acados, applied to Gaussian process-based predictive control - Experimental data*. en. Dec. 2024. DOI: 10.3929/ETHZ-B-000707631. URL: http://hdl.handle.net/20.500.11850/707631.

[28] David Portugal, Rui P. Rocha, and João P. Castilho. "Inquiring the robot operating system community on the state of adoption of the ROS 2 robotics middleware". en. In: *International Journal of Intelligent Robotics and Applications* 9.2 (June 2025), pp. 454–479. ISSN: 2366-598X. DOI: 10.1007/s41315-024-00393-4. URL: https://doi.org/10.1007/s41315-024-00393-4.

[29] Andrew Gray, Janick Frasch, and Mario Zanon. "Vehicle Parameters for a Jaguar X-Type". en. In: ().