FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Collision Avoidance Strategies for Autonomous Racing Vehicles

Tomás Gomes Monteiro Sanhudo



Mestrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Luís Tiago de Freixo Ramos Paiva

July 23, 2025

© Tomás Sanhudo, 2025

Resumo

Os veículos autónomos têm vindo a assumir um papel cada vez mais relevante, impulsionando o desenvolvimento de corridas autónomas como plataforma de teste para algoritmos de navegação, otimização de trajetórias e segurança em condições extremas. As corridas autónomas permitem aos investigadores enfrentar desafios como a tomada de decisão em tempo real e a prevenção de colisões, os quais, por sua vez, têm implicações mais amplas na melhoria da segurança dos veículos autónomos em contextos de transporte real.

Esta dissertação apresenta o desenvolvimento de um sistema híbrido de prevenção de colisões para corridas autónomas, que combina o Campo Potencial Artificial (APF) para o planeamento dinâmico de trajetórias, com Controlo Preditivo (MPC) para o seguimento preciso das trajetórias. O sistema foi implementado e testado num ambiente MATLAB/Simulink, com especial foco no desvio de obstáculos e na aderência às trajetórias ideais de corrida. Esta abordagem híbrida tira partido da eficiência computacional dos APF para uma geração rápida de trajetórias, e da capacidade do MPC para lidar com restrições do sistema, assegurando um movimento do veículo suave, viável e compatível com a sua dinâmica, constituindo, assim, uma solução promissora para futuras aplicações em corridas autónomas.

Um elemento central da pesquisa é o planeador de trajetórias baseado no APF, que foi testado em diferentes configurações de pista e conseguiu gerar com sucesso trajetórias livres de colisões, demonstrando a sua adaptabilidade a ambientes de corrida complexos.

O sistema híbrido foi validado em simulação, demonstrando a sua capacidade de gerar trajetórias livres de colisões e segui-las com elevada precisão num ambiente virtual controlado. Estes resultados constituem uma base sólida para futuras aplicações em cenários reais de corridas autónomas e contribuem para os esforços contínuos no avanço de sistemas autónomos seguros e eficientes. Ao abordar os desafios inerentes às corridas autónomas, este trabalho contribui para a melhoria da segurança e desempenho tanto no contexto de corridas autónomas como em aplicações mais amplas de veículos autónomos.

Palavras-chave: corridas autónomas, prevenção de colisões, Campo Potencial Artificial (APF), Controlo Preditivo (MPC), planeamento de trajetórias, seguimento de trajetórias, MATLAB/Simulink

Abstract

Self-driving cars are becoming increasingly important, driving advancements in autonomous racing as a platform for testing navigation, trajectory optimization, and safety algorithms under extreme conditions. Autonomous racing allows researchers to address challenges such as real-time decision-making and collision avoidance, which have broader implications for improving autonomous vehicle safety in general transportation contexts.

The dissertation presents the development of a hybrid collision avoidance system for autonomous racing, combining Artificial Potential Field (APF) for reactive path planning with Model Predictive Control (MPC) for accurate path tracking. The system is implemented and tested in a MATLAB/Simulink environment, with a focus on obstacle avoidance and adherence to racing lines. This hybrid approach leverages the computational efficiency of APF for fast path generation and the ability of MPC to handle system constraints, ensuring smooth, feasible, and dynamically consistent vehicle motion, making it a promising solution for future autonomous racing applications.

A key part of the research is the APF-based path planner, which was tested on multiple track configurations and successfully generated collision-free trajectories, demonstrating its adaptability to complex racing environments.

The hybrid system was validated through simulation, demonstrating its ability to generate collision-free paths and track them accurately in a controlled virtual environment. These results provide a foundation for future application in real-world autonomous racing scenarios and contribute to ongoing efforts in advancing safe and efficient autonomous racing systems. By addressing the challenges of autonomous racing, this work contributes to improving safety and performance in both racing and broader applications of autonomous vehicles.

Keywords: autonomous racing, collision avoidance, Artificial Potential Field (APF), Model Predictive Control (MPC), path planning, path tracking, MATLAB/Simulink

UN Sustainable Development Goals (SDGs)

The Sustainable Development Goals (SDGs) are the blueprint to achieve a better and more sustainable future for all. They address the global challenges we face, including those related to poverty, inequality, climate change, environmental degradation, peace and justice. It consists of 17 Goals that are all interconnected, and in order to leave no one behind, it is important that we achieve them all by 2030 [1].

The SDGs represent an urgent call to action for all countries, whether poor, rich, or middle-income, to promote shared prosperity while protecting the planet. Figure 1 presents the 17 SDGs, which since their creation have inspired thousands of events, publications, and initiatives worldwide.



Figure 1: United Nations 17 Sustainable Development Goals [2].

This thesis focuses on developing collision avoidance algorithms for autonomous vehicle racing. While it does not explicitly address any of the 17 United Nations SDGs, its broader impact aligns with several of them. By reflecting on the wider context of the project, it becomes clear that there are meaningful connections to specific SDGs.

Autonomous vehicle racing has broader implications for autonomous vehicles technologies, which have the potential to improve road safety, optimize energy efficiency, and promote innovation within the transportation sector. These advancements can contribute to more sustainable cities, reduced resource consumption, and economic growth through technological development. Although the primary aim of this thesis is technical, its impacts extend into environmental, social, and economic domains that align with the SDGs.

The specific Sustainable Development Goals related to this thesis are:

SDG 4 Quality Education

SDG 9 Industry, Innovation and Infrastructure

SDG 11 Sustainable Cities and Communities

Table 1 illustrates the SDGs and the corresponding performance indicators established by the United Nations to which this work is related.

Table 1: Contribution to SDGs and Performance Indicators.

SDG	**Target**	**Contribution**	**Performance Indicators and Metrics**
4	4.4	Development of autonomous racing technologies promotes STEM[1] education and technical skill development.	Indicator 4.4.1: Proportion of youth and adults with ICT[2] skills relevant to autonomous systems.
	4.7	Research contributes to education for sustainable development in transportation technologies.	Indicator 4.7.1: Extent to which sustainable development is mainstreamed in technical education.
9	9.5	Advancements in collision avoidance algorithms represent technological innovation in autonomous systems.	Indicator 9.5.1: Research expenditure in autonomous vehicle technologies as proportion of GDP[3].
	9.b	Development of domestic technology for autonomous racing systems.	Indicator 9.b.1: Proportion of high-tech industry value added.
11	11.2	Improved collision avoidance contributes to safer future transport systems.	Indicator 11.2.1: Proportion of population with access to safe transport systems.
	11.3	Research informs urban planning for autonomous vehicle integration.	Indicator 11.3.2: Proportion of cities integrating autonomous vehicle research into public urban planning forums.
	11.6	Autonomous racing research leads to more efficient vehicles with lower environmental impact.	Indicator 11.6.2: Reduction in pollution levels from optimized autonomous transport.

[1] Science, Technology, Engineering, and Mathematics (STEM)
[2] Information and Communication Technology (ICT)
[3] Gross Domestic Product (GDP)

Acknowledgements

This thesis represents far more than academic work, it is the culmination of years of support, mentorship, and opportunities provided by extraordinary individuals and institutions. With this, I would like to thank:

My supervisor, Professor Luís Tiago Paiva, for his continuous support and invaluable advice, especially during critical moments of this research. His guidance was indispensable to the completion of this work.

All my teachers throughout my studies, for sharing their knowledge and guidance that shaped me as an engineer and made this journey possible.

Everyone involved in the UP2Speed project and all members of LSCOE (Laboratory of Systems, Control, Optimization, and Estimation) SYSTEC-ARISE located in Lab i202 DEEC-FEUP. I am truly thankful for your assistance, collaborative environment, and for providing me with all the necessary conditions and resources to complete this project.

My family, for their unwavering support, endless patience, and for providing me with all the conditions to pursue and complete my studies.

My friends, for making this journey unforgettable through shared moments and for all the help along the way.

Sincerely,

Tomás Gomes Monteiro Sanhudo

"Success is also a way to say thank you to all the people who help you to achieve this."

Jannik Sinner

Contents

1 Introduction **1**

2 State-of-the-Art **3**
2.1 Autonomous Vehicle Racing - Background . . . 3
2.1.1 Perception . . . 4
2.1.2 Planning . . . 4
2.1.3 Control . . . 4
2.2 Collision Avoidance - State-of-the-Art . . . 5
2.2.1 Problem Description . . . 5
2.2.2 Overview . . . 5
2.2.3 Algorithms . . . 6
2.2.3.1 APF and MPC: Hybrid Approach for Collision Avoidance . . . 8
2.2.3.2 RL: Reward Signal Design for Autonomous Racing . . . 11
2.3 Laboratory Facilities . . . 13
2.3.1 Race track . . . 13
2.3.2 Race car . . . 14

3 Collision Avoidance **16**
3.1 Potential Field Equations . . . 16
3.1.1 Attractive Potential Field . . . 16
3.1.2 Repulsive Potential Field . . . 19
3.1.3 Universal Potential Field . . . 20
3.2 Induced Force and Trajectory Generation . . . 21
3.2.1 Induced Force . . . 21
3.2.2 Trajectory Generation . . . 22
3.3 MATLAB Implementation . . . 23
3.3.1 General Code Logic . . . 23
3.3.2 Scaling Between Laboratory and Simulation . . . 26
3.3.3 Segment Types . . . 26
3.3.3.1 Segments with Obstacles . . . 27
3.3.3.2 Straights Segments without Obstacles . . . 35
3.3.3.3 Curved Segments without Obstacles . . . 38
3.4 Path Planning Results . . . 42

4 Model Predictive Control **48**
4.1 Introduction to MPC . . . 48
4.1.1 MPC in Autonomous Racing . . . 49
4.1.2 MPC Parameters . . . 50

4.1.3 MPC Types . . . 51
4.2 MPC Design and Implementation . . . 53
4.2.1 Step 1: MPC Virtual Lab . . . 53
4.2.1.1 Objective . . . 53
4.2.1.2 Vehicle Dynamic Model and State-Space Description . . . 53
4.2.1.3 Reference Trajectory Definition . . . 56
4.2.1.4 MPC Controller Design and Tuning . . . 56
4.2.1.5 Simulation Results and Analysis . . . 58
4.2.2 Step 2: Implementing the Control Model and Approach of [5] . . . 60
4.2.2.1 Objective . . . 60
4.2.2.2 Vehicle Dynamic Model and State-Space Description . . . 60
4.2.2.3 Reference Trajectory Definition . . . 63
4.2.2.4 MPC Controller Design and Tuning . . . 65
4.2.2.5 Simulation Results and Analysis . . . 66
4.2.3 Step 3: Adaptive MPC Implementation . . . 69
4.2.3.1 Objective . . . 69
4.2.3.2 Vehicle Dynamic Model and State-Space Description . . . 69
4.2.3.3 Reference Trajectory Definition . . . 74
4.2.3.4 MPC Controller Design and Tuning . . . 75

5 Integration of Path Planning and Control **77**
5.1 Test Scenarios . . . 77
5.2 Results and Discussion . . . 77
5.2.1 Scenario 1: Simpler Track . . . 78
5.2.2 Scenario 2: Laboratory Track . . . 83

6 Conclusions and Future Work **89**

References **91**

List of Figures

1 United Nations 17 Sustainable Development Goals [2]. iii

2.1 Autonomous racing pipeline showing perception, planning, and control integration for real-time decision-making[3]. 3
2.2 Autonomous racing pipeline showing perception, planning, and control integration for real-time decision-making [5]. 6
2.3 The concept of the APF method [6]. 7
2.4 Universal potential field of road and obstacle [5]. 9
2.5 Vehicle trajectories of controllers A and B and the planned trajectory [5]. 10
2.6 Stability performance of controllers A and B [5]. 11
2.7 Vehicle position projected onto a reference line to calculate progress [8]. 12
2.8 Cross-track distance and heading error used in velocity-based rewards [8]. 12
2.9 Trajectory using the CTH reward framework with dynamic obstacle avoidance [8]. 13
2.10 Completed race track with defined boundaries [11]. 14
2.11 Three race cars uniquely colored for tracking [11]. 15
2.12 Race car with PCB and ESP32 installed [11]. 15

3.1 $A(X)$ Potential Variation. 17
3.2 $A(Y)$ Potential Variation. 18
3.3 Forward Motion Potential $P_A(X,Y)$. 18
3.4 Attractive Potential Field P_R. 19
3.5 Repulsive Potential Field P_O. 20
3.6 Universal Potential Field P_U. 20
3.7 Induced force $\vec{F}_U(X,Y)$ across the road. 22
3.8 Zoomed-in views of the induced force field for better visualization of local vector behavior. 22
3.9 DSD coordinate system. 24
3.10 Flowchart with the logic behind the code. 25
3.11 Laboratory car dimensions. 26
3.12 Car model dimensions. 26
3.13 Straight segment with respective start and end center points C_1 and C_2, and obstacle position P_{obs}. 28
3.14 Segment with corresponding APF local coordinate axes. 29
3.15 Segment with corresponding APF local coordinate axes. 29
3.16 Segment with respective local reference planned trajectory. 33
3.17 APF contour plot illustrating the trajectory overlaid on the potential field. 33
3.18 APF contour plot with overlaid circles and rectangle representing the vehicle and obstacle dimensions, respectively. 34
3.19 Segment with respective local reference planned trajectory. 34

3.20 APF contour plot illustrating the trajectory overlaid on the potential field. 34
3.21 Segment with respective local reference planned trajectory. 35
3.22 APF contour plot illustrating the trajectory overlaid on the potential field. 35
3.23 Straight segment with respective start and end center points C_1 and C_2. 35
3.24 Segment with respective local reference planned trajectory. 37
3.25 Reference orientation angles based on the DSD convention. 38
3.26 Example 1: Curved segment with respective start and end center points C_1 and C_2. 39
3.27 Example 2: Curved segment with respective start and end center points C_1 and C_2. 39
3.28 Illustration of the ellipse used in trajectory generation. 41
3.29 Example 1: Segment with respective local reference planned trajectory. 42
3.30 Example 2: Segment with respective local reference planned trajectory. 42
3.31 Laboratory Race Track. 43
3.32 Laboratory Race Track in DSD. 43
3.33 Reference trajectory on the laboratory track. 43
3.34 Reference trajectory on the same track with obstacles repositioned. 44
3.35 Reference trajectory on a modified track configuration. 45
3.36 Reference trajectory on an alternative track layout. 46

4.1 The MPC process [20]. 49
4.2 MPC general design. 50
4.3 MPC types [23]. 52
4.4 Memory and speed comparision for different types of MPC [20]. 53
4.5 Representation of the car model. 54
4.6 Simulink model of [24]. 58
4.7 Steering angle response. 58
4.8 Lateral position tracking: Y_v vs. $Y_{v,\text{ref}}$. 59
4.9 Yaw angle tracking: ψ vs. ψ_{ref}. 59
4.10 Vehicle executing the lane change in the DSD scenario. 60
4.11 Representation of the vehicle dynamic model. 60
4.12 Generated reference trajectory from DSD. 63
4.13 Simulink model used in Step 2 implementation. 66
4.14 Steering angle applied by the controller. 66
4.15 Lateral position tracking: Y_v vs. $Y_{v,\text{ref}}$. 67
4.16 Body sideslip angle tracking: β vs. β_{ref}. 67
4.17 Yaw rate tracking: $\dot{\psi}$ vs. $\dot{\psi}_{\text{ref}}$. 68
4.18 Start of the maneuver at $Y_v = 6$ m. 68
4.19 Mid-maneuver near $Y_v = 2$ m. 68
4.20 End of the maneuver returning to $Y_v = 6$ m. 68
4.21 Internal prediction vehicle model. 70
4.22 Simulink Lookup Table block. 74
4.23 Simulink model used in Step 3 implementation [30]. 76

5.1 Path tracking: Vehicle position vs. Reference trajectory in Scenario 1. 78
5.2 Steering angle δ applied by the controller in Scenario 1. 79
5.3 Longitudinal acceleration command $a_{x,\text{cmd}}$ applied by the controller in Scenario 1. 79
5.4 Longitudinal position tracking: X_v vs. $X_{v,\text{ref}}$ in Scenario 1. 80
5.5 Lateral position tracking: Y_v vs. $Y_{v,\text{ref}}$ in Scenario 1. 80
5.6 Velocity tracking: V vs. V_{ref} in Scenario 1. 81
5.7 Yaw angle tracking: ψ vs. ψ_{ref} in Scenario 1. 82

5.8 Yaw rate tracking: $\dot{\psi}$ vs. $\dot{\psi}_{\text{ref}}$ in Scenario 1. 82
5.9 Lateral tracking error (%) in Scenario 1. 83
5.10 Path tracking: Vehicle position vs. Reference trajectory in Scenario 2. 84
5.11 Steering angle δ applied by the controller in Scenario 2. 84
5.12 Longitudinal acceleration command $a_{x,\text{cmd}}$ applied by the controller in Scenario 2. 85
5.13 Longitudinal position tracking: X_v vs. $X_{v,\text{ref}}$ in Scenario 2. 85
5.14 Lateral position tracking: Y_v vs. $Y_{v,\text{ref}}$ in Scenario 2. 86
5.15 Velocity tracking: V vs. V_{ref} in Scenario 2. 86
5.16 Yaw angle tracking: ψ vs. ψ_{ref} in Scenario 2. 87
5.17 Yaw rate tracking: $\dot{\psi}$ vs. $\dot{\psi}_{\text{ref}}$ in Scenario 2. 87
5.18 Lateral tracking error (%) in Scenario 2. 88

List of Tables

1 Contribution to SDGs and Performance Indicators. iv

3.1 Summary of constants used in the APF algorithm. 32
3.2 Track segment coordinates and corresponding attributes. 44
3.3 Track segment coordinates and corresponding attributes. 46
3.4 Track segment coordinates and corresponding attributes. 47

4.1 Vehicle parameters used in the MPC Virtual Lab [24]. 55
4.2 Vehicle parameters used for the dynamic model [5]. 62
4.3 Vehicle parameters of the internal prediction model. 73

Abbreviations and Symbols

2D	Two-Dimensional
APF	Artificial Potential Field
CG	Center of Gravity
CTH	Cross-Track and Heading
DSD	Driving Scenario Designer
GDP	Gross Domestic Product
ICT	Information and Communication Technology
LTI	Linear Time-Invariant
MIMO	Multi-Input Multi-Output
MMPC	Multiconstrained Model Predictive Control
MPC	Model Predictive Control
OCP	Optimal Control Problem
PCB	Printed Circuit Board
PID	Proportional-Integral-Derivative
RC	Radio-Controlled
RL	Reinforcement Learning
SDGs	Sustainable Development Goals
SLAM	Simultaneous Localization And Mapping
STEM	Science, Technology, Engineering, and Mathematics
$A(X)$	amplitude of the potential field along the road width
$A(Y)$	amplitude of the potential field along the road length
a, b	ellipse semi-axes (m)
a_x	vehicle longitudinal acceleration (m/s^2)
$a_{x,\text{cmd}}$	commanded longitudinal acceleration (m/s^2)
$C_1 = (x_1, y_1)$	segment start center point (m)
$C_2 = (x_2, y_2)$	segment end center point (m)
C_f	front wheels cornering stiffness (N/rad)
C_r	rear wheels cornering stiffness (N/rad)
C_T	instant curvature (1/m)
c_1	lateral repulsion weight
c_2	longitudinal repulsion weight
D_b	shortest braking distance (m)
D_s	scope of the universal potential field in the Y-direction (m)
D_t	scope of the repulsive potential field in the Y-direction (m)
d_{total}	total distance (m)
F_m	maximum braking force per tire (N)
$F_{U_X}(X, Y)$	scalar induced force in the X-direction (N)

$F_{U_Y}(X,Y)$	scalar induced force in the Y-direction (N)
F_{wf}	front wheels total forces (N)
F_{wr}	rear wheels total forces (N)
F_{yf}	front lateral tire force (N)
F_{yr}	rear lateral tire force (N)
I_z	yaw moment of inertia (kg·m^2)
L_{arc}	arc length (m)
L_{car}	vehicle length (m)
L_{center}	lane center (m)
L_{ellipse}	ellipse perimeter (m)
L_l	obstacle spatial influence (m)
L_{obs}	obstacle length (m)
L_w	lane width (m)
l_f	distance from CG of the vehicle to front axle (m)
l_r	distance from CG of the vehicle to rear axle (m)
m	vehicle mass (kg)
N	number of integration steps
N_c	control horizon
N_p	prediction horizon
$P_A(X,Y)$	forward motion potential
$P_O(X,Y)$	repulsive potential field
$P_R(X,Y)$	attractive potential field
$P_U(X,Y)$	universal potential field
$P_{\text{center}} = (x_c,\ y_c)$	ellipse center point (m)
$P_{\text{end}} = (x_{\text{end}},\ y_{\text{end}})$	end reference trajectory point (m)
$P_i = (X_{v,i},\ Y_{v,i})$	initial position of the vehicle point (m)
P_m	amplitude of potential of the middle of the road
$P_{\text{obs}} = (X_o, Y_o)$	obstacle position point (m)
$P_{\text{start}} = (x_{\text{start}},\ y_{\text{start}})$	start reference trajectory point (m)
P_t	threshold that determines the scope of the repulsive potential field
R_l	road length (m)
R_w	road width (m)
T_s	sample time (s)
t_{ref}	reference trajectory time vector (s)
t_{total}	total simulation time (s)
u	input variables vector
V	vehicle velocity (m/s)
V_r	characteristic speed (m/s)
V_x	vehicle longitudinal velocity (m/s)
V_y	vehicle lateral velocity (m/s)
V_{scalar}	reference velocity used to advance the trajectory (m/s)
v_{obs}	obstacle longitudinal velocity (m/s)
W_{obs}	obstacle width (m)
X_l	centerline of the left lane (m)
$X_{\text{min}}, X_{\text{max}}$	minimum and maximum lateral coordinate of the road (m)
X_o	position of the obstacle in the X-direction (m)
X_r	centerline of the right lane (m)

X_v	position of the vehicle in the X-direction (m)
$X_{v,i}$	initial position of the vehicle in the X-direction (m)
$X_{v,\text{ref}}$	reference longitudinal position (m)
x	state variables vector
$Y_{\min}, Y_{\max}$	minimum and maximum longitudinal coordinate of the road (m)
Y_o	position of the obstacle in the Y-direction (m)
Y_v	position of the vehicle in the Y-direction (m)
$Y_{v,i}$	initial position of the vehicle in the Y-direction (m)
$Y_{v,\text{ref}}$	reference lateral position (m)
$Y_{v,\min}, Y_{v,\max}$	minimum and maximum reference lateral position (m)
y	output variables vector
α_f	front tire slip angle (rad)
α_r	rear tire slip angle (rad)
β	body sideslip angle (rad)
β_{ref}	reference body sideslip angle (rad)
$\beta_{\max}$	maximum reference body sideslip angle (rad)
δ	steering angle (rad)
Δt	simulation integration time step (s)
θ_i	segment orientation initial angle ($^\circ$)
θ_f	segment orientation final angle ($^\circ$)
θ_{ref}	reference angular vector ($^\circ$)
ψ	yaw angle (rad)
ψ_i	initial yaw angle (rad)
ψ_{ref}	reference yaw angle (rad)
$\dot{\psi}$	yaw rate (rad/s)
$\dot{\psi}_{\text{ref}}$	reference yaw rate (rad/s)
$\dot{\psi}_{\max}$	maximum reference yaw rate (rad/s)
τ	longitudinal acceleration tracking time constant (s)
$\vec{F}_U(X,Y)$	induced force (N)

Chapter 1

Introduction

The race to develop autonomous vehicles, particularly in high-speed competitions such as autonomous racing, has gained significant attention in recent years. This surge is driven not only by the interest in self-driving technology but also by the potential to explore and test new approaches to navigation, trajectory optimization, and safety in extreme conditions, such as those found on racing tracks [3]. Autonomous racing has become an ideal testbed for evaluating the robustness and responsiveness of vehicle control algorithms [3].

As autonomous vehicles leave the research laboratory and enter public roads, they must react to emergency scenarios, some of which may necessitate maneuvering up to the handling limits of the vehicle in order to avoid a collision [4]. This is why one of the main research areas in autonomous vehicle racing involves developing collision avoidance algorithms, which are crucial for ensuring both safety and performance, especially when vehicles must respond rapidly to obstacles and other moving objects. In these scenarios, the car must be able to predict and avoid collisions in real time while optimizing its path and maintaining maximum efficiency. Various techniques can address this challenge, showing great potential by predicting future system behavior and optimizing control decisions.

In this research, several methodologies for collision avoidance in autonomous vehicle racing were explored, with particular focus on a hybrid approach that combines the APF technique with a MPC framework. The APF method is responsible for generating a collision-free reference trajectory, while the MPC tracks this trajectory by considering the dynamic constraints of the vehicle. The proposed system was implemented using MATLAB/Simulink and tested in simulation across multiple scenarios, including a representative racing circuit based on the laboratory track. This setup allowed for a realistic and controlled validation of the system's performance.

The goal of this work is to develop and implement collision avoidance algorithms for autonomous racing vehicles, using suitable techniques to generate safe trajectories in dynamic and competitive environments. Solving this problem is not only critical for reducing accidents but also for enhancing overall vehicle performance and safety, enabling smoother and more efficient driving even in unpredictable scenarios. By focusing on the integration of planning and control methods within a simulation framework, the project aims to contribute toward safer and more

efficient autonomous navigation, not just in high-speed racing, but also in broader autonomous driving contexts.

Chapter 2 of this dissertation presents a comprehensive review of the state of the art in autonomous vehicle racing, highlighting key developments in perception, planning, and control strategies, with a particular focus on collision avoidance methods such as APF, MPC, and Reinforcement Learning (RL). This chapter serves to contextualize the current research within the broader landscape of autonomous driving technologies. It also introduces the laboratory environment that served as the basis for simulation testing, describing the physical race track and the autonomous vehicle prototypes.

Chapter 3 focuses on the development of the APF-based path planner, explaining its implementation in MATLAB and detailing how the method was used to generate collision-free reference trajectories suitable for autonomous racing scenarios. The chapter also discusses how the APF's parameters were tuned to ensure effective obstacle avoidance and track following.

Chapter 4 turns to the control strategy, presenting an introduction to MPC, as well as the design and implementation of several MPC-based controllers using MATLAB/Simulink. Three different controller configurations were explored in a progressive manner, each contributing to a deeper understanding of MPC-based control. This step-by-step implementation played a crucial role in helping to understand the final Adaptive MPC controller, which was ultimately used to track the reference trajectories.

Chapter 5 brings together the planning and control components of the system, showcasing simulation results from a range of test scenarios, including a circuit inspired by the laboratory track. The performance of the integrated system is evaluated using key metrics such as the lateral tracking error percentage.

Finally, Chapter 6 concludes the dissertation by summarizing the main contributions of the work, reflecting on its capabilities and limitations, and outlining directions for future research.

Chapter 2

State-of-the-Art

In this chapter, we provide an overview of the current state of autonomous vehicle racing and its key components. Particular attention is given to collision avoidance techniques, including MPC, APF and RL. The chapter also highlights the hybridization of APF and MPC[1] to address their individual limitations, offering insight into their complementary roles in ensuring safe and efficient navigation in competitive racing scenarios. In addition, the laboratory facilities are described to provide context for the experimental setup.

2.1 Autonomous Vehicle Racing - Background

This section provides a fundamental understanding of autonomous racing and its key components, perception, planning, and control, that form the basis of this work, along with relevant collision avoidance techniques and background. Each subsection offers an overview of the core concepts and methods supporting autonomous racing, supported by relevant literature.

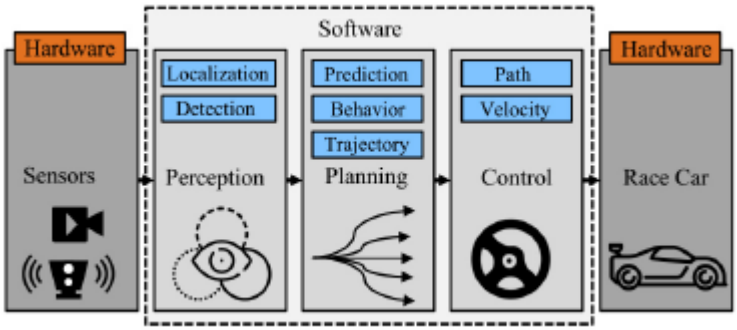


Figure 2.1: Autonomous racing pipeline showing perception, planning, and control integration for real-time decision-making[3].

[1] Referred to as Multiconstrained Model Predictive Control (MMPC) in [5], where the distinction is based on the number of constraints applied. In this work, we uniformly refer to it as MPC.

Autonomous racing is an emerging field that focuses on developing autonomous vehicles capable of competing in high-speed, dynamic environments without human intervention. These vehicles are equipped with a range of sensors and microprocessors to make real-time decisions at extreme speeds, relying entirely on onboard algorithms to detect obstacles, calculate optimal paths, and control vehicle dynamics. Autonomous race cars generally have four wheels and are powered by either an electric or combustion engine [3]. They must operate within strict parameters, including minimal reaction times and optimized racing strategies, to avoid collisions and complete a race as quickly as possible.

2.1.1 Perception

Perception in autonomous racing refers to the ability of the vehicle to interpret its surroundings through sensors that gather data on objects, other vehicles, and the track environment. Perception algorithms process this data to detect objects, estimate distances, and identify the position of the car relative to the track and competitors. In autonomous racing, perception is especially challenging due to the high speeds and rapid response requirements. Techniques often involve object detection, localization, and Simultaneous Localization And Mapping (SLAM) approaches, which are tailored for high-speed applications in dynamic racing environments [3]. By processing this information quickly, the vehicle can adapt to the constantly changing environment on the track, a crucial capability for both performance and safety in racing scenarios.

2.1.2 Planning

Planning in autonomous racing involves determining the trajectory of the vehicle and the strategy to optimize its path around the track while avoiding obstacles and minimizing lap time. This process includes both global and local planning. Global planning focuses on finding the fastest possible route around the entire track, often by setting up an Optimal Control Problem (OCP) to determine the optimal trajectory. Local planning, on the other hand, dynamically adjusts the path based on nearby objects or vehicles. Common planning strategies in autonomous racing utilize optimization methods, such as generating trajectories that minimize time, and various MPC-based methodologies that adapt the trajectory of the vehicle in response to changing conditions on the track. Although it is primarily a control method, MPC is often used in planning due to its adaptability to dynamic conditions and its effectiveness in local trajectory planning [3]. Planning is essential for competitive autonomous racing, as it enables the vehicle to maneuver efficiently, avoid obstacles, and make real-time adjustments based on current and anticipated conditions.

2.1.3 Control

Control is the function that computes specific actions, such as steering, braking, and acceleration, according to the planned trajectory. The control systems in autonomous racing must operate at high frequencies to ensure that the vehicle follows the planned path accurately, especially during high-speed maneuvers. Control techniques range from classic Proportional-Integral-Derivative

(PID) control to advanced MPC and RL approaches. In autonomous racing, control is particularly challenging due to the need for precise and stable handling at high speeds and the requirement to stay within the dynamic limits of the vehicle [3]. Effective control is critical to maintain vehicle stability, minimize errors, and ensure that race car performance is optimized under constantly changing conditions.

2.2 Collision Avoidance - State-of-the-Art

2.2.1 Problem Description

Collision avoidance in autonomous racing presents unique challenges compared to conventional autonomous driving. At high speeds, vehicles must react quickly to dynamic obstacles, track boundaries, and competitors while maintaining optimal performance. Unlike structured road environments, racing requires precise trajectory adjustments within milliseconds to prevent crashes without compromising speed.

In this work, perception is performed using an external camera, which serves as a sensor and introduces challenges such as potential delays in data acquisition and processing. The system must process this information in real-time to predict obstacles and generate a collision-free trajectory while ensuring minimal deviation from the optimal racing line.

2.2.2 Overview

The architecture of a collision avoidance system for autonomous racing integrates perception, planning, and control modules that work together in real-time. Figure 2.2 illustrates this architecture, where the perception module detects track boundaries, obstacles, and competitors. The planning module, represented by the path planning block, generates safe trajectories in response to the detected environment. Finally, the control module, illustrated as the path tracking block, executes the necessary adjustments, such as steering and acceleration, to ensure that the vehicle follows the planned trajectory and maintains a collision-free path.

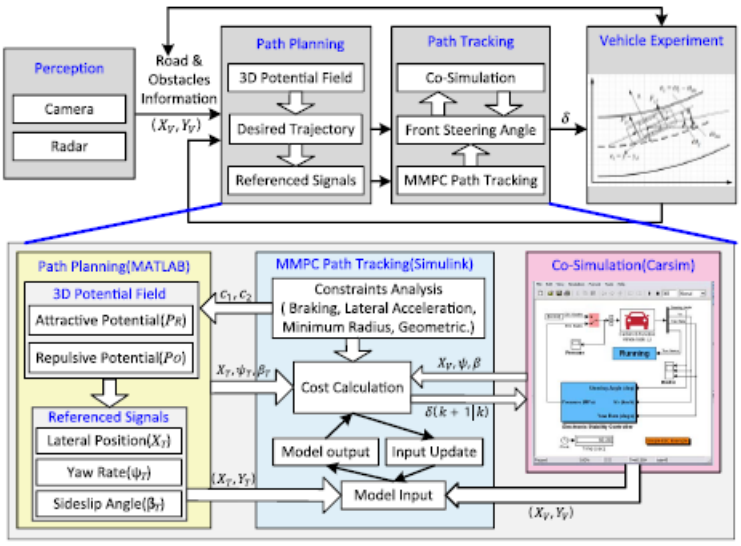


Figure 2.2: Autonomous racing pipeline showing perception, planning, and control integration for real-time decision-making [5].

Building on this foundation, the following subsection dives into the specific algorithms used to achieve collision avoidance in high-speed racing environments.

2.2.3 Algorithms

In autonomous vehicle racing, effective collision avoidance is crucial for both safety and performance. Racing vehicles must maintain high speeds while dynamically adjusting their paths in real-time to avoid collisions with track boundaries, obstacles, and other competing vehicles. This requires an approach that can respond to immediate environmental changes while preserving optimal racing performance. The motivation for developing advanced collision avoidance algorithms is to improve vehicle safety without compromising speed or maneuverability, particularly given the high-speed environment, which demands that calculations be efficient enough to avoid accidents within milliseconds [5].

Among various collision avoidance strategies, MPC stands out due to its ability to handle both planning and control within a predictive framework. Model predictive controllers continuously evaluate the position of the vehicle, anticipated path, and proximity to obstacles, calculating optimal control actions, such as steering and acceleration, based on these predictions. Mathematically, the algorithm continuously minimizes a predefined cost function, balancing factors such as tracking error (difference between the desired and predicted output) and the control effort required for steering and acceleration [6]. This balance ensures efficient trajectory planning and system responsiveness.

For autonomous racing, the ability of this algorithm to anticipate and adjust in real-time is indispensable, as it formulates collision avoidance as a constrained optimization problem, maintaining a collision-free path through fast, calculated adjustments. This predictive feature allows

MPC to optimize control actions, that is, handle system constraints, within a short time horizon, ensuring stability and responsiveness even during high-speed maneuvers [7]. Additionally, this method can be applied to solve problems with multiple constraints, including vehicle dynamics, track boundaries, and obstacle proximity, making it highly suitable for the demands of competitive racing environments. Essentially, MPC solves a rolling optimization problem by incorporating vehicle dynamics and environmental constraints into its calculations. While highly efficient, further advances are needed to improve its real-time performance [5, 6].

While MPC is commonly employed in autonomous racing, additional techniques can complement it. For example, the APF method may be used to create repulsive forces around obstacles, guiding the vehicle away from potential collisions. At the same time, it creates an attractive force to the goal point. However, APF approaches often encounter issues with local minima, which can prevent the vehicle from arriving at its required destination. Still, the APF method is a valuable tool for path planning in uncertain and dynamic environments [6].

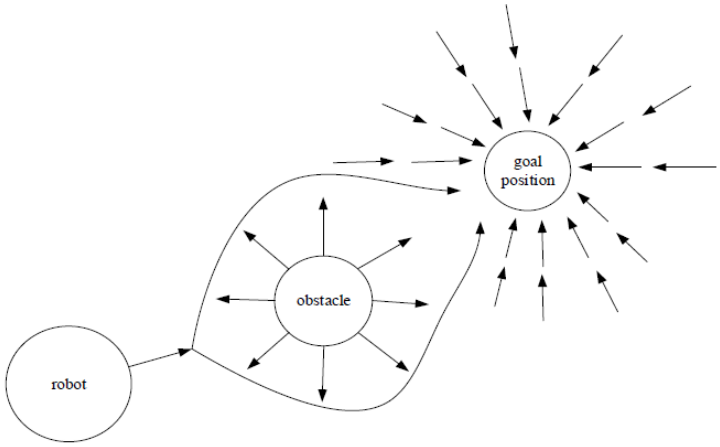


Figure 2.3: The concept of the APF method [6].

RL offers another promising approach for adaptive collision avoidance. As described in reference [8], this technique is used in parallel with a pure pursuit path-following algorithm. Pure pursuit initially generates a path based on the desired trajectory, while the RL agent refines this path in real-time by making adjustments to the path-following commands. The RL model receives the current velocity, steering commands, and laser range readings as inputs, allowing it to optimize for both speed and obstacle avoidance by adjusting steering and acceleration in response to dynamic obstacles.

In this setup, reward signal design plays a crucial role in guiding the RL behavior of the agent. Three types of reward signals are commonly used:

- Position-based rewards.
- Velocity-based rewards.
- Action-based rewards.

One effective RL reward approach in autonomous racing is the Cross-Track and Heading (CTH) reward, which incentivizes speed along the track center line while penalizing lateral deviations. This reward is based on the alignment of the vehicle with the track, it rewards the velocity in the desired direction and penalizes deviations in position and heading. By referencing either the track center line or a minimum curvature path, the CTH reward encourages a balanced trajectory, helping the vehicle to stay aligned while maintaining optimal speed [8].

By integrating RL as an adaptive modification layer to pure pursuit path-following, this approach enables racing vehicles to navigate complex tracks and respond to unexpected obstacles, enhancing both performance and safety in high-speed autonomous racing. One downside of this method is that it requires a large number of training samples as it is a machine learning method [6].

The methodologies reviewed in this section form the foundation for exploring practical implementations in autonomous vehicle racing. Now, we dive deeper into these methods by applying them to specific case studies and discussing their integration into a comprehensive hybrid approach.

2.2.3.1 APF and MPC: Hybrid Approach for Collision Avoidance

The combination of APF and MPC creates a practical solution for collision avoidance, bringing together the advantages of both methods to address the challenges of high-speed autonomous racing. This hybrid system is particularly valuable as it takes advantage of the computational efficiency of APF for rapid path generation and the complete tracking capabilities of MPC to ensure stability and dynamic feasibility.

APF is excellent in generating collision-free trajectories in real-time by imposing attractive and repulsive potential fields. The attractive potential aligns the vehicle with its destination or the road centerline, while the repulsive potential prevents collisions by steering the vehicle away from obstacles. However, APF methods are inherently susceptible to local minima, a situation where the car can become trapped between conflicting potential forces, unable to progress toward its goal. This limitation is handled in the hybrid system by incorporating MPC, which dynamically optimizes the trajectory of the vehicle, resolving local minima issues and adapting to environmental changes.

Figure 2.4 illustrates the universal potential field, showing how the attractive potential, pulling the vehicle toward the centerline, combines with the repulsive potentials from the road boundaries and an obstacle to produce a collision-free trajectory.

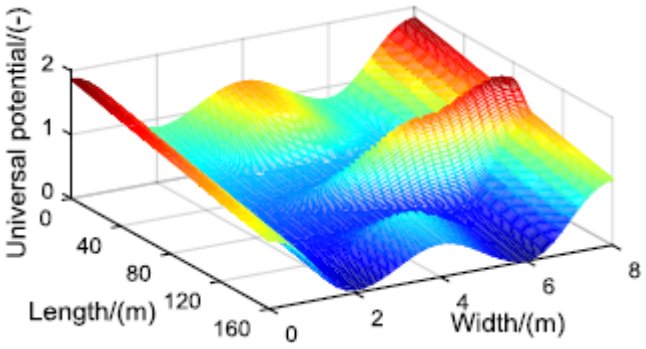


Figure 2.4: Universal potential field of road and obstacle [5].

As we can observe in Figure 2.2 the hybrid architecture integrates path planning, represented by the generation of collision-free trajectories using the 3D potential field approach, and path tracking, executed by MPC. In this setup, the path planning block calculates a desired trajectory based on attractive and repulsive potentials, producing referenced signals such as lateral position, yaw rate, and sideslip angle. These signals serve as inputs for the path tracking module, where MPC analyzes constraints, including braking, lateral acceleration, and minimum turning radius, to ensure precise vehicle control. The integration of APF and MPC was tested in numeric simulations detailed in [5]. These simulations were conducted using vehicle simulation software, namely CarSim and MATLAB/Simulink.

The study in [5] demonstrates the capability of using APF for local trajectory planning, noting that the approach can modify collision-free paths in real-time. This, combined with the fact that potential function approaches have already seen substantial use for robot motion planning, makes APF particularly suitable for scenarios requiring quick decisions, such as obstacle avoidance during high-speed racing and justifies its considerable use for path planning of autonomous vehicles [9]. To simplify the derivation of trajectories using the APF method, certain assumptions are employed [5, 10]:

- The road is straight and has boundaries on both sides.
- Only a single obstacle appears in front of the host vehicle.
- The obstacle moves consistently along the road.

These assumptions clarify the computational requirements and make the method feasible for dynamic racing environments. Under these conditions, APF efficiently identifies a safe path for the vehicle. However, when deviations from these assumptions occur, such as in curved roads or multi-obstacle scenarios, MPC becomes crucial.

MPC complements APF by transforming the initial trajectory into a dynamically feasible path that adheres to both physical and control constraints. The predictive capabilities of MPC ensure that the vehicle remains stable and responsive, even in the face of high-speed maneuvers or abrupt environmental changes. For instance, model predictive controllers consider constraints on steering

angles, yaw rates, and sideslip angles [5], thereby preventing instabilities caused by actuator saturation or extreme maneuvers. By continuously optimizing the trajectory, MPC ensures a balance between maintaining the planned path and dynamically reacting to obstacles or road conditions, while ensuring the stability required to maintain control over longer periods, creating a balanced solution for collision avoidance.

Figure 2.5 illustrates the trajectories of two controllers under challenging conditions. Controller A (red line), an MPC implementation solving an optimization problem with only a single input constraint on the front steering angle[2], fails to maintain stability when the planned path includes sharp turns or sudden deviations, causing the vehicle to leave the track completely. In contrast, controller B (blue line), also based on MPC but solving an optimization problem with multiple constraints, including an input constraint on the front steering angle and state constraints on lateral tracking error, yaw rate, and sideslip angle[2], successfully keeps the vehicle within safe dynamic limits.

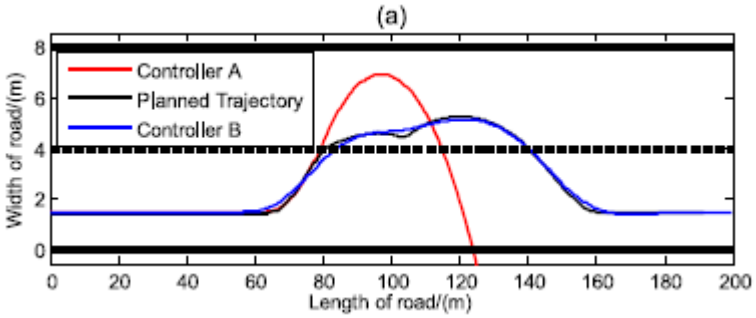


Figure 2.5: Vehicle trajectories of controllers A and B and the planned trajectory [5].

The effectiveness of the more advanced MPC approach, that considers multiple constraints, becomes even more evident when compared to simpler implementations. Simulations in [5] compare an MPC implementation solving an optimization problem with multiple constraints (controller B) with a simpler MPC implementation solving a similar problem but with only a single input constraint (controller A). While controller A struggled to maintain stability when the vehicle deviated significantly from the planned path, controller B successfully constrained the vehicle within safe dynamic limits. This resulted in smoother control inputs, reduced tracking errors, and more precise trajectory tracking.

Figure 2.6 also illustrates the stability differences between these controllers, showing how controller B (blue line) avoids excessive yaw rates and sideslip angles, ensuring the vehicle remains stable during aggressive maneuvers, while controller A (red line) experiences significant instability, with excessive yaw rates and sideslip angles causing the vehicle to lose control.

[2]These vehicle dynamic variables are formally introduced later in the report, within the vehicle model sections.

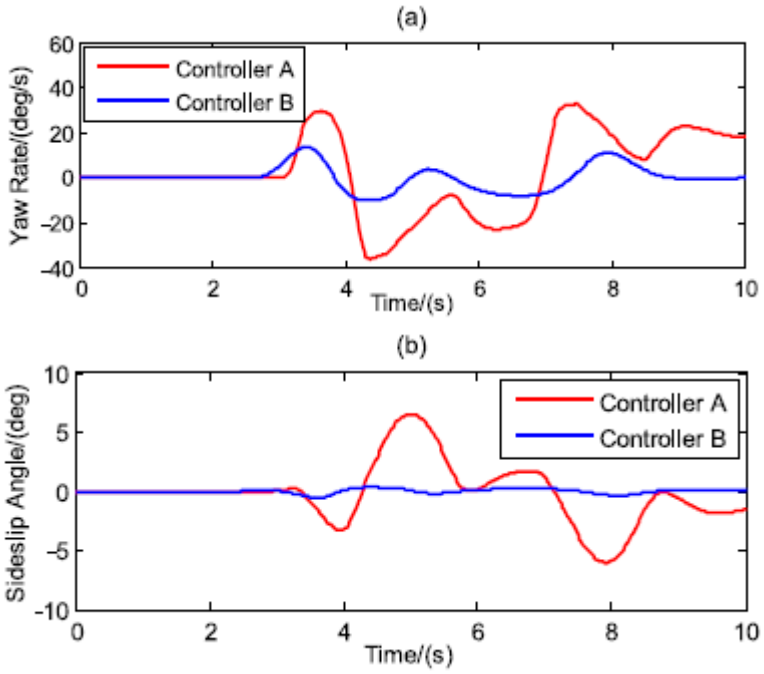


Figure 2.6: Stability performance of controllers A and B [5].

In conclusion, while MPC addresses the local minima problem inherent to APF, its role in the hybrid system goes beyond that. It ensures the dynamic feasibility of trajectories, enhances stability under constraints, and optimizes control inputs, making it an indispensable component of this advanced collision avoidance framework. The complementary nature of APF and MPC offers a powerful solution for navigating the complexities of autonomous vehicle racing.

2.2.3.2 RL: Reward Signal Design for Autonomous Racing

A key component of the RL approach for autonomous vehicle racing is reward signal design, which has a direct influence on how an agent prioritizes actions like maximizing speed, maintaining safety, and avoiding obstacles. The study on reward signal design for autonomous racing explores different reward frameworks to balance these competing objectives, contributing valuable insights to high-speed environments like autonomous racing [8], such as:

- Position-based rewards: These measure vehicle progress along a predefined trajectory, such as the centerline of the race track or a minimum curvature path. This encourages forward motion by providing incremental rewards for reaching closer to the goal. The Figure 2.7 shows how the position of the vehicle is projected onto the reference line to calculate progress.

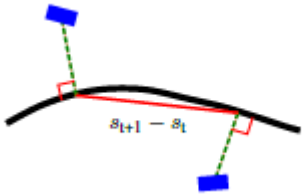


Figure 2.7: Vehicle position projected onto a reference line to calculate progress [8].

- Velocity-based rewards: These focus on optimizing the speed and alignment of the vehicle with the reference line. By considering directional movement, this method optimizes performance over time while penalizing lateral deviations. The reward is determined by calculating heading errors and cross-track distance.

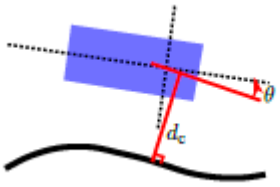


Figure 2.8: Cross-track distance and heading error used in velocity-based rewards [8].

- Action-based rewards: These aim to reduce steering effort, to promote smoother trajectories. They use a minimum curvature trajectory as the reference, aligning well with the dynamics of high-speed racing. This increases overall efficiency and reduces harsh turns.

A key contribution of [8] is the use of minimum curvature trajectories as a reward reference. Traditional centerline-based methods often lack precision in high-speed scenarios, particularly during sharp turns. By contrast, it is expected that the minimum curvature approach results in faster lap times and improved adherence to the path [8]. This innovation is particularly relevant for environments demanding dynamic responses to obstacles and rapid decision-making which is the case we want to reproduce.

Simulation tests conducted in F1/10th scale racing validate the proposed methodologies. For example, the quickest lap times were generated by the CTH reward signal, while successfully balancing speed and obstacle avoidance. A typical trajectory using this reward signal is shown in Figure 2.9, demonstrating its ability to maintain path adherence and dynamically avoid obstacles. The grey area is the track, the green line is the reference and the red line is the path taken [8].

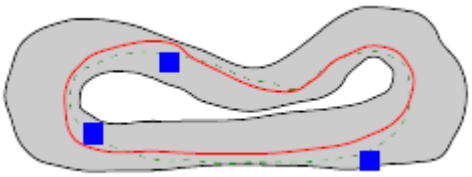


Figure 2.9: Trajectory using the CTH reward framework with dynamic obstacle avoidance [8].

The findings highlight the advantages of combining velocity considerations with dynamically optimized reference trajectories, particularly in demanding racing environments. While simpler approaches, such as steering-only rewards, improve basic performance, they fall short in complex scenarios that require adaptability and precision [8].

In summary, this study provides a robust framework for designing reward signals that enhance the performance of autonomous vehicles in high-speed racing. By using innovative methodologies such as the minimum curvature trajectory, it establishes the groundwork for future advancements in collision avoidance and autonomous racing strategies.

2.3 Laboratory Facilities

The following section presents the laboratory facilities that form the foundation of this work. Understanding the conditions and components available in the lab environment is essential, as they define the context in which the system is designed and developed. The racetrack and autonomous race cars described here represent the basis for the experimental setup and are central to the structure and goals of this project.

2.3.1 Race track

The race track provides a controlled environment for developing and testing collision avoidance algorithms in autonomous racing scenarios. Its design ensures a balance between realistic challenges and consistency, allowing algorithms to handle various situations such as sharp turns, sudden changes in trajectory, and boundary constraints. The track was constructed with stability and precision as primary objectives, ensuring reliable performance during testing and data collection.

The base of the track comprises four tongue and groove plywood boards reinforced with wooden battens to create a sturdy surface capable of withstanding repeated use. The racing surface is composed of urethane pieces arranged in a 7x8 grid, providing adequate traction for the vehicles while simulating real-world conditions. The outer boundaries are enclosed with solid walls to prevent cars from leaving the track, and the inner boundaries are marked with tape, allowing for easy reconfiguration of the layout. This modular approach enhances flexibility in testing different track designs [11].

The final configuration of the race track includes five sharp curves that challenge the maneuverability of the vehicle, requiring precise steering and velocity control. This setup creates a

comprehensive platform for evaluating collision avoidance strategies in high-speed environments [11]. The completed track is illustrated in Figure 2.10.

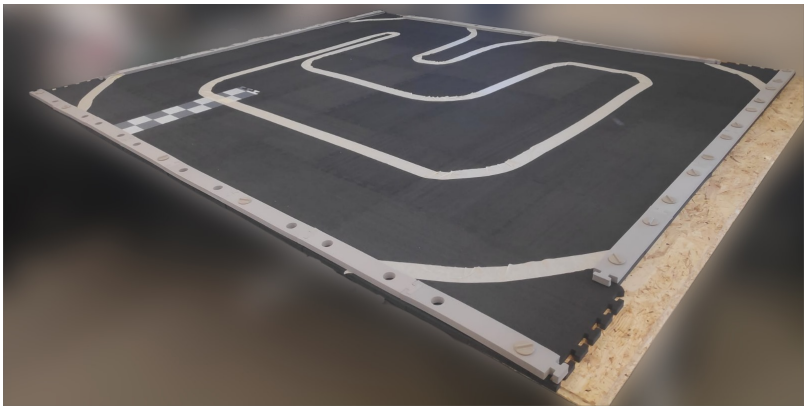


Figure 2.10: Completed race track with defined boundaries [11].

By incorporating realistic obstacles and boundary conditions, the race track provides an effective testing ground for assessing the performance of collision avoidance algorithms under controlled yet dynamic scenarios.

2.3.2 Race car

The race car used in this project serves as the experimental platform for validating collision avoidance algorithms in autonomous racing. Scaled at 1:43, the Radio-Controlled (RC) cars are equipped with essential hardware to ensure precise motion control and real-time responsiveness during tests.

Each car is equiped with an ESP32 S2 Mini microprocessor and a custom-designed Printed Circuit Board (PCB). This setup enables efficient communication with a central computer, which processes sensor data and computes control commands [11]. The motors and steering mechanisms of the car are controlled precisely to execute planned trajectories while maintaining stability during sharp turns and high-speed maneuvers.

The design includes lightweight materials for agility and components optimized for racing performance. The front and rear wheels are of varying diameters, and the robust axle structure ensures durability during repeated testing. To facilitate visual tracking and identification during experiments, each car is uniquely colored, like for example: yellow, purple, and blue, as we can observe in Figure 2.11 [11].



Figure 2.11: Three race cars uniquely colored for tracking [11].

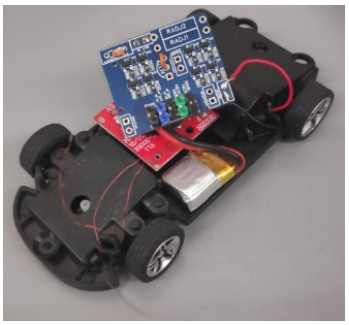


Figure 2.12: Race car with PCB and ESP32 installed [11].

These cars serve as the foundation for evaluating collision avoidance algorithms in real conditions. By providing a consistent platform for experiments, they ensure the reproducibility of results and allow detailed analysis of the effectiveness of algorithms in navigating complex tracks and avoiding obstacles under high-speed conditions.

Wrapping up chapter 2, the complementary nature of APF and MPC offers a robust solution for navigating the complexities of autonomous vehicle racing and that is the reason why we select this hybrid technique to use in this dissertation. The decision to use this hybrid technique in this dissertation is based in its demonstrated ability to combine computational efficiency with stability and dynamic feasibility, making it suitable to the demands of high-speed environments, particularly those defined by the laboratory setup.

Chapter 3

Collision Avoidance

This chapter describes the implementation of the APF method for collision avoidance path planning in MATLAB. The approach is based on the methodology presented in [5]. The APF method creates a virtual force field that influences the movement of the vehicle by attracting it toward the center of the lane while repelling it from obstacles.

Using this method, it is possible to generate complete reference trajectories for arbitrary track layouts, enabling the creation of safe, collision-free paths adaptable to various environments and scenarios.

3.1 Potential Field Equations

The potential field is composed of two key components: an attractive potential field, which guides the vehicle toward the desired path, and a repulsive potential field, which pushes it away from obstacles to avoid collisions. The universal potential field, $P_U(X,Y)$, is obtained by summing these two contributions, where $P_R(X,Y)$ represents the attractive potential of the road, and $P_O(X,Y)$ corresponds to the repulsive potential generated by obstacles.

The potential field plots presented in this section are generated based on a straight segment of the racing track used in our laboratory setup, providing a concrete basis for illustrating the behavior of the potential fields.

3.1.1 Attractive Potential Field

The attractive potential field is designed to keep the vehicle within the road limits while encouraging forward motion.

$$P_R(X,Y) = A(X) \cdot A(Y) + P_A(X,Y) \tag{3.1}$$

Here $A(X)$ represents the amplitude of the potential field along the road width and $A(Y)$ represents the amplitude along the road length. Additionally, $P_A(X,Y)$ introduces a forward motion

potential that encourages the vehicle to move forward and leads the vehicle toward the centerline of the left lane when it has passed the obstacles ahead.

The function $A(X)$ is given by:

$$A(X) = \frac{(|\text{sign}(X - X_l) + \text{sign}(X - X_r)|\,(1 - P_m) + 2P_m)}{4} \cdot \left(\cos\left(\frac{2\pi X}{L_w}\right) + 1\right) \tag{3.2}$$

where X_l and X_r represent the centerlines of the left and right lanes, respectively. The term P_m indicates the amplitude of potential of the middle of the road, while L_w denotes the lane width.

Similarly, the function $A(Y)$ introduces a scaling factor that adjusts the potential based on the vehicle's distance along the road. It is defined as:

$$A(Y) = \begin{cases} 0, & |Y - Y_o| \le D_b \\ \frac{|Y - Y_o| - D_b}{D_t - D_b}, & D_b < |Y - Y_o| \le D_t \\ 1, & |Y - Y_o| > D_t \end{cases} \tag{3.3}$$

where Y_o is the position of the obstacle in the Y-direction of the fixed earth axis system. The parameters D_t and D_b define the scope of the repulsive potential field in the Y-direction and the shortest braking distance, respectively.

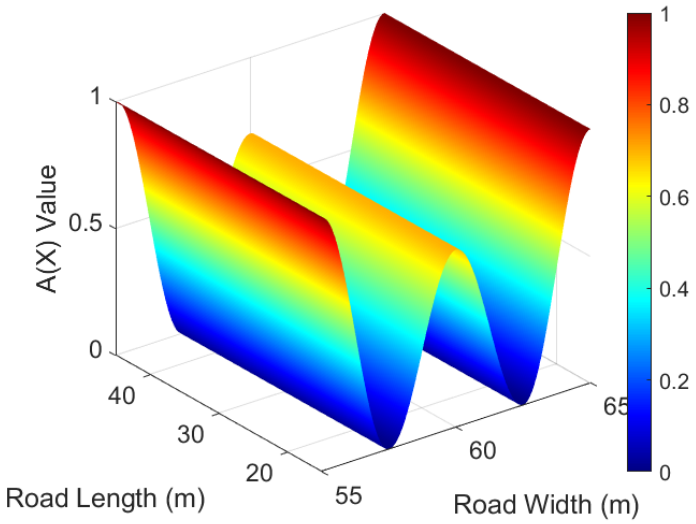


Figure 3.1: $A(X)$ Potential Variation.

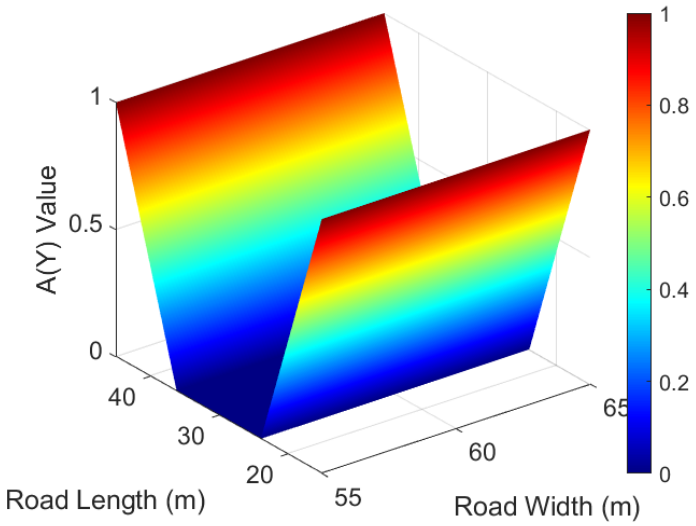


Figure 3.2: $A(Y)$ Potential Variation.

To encourage forward motion, a small potential gradient, $P_A(X,Y)$, is introduced[1]:

$$P_A(X,Y) = \frac{|X - X_l| + |Y - (Y_o + 0.5D_s)|}{100} \tag{3.4}$$

where D_s represents the scope of the universal potential field in the Y-direction.

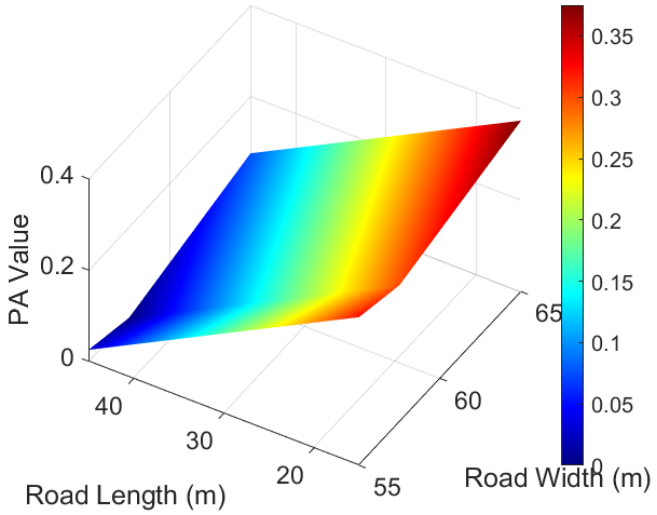


Figure 3.3: Forward Motion Potential $P_A(X,Y)$.

[1]The term X_l appears in the equation in that specific position to encourage the vehicle to remain in the center of the left lane whenever possible. While the user could alternatively choose X_r, since the APF formulation assumes the road consists of two lanes, the decision to use X_l is motivated by the assumption that the vehicle always moves in a counterclockwise direction, as will be seen later in the report. This makes the left lane the inner lane, resulting in shorter and more efficient trajectories.

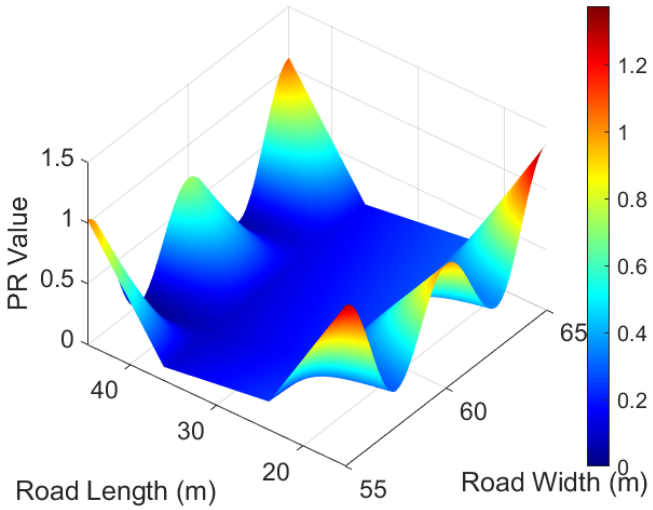


Figure 3.4: Attractive Potential Field P_R.

3.1.2 Repulsive Potential Field

The repulsive potential field ensures that the vehicle avoids obstacles by generating a repelling force. It is modeled as follows:

$$P_O(X,Y) = \frac{|e^{-c_1(X-X_o)^2-c_2(Y-Y_o)^2} - P_t|}{1-P_t} \tag{3.5}$$

where X_o and Y_o, define the position of the obstacle, and P_t represents a threshold potential value that determines the scope of the repulsive potential field, i.e., the minimum potential level below which the repulsive effect is considered negligible.

The parameters c_1 and c_2 define the lateral and longitudinal weights of the repulsion, respectively, and are given by:

$$c_1 = \begin{cases} \frac{-\log(P_t)-c_2(Y-Y_o)^2}{(0.5L_w(\sin(\frac{(Y-Y_o+D_b)}{D_b}\pi-\frac{\pi}{2})+1))^2}, & |Y-Y_o| \le D_b \\ 0, & |Y-Y_o| > D_b \end{cases} \tag{3.6}$$

$$c_2 = \frac{-\ln(P_t)}{D_b^2} \tag{3.7}$$

where D_b is the shortest braking distance, a critical parameter that defines the area in which the repulsive forces act. It is calculated as:

$$D_b = \frac{m(V_x^2 - v_{\text{obs}}^2)}{4F_m} + \frac{L_l}{2} \tag{3.8}$$

where m is the mass of the vehicle, V_x and v_{obs} are the longitudinal velocities of the vehicle and the obstacle, respectively, F_m is the maximum braking force of each tire and L_l is the obstacle spatial influence.

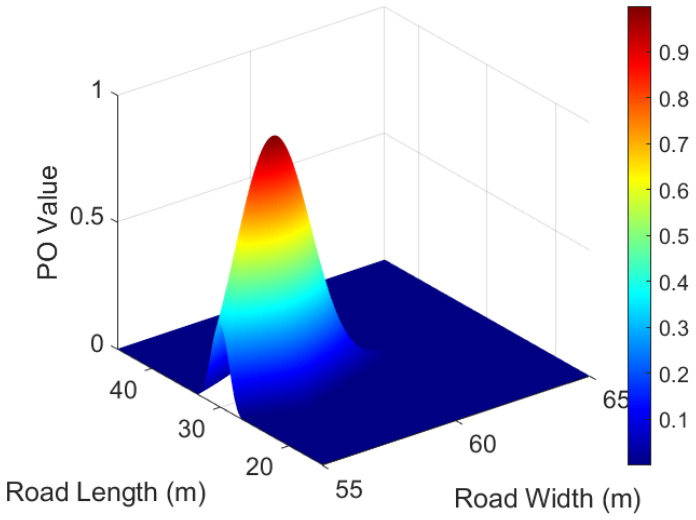


Figure 3.5: Repulsive Potential Field P_O.

3.1.3 Universal Potential Field

Finally, the universal potential field is obtained by summing the attractive and repulsive potentials:

$$P_U(X,Y) = P_R(X,Y) + P_O(X,Y) \tag{3.9}$$

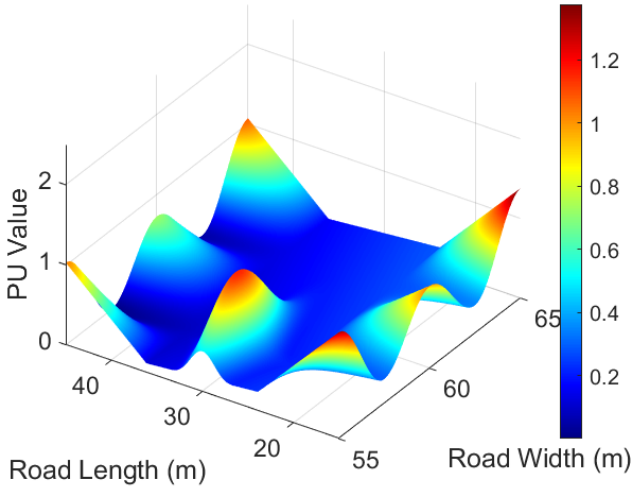


Figure 3.6: Universal Potential Field P_U.

3.2 Induced Force and Trajectory Generation

Having established the structure of the universal potential field, the next step involves computing the resulting induced force that guides the vehicle's motion and ultimately generates a reference trajectory for navigation.

3.2.1 Induced Force

From the universal potential field $P_U(X,Y)$ described previously, we derive an induced force that serves as a guidance vector for the vehicle at each point in space. The induced force $\vec{F}_U(X,Y)$ is calculated as the negative gradient of the universal potential field:

$$\vec{F}_U(X,Y) = -\nabla P_U(X,Y) = -\left[\frac{\partial P_U}{\partial X}, \frac{\partial P_U}{\partial Y}\right] \tag{3.10}$$

This force points in the direction of the steepest descent of the universal potential field, leading the vehicle towards the goal while steering it away from obstacles.

In practice, the partial derivatives of the potential field $P_U(X,Y)$ are approximated over a discrete grid using central finite differences. This yields the scalar components of the induced force field:

$$F_{U_X}(X,Y) = -\frac{\partial P_U}{\partial X}(X,Y) \tag{3.11}$$

$$F_{U_Y}(X,Y) = -\frac{\partial P_U}{\partial Y}(X,Y) \tag{3.12}$$

These expressions represent the negative gradient of the universal potential field in the X and Y directions, respectively. Since the induced force field is originally computed on a discrete grid, interpolation techniques are applied to estimate the force at arbitrary positions in the continuous domain, enabling smooth trajectory generation.

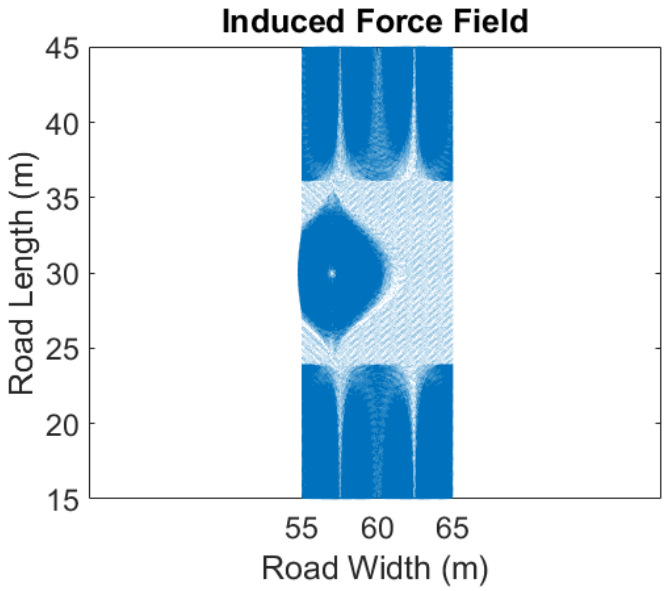
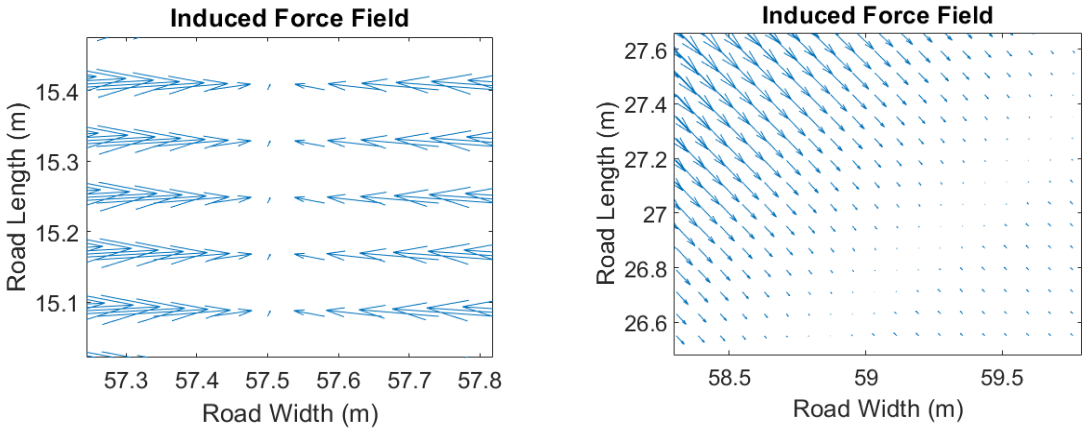


Figure 3.7: Induced force $\vec{F}_U(X,Y)$ across the road.



(a) Region illustrating the induced forces guiding the vehicle smoothly along the left lane.

(b) Region illustrating the induced forces guiding the vehicle away from the obstacle.

Figure 3.8: Zoomed-in views of the induced force field for better visualization of local vector behavior.

3.2.2 Trajectory Generation

Given an initial position of the vehicle $P_i = (X_{v,i},\ Y_{v,i})$ here considered $(x_1,\ y_1)$, a reference trajectory is generated by iteratively advancing the position using discrete-time integration in the direction of the induced force. At each iteration k, the position is updated using:

$$X_{v,\text{ref},k} = x_{k-1} + V_{scalar} \cdot \frac{F_{U_X}}{\|\vec{F}_U\|} \cdot \Delta t, \quad k = 2,3,\ldots,N \tag{3.13}$$

$$Y_{v,\text{ref},k} = y_{k-1} + V_{scalar} \cdot \frac{F_{U_Y}}{\|\vec{F}_U\|} \cdot \Delta t, \quad k = 2,3,\ldots,N \tag{3.14}$$

Several parameters are needed for the trajectory generation process. Δt is the integration time step, determining how frequently the vehicle's position is updated. V_{scalar} is a scalar reference velocity used to determine the distance advanced at each integration step, t_{total} is the total simulation time, and it is computed as the track length divided by the velocity, i.e., $t_{\text{total}} = R_l/V_{scalar} + \Delta t$, where the small offset ensures inclusion of the final point. The total number of integration steps is then given by $N = \text{round}(t_{\text{total}}/\Delta t)$.

At each time step, the induced force vector $\vec{F}_U = [F_{U_X}, F_{U_Y}]$ is evaluated at the current position of the vehicle using bilinear interpolation over the discretized universal potential field. This force vector is then normalized to obtain its direction, and the vehicle is moved at the constant speed V_{scalar} along this direction. However, in practice, to ensure consistent forward motion along the track, the update equation for the y coordinate was simplified to $y_k = y_{k-1} + V_{scalar} \cdot \Delta t$, decoupling it from the y component of the force. The x coordinate remains influenced by the normalized force direction, following 3.13. This adjustment, although a deviation from the standard APF formulation, was necessary to ensure that the vehicle consistently advanced along the track.

This approach provides a smooth and dynamically consistent reference path, which can be used as input for control algorithms such as MPC. From this trajectory, a reference signal is constructed for the controller, containing the relevant vehicle states over time:

$$\text{reference_planned_trajectory}(t) = \begin{bmatrix} t_{\text{ref}} & X_{v,\text{ref}}(t) & Y_{v,\text{ref}}(t) \end{bmatrix} \tag{3.15}$$

This basic reference trajectory can then be extended as needed, depending on the vehicle model used. For example, it is possible to calculate the path curvature, which in turn allows the estimation of other variables such as the yaw angle, yaw rate, and even the body sideslip angle, depending on the kinematic or dynamic vehicle model considered. Therefore, the reference trajectory can be adapted to suit the requirements of the control framework.

3.3 MATLAB Implementation

3.3.1 General Code Logic

The core of the trajectory generation system is structured around a MATLAB script that interprets the data provided by a perception system to define and construct the road layout, which then serves as the basis for trajectory planning. The perception module provides key information such as the vehicle's relative position, the lateral and longitudinal boundaries of the road, the segment information and the location of any obstacles present in the environment. Using this data, the racing

track can be reconstructed within the Driving Scenario Designer (DSD) [12], an application available in MATLAB Automated Driving Toolbox [13], while the reference trajectory is computed based on this constructed environment.

It is important to note that the DSD operates with its own coordinate system, shown in Figure 3.9, where the X-axis points upward and the Y-axis to the left. This convention must be taken into account when calculating the reference trajectory, as explained in the following sections.



Figure 3.9: DSD coordinate system.

The track is composed of a sequence of segments, such as straights and curves, and the trajectory generation logic processes these segments iteratively, one by one. It is assumed that the vehicle moves in a counterclockwise direction throughout the entire track.

The interpretation of each segment depends on whether there are obstacles within that segment and on the position and orientation of the start and end center points of the road. These elements determine how the segment should be handled, whether it corresponds to a straight path with no obstacles, a straight that requires obstacle avoidance, or a curved section such as a 90° or 180° turn.

The code also includes checks to ensure smooth and continuous transitions between segments. This guarantees that the end center point of one segment aligns correctly with the start center point of the next, resulting in a physically consistent track and, as a result, a coherent reference trajectory.

To help understand how this logic works, the following flowchart in Figure 3.10 gives a simplified visual summary of how the script processes each segment and makes decisions for trajectory generation.

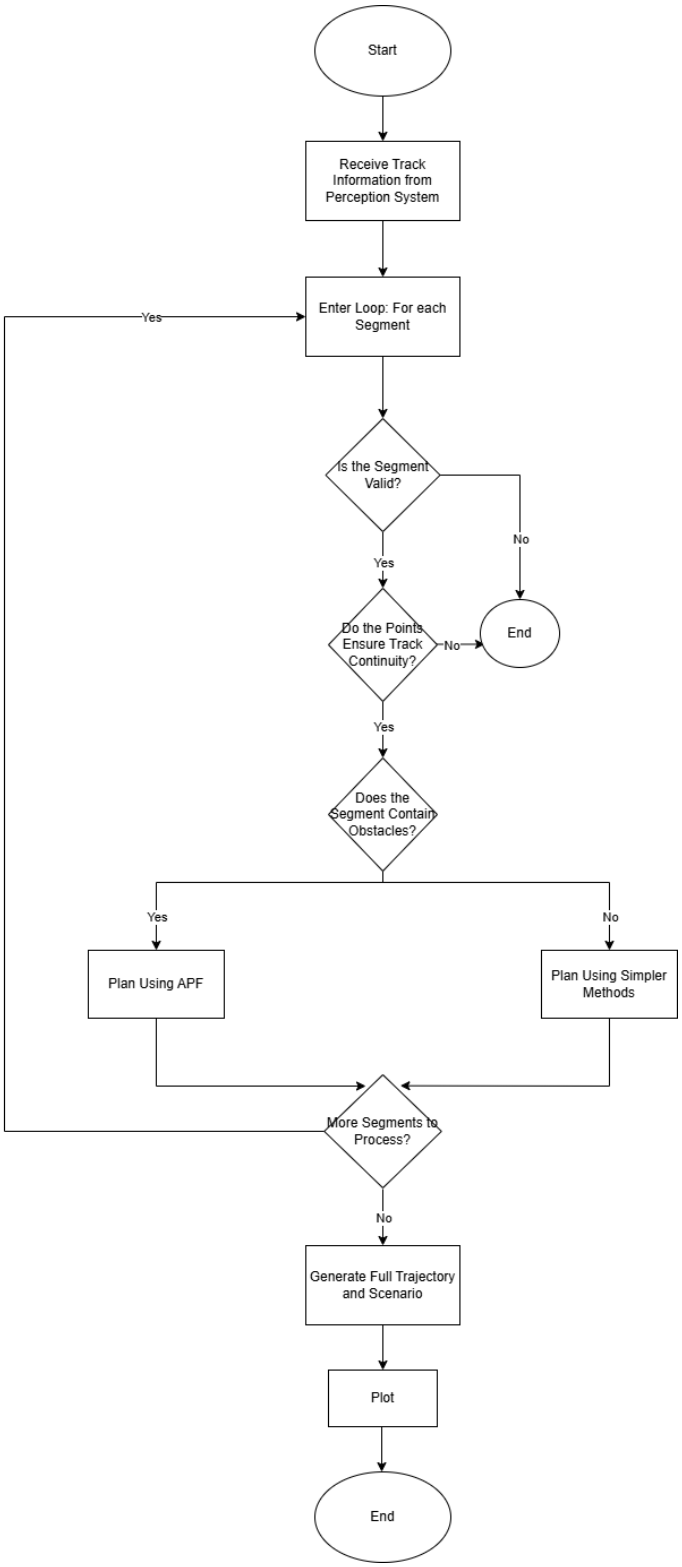


Figure 3.10: Flowchart with the logic behind the code.

3.3.2 Scaling Between Laboratory and Simulation

Given that the controller used in the control part of the problem, detailed in Chapter 4, was specifically designed and tuned for a vehicle model with a total length of 3 meters, calculated as $L_{\text{car}} = l_f + l_r$[2], it was necessary to adopt the same vehicle configuration during the planning phase to ensure consistency.

To maintain the same length to width ratio as the laboratory vehicle, this total length was used to compute a corresponding width of 1.25 meters for the simulated planning model.

As a result, the simulated track was dimensioned to preserve the same vehicle to road proportions as those found in the laboratory setup. In the lab, the autonomous vehicle prototype measured approximately 12 cm in length and 5 cm in width. To reflect the same relative spatial relationships, the simulated environment was scaled by a factor of approximately 25. This ensures that the behavior of the APF algorithm is evaluated under conditions that closely mirror the real-world experimental scenario available in the laboratory.

Figure 3.11 and 3.12 provide a visual comparison between the laboratory vehicle and the scaled simulated vehicle used in both the planning and control environments.

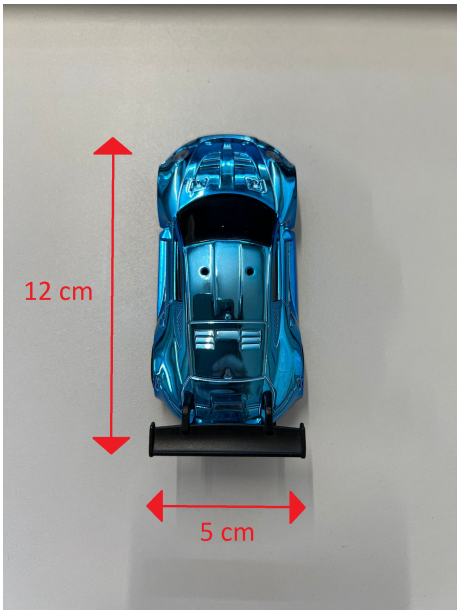


Figure 3.11: Laboratory car dimensions.

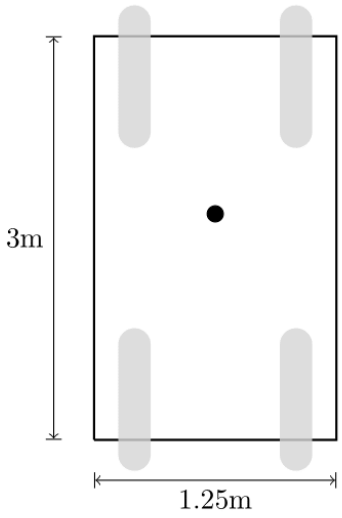


Figure 3.12: Car model dimensions.

3.3.3 Segment Types

To plan the trajectory of the vehicle along the road network, we first identify whether a segment contains obstacles or not. If obstacles are present, the segment is generated using APF to ensure safe and adaptable obstacle avoidance. For segments without obstacles, a simpler method is used

[2]Using $l_f = 1.4$ m and $l_r = 1.6$ m, which represent the distances from the vehicle's center of gravity to the front and rear axles, respectively.

depending on whether the segment is straight or curved. This separation allows us to balance safety and efficiency across different driving scenarios.

The road is assumed to consist of two lanes. In all cases, we force the vehicle to remain in the left lane, as dictated by the APF. This decision is motivated by the overall counterclockwise direction of the track, making the left lane the shorter and generally more efficient path.

Based on this logic, each segment is handled using one of the following strategies:

- **Segments with Obstacles:**
  Path planning is performed using APF to avoid obstacles.
- **Segments without Obstacles:**
  - **Straight Segments:**
    A straight-line trajectory is defined, aligned with the segment's direction and offset to keep the vehicle centered in the left lane.
  - **Curved Segments:**
    An elliptical arc is used to generate a smooth turning path while maintaining the vehicle in the left lane.

Each segment is handled differently due to the specific requirements of its geometry and dynamic constraints. This subsection details the computation logic and trajectory generation strategy used for each case.

It is important to note that the chosen longitudinal velocity of the vehicle of $V_{scalar} = 4.5$ m/s and time step $\Delta t = 0.01$ s, which will be introduced in the following subsubsections, are only used for the path planning phase, particularly for computing the reference trajectory through discrete integration. These values do not necessarily correspond to the actual velocity profile that the vehicle will follow, which will be later determined and adjusted by the controller. The selected parameters serve simply to generate a feasible and smooth path.

3.3.3.1 Segments with Obstacles

For straight segments containing obstacles, the trajectory is generated using the APF method, as described in Sections 3.1 and 3.2. This method constructs a universal potential field that guides the vehicle away from obstacles while maintaining progress along the segment.

Let us consider a straight road segment defined by its start and end center point $C_1 = (x_1, y_1)$ and $C_2 = (x_2, y_2)$, respectively, the position of an obstacle given by its center of mass $P_{\text{obs}} = (X_o, Y_o)$, and the road width, denoted by R_w:

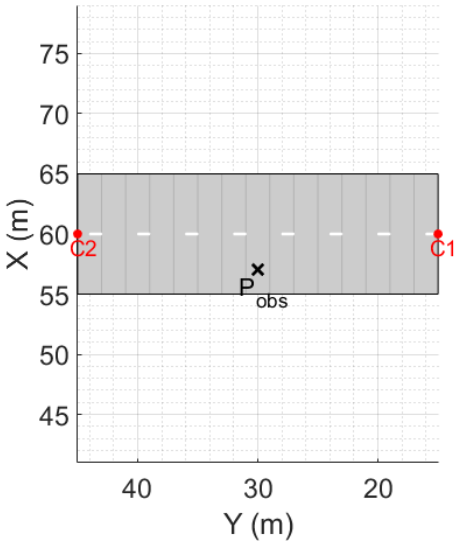


Figure 3.13: Straight segment with respective start and end center points C_1 and C_2, and obstacle position P_{obs}.

The APF method assumes a Cartesian coordinate system where the road always extends along the positive Y-axis (upward) and the lateral direction along the X-axis (rightward). This local coordinate system is intrinsic to the APF logic and does not match the DSD coordinate system shown in Figure 3.9.

Therefore, a transformation is required to reorient the road and obstacle positions into a local APF-aligned frame. This ensures consistent application of the APF method, regardless of the orientation of the road in the DSD map.

The illustrations below show how the APF axes are applied relative to different road segment directions in the DSD. In the Figure 3.14, the DSD and APF axes align naturally. In the Figure 3.15, a transformation is needed to match the APF coordinate system.

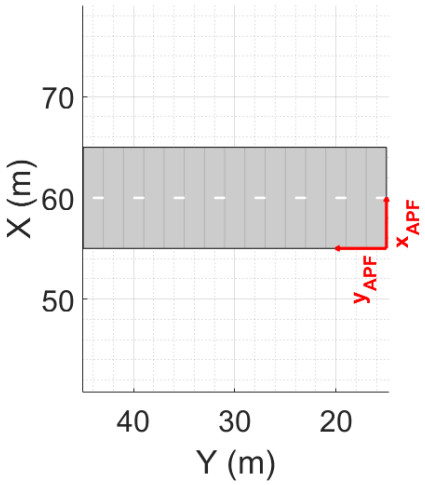


Figure 3.14: Segment with corresponding APF local coordinate axes.

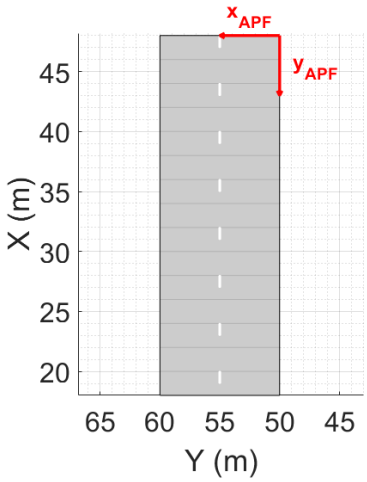


Figure 3.15: Segment with corresponding APF local coordinate axes.

With this established, based on the positions of the points C_1 and C_2, several parameters need to be adjusted to correctly calculate the reference trajectory with APF, for all road orientations possible. The minimum lateral coordinate of the road, $X_{\min}$, is calculated as:

$$X_{\min} = \begin{cases} x_1 - \frac{R_w}{2}, & \text{if } x_1 = x_2 \wedge y_2 > y_1 \quad \text{(horizontal left)} \\ y_1 - \frac{R_w}{2}, & \text{if } y_1 = y_2 \wedge x_1 > x_2 \quad \text{(vertical down)} \\ y_1 - \frac{R_w}{2}, & \text{if } y_1 = y_2 \wedge x_1 < x_2 \quad \text{(vertical up)} \\ x_1 - \frac{R_w}{2}, & \text{if } x_1 = x_2 \wedge y_1 > y_2 \quad \text{(horizontal right)} \end{cases} \tag{3.16}$$

The maximum lateral coordinate of the road, $X_{\max}$, is determined as:

$$X_{\max} = \begin{cases} x_1 + \frac{R_w}{2}, & \text{if } x_1 = x_2 \wedge y_2 > y_1 \\ y_2 + \frac{R_w}{2}, & \text{if } y_1 = y_2 \wedge x_1 > x_2 \\ y_2 + \frac{R_w}{2}, & \text{if } y_1 = y_2 \wedge x_1 < x_2 \\ x_1 + \frac{R_w}{2}, & \text{if } x_1 = x_2 \wedge y_1 > y_2 \end{cases} \tag{3.17}$$

The minimum longitudinal coordinate of the road, $Y_{\min}$, is computed as:

$$Y_{\min} = \begin{cases} y_1, & \text{if } x_1 = x_2 \wedge y_2 > y_1 \\ x_2, & \text{if } y_1 = y_2 \wedge x_1 > x_2 \\ x_1, & \text{if } y_1 = y_2 \wedge x_1 < x_2 \\ y_2, & \text{if } x_1 = x_2 \wedge y_1 > y_2 \end{cases} \tag{3.18}$$

The maximum longitudinal coordinate of the road, $Y_{\max}$, is calculated as:

$$Y_{\max} = \begin{cases} y_2, & \text{if } x_1 = x_2 \wedge y_2 > y_1 \\ x_1, & \text{if } y_1 = y_2 \wedge x_1 > x_2 \\ x_1, & \text{if } y_1 = y_2 \wedge x_1 < x_2 \\ y_2, & \text{if } x_1 = x_2 \wedge y_1 > y_2 \end{cases} \tag{3.19}$$

The obstacle position $P_{\text{obs}} = (X_o, Y_o)$ is adjusted as:

$$(X_o, Y_o) = \begin{cases} (X_o,\ Y_o), & \text{if } x_1 = x_2 \wedge y_2 > y_1 \\ (Y_o,\ Y_{\max} + Y_{\min} - X_o), & \text{if } y_1 = y_2 \wedge x_1 > x_2 \\ (X_{\max} + X_{\min} - Y_o,\ X_o), & \text{if } y_1 = y_2 \wedge x_1 < x_2 \\ (X_{\max} + X_{\min} - X_o,\ Y_{\max} + Y_{\min} - Y_o), & \text{if } x_1 = x_2 \wedge y_1 > y_2 \end{cases} \tag{3.20}$$

The vehicle's initial position $(X_{v,i}, Y_{v,i})$ in the APF reference frame is defined as:

$$(X_{v,i}, Y_{v,i}) = \begin{cases} (x_1 - \frac{R_w}{4},\ y_1), & \text{if } x_1 = x_2 \wedge y_2 > y_1 \\ (y_1 - \frac{R_w}{4},\ x_2), & \text{if } y_1 = y_2 \wedge x_1 > x_2 \\ (y_1 - \frac{R_w}{4},\ x_1), & \text{if } y_1 = y_2 \wedge x_1 < x_2 \\ (x_1 - \frac{R_w}{4},\ y_2), & \text{if } x_1 = x_2 \wedge y_1 > y_2 \end{cases} \tag{3.21}$$

The lane center coordinate L_{center}, used to define lane boundaries in the APF model, is given by:

$$L_{\text{center}} = \begin{cases} x_1, & \text{if } x_1 = x_2 \quad \text{(horizontal cases)} \\ y_1, & \text{if } y_1 = y_2 \quad \text{(vertical cases)} \end{cases} \tag{3.22}$$

With all necessary transformations established, the APF logic operates within a consistent reference frame, regardless of the orientation of the road segment in the global DSD coordinate system.

These transformations ensure that the obstacle, vehicle, and road geometry are always perceived by the APF method as aligned with its fixed Cartesian reference frame, as illustrated in Figures 3.14 and 3.15.

Based on the adjusted parameters, several crucial quantities are computed for the subsequent steps of the algorithm. The road length R_l is computed as:

$$R_l = |Y_{\max} - Y_{\min}| \tag{3.23}$$

The lane width L_w is defined as half of the total road width since we assume a road with two lanes:

$$L_w = \frac{R_w}{2} \tag{3.24}$$

Finally, the centerlines of the left and right lanes, X_l and X_r, are calculated as:

$$X_l = L_{\text{center}} - \frac{L_w}{2} \tag{3.25}$$

$$X_r = L_{\text{center}} + \frac{L_w}{2} \tag{3.26}$$

Before computing the potential field, the road segment is discretized into a uniform computational grid that spans the full road width and length. The resolution of this grid is adapted to the road geometry, enabling efficient and accurate evaluation of the potential field across the spatial domain. This grid is defined using MATLAB functions [14], [15] and [16], as follows:

$$Xrange = \text{unique}([\text{linspace}(X_{\min}, X_{\max}, \text{round}(12.5 \cdot R_w)), X_l, X_r]) \tag{3.27}$$

$$Yrange = \text{linspace}(Y_{\min}, Y_{\max}, 12.5 \cdot R_l) \tag{3.28}$$

$$[X, Y] = \text{meshgrid}(Xrange, Yrange) \tag{3.29}$$

where X and Y form a meshgrid of points evenly distributed across the road's width and length. The number of grid points scales proportionally with the dimensions of the road, maintaining a consistent spatial resolution for different scenarios. This matrix provides a structured grid on which the potential field values are computed.

The APF calculation relies on several predefined constants to determine the attractive and repulsive forces acting on the vehicle. These constants were determined through a combination of empirical testing and visual analysis of the figures in [5].

The constants used in the APF implementation are as follows. The vehicle's mass is defined as $m = 2000$ kg, and it travels at a constant longitudinal velocity of $V_x = 4.5$ m/s. The obstacle is assumed to be static, with a velocity of $v_{\text{obs}} = 0$ m/s. A maximum braking force per tire of $F_m = 3240$ N is considered, based on typical vehicle dynamics. The parameter $P_m = 0.7$ controls the amplitude of the potential at the center of the road.

To ensure flexibility across different road lengths, several spatial parameters are defined proportionally to the road length. These include the repulsive potential field scope, $D_t = 0.5 \cdot R_l$, and the universal potential field scope, $D_s = R_l$.

The obstacle's spatial influence is set to $L_l = 6$ m. This is because the obstacle's actual dimensions are defined as $L_{\text{obs}} = \frac{L_l}{2}$ in length and $W_{\text{obs}} = \frac{L_l}{4.8}$ in width. With this configuration, the obstacle matches the size of the vehicle used for trajectory planning and control, as detailed in Section 3.3.2. This design choice, as will be demonstrated later in the report, helps ensure that the vehicle can safely navigate around obstacles without collisions in the plotted simulation results.

Finally, a small threshold value, $P_t = 0.001$, is chosen to regulate the repulsive potential's influence and the time step for simulation is set to $\Delta t = 0.01$ s.

The constants used in the APF implementation are summarized in Table 3.1.

Table 3.1: Summary of constants used in the APF algorithm.

Symbol	Description	Value [units]
m	Vehicle mass	2000 [kg]
V_x	Vehicle constant longitudinal velocity	4.5 [m/s]
v_{obs}	Obstacle velocity	0 [m/s]
F_m	Maximum braking force per tire	3240 [N]
P_m	Potential amplitude	0.7 [–]
D_t	Repulsive potential field scope	$0.5 \cdot R_l$ [m]
D_s	Universal potential field scope	R_l [m]
L_l	Obstacle spatial influence	6 [m]
P_t	Potential threshold	0.001 [–]
Δt	Time step	0.01 [s]

With all necessary geometric transformations and APF constants defined, as discussed in the previous subsections, the reader can now apply the methodology outlined in Sections 3.1 and 3.2.

However, the generated trajectory is still expressed in a local coordinate frame and must be transformed back into the global coordinate frame used by the DSD. This step ensures the trajectory aligns correctly with the overall track orientation.

This transformation is performed according to the orientation of each segment and involves conditional operations such as axis swapping, vector reversal, and coordinate shifting. In particular, the MATLAB function `flip` is used to reverse the order of a vector's elements when needed [17].

The resulting global reference trajectory is computed as follows:

$$(X_{v,\text{ref}}(t),\ Y_{v,\text{ref}}(t)) = \begin{cases} (X_{v,\text{APF}}(t),\ Y_{v,\text{APF}}(t)) & \text{if } x_1 = x_2 \wedge y_2 > y_1 \\ (\text{flip}(Y_{v,\text{APF}}(t)),\ X_{v,\text{APF}}(t)) & \text{if } y_1 = y_2 \wedge x_1 > x_2 \\ (Y_{v,\text{APF}}(t),\ X_{\max} - X_{v,\text{APF}}(t) + X_{\min}) & \text{if } y_1 = y_2 \wedge x_1 < x_2 \\ (X_{\max} - X_{v,\text{APF}}(t) + X_{\min},\ \text{flip}(Y_{v,\text{APF}}(t))) & \text{if } x_1 = x_2 \wedge y_1 > y_2 \end{cases} \tag{3.30}$$

The corresponding time vector is constructed in MATLAB as:

$$t_{\text{ref}} = (0 : length(X_{v,\text{ref}}) - 1)' \cdot \Delta t \tag{3.31}$$

Finally, the full local trajectory is expressed as:

$$\text{reference_planned_local_trajectory}(t) = \begin{bmatrix} t_{\text{ref}} & X_{v,\text{ref}}(t) & Y_{v,\text{ref}}(t) \end{bmatrix} \tag{3.32}$$

Figure 3.16 and its corresponding APF contour plot in Figure 3.17,show the resulting local reference trajectory for the segment defined with the start and end center points C_1 and C_2, and the obstacle position $P_{\text{obs}} = (X_o, Y_o)$, depicted in Figure 3.13.

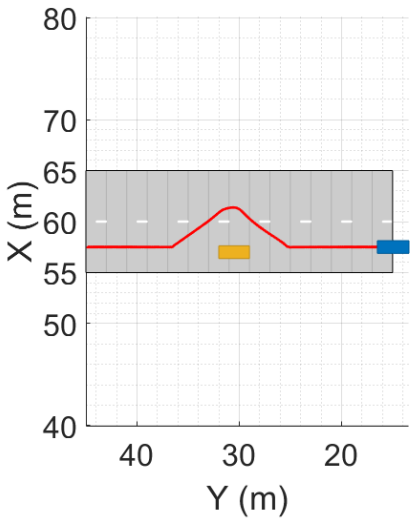


Figure 3.16: Segment with respective local reference planned trajectory.

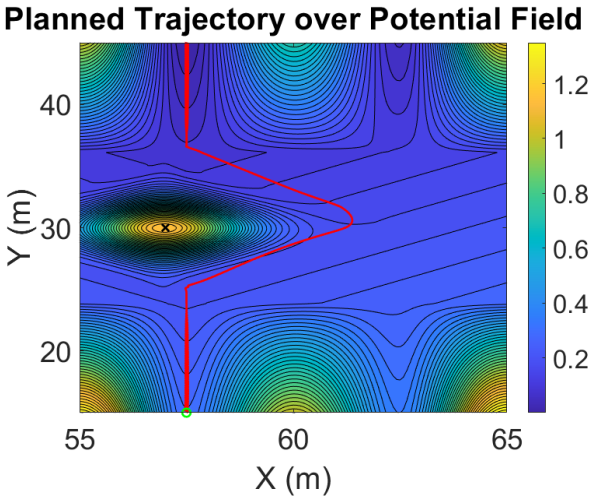


Figure 3.17: APF contour plot illustrating the trajectory overlaid on the potential field.

Since the planned reference trajectory only accounts for the vehicle's center of mass as a point, it is necessary to verify that the vehicle's actual physical dimensions do not cause collisions with obstacles. To this end, the vehicle was approximated by a circle with a diameter equal to the diagonal length of its rectangular footprint, while the obstacle was represented by a rectangle defined by the dimensions introduced earlier.

As described previously, the obstacle's spatial influence parameter $L_l = 6$ m was chosen because it provides a safety margin that ensures the obstacle size corresponds to that of the vehicle, according to the formulas $L_{\text{obs}} = \frac{L_l}{2} = \frac{6}{2} = 3$ m and $W_{\text{obs}} = \frac{L_l}{4.8} = \frac{6}{4.8} = 1.25$ m.

Figure 3.18 shows the APF contour plot with these geometric shapes overlaid, confirming that the planned trajectory maintains sufficient clearance to avoid collisions. This validation supports the effectiveness of the selected parameters in safely navigating the vehicle around obstacles.

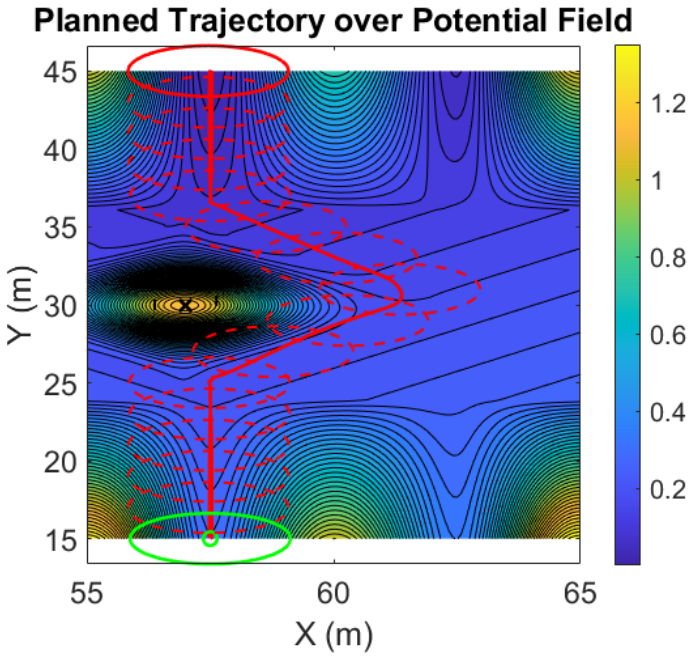


Figure 3.18: APF contour plot with overlaid circles and rectangle representing the vehicle and obstacle dimensions, respectively.

It is important to note that the circles appear elliptical in the plot due to the different scales used on the axes. Specifically, the longitudinal axis is three times longer than the lateral axis, which causes this visual distortion.

Ultimately, to further demonstrate the robustness and effectiveness of the APF algorithm, additional segment examples with their corresponding contour plots are presented below. These illustrate the method's capacity to handle a variety of obstacle configurations while ensuring collision-free path planning.

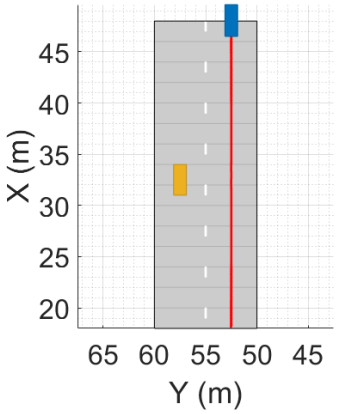


Figure 3.19: Segment with respective local reference planned trajectory.

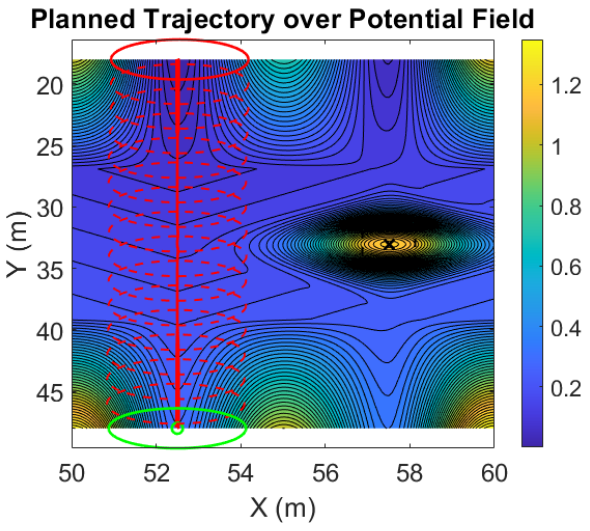


Figure 3.20: APF contour plot illustrating the trajectory overlaid on the potential field.

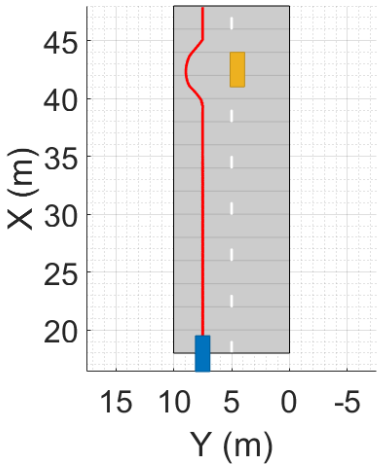


Figure 3.21: Segment with respective local reference planned trajectory.

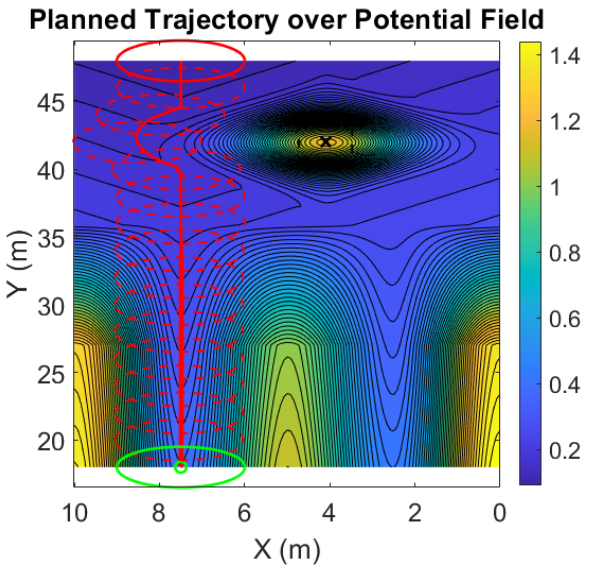


Figure 3.22: APF contour plot illustrating the trajectory overlaid on the potential field.

3.3.3.2 Straights Segments without Obstacles

For these segments, the trajectory generation consists of computing a straight line along the direction of motion, with an offset to keep the vehicle centered in the left lane. This choice aligns with the counterclockwise navigation assumption and ensures consistency across the track.

Given two center points, $C_1 = (x_1, y_1)$ and $C_2 = (x_2, y_2)$, and the road width R_w:

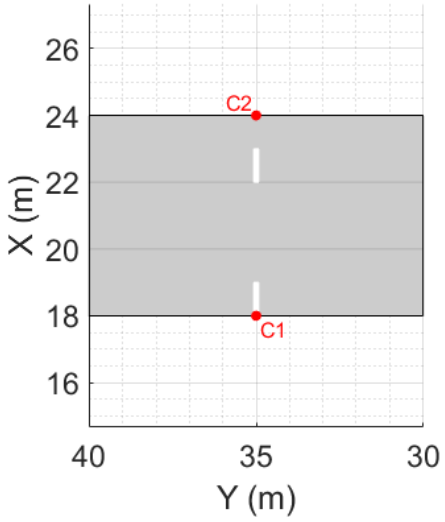


Figure 3.23: Straight segment with respective start and end center points C_1 and C_2.

which define the start and end of the segment along the centerline of the road, the algorithm calculates the direction of movement and constructs a straight reference path with an offset to the left lane. The offset is defined as:

$$\text{offset} = \frac{R_w}{4} \tag{3.33}$$

This offset positions the vehicle at the center of the left lane, assuming a road with two lanes. We define the start and end points of the reference trajectory as $P_{\text{start}} = (x_{\text{start}},\ y_{\text{start}})$ and $P_{\text{end}} = (x_{\text{end}},\ y_{\text{end}})$ and depending on the orientation of the segment, the offset is applied either in the positive or negative direction, as shown below:

$$(P_{\text{start}},\ P_{\text{end}}) = \begin{cases} ((x_1,\ y_1 + \text{offset}),\ (x_2,\ y_2 + \text{offset})), & \text{if } x_2 > x_1 \wedge y_2 = y_1 \quad \text{(Case 1)} \\ ((x_1,\ y_1 - \text{offset}),\ (x_2,\ y_2 - \text{offset})), & \text{if } x_2 < x_1 \wedge y_2 = y_1 \quad \text{(Case 2)} \\ ((x_1 - \text{offset},\ y_1),\ (x_2 - \text{offset},\ y_2)), & \text{if } x_2 = x_1 \wedge y_2 > y_1 \quad \text{(Case 3)} \\ ((x_1 + \text{offset},\ y_1),\ (x_2 + \text{offset},\ y_2)), & \text{if } x_2 = x_1 \wedge y_2 < y_1 \quad \text{(Case 4)} \end{cases} \tag{3.34}$$

Each line of the equation corresponds to one of the four possible orientations of straight segments:

- Case 1: Vertical segment from bottom to top, the offset is applied in the positive direction of Y.
- Case 2: Vertical segment from top to bottom, the offset is applied in the negative direction of Y.
- Case 3: Horizontal segment from right to left, the offset is applied in the negative direction of$X$.
- Case 4: Horizontal segment from left to right, the offset is applied in the positive direction of X.

Once the start and end points of the trajectory P_{start} and P_{end} are determined, the total distance between them is computed using the Euclidean norm:

$$d_{\text{total}} = ||P_{\text{end}} - P_{\text{start}}|| \tag{3.35}$$

To proceed, the trajectory is sampled at a fixed rate using a time step $\Delta t = 0.01$ s and a constant scalar vehicle speed $V_{scalar} = 4.5$ m/s, leading to a set of equidistant points along the segment. The total time is then obtained by:

$$t_{\text{total}} = \frac{d_{\text{total}}}{V_{scalar}} \tag{3.36}$$

From this, the number of reference points N is calculated as:

$$N = \frac{t_{\text{total}}}{\Delta t} + 1 \tag{3.37}$$

To associate each spatial position with a corresponding time value, a time reference vector is constructed to reflect the sampling times of the trajectory, in MATLAB using:

$$t_{\text{ref}} = (0 : \Delta t : t_{\text{total}})' \tag{3.38}$$

Finally, the trajectory is built using linear spacing between the start and end points, already shifted by the lane offset. The `linspace` function in MATLAB [14] is used to generate a set of uniformly spaced values between the start and end positions:

$$X_{v,\text{ref}} = \text{linspace}(x_{\text{start}}, x_{\text{end}}, N)' \tag{3.39}$$

$$Y_{v,\text{ref}} = \text{linspace}(y_{\text{start}}, y_{\text{end}}, N)' \tag{3.40}$$

This results in the local reference trajectory for the segment, ready to be appended to the overall track reference trajectory:

$$\text{reference_planned_local_trajectory}(t) = \begin{bmatrix} t_{\text{ref}} & X_{v,\text{ref}}(t) & Y_{v,\text{ref}}(t) \end{bmatrix} \tag{3.41}$$

Figure 3.24 shows an example of a local reference trajectory generated between the start and end center points C_1 and C_2 illustrated in Figure 3.23.

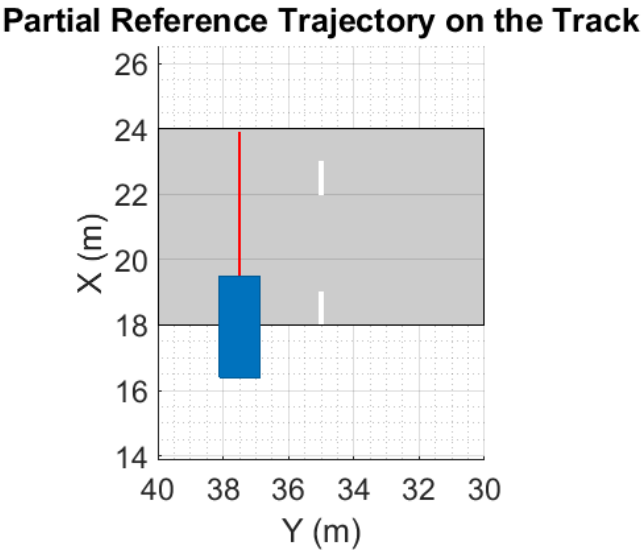


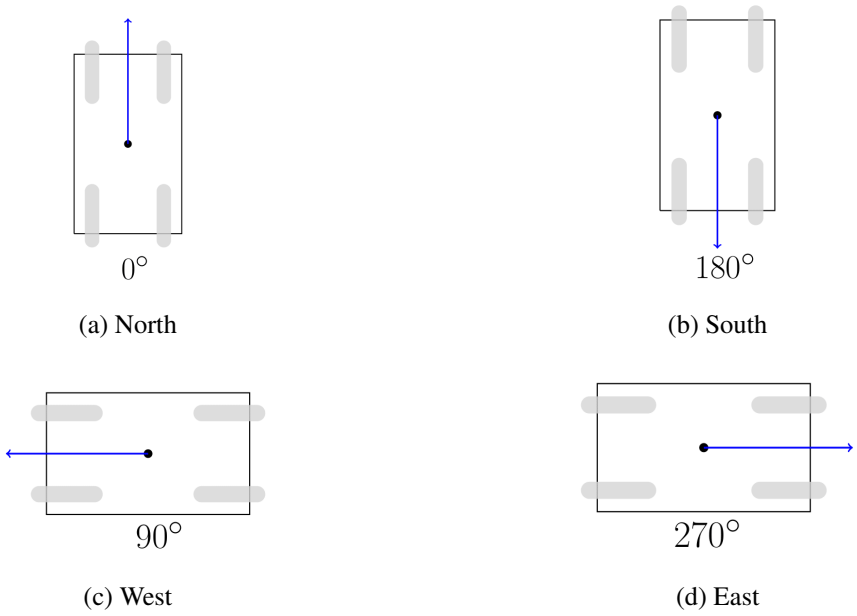
Figure 3.24: Segment with respective local reference planned trajectory.

This approach ensures the resulting trajectory is both geometrically simple and efficient to compute, providing accurate lane alignment throughout the segment by applying the offset in the appropriate direction.

3.3.3.3 Curved Segments without Obstacles

Trajectory generation for these segments uses an elliptical arc to ensure a smooth turning path that keeps the vehicle centered in the left lane. This method adheres to the counterclockwise navigation assumption and maintains consistency across the full trajectory.

To ensure consistency with the simulation environment, the trajectory generation process follows the angle conventions defined by the DSD framework. The conventions adopted are illustrated in Figure 3.25.



(a) North (b) South

(c) West (d) East

Figure 3.25: Reference orientation angles based on the DSD convention.

Given two example segments, each defined by start and end center points $C_1 = (x_1, y_1)$ and $C_2 = (x_2, y_2)$, and their corresponding orientation initial and final angles θ_i and θ_f according to the DSD convention shown in Figure 3.25, and the road width R_w:

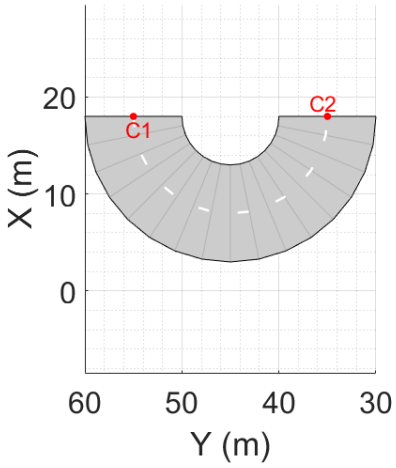


Figure 3.26: Example 1: Curved segment with respective start and end center points C_1 and C_2.

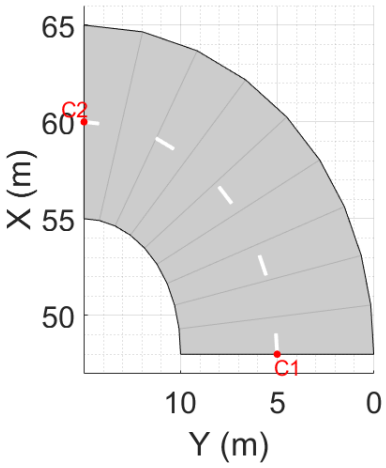


Figure 3.27: Example 2: Curved segment with respective start and end center points C_1 and C_2.

These points define the geometric centerline start and end of the segment, from which a series of parameters are calculated to generate the trajectory. From them, the algorithm determines the direction of motion and generates a reference path using an elliptical arc, with a lateral offset towards the left lane. The offset is defined as:

$$\text{offset} = \frac{R_w}{4} \tag{3.42}$$

Additionally, if $\theta_f = 0°$, it is adjusted to $\theta_f = 360°$ to ensure correct ellipse computation in MATLAB. The ellipse center $P_{\text{center}} = (x_c,\ y_c)$ is then determined based on the relative positions of the start and end points, following the logic:

$$(x_c,\ y_c) = \begin{cases} (x_1,\ \min(y_1,y_2) + \frac{|y_1-y_2|}{2}), & \text{if } (x_2 = x_1) \wedge (y_2 \neq y_1) \\ (\min(x_1,x_2) + \frac{|x_1-x_2|}{2},\ y_1), & \text{if } (y_2 = y_1) \wedge (x_2 \neq x_1) \\ (x_1,\ y_2), & \text{if } (x_2 > x_1) \wedge (y_2 > y_1) \\ (x_2,\ y_1), & \text{if } (x_2 < x_1) \wedge (y_2 > y_1) \\ (x_1,\ y_2), & \text{if } (x_2 < x_1) \wedge (y_2 < y_1) \\ (x_2,\ y_1), & \text{if } (x_2 > x_1) \wedge (y_2 < y_1) \end{cases} \tag{3.43}$$

The start and end points of the reference trajectory $P_{\text{start}} = (x_{\text{start}},\ y_{\text{start}})$ and $P_{\text{end}} = (x_{\text{end}},\ y_{\text{end}})$, are calculated based on the geometry and direction of the road segment (e.g., whether the curve ascends to the left or descends to the right):

$$(P_{\text{start}},\ P_{\text{end}}) = \begin{cases} ((x_1,\ y_1+\text{offset}),\ (x_2,\ y_2-\text{offset})), & \text{if } (x_2 = x_1) \wedge (y_2 \neq y_1) \wedge (\theta_f > \theta_i) \\ ((x_1,\ y_1-\text{offset}),\ (x_2,\ y_2+\text{offset})), & \text{if } (x_2 = x_1) \wedge (y_2 \neq y_1) \wedge (\theta_f < \theta_i) \\ ((x_1-\text{offset},\ y_1),\ (x_2+\text{offset},\ y_2)), & \text{if } (y_2 = y_1) \wedge (x_2 \neq x_1) \wedge (\theta_f > \theta_i) \\ ((x_1+\text{offset},\ y_1),\ (x_2-\text{offset},\ y_2)), & \text{if } (y_2 = y_1) \wedge (x_2 \neq x_1) \wedge (\theta_f > \theta_i) \\ ((x_1,\ y_1+\text{offset}),\ (x_2-\text{offset},\ y_2)), & \text{if } (x_2 > x_1) \wedge (y_2 > y_1) \\ ((x_1-\text{offset},\ y_1),\ (x_2,\ y_2-\text{offset})), & \text{if } (x_2 < x_1) \wedge (y_2 > y_1) \\ ((x_1,\ y_1-\text{offset}),\ (x_2+\text{offset},\ y_2)), & \text{if } (x_2 < x_1) \wedge (y_2 < y_1) \\ ((x_1+\text{offset},\ y_1),\ (x_2,\ y_2+\text{offset})), & \text{if } (x_2 > x_1) \wedge (y_2 < y_1) \end{cases} \tag{3.44}$$

Finally, the distances a and b that define the semi-axes of the ellipse can be obtained by:

$$a = \begin{cases} |y_{\text{start}} - y_c|, & \text{if } (x_2 = x_1) \wedge (y_2 \neq y_1) \\ |x_{\text{start}} - x_c|, & \text{if } (y_2 = y_1) \wedge (x_2 \neq x_1) \\ |(y_1+\text{offset}) - y_c|, & \text{if } (x_2 > x_1) \wedge (y_2 > y_1) \\ |(y_2-\text{offset}) - y_c|, & \text{if } (x_2 < x_1) \wedge (y_2 > y_1) \\ |(y_1-\text{offset}) - y_c|, & \text{if } (x_2 < x_1) \wedge (y_2 < y_1) \\ |(y_2+\text{offset}) - y_c|, & \text{if } (x_2 > x_1) \wedge (y_2 < y_1) \end{cases} \tag{3.45}$$

$$b = \begin{cases} |y_{\text{start}} - y_c|, & \text{if } (x_2 = x_1) \wedge (y_2 \neq y_1) \\ |x_{\text{start}} - x_c|, & \text{if } (y_2 = y_1) \wedge (x_2 \neq x_1) \\ |(x_2-\text{offset}) - x_c|, & \text{if } (x_2 > x_1) \wedge (y_2 > y_1) \\ |(x_1-\text{offset}) - x_c|, & \text{if } (x_2 < x_1) \wedge (y_2 > y_1) \\ |(x_2+\text{offset}) - x_c|, & \text{if } (x_2 < x_1) \wedge (y_2 < y_1) \\ |(x_1+\text{offset}) - x_c|, & \text{if } (x_2 > x_1) \wedge (y_2 < y_1) \end{cases} \tag{3.46}$$

With these parameters defined, the trajectory can now be obtained using an elliptical arc. The general equation of an ellipse centered at (x_c, y_c), with semi-axes a and b, is:

$$\frac{(x-x_c)^2}{b^2} + \frac{(y-y_c)^2}{a^2} = 1 \tag{3.47}$$

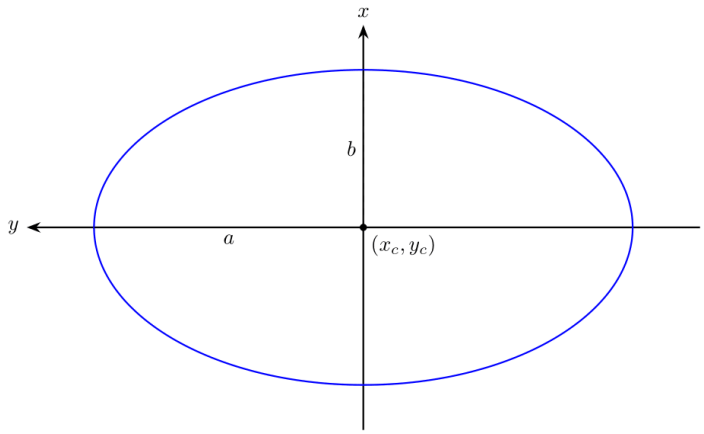


Figure 3.28: Illustration of the ellipse used in trajectory generation.

To determine how many points to sample along the elliptical arc, we first approximate its arc length using the Ramanujan's first formula for the complete perimeter of an ellipse [18]:

$$L_{\text{ellipse}} \approx \pi \left(3(a+b) - \sqrt{(3a+b)(a+3b)}\right) \tag{3.48}$$

Given that only a part of the ellipse is used (e.g, a quarter arc for 90° or a half arc for 180°), the effective arc length L_{arc} is scaled according to the angular range used:

$$L_{arc} = L_{\text{ellipse}} \cdot \frac{|\theta_f - \theta_i|}{2\pi} \tag{3.49}$$

Considering a scalar velocity $V_{scalar} = 4.5$ m/s, the total time to complete this segment is:

$$t_{\text{total}} = \frac{L_{arc}}{V_{scalar}} \tag{3.50}$$

With a fixed time step of $\Delta t = 0.01$, the number of reference points to sample along the arc is determined as:

$$N = \left\lfloor \frac{t_{\text{total}}}{\Delta t} \right\rfloor \tag{3.51}$$

A time vector t_{ref} is then defined to map each trajectory point to its respective timestamp. This sequence of sampling instants is constructed in MATLAB using:

$$t_{\text{ref}} = (0 : \Delta t : t_{\text{total}})' \tag{3.52}$$

To generate the elliptical trajectory, a vector of angular values θ_{ref} is defined to specify the portion of the ellipse to be followed. This vector has length N, corresponding to the number of trajectory points sampled uniformly along the arc. The angular values are computed in MATLAB using:

$$\theta_{\text{ref}} = \text{linspace}(\theta_i - 90^\circ,\ \theta_f, N)'. \tag{3.53}$$

The subtraction of 90° accounts for the fact that the axes used in DSD coordinate system (Figure 3.9) are rotated 90° counterclockwise relative to the standard Cartesian coordinate system.

With the reference angular vector θ_{ref} defined, the corresponding reference positions along the elliptical arc are computed using the parametric equations of the ellipse:

$$\begin{cases} X_{v,\text{ref}} = x_c + b\cos(\theta_{\text{ref}}) \\ Y_{v,\text{ref}} = y_c + a\sin(\theta_{\text{ref}}) \end{cases} \tag{3.54}$$

Here, both $X_{v,\text{ref}}$ and $Y_{v,\text{ref}}$ are vectors of length N, containing the sampled reference positions along the elliptical segment for each value in θ_{ref}.

This results in the local reference trajectory for the segment, ready to be appended to the overall track reference trajectory:

$$\text{reference_planned_local_trajectory}(t) = \begin{bmatrix} t_{\text{ref}} & X_{v,\text{ref}}(t) & Y_{v,\text{ref}}(t) \end{bmatrix} \tag{3.55}$$

Figures 3.29 and 3.30 illustrate the resulting local reference trajectories for two different curved segments, generated between the corresponding start and end center points C_1 and C_2 shown in Figures 3.26 and 3.27, respectively.

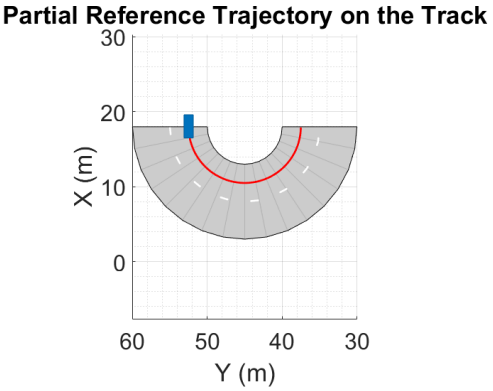


Figure 3.29: Example 1: Segment with respective local reference planned trajectory.

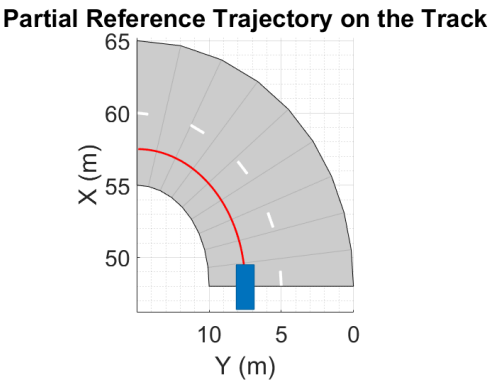


Figure 3.30: Example 2: Segment with respective local reference planned trajectory.

This method enables smooth and consistent trajectory generation through curved segments, ensuring accurate lane placement by applying the offset appropriately.

3.4 Path Planning Results

This section presents the results of the total reference trajectory generation, constructed segment by segment as described in Section 3.3. To validate the planning approach and visualize its practical applicability, several trajectory configurations were analyzed.

The evaluation begins by comparing the real laboratory race track, used for experimental validation, with its simulated counterpart constructed in DSD. As discussed in Section 3.3.2, the dimensions of the simulated track are scaled to maintain proportionality with the laboratory setup, ensuring consistency in the spatial relationship between the vehicle and the track in both environments.

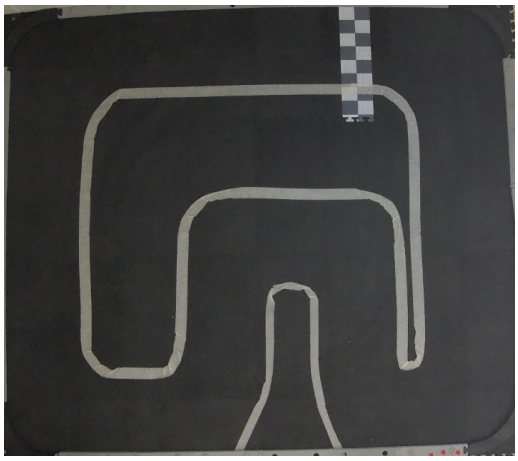


Figure 3.31: Laboratory Race Track.

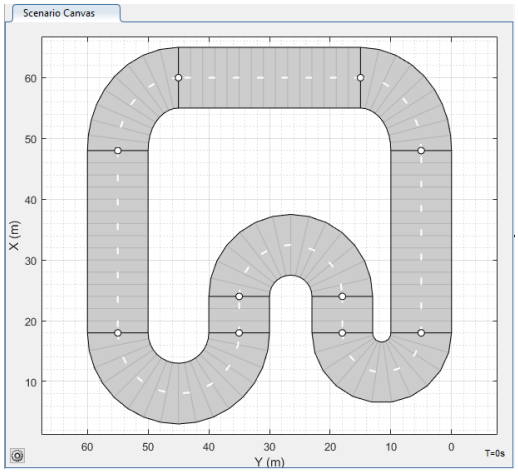


Figure 3.32: Laboratory Race Track in DSD.

Figure 3.33 illustrates an example of the total reference trajectory on the laboratory track, with obstacles placed at selected positions for testing purposes.

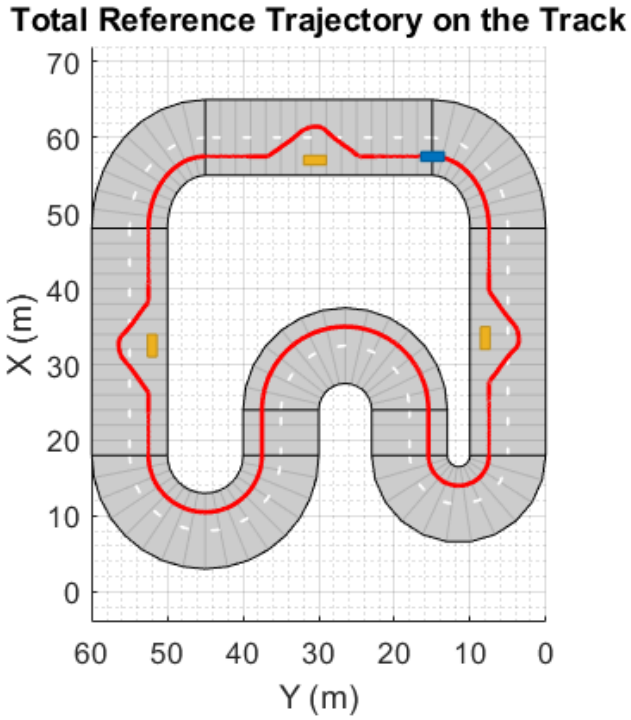


Figure 3.33: Reference trajectory on the laboratory track.

Table 3.2 presents the characteristics of each trajectory segment, including start and end center coordinates, curve angles, and any obstacle positions.

Table 3.2: Track segment coordinates and corresponding attributes.

Segment	$C1_n$	$C2_n$	$\theta_{i,n}$	$\theta_{f,n}$	**Obstacle** ($P_{\mathbf{obs},n}$)
1	(60, 15)	(60, 45)	–	–	(57, 30)
2	(60, 45)	(48, 55)	90°	180°	–
3	(48, 55)	(18, 55)	–	–	(33, 52)
4	(18, 55)	(18, 35)	180°	0°	–
5	(18, 35)	(24, 35)	–	–	–
6	(24, 35)	(24, 18)	0°	180°	–
7	(24, 18)	(18, 18)	–	–	–
8	(18, 18)	(18, 5)	180°	0°	–
9	(18, 5)	(48, 5)	–	–	(33, 8)
10	(48, 5)	(60, 15)	0°	90°	–

Note: All segments share a constant road width of $R_w = 10$ m.

Figure 3.34 shows another example of the total reference trajectory on the laboratory track, this time with obstacles placed in different locations to further evaluate the planner's behavior under new conditions.

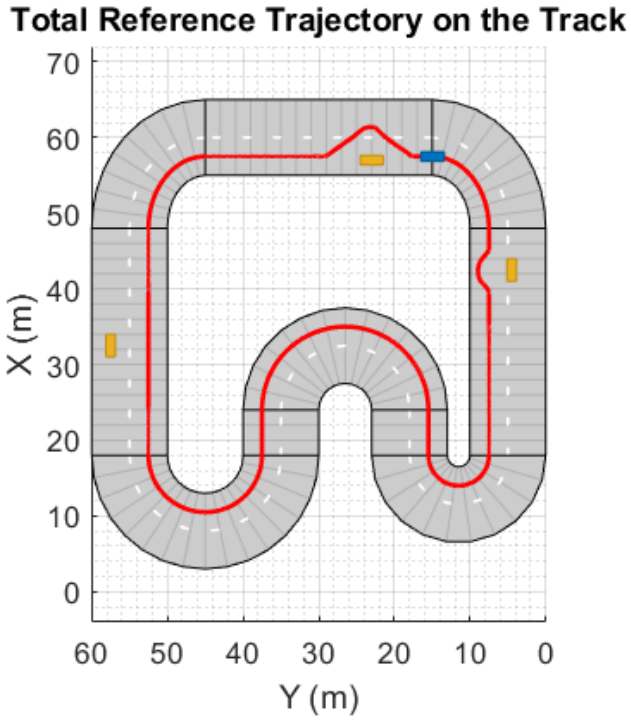


Figure 3.34: Reference trajectory on the same track with obstacles repositioned.

While the track geometry and planning parameters remain unchanged from the previous configuration, the obstacles were repositioned, for instance, $P_{\text{obs},1}$ to (57, 22.5), $P_{\text{obs},3}$ to (33, 57.5), and $P_{\text{obs},9}$ to (42, 4.5). All other track specifications, including segment coordinates, curve angles, and road width (10 m), remain consistent with those defined in Table 3.2.

Figure 3.35 depicts another example of the total reference trajectory on a different track, with obstacles placed at selected positions for testing purposes, demonstrating the scalability of the method.

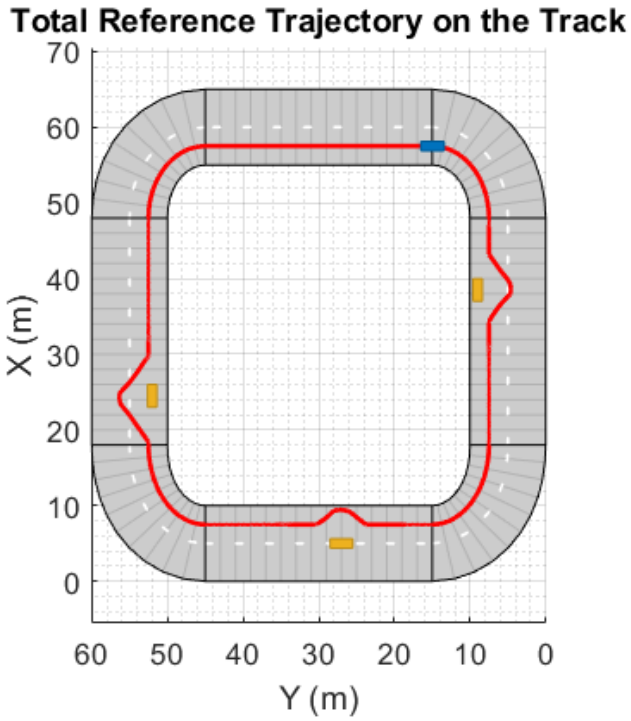


Figure 3.35: Reference trajectory on a modified track configuration.

Table 3.3 presents the characteristics of each trajectory segment, including start and end center coordinates, curve angles, and any obstacle positions.

Table 3.3: Track segment coordinates and corresponding attributes.

Segment	$C1_n$	$C2_n$	$\theta_{i,n}$	$\theta_{f,n}$	**Obstacle** ($P_{\mathbf{obs},n}$)
1	(60, 15)	(60, 45)	–	–	(57, 30)
2	(60, 45)	(48, 55)	90°	180°	–
3	(48, 55)	(18, 55)	–	–	(25, 52)
4	(18, 55)	(5, 45)	180°	270°	–
5	(5, 45)	(5, 15)	–	–	(5, 27.5)
6	(5, 15)	(18, 5)	270°	0°	–
7	(18, 5)	(48, 5)	–	–	(38, 9)
8	(48, 5)	(60, 15)	0°	90°	–

Note: All segments share a constant road width of $R_w = 10$ m.

Finally, Figure 3.36 demonstrates another example of the total reference trajectory on a different track, with obstacles placed at selected positions for testing purposes, further demonstrating the scalability and capabilities of the method.

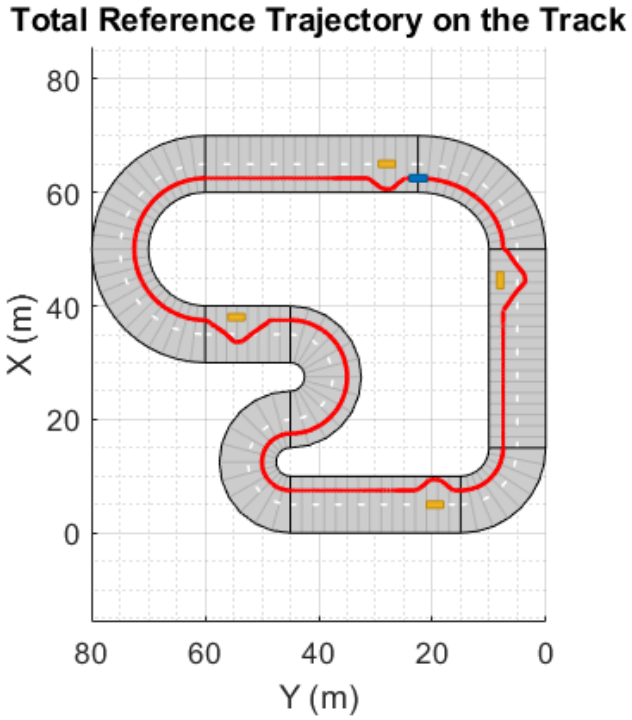


Figure 3.36: Reference trajectory on an alternative track layout.

Table 3.4 shows the characteristics of each trajectory segment, including start and end center coordinates, curve angles, and any obstacle positions.

Table 3.4: Track segment coordinates and corresponding attributes.

Segment	$C1_n$	$C2_n$	$\theta_{i,n}$	$\theta_{f,n}$	**Obstacle** ($P_{\mathbf{obs},n}$)
1	(65, 22.5)	(65, 60)	–	–	(65, 27.5)
2	(65, 60)	(35, 60)	90°	270°	–
3	(35, 60)	(35, 45)	–	–	(38, 55)
4	(35, 45)	(20, 45)	270°	90°	–
5	(20, 45)	(5, 45)	90°	270°	–
6	(5, 45)	(5, 15)	–	–	(5, 20)
7	(5, 15)	(15, 5)	270°	0°	–
8	(15, 5)	(50, 5)	–	–	(44, 8)
9	(50, 5)	(65, 22.5)	0°	90°	–

Note: All segments share a constant road width of $R_w = 10$ m.

Chapter 4

Model Predictive Control

In this chapter, we discuss the implementation of several model predictive controllers, both in theory and through MATLAB and Simulink, to track the reference trajectory generated during the path planning phase of our collision avoidance system. We also explore different approaches and solutions within the MPC framework to address this path tracking problem.

4.1 Introduction to MPC

MPC first emerged in the early 1960s, when the core idea for discrete-time linear systems was introduced as a static optimization problem [19]. As microprocessor capabilities grew, the use of MPC has spread to real-time embedded applications, often used in the automotive and aerospace industries [20].

One of its key strengths is the ability to handle constraints on the inputs, outputs, and states, which is important since real-world systems have physical limits that need to be respected [20]. Additionally, MPC also predicts several steps into the future using an internal model of the controlled system, allowing improved performance by anticipating system behavior [21].

MPC operates as a cyclic, real-time optimization process with four iterative steps:

1. Define/Update Optimization Problem

   At each time step k, MPC formulates a constrained optimization problem to:

   - Minimize a cost function (e.g., trajectory tracking error).
   - Satisfy input/output/state constraints (e.g., actuator limits, safety bounds).
   - Incorporate the dynamic model of the system (linear or nonlinear).

2. Solve Optimization Problem in step k

   The solver computes an open-loop control sequence over the prediction horizon p.

   - For linear MPC: Quadratic cost with linear constraints (fast computation).

- For nonlinear MPC: Non-quadratic cost with nonlinear constraints (higher complexity).

3. Apply First Control Move

   Only the first optimized control action is executed on the system.

   - Subsequent moves are discarded (reduces computational overhead).
   - Ensures feedback responsiveness to disturbances.

4. Shift Horizon and Update Feedback

   - Advance the time horizon: k←k+1.
   - Measure/estimate new system states (e.g., via sensors/Kalman filters).

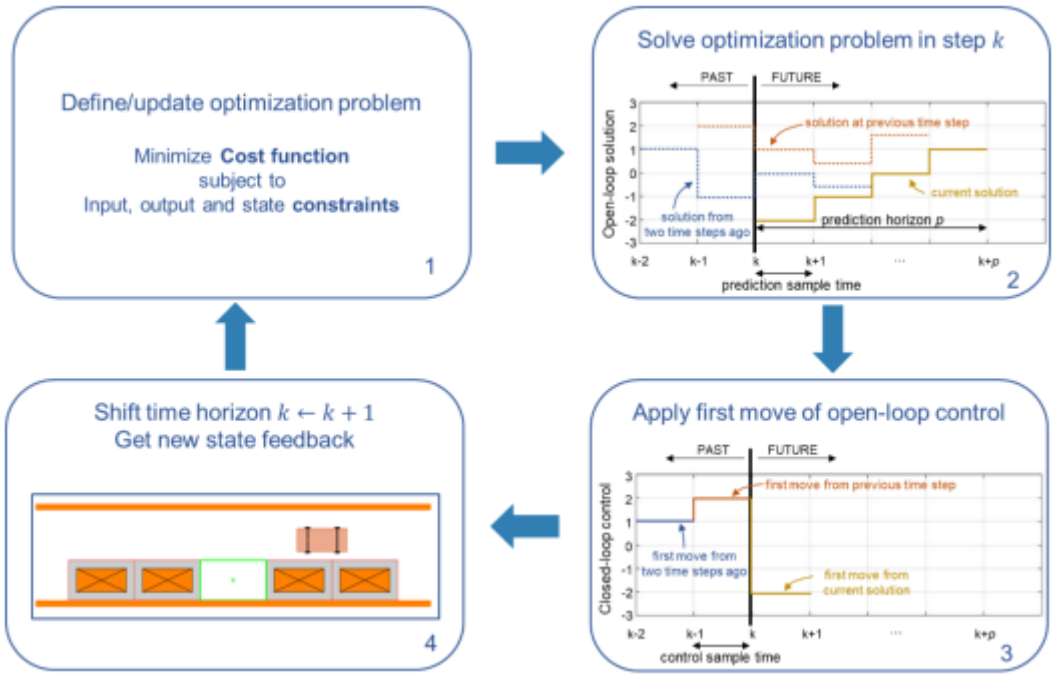


Figure 4.1: The MPC process [20].

4.1.1 MPC in Autonomous Racing

MPC is one of the most popular control strategies for engineers. It is a feedback control algorithm that uses a model to make predictions about future outputs of a process [22].

In the context of autonomous racing, the model represents the vehicle dynamics, which is based on the car characteristics and how it behaves in response to the given control actions. Because the car's behavior over time is known, the MPC will allow us to compute the optimal control inputs needed to achieve the desired outputs, this is, to follow as closely as possible the reference trajectory. It does this by solving an online optimization problem [23].

This approach is well-suited to our problem because it handles Multi-Input Multi-Output (MIMO) systems that might have interactions between their inputs and outputs. Beyond this, it can handle constraints and has preview capabilities, which are extremely important in a racing scenario, where we have tight curves and we have to keep the car within the limits of the track while avoiding obstacles. MPC can easily incorporate future reference information into the control problem to improve controller performance [23].

In a control problem, the controller calculates the input to the plant so that the plant output follows a desired reference. A model predictive controller's strategy to compute this input is to predict the future. These predictions are derived from the internal model of the system. It also uses an optimizer, which ensures that the predicted future plant output tracks the desired reference [21].

Because of the forward-moving nature of the prediction horizon, MPC is also referred to as receding horizon control [19]. A general MPC design scheme is shown in Figure 4.2.

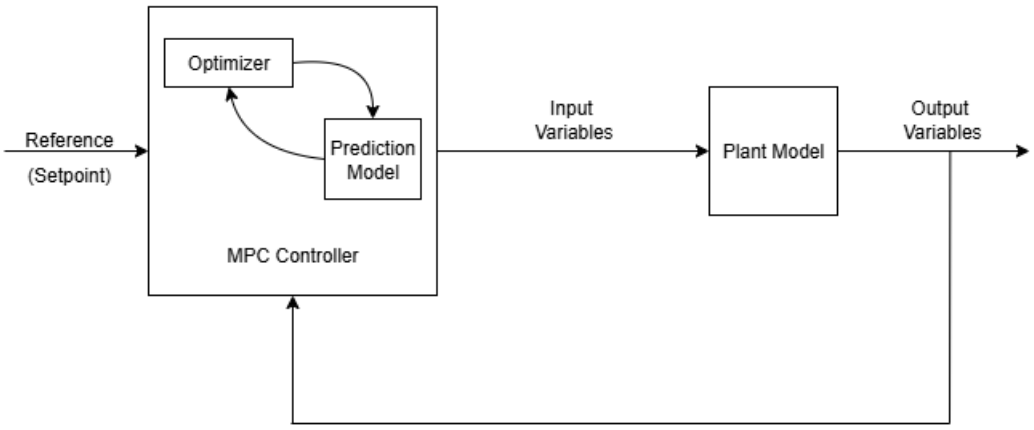


Figure 4.2: MPC general design.

4.1.2 MPC Parameters

Choosing appropriate MPC parameters is crucial, as they affect not only the controller performance but also the computational complexity of the MPC algorithm. These parameters are:

- Sample time.
- Prediction horizon.
- Control horizon.
- Constraints.
- Weights.

The sample time is the rate at which the controller executes the control algorithm. To find the right balance between performance and computational effort, the recommendation is to fit 10 to 20 samples within the rise time of the open-loop system response [23].

The prediction horizon shows how far the controller predicts into the future. Assuming the sample time is chosen based on what was said before, the recommendation is to choose a prediction horizon that has 20 to 30 samples covering the open-loop transient response [23].

The control horizon is the number of control moves in the future into a step we choose. The rest of the inputs are held constant. Increasing it can improve prediction accuracy at the cost of an increased computational complexity. However, usually only the first control moves have significant impact on the predicted output behavior. A good practise is to choose the control horizon as 10 to 20% of the prediction horizon, with a minimum of 2 to 3 steps [23].

MPC can add constraints on the inputs, rate of change of inputs, and the outputs. There are two types of constraints: hard constraints and soft constraints. Hard constraints cannot be violated, in contrast soft constraints can be violated. These have to be carefully adjusted, since having hard constraints on both inputs and outputs can lead to an unfeasible solution for the optimization problem, due to these constraints conflicting with each other. It is recommended that output constraints are soft and to avoid having hard constraints both on the inputs and their rate of change [23].

Finally, MPC has multiple goals and the defined weights determine how closely the controller tracks the setpoints. These weights have to be carefully chosen, since smooth control moves are generally preferable. Typically, larger weights are assigned to the more important output variables to prioritize their tracking [23].

4.1.3 MPC Types

Different types of MPC systems can be used depending on the problem we are trying to solve.

If we have a linear system that is represented by a linear plant model, with linear constraints and a quadratic cost function, we can use a Linear Time-Invariant (LTI) MPC to control the system. An MPC problem with these properties gives rise to a convex optimization problem where the cost function has a single global optimum, and the goal of the optimization is to find this optimum [23].

If we have a nonlinear system, with linear constraints and a quadratic cost function, we can use linear MPC and still benefit from the properties of the convex optimization problem. There are two different available methods, the Adaptive MPC and the Gain-scheduled MPC. These controllers deal with the nonlinear system by linearizing them at an operating point, and this gives a linear function that approximates the nonlinear system well near that point. However, outside of this region, that linear function will not work well [23].

That is why in Adaptive MPC, a linear model is computed in real-time as the operating conditions change, so at each time step we update the internal plant model used by the MPC controller with this linear model. Note that the structure of the optimization problem remains the same across different operating points, this is, the number of states and of constraints don't change for different operating conditions over the prediction horizon [23].

To be able to change them, a Gain-scheduled MPC should be used, this method linearizes offline at the operating points of interest and for each operating point, a linear MPC controller is designed. Each controller is independent from each other, so there could be controllers with different numbers of states and constraints. In this approach, an algorithm to change the predefined MPC controllers for different operating conditions is necessary. This approach uses more memory than Adaptive MPC [23].

Adaptive MPC is recommended when a linear plant model can be recalculated in real-time and when the structure of the optimization problem remains fixed across different operating conditions. If they change, a Gain-scheduled MPC is recommended, where independent MPC controllers can be designed for the changing operating conditions [23].

When a nonlinear system can't be approximated well by a linear model, nonlinear MPC should be used. This method is the most powerful one, as it uses the most accurate representation of the plant, leading to more accurate predictions and better control actions. The setback is that it is the most difficult to implement and solve in real-time, because with nonlinear constraints and a nonlinear cost function, the optimization problem becomes nonconvex. The cost function may have multiple local optima, and finding the global optimum may be hard. The efficiency of solving this optimization problem, which requires a large number of computations, depends on the nonlinear solver that is used [23].

In Figure 4.3, we see the different types of MPC. It compares Linear (L) and Nonlinear (NL) MPC, showing how they handle constraints, costs, and plant models. It also categorizes systems based on their behavior, linear, nonlinear, or a mix of both. The figure also distinguishes Adaptive MPC and Gain-Scheduled MPC, which adjust controller parameters for changing operating conditions.

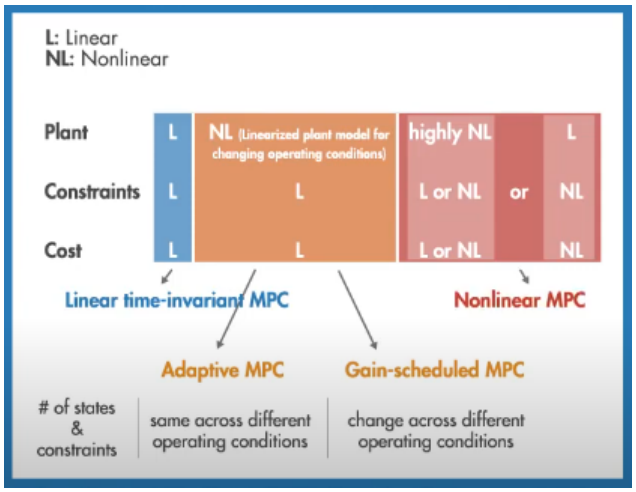


Figure 4.3: MPC types [23].

Meanwhile, Figure 4.4 provides a comparison of memory requirements and computational speed for these different MPC types.

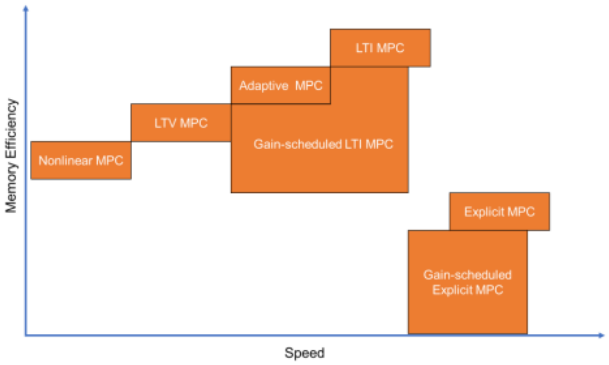


Figure 4.4: Memory and speed comparision for different types of MPC [20].

4.2 MPC Design and Implementation

This section documents the progressive development of the MPC system implemented throughout this work. The learning process began with basic trajectory tracking using a LTI model and evolved through custom trajectory generation and advanced model integration, concluding with the attempt to implement an Adaptive MPC controller for more realistic racing scenarios. Each step involved its own challenges and technical milestones, detailed below.

4.2.1 Step 1: MPC Virtual Lab

4.2.1.1 Objective

The goal of this first step was to understand the basic principles of MPC in the context of autonomous driving, particularly, autonomous steering. To achieve this, the virtual lab proposed in [24] was used as a guided exercise to design and simulate a LTI MPC controller capable of performing a single lane change maneuver.

4.2.1.2 Vehicle Dynamic Model and State-Space Description

To design an MPC controller capable of steering the vehicle during a lane change maneuver, we begin by modeling the lateral dynamics of the vehicle using a simplified bicycle model, as illustrated in Figure 4.5. The model includes two degrees of freedom, lateral motion and yaw angle [24]. It is worth noting that in this implementation, the controller's internal prediction model and the external plant model use identical dynamics, a simplification that facilitates initial controller design but would require adaptation for real-world applications.

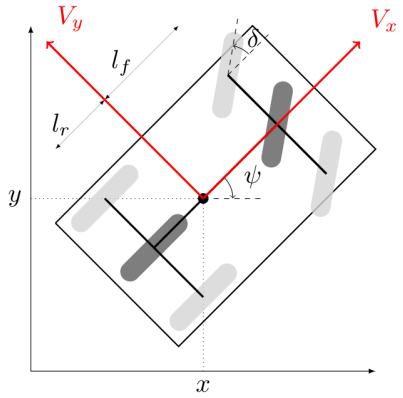


Figure 4.5: Representation of the car model.

In this model, X_v and Y_v represent the position of the vehicle in the global coordinate system, while V_x and V_y correspond to the longitudinal and lateral velocities in the body-fixed coordinate system, measured with respect to the vehicle's center of gravity. The variable ψ denotes the yaw angle of the vehicle, and δ is the front steering angle applied to control its direction [24].

Additionally, $Y_{v,\text{ref}}$ and ψ_{ref} represent the reference lateral position and reference yaw angle, respectively, which the controller aims to track.

In this example [24], several simplifying assumptions were made to keep the analysis suitable for linear MPC design. First, the vehicle's longitudinal velocity V_x was assumed to be constant throughout the maneuver. Additionally, the model employed is based on a bicycle configuration, meaning that the left and right wheels at each axle are lumped into a single equivalent wheel to simplify the lateral dynamics. Lastly, the model neglects the effects of suspension movements, tire slip phenomena, and aerodynamic influences, which are common simplifications when using a linearized dynamic model for control design purposes.

The continuous-time linear lateral dynamics of the vehicle can be described by the following equations:

$$\dot{V}_y = -\frac{2C_f + 2C_r}{mV_x} V_y + \left(-V_x - \frac{2C_f l_f - 2C_r l_r}{mV_x} \right) \dot{\psi} + \frac{2C_f}{m} \delta \tag{4.1}$$

$$\dot{\psi} = \dot{\psi} \tag{4.2}$$

$$\ddot{\psi} = -\frac{(2C_f l_f - 2C_r l_r)}{I_z V_x} V_y - \frac{(2C_f l_f^2 + 2C_r l_r^2)}{I_z V_x} \dot{\psi} + \frac{2C_f l_f}{I_z} \delta \tag{4.3}$$

$$\dot{Y}_v = V_y + V_x \psi \tag{4.4}$$

To model the system in state-space form, [24] defines the state, input, and output variables as:

$$x = \begin{bmatrix} V_y \\ \psi \\ \dot{\psi} \\ Y_v \end{bmatrix}, \quad u = \begin{bmatrix} \delta \end{bmatrix}, \quad y = \begin{bmatrix} Y_v \\ \psi \end{bmatrix} \tag{4.5}$$

This means that the state variables are the lateral velocity V_y, yaw angle ψ, yaw rate $\dot{\psi}$, and the global Y_v position. The input variable is the front steering angle δ, and the output variables are the global lateral position Y_v and the yaw angle ψ.

The continuous-time linear system can be expressed in standard state-space form as:

$$\dot{x} = Ax + Bu \tag{4.6}$$

$$y = Cx + Du \tag{4.7}$$

where the system matrices are:

$$\begin{aligned} A &= \begin{bmatrix} \frac{-(2C_f+2C_r)}{mV_x} & 0 & -V_x - \frac{(2C_f l_f - 2C_r l_r)}{mV_x} & 0 \\ 0 & 0 & 1 & 0 \\ \frac{-(2C_f l_f - 2C_r l_r)}{I_z V_x} & 0 & \frac{-(2C_f l_f^2 + 2C_r l_r^2)}{I_z V_x} & 0 \\ 1 & V_x & 0 & 0 \end{bmatrix}, \\ B &= \begin{bmatrix} \frac{2C_f}{m} \\ 0 \\ \frac{2C_f l_f}{I_z} \\ 0 \end{bmatrix}, \quad C = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \quad D = \begin{bmatrix} 0 \end{bmatrix} \end{aligned} \tag{4.8}$$

Table 4.1 summarizes all the constant parameters used in the model along with their descriptions and values.

Table 4.1: Vehicle parameters used in the MPC Virtual Lab [24].

Symbol	Description	Value [units]
m	Vehicle mass	1575 [kg]
I_z	Yaw moment of inertia	2875 [kg·m^2]
V_x	Vehicle constant longitudinal velocity	15 [m/s]
l_f	Distance from CG[1] of the vehicle to front axle	1.2 [m]
l_r	Distance from CG of the vehicle to rear axle	1.6 [m]
C_f	Front wheels cornering stiffness	19000 [N/rad]
C_r	Rear wheels cornering stiffness	33000 [N/rad]

These equations define a continuous-time linear system that captures the lateral and yaw motion of the vehicle. This model was used as the plant representation for the linear MPC controller

[1] Center of Gravity (CG)

developed in the following design phase. The parameters and the model were implemented in MATLAB, with the latter created using the `ss` function [25], which is used to define state-space models.

4.2.1.3 Reference Trajectory Definition

The reference trajectory used in this example [24] is a simple step profile, designed to test the controller's ability to perform a single lane change maneuver. It is defined as a discrete time-indexed vector that provides the desired lateral position and yaw angle the vehicle should follow during the simulation.

Formally, the reference trajectory can be represented as:

$$\text{reference trajectory}(t) = \begin{bmatrix} t_{\text{ref}} & Y_{v,\text{ref}}(t) & \psi_{\text{ref}}(t) \end{bmatrix} \tag{4.9}$$

In the actual implementation, this trajectory is structured as a matrix with three columns:

- The first column corresponds to the time vector, ranging from 0 to 10 seconds (which matches the simulation duration), with a spacing of 0.01 seconds, resulting in 1001 total time steps.
- The second column defines the desired global lateral position $Y_{v,\text{ref}}$, which is 0 until $t = 3.01$ seconds, at which point it steps to 4 meters and remains constant thereafter.
- The third column specifies the desired yaw angle ψ_{ref}, which is set to zero for the entire duration of the simulation.

Although the spacing of 0.01 seconds in the time vector does not match the sample time of the MPC controller ($T_s = 0.1$ s), this finer resolution in the reference vector serves the internal interpolation and alignment processes of MATLAB. The output variables to be tracked, Y_v and ψ, align directly with the second and third columns of the reference matrix.

4.2.1.4 MPC Controller Design and Tuning

With the plant model already implemented in MATLAB and Simulink, the next step was to design and tune the MPC controller. The overall objective was to allow the vehicle to perform a single lane change maneuver by tracking a predefined reference trajectory for the lateral position and yaw angle. This was accomplished through the approach proposed in the MPC Virtual Lab [24], using the MPC Designer app [26], which provides an easy interface to define the controller structure, tune parameters, and simulate performance.

The Simulink model for the closed-loop system was constructed by first inserting an `LTI System` block, in which the previously created state-space model of the vehicle was loaded. This `LTI System` block was connected to an `MPC Controller` block, where the actual design of the MPC would take place. Through this connection, the MPC block automatically identified the

manipulated variables of the system, which is the steering angle δ, and the measured outputs, which are the lateral position Y_v and yaw angle ψ.

The reference trajectory, previously defined as a time-dependent signal containing $Y_{v,\text{ref}}$ and ψ_{ref}, was imported into the Simulink model using a `From Workspace` block. This block allowed the reference trajectory to be applied to the controller.

Once the system structure was set up, the MPC Designer app was launched directly from the MPC block to begin the design process. The initial sample time was set to $T_s = 0.1$ s, a value that was chosen based on the estimated maneuver time of approximately 2 seconds. This ensures an adequate balance between performance and computational efficiency.

In the MPC Designer app, the controller structure was defined by adjusting the number of measured outputs to two, corresponding to Y_v and ψ. The input/output attributes were then configured, including the names, units, and scale factors. The yaw angle ψ, due to its smaller numerical range compared to Y_v, was assigned a smaller scale factor. These scale factors are important to balance the influence of each output in the cost function.

Then, the prediction and control horizons were set to $N_p = 10$ and $N_c = 2$, respectively. Several tuning attempts were performed, but the initial configuration already provided satisfactory results. In later stages, it was confirmed that increasing the horizons led to higher memory requirements without substantial performance improvements, so the original values were retained.

The output and reference signals, as well as the steering angle, were logged in the Simulink model in order to visualize the results in the Simulation Data Inspector [27]. As shown in the Simulink model in Figure 4.6, the logging is indicated by the small Wi-Fi-like symbols on the signal lines. This step enabled an easy comparison of the tracking performance against the reference signals.

Regarding constraints, bounds were imposed on both the steering angle and its rate of change. The steering angle was limited to $\pm\pi/6$ rad, and the steering rate was limited to $\pm\pi/12$ rad/s, helping to ensure smoother trajectory tracking. Additionally, the lateral position Y_v was constrained to stay within the range $[-2, 6]$ meters to ensure the vehicle stayed within the track boundaries.

The output weights were tuned to prioritize lateral position tracking over yaw angle tracking. A higher weight of 1 was assigned to Y_v, and a lower weight of 0.1 to ψ, reflecting their relative importance in the lane change maneuver.

All of these configurations and tuning operations were iteratively tested using the simulation scenario embedded in the MPC Designer app. The resulting plots and feedback helped verify constraint satisfaction, trajectory tracking accuracy, and control smoothness. Once tuning was finalized, the controller was exported to the Simulink workspace and integrated into the closed-loop simulation model.

A visual overview of the Simulink control system architecture is shown in Figure 4.6, where the connections between the reference input, MPC controller, vehicle model, and logged outputs can be observed.

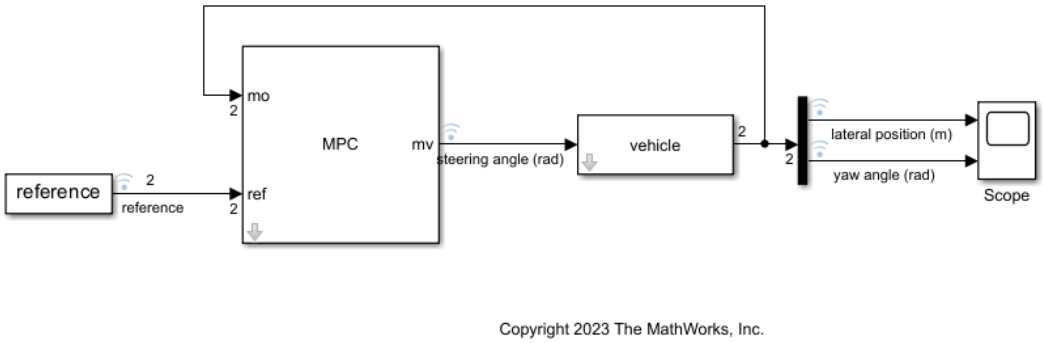


Figure 4.6: Simulink model of [24].

4.2.1.5 Simulation Results and Analysis

The final simulation results are shown below. Figure 4.7 presents the steering angle applied by the MPC controller during the lane change maneuver. Figures 4.8 and 4.9 show the tracking performance of the controller for the lateral position Y_v and yaw angle ψ, respectively, compared to their reference values.

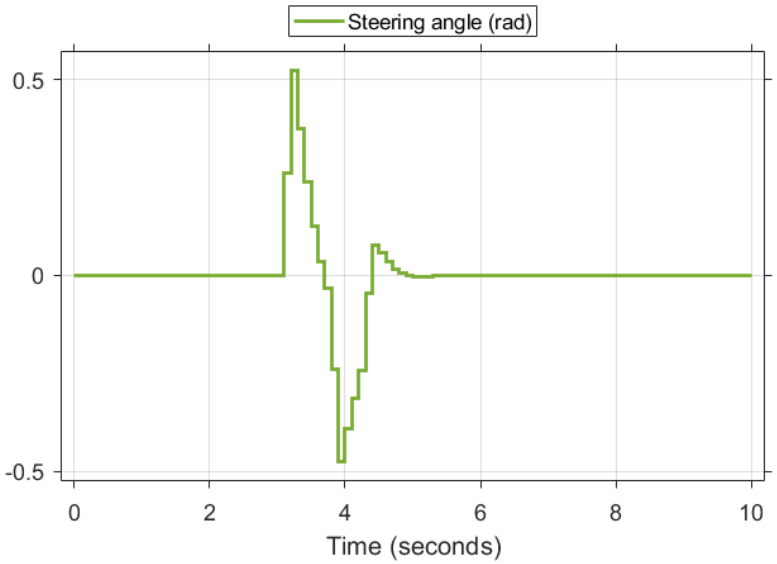


Figure 4.7: Steering angle response.

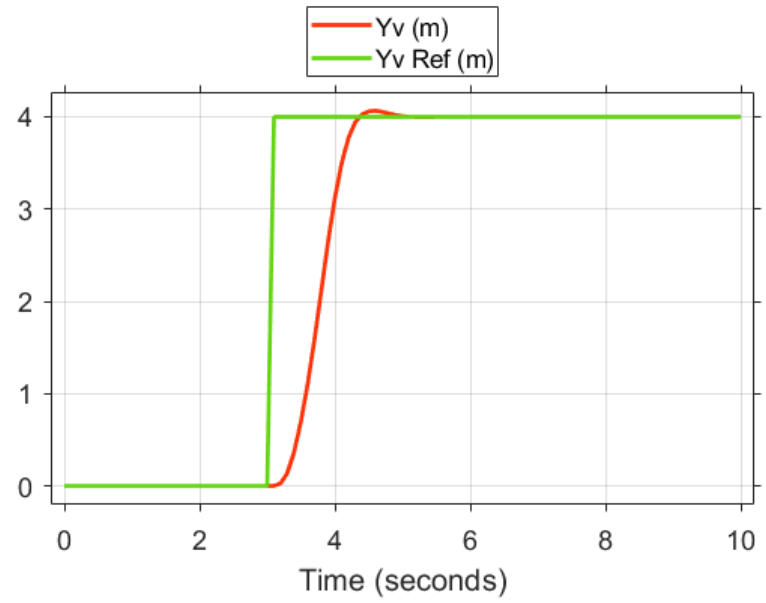


Figure 4.8: Lateral position tracking: Y_v vs. $Y_{v,\text{ref}}$.

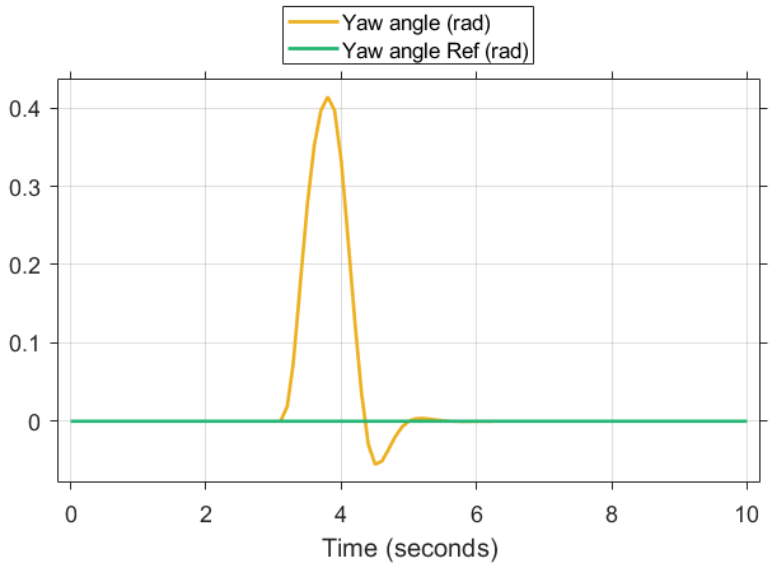


Figure 4.9: Yaw angle tracking: ψ vs. ψ_{ref}.

The controller achieves good tracking performance in the constant longitudinal velocity of $V_x = 15\,\text{m/s}$. However, when the longitudinal speed is changed significantly, the controller becomes unstable or ineffective. This highlights the importance of using Adaptive MPC techniques in more dynamic scenarios.

Finally, taking advantage of a scenario already designed in [24] using the DSD app [12], it was possible to visualize the vehicle following the trajectory generated by the controller. In Figure 4.10, we can see a snapshot of the vehicle behavior during the lane change.

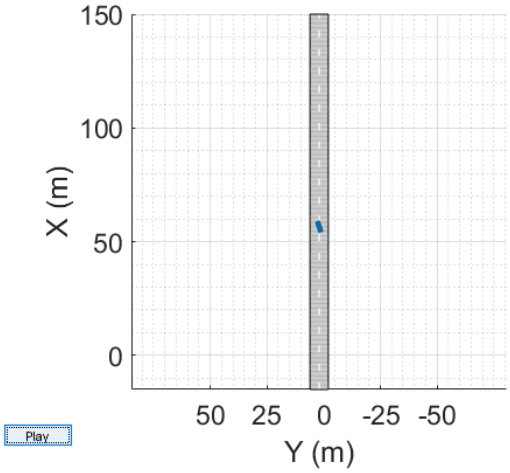


Figure 4.10: Vehicle executing the lane change in the DSD scenario.

4.2.2 Step 2: Implementing the Control Model and Approach of [5]

4.2.2.1 Objective

The objective of this second step[2] was to transfer the knowledge acquired in Step 1 (4.2.1) to a slightly more complex scenario, still using a linear vehicle model and a Linear MPC controller. The goal was to test the ability of the controller to follow a more challenging trajectory, emulating the behavior of a car avoiding an obstacle, based on the control approach developed in [5].

4.2.2.2 Vehicle Dynamic Model and State-Space Description

To develop a model predictive controller capable of path tracking a collision avoidance trajectory, it is crucial to construct a dynamic model that accurately describes the motion of the vehicle. For this purpose, we adopt a simplified dynamic model derived from Newton's laws, as illustrated in Figure 4.11, which captures the lateral and yaw dynamic aspects of the vehicle [5].

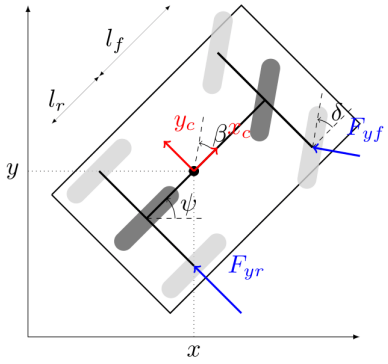


Figure 4.11: Representation of the vehicle dynamic model.

[2]Note that in this Step 2 subsection, the angular quantities are expressed in degrees (°), following the original reference example.

In this model, X_v and Y_v represent the position of the vehicle in the fixed earth coordinate system. The variable V_x denotes the longitudinal velocity at the CG of the vehicle, and β is the body sideslip angle. The yaw angle is represented by ψ, while $\dot{\psi}$ corresponds to the yaw rate. The steering angle applied at the front wheels is given by δ. Finally, F_{yf} and F_{yr} denote the lateral tire forces at the front and rear wheels, respectively.

The lateral position $Y_{v,\text{ref}}$, body sideslip angle β_{ref}, and yaw rate $\dot{\psi}_{\text{ref}}$ represent the main reference values the controller tracks.

For the path tracking problem, the following modeling assumptions were made. First, the longitudinal velocity V_x is constant throughout the motion. Second, the vehicle is represented using a single-track bicycle model, in which the left and right wheels of each axle are lumped into a single equivalent wheel. Third, suspension movements, slip phenomena, and aerodynamic influences are neglected.

The vehicle lateral dynamics can be described by the following equations:

$$mV(\dot{\beta} + \dot{\psi}) = F_{yf} + F_{yr} \tag{4.10}$$

$$I_z\ddot{\psi} = l_f F_{yf} - l_r F_{yr} \tag{4.11}$$

In this approach the lateral tire forces were approximated as linear functions of the front and rear tire slip angles, α_f and α_r. Therefore, the equations that describe F_{yf} and F_{yr} are:

$$F_{yf} = C_f \cdot \alpha_f = C_f\left(\delta - \beta - \frac{l_f\dot{\psi}}{V_x}\right) \tag{4.12}$$

$$F_{yr} = C_r \cdot \alpha_r = C_r\left(-\beta + \frac{l_r\dot{\psi}}{V_x}\right) \tag{4.13}$$

Substituting (4.12) and (4.13) into (4.10) and (4.11), we obtain the equations that determine the lateral and yaw dynamics in the vehicle:

$$\dot{Y}_v = V_x\beta + V_x\psi \tag{4.14}$$

$$\dot{\beta} = -\frac{(C_r + C_f)}{mV_x}\beta + \left(\frac{C_r l_r - C_f l_f}{mV_x^2} - 1\right)\dot{\psi} + \frac{C_f}{mV_x}\delta \tag{4.15}$$

$$\dot{\psi} = \dot{\psi} \tag{4.16}$$

$$\ddot{\psi} = \frac{C_r l_r - C_f l_f}{I_z}\beta - \frac{C_r l_r^2 + C_f l_f^2}{I_z V_x}\dot{\psi} + \frac{C_f l_f}{I_z}\delta \tag{4.17}$$

To represent the model in state-space form, we define the state vector x, control input u, and output y as follows:

$$x = \begin{bmatrix} Y_v \\ \beta \\ \psi \\ \dot{\psi} \end{bmatrix}, \quad u = \begin{bmatrix} \delta \end{bmatrix}, \quad y = \begin{bmatrix} Y_v \\ \beta \\ \dot{\psi} \end{bmatrix} \tag{4.18}$$

Accordingly, the state variables are the lateral position Y_v, body sideslip angle β, yaw angle ψ, and the yaw rate $\dot{\psi}$. The input variable is the front steering angle δ, and the output variables are the global lateral position Y_v, body sideslip angle β and the yaw rate $\dot{\psi}$.

The continuous-time system can then be written in the standard linear state-space form:

$$\dot{x} = Ax + Bu \tag{4.19}$$

$$y = Cx + Du \tag{4.20}$$

where the system matrices are given by:

$$\begin{aligned} A &= \begin{bmatrix} 0 & V_x & V_x & 0 \\ 0 & -\frac{C_r+C_f}{mV_x} & 0 & \frac{C_r l_r - C_f l_f}{mV_x^2} - 1 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{C_r l_r - C_f l_f}{I_z} & 0 & -\frac{C_f l_f^2 + C_r l_r^2}{I_z V_x} \end{bmatrix}, \\ B &= \begin{bmatrix} 0 \\ \frac{C_f}{mV_x} \\ 0 \\ \frac{C_f l_f}{I_z} \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad D = \begin{bmatrix} 0 \end{bmatrix} \end{aligned} \tag{4.21}$$

An overview of the constant parameters employed in the model, including their definitions and numerical values, is provided in Table 4.2.

Table 4.2: Vehicle parameters used for the dynamic model [5].

Symbol	Description	Value [units]
m	Vehicle mass	2020 [kg]
I_z	Yaw moment of inertia	3234 [kg·m^2]
V_x	Vehicle constant longitudinal velocity	20 [m/s]
l_f	Distance from CG of the vehicle to front axle	1.40 [m]
l_r	Distance from CG of the vehicle to rear axle	1.65 [m]
C_f	Front wheels cornering stiffness	1420 [N/°]
C_r	Rear wheels cornering stiffness	1420 [N/°]

This state-space model captures the essential dynamics of the vehicle's lateral motion and yaw behavior under the assumptions mentioned previously. It forms the basis for the trajectory tracking controller developed in the following sections. The model and its associated parameters

were implemented in MATLAB. The state-space representation of the system was constructed using the `ss` function, which defines models in state-space form.

4.2.2.3 Reference Trajectory Definition

In this second step, the reference trajectory was obtained using the DSD app, a tool available in MATLAB that enables the creation of custom roads, actors, and sensor setups. Following the guidance provided by [28] and [29], we created a road with boundaries spanning from $Y = 0$ to $Y = 8$ meters in width and $X = 0$ to $X = 200$ meters in length. An ego vehicle actor was placed on the road, and a realistic lane change trajectory, simulating obstacle avoidance, was created by defining a set of waypoints for it to follow.

We also attached a sensor of type vision to the vehicle, which was required to enable the scenario export functionality. Once the road, actor, sensor, and waypoints were fully defined, the entire setup was exported from DSD into MATLAB as a function. This function script contains all the data generated in the scenario, including time, position, and yaw angle for each simulation step, which was later used to construct the reference trajectory.

A snapshot of the planned reference trajectory, created in DSD, which shows a realistic collision avoidance maneuver from $Y_{v,\text{ref}} = 6$ m to $Y_{v,\text{ref}} = 2$ m and back, is presented in Figure 4.12.

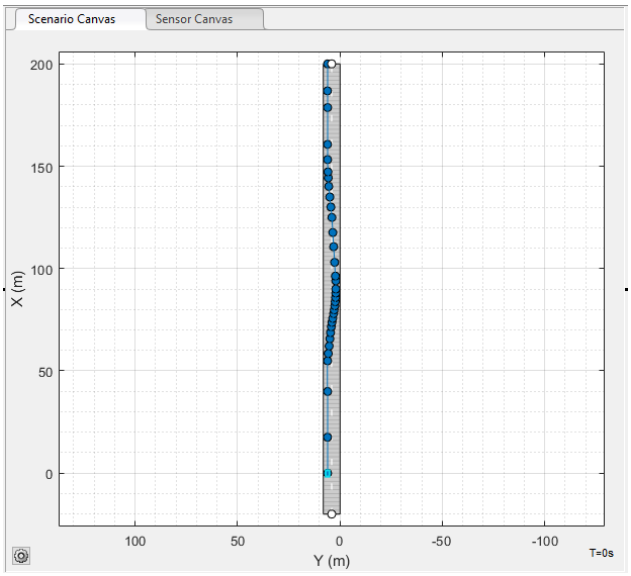


Figure 4.12: Generated reference trajectory from DSD.

Before using this trajectory as a reference for the controller, further processing was required to compute additional variables and enforce physical constraints based on the curvature. This was done through a custom script, which implemented equations from the paper to compute the curvature C_T, body sideslip angle reference β_{ref}, and yaw rate reference $\dot{\psi}_{\text{ref}}$, and to limit them according to vehicle dynamics constraints. These were computed using formulas inspired by the approach in [5].

First, the curvature C_T of the planned trajectory at a certain instant was estimated based on the vehicle reference position coordinates $X_{v,\text{ref}}(t)$ and $Y_{v,\text{ref}}(t)$ using:

$$C_T(t) = \frac{\frac{d^2 X_{v,\text{ref}}}{dY_{v,\text{ref}}^2}}{\left(1 + \left(\frac{dX_{v,\text{ref}}}{dY_{v,\text{ref}}}\right)^2\right)^{3/2}} \tag{4.22}$$

Then β_{curv} was calculated as:

$$\beta_{\text{curv}} = C_T(t) \cdot \left(l_r - \frac{l_f m V_x^2}{2 C_r L_{car}}\right) \tag{4.23}$$

where $L_{car} = l_f + l_r$ is the length of the vehicle.

To ensure lateral stability, the reference body sideslip angle was bounded as follows:

$$\beta_{\max} = \begin{cases} \frac{2(k_1-k_2)}{V_r^3} V_x^3 - \frac{3(k_1-k_2)}{V_r^2} V_x^2 + k_1, & \text{if } V_x < V_r \\ k_2, & \text{if } V_x \geq V_r \end{cases} \tag{4.24}$$

with reasonable values being $k_1 = \pi/18$, $k_2 = \pi/60$, where V_r is a characteristic speed, typically slightly higher than the vehicle's velocity. The final reference sideslip angle is clamped within $[-\beta_{\max}, \beta_{\max}]$ as:

$$\beta_{\text{ref}}(k) = \begin{cases} \beta_{\max}, & \text{if } \beta_{\text{ref}}(k) > \beta_{\max} \\ \beta_{\text{curv}}, & \text{if } \beta_{\text{ref}}(k) \in [-\beta_{\max}, \beta_{\max}] \\ -\beta_{\max}, & \text{if } \beta_{\text{ref}}(k) < -\beta_{\max} \end{cases} \tag{4.25}$$

Similarly, the yaw rate reference $\dot{\psi}_{\text{ref}}$ is derived from the curvature using:

$$\dot{\psi}_{\text{curv}}(t) = \frac{V_x \cdot C_T(t)}{\cos(\beta_{\text{curv}})} \tag{4.26}$$

and then constrained by the maximum reference yaw rate:

$$\dot{\psi}_{\max} = \frac{\mu g}{V_x} \tag{4.27}$$

with $\mu = 0.5$ and $g = 9.81\,\text{m/s}^2$. The final yaw rate reference contained in $[-\dot{\psi}_{\max}, \dot{\psi}_{\max}]$ is given by:

$$\dot{\psi}_{\text{ref}}(k) = \begin{cases} \dot{\psi}_{\max}, & \text{if } \dot{\psi}_{\text{ref}}(k) > \dot{\psi}_{\max} \\ \dot{\psi}_{\text{curv}}, & \text{if } \dot{\psi}_{\text{curv}}(k) \in [-\dot{\psi}_{\max}, \dot{\psi}_{\max}] \\ -\dot{\psi}_{\max}, & \text{if } \dot{\psi}_{\text{ref}}(k) < -\dot{\psi}_{\max} \end{cases} \tag{4.28}$$

Finally, the reference vector for the simulation was built as:

$$\text{reference trajectory}(t) = \begin{bmatrix} t_{\text{ref}} & Y_{v,\text{ref}}(t) & \beta_{\text{ref}}(t) & \dot{\psi}_{\text{ref}}(t) \end{bmatrix} \tag{4.29}$$

$Y_{v,\text{ref}}$ corresponds to the lateral position path in the fixed earth coordinate system, while β_{ref} and $\dot{\psi}_{\text{ref}}$ represent the constrained sideslip and yaw rate values, respectively. The time vector t_{ref} is aligned with the simulation step size used by the exported scenario in DSD, which was defined as 0.01 seconds.

4.2.2.4 MPC Controller Design and Tuning

In this second step, the same modeling and control design framework from the previous step was adopted to implement the MPC controller. The vehicle model was inserted into the Simulink model using an `LTI System` block. This block was connected to an `MPC Controller` block, and the newly generated and more realistic reference trajectory was introduced into the model via a `From Workspace` block.

The tuning procedure followed the same general approach as before, using the MPC Designer App. The prediction and control horizons $N_p = 25$ and $N_c = 15$, input/output attributes, and scaling factors were configured similarly to ensure consistency. Signal logging was also performed to analyze the reference tracking performance in the Simulation Data Inspector.

As for constraints, the same bounds were applied to the steering angle and its rate, limited to $\pm 30^\circ$ and $\pm 15^\circ/\text{s}$, respectively, to ensure stability and comfort. In addition, the output variables were constrained according to the following inequality expressions:

$$Y_{v,\min} \leq Y_v(k) \leq Y_{v,\max} \tag{4.30}$$

$$-\beta_{\max} \leq \beta(k) \leq \beta_{\max} \tag{4.31}$$

$$-\dot{\psi}_{\max} \leq \dot{\psi}(k) \leq \dot{\psi}_{\max} \tag{4.32}$$

where $Y_{v,\min} = 0$ m, $Y_{v,\max} = 8$ m define the road boundaries, while $\beta_{\max}$ and $\dot{\psi}_{\max}$ were computed during reference trajectory generation based on vehicle dynamics according to 4.24 and 4.27, respectively, following the approach in [5].

Given that the modeling and tuning logic remained fundamentally the same, this step primarily served to validate that the controller can successfully follow a realistic, smooth lane-change reference trajectory under a constrained control structure.

This work was carried out by adapting the approach proposed in [5] to the framework established in Step 1. The implementation confirmed that the method presented in the paper integrates naturally with the previously developed setup, resulting in a trajectory that more accurately reflects a collision avoidance maneuver.

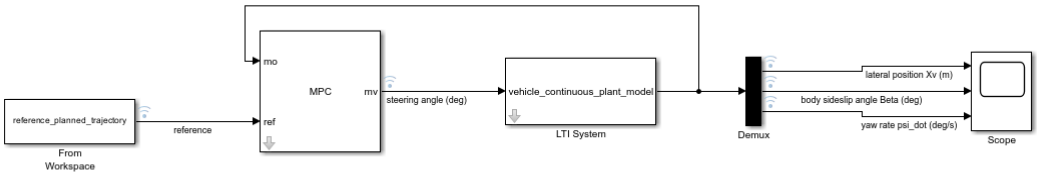


Figure 4.13: Simulink model used in Step 2 implementation.

4.2.2.5 Simulation Results and Analysis

The following figures present the controller's performance in tracking the reference trajectory under the defined constraints and dynamics. Each plot shows the behavior of a key variable relevant to tracking performance.

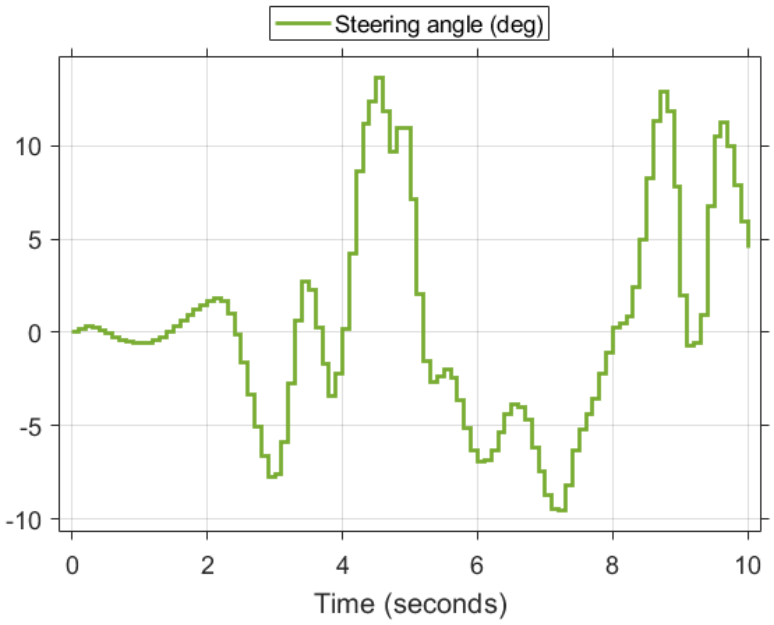


Figure 4.14: Steering angle applied by the controller.

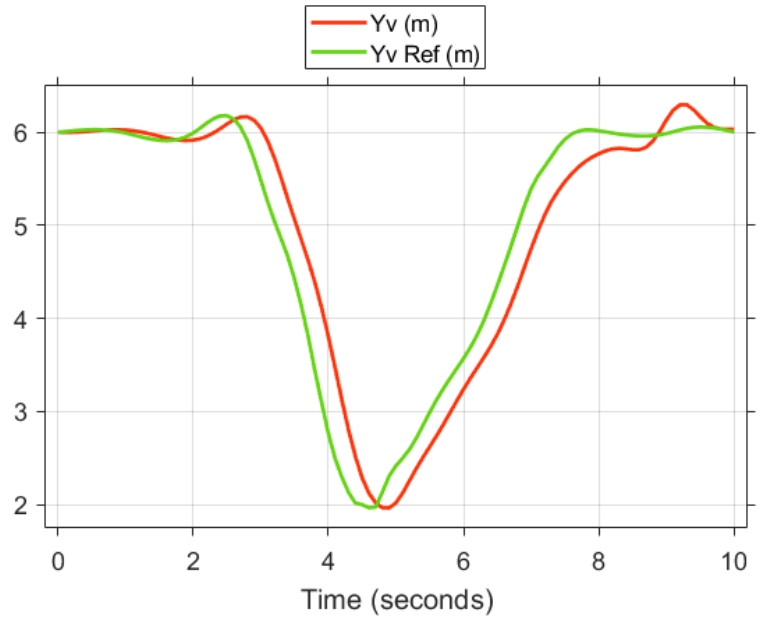


Figure 4.15: Lateral position tracking: Y_v vs. $Y_{v,\text{ref}}$.

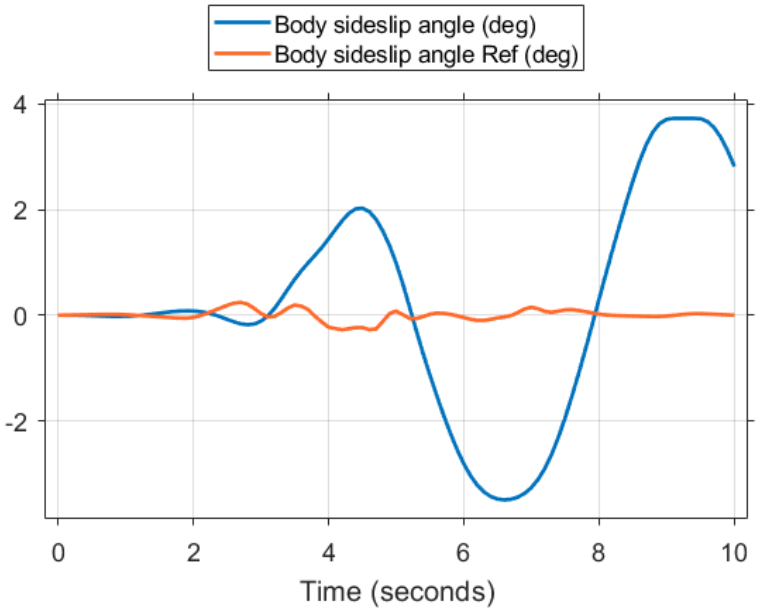


Figure 4.16: Body sideslip angle tracking: β vs. β_{ref}.

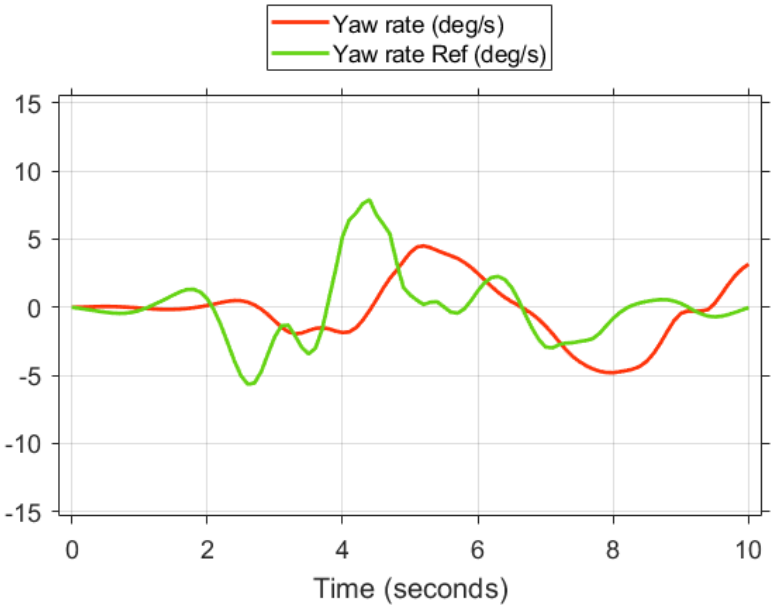


Figure 4.17: Yaw rate tracking: $\dot{\psi}$ vs. $\dot{\psi}_{\text{ref}}$.

Ultimately, snapshots of the vehicle following the reference trajectory are shown below, illustrating the collision avoidance maneuver performed by the controller. The sequence highlights the car smoothly transitioning from $Y_v = 6$ m to $Y_v = 2$ m and back.

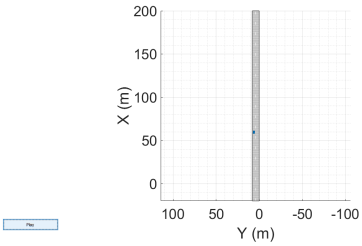


Figure 4.18: Start of the maneuver at $Y_v = 6$ m.

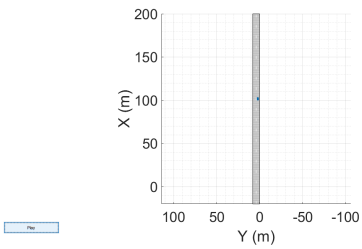


Figure 4.19: Mid-maneuver near $Y_v = 2$ m.

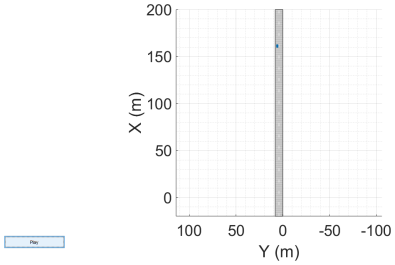


Figure 4.20: End of the maneuver returning to $Y_v = 6$ m.

4.2.3 Step 3: Adaptive MPC Implementation

4.2.3.1 Objective

The goal of this third step was to address the limitations of the previous linear MPC controller used in Step 1 (4.2.1) and Step 2 (4.2.2) by implementing an Adaptive MPC capable of tracking a more complex reference trajectory. This implementation was directly based on an example provided by the MATLAB and Simulink Racing Lounge [30], which was carefully studied, understood, and adapted to fit the needs of this project. The objective was to evaluate its performance in a dynamic scenario such as the laboratory racing track, where the operating conditions vary depending on road features, requiring continuous updates of the internal model to maintain accurate control.

4.2.3.2 Vehicle Dynamic Model and State-Space Description

For the design of the closed-loop system with the adaptive model predictive controller, two distinct representations of the vehicle model are considered:

- **Internal Prediction Model:** The simplified model used by the MPC controller for online optimization and prediction.
- **External Plant Model:** The high-fidelity nonlinear vehicle model used in the simulation environment to represent the real system behavior.

This separation highlights the distinction between the model available to the controller for prediction and control, and the simulated vehicle behavior, which evolves independently of the controller's internal representation. The Adaptive MPC updates its internal model parameters online to reduce the mismatch between these two models.

Internal Prediction Model

For the design of the Adaptive MPC used within the `Path Following Control System` block in Simulink [31], the internal predictive model of the vehicle combines two state-space representations, one for longitudinal (Adaptive Cruise Control) and one for lateral (Lane Keeping) dynamics.

This dual-model structure shown in Figure 4.5 and presented in [31] allows the controller to simultaneously regulate vehicle speed and steering in order to follow a desired path while maintaining stability and control. Note that in this internal model, suspension dynamics, tire slip saturation effects, and aerodynamic forces are neglected in order to simplify both the longitudinal and lateral vehicle dynamics representations. Also, unlike the linear MPC controller used in Step 1 (4.2.1) and Step 2 (4.2.2), where the longitudinal velocity was assumed constant, here the longitudinal dynamics are explicitly modeled, allowing them to vary as part of the vehicle's state.

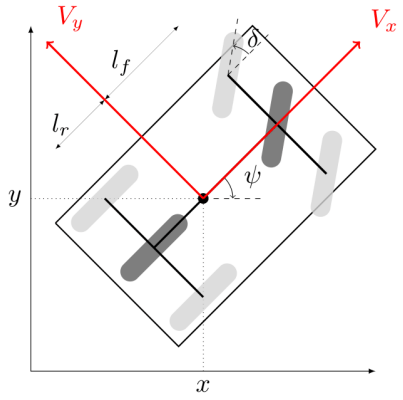


Figure 4.21: Internal prediction vehicle model.

In this instance, X_v and Y_v represent the position of the vehicle in the global coordinate system, V_x and V_y correspond to the longitudinal and lateral velocities in the body-fixed coordinate system, measured with respect to the vehicle's center of gravity. Lastly, the variable ψ denotes the yaw angle of the vehicle, and δ is the front steering angle applied to control its direction.

Longitudinal Model

The longitudinal behavior of the vehicle is represented by a second-order linear model describing the vehicle's acceleration and velocity dynamics. In this model, a_x represents the longitudinal acceleration of the vehicle in the vehicle body-fixed coordinate system, while V_x denotes the longitudinal velocity measured at the vehicle's center of gravity. The variable $a_{x,\text{cmd}}$ denotes the commanded longitudinal acceleration provided by the controller.

The continuous-time linear longitudinal dynamics of the vehicle can be described by the following equations:

$$\dot{a}_x = -\frac{1}{\tau}a_x + \frac{1}{\tau}a_{x,\text{cmd}} \tag{4.33}$$

$$\dot{V}_x = a_x \tag{4.34}$$

To model the system in state-space form, the state, input, and output variables are defined as:

$$x_1 = \begin{bmatrix} a_x \\ V_x \end{bmatrix}, \quad u_1 = \begin{bmatrix} a_{x,\text{cmd}} \end{bmatrix}, \quad y_1 = \begin{bmatrix} V_x \end{bmatrix} \tag{4.35}$$

Thus, the state vector includes the longitudinal acceleration a_x and the longitudinal velocity V_x, the input variable is the commanded longitudinal acceleration $a_{x,\text{cmd}}$ generated by the controller, and the output corresponds to the longitudinal velocity V_x.

The continuous-time linear system can be expressed in standard state-space form as:

$$\dot{x}_1 = A_1 x_1 + B_1 u_1 \tag{4.36}$$

$$y_1 = C_1 x_1 + D_1 u_1 \tag{4.37}$$

where the system matrices are:

$$\begin{aligned} A_1 &= \begin{bmatrix} -\frac{1}{\tau} & 0 \\ 1 & 0 \end{bmatrix}, \quad B_1 = \begin{bmatrix} \frac{1}{\tau} \\ 0 \end{bmatrix}, \\ C_1 &= \begin{bmatrix} 0 & 1 \end{bmatrix}, \quad D_1 = \begin{bmatrix} 0 \end{bmatrix} \end{aligned} \tag{4.38}$$

in which τ is the longitudinal acceleration tracking time constant parameter.

Lateral Model

The lateral motion and yaw dynamics of the vehicle are modeled using a simplified bicycle model. This second-order linear system describes the vehicle's response to steering commands in terms of both lateral motion and yaw rotation. In this formulation, V_y represents the lateral velocity of the vehicle at its center of gravity, and $\dot{\psi}$ denotes the yaw rate, describing the rate of change of the yaw angle. The variable δ is the steering angle, which is applied to control the vehicle's direction.

The continuous-time linear lateral dynamics of the vehicle can be described by the following equations:

$$\dot{V}_y = -\frac{2(C_f + C_r)}{mV_x} V_y + \left(-V_x - \frac{2(C_f l_f - C_r l_r)}{mV_x} \right) \dot{\psi} + \frac{2C_f}{m} \delta \tag{4.39}$$

$$\ddot{\psi} = -\frac{2(C_f l_f - C_r l_r)}{I_z V_x} V_y - \frac{2(C_f l_f^2 + C_r l_r^2)}{I_z V_x} \dot{\psi} + \frac{2C_f l_f}{I_z} \delta \tag{4.40}$$

To model the system in state-space form, the state, input, and output variables are:

$$x_2 = \begin{bmatrix} V_y \\ \dot{\psi} \end{bmatrix}, \quad u_2 = \begin{bmatrix} \delta \end{bmatrix}, \quad y_2 = \begin{bmatrix} V_y \\ \dot{\psi} \end{bmatrix}. \tag{4.41}$$

Correspondingly, the state variables are the lateral velocity V_y and the yaw rate $\dot{\psi}$. The input variable is the steering angle δ provided by the controller, and both states are also considered as the output variables.

The continuous-time linear system can be expressed in standard state-space form as:

$$\dot{x}_2 = A_2 x_2 + B_2 u_2 \tag{4.42}$$

$$y_2 = C_2 x_2 + D_2 u_2 \tag{4.43}$$

where the system matrices are:

$$
\begin{aligned}
A_2 &= \begin{bmatrix} -\frac{2(C_f+C_r)}{mV_x} & -V_x - \frac{2(C_f l_f - C_r l_r)}{mV_x} \\ -\frac{2(C_f l_f - C_r l_r)}{I_z V_x} & -\frac{2(C_f l_f^2 + C_r l_r^2)}{I_z V_x} \end{bmatrix}, \\
B_2 &= \begin{bmatrix} \frac{2C_f}{m} \\ \frac{2C_f l_f}{I_z} \end{bmatrix}, \quad C_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad D_2 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}
\end{aligned}
\tag{4.44}
$$

The parameters of this model include the total vehicle mass m, yaw moment of inertia I_z, distances from the center of gravity to the front and rear axles l_f and l_r, as well as the front and rear tire cornering stiffness coefficients C_f and C_r. The longitudinal velocity V_x corresponds to the vehicle's instantaneous longitudinal velocity, determined by the longitudinal control system and potentially varying according to the driving scenario or simulation input.

Combined Model

Following the formulation of the longitudinal and lateral vehicle dynamics separately, these two subsystems are combined into a single state-space model. This model describes the complete motion of the vehicle in both the longitudinal and lateral directions and serves as the internal predictive model for the path-following controller.

The combined continuous-time linear system can be represented by the following standard state-space equations:

$$\dot{x} = Ax + Bu \tag{4.45}$$

$$y = Cx + Du \tag{4.46}$$

where the system matrices are constructed as:

$$
\begin{aligned}
A &= \begin{bmatrix} A_1 & 0 \\ 0 & A_2 \end{bmatrix}, \quad B = \begin{bmatrix} B_1 & 0 \\ 0 & B_2 \end{bmatrix}, \\
C &= \begin{bmatrix} C_1 & 0 \\ 0 & C_2 \end{bmatrix}, \quad D = \begin{bmatrix} D_1 & 0 \\ 0 & D_2 \end{bmatrix}
\end{aligned}
\tag{4.47}
$$

Here, A_1, B_1, C_1, D_1 represent the longitudinal dynamics matrices, while A_2, B_2, C_2, D_2 represent the lateral dynamics matrices derived in the previous sections. The state vector x, input vector u, and output vector y are accordingly defined as:

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \quad u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}, \quad y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \tag{4.48}$$

with x_1, u_1, y_1 referring to the longitudinal subsystem, and x_2, u_2, y_2 to the lateral subsystem.

In addition to this combined vehicle model, the internal predictive model used by the path-following controller is augmented to incorporate the road curvature as a measured disturbance

input. This augmentation allows the controller to account for the influence of the road geometry on the vehicle's future path and improve tracking accuracy.

Table 4.3 summarizes the parameters used by the controller's internal predictive model.

Table 4.3: Vehicle parameters of the internal prediction model.

Symbol	Description	Value [units]
τ	Longitudinal acceleration time constant	0.5 [s]
m	Vehicle mass	2000 [kg]
I_z	Yaw moment of inertia	4000 [kg·m^2]
l_f	Distance from CG of the vehicle to front axle	1.4 [m]
l_r	Distance from CG of the vehicle to rear axle	1.6 [m]
C_f	Front wheels cornering stiffness	12×10^3 [N/rad]
C_r	Rear wheels cornering stiffness	11×10^3 [N/rad]

This internal predictive model is used by the Adaptive MPC controller to compute optimal longitudinal acceleration and steering angle inputs, ensuring the vehicle follows the desired path. This model was implemented in MATLAB by configuring the corresponding parameters in the `Path Following Control System` block in Simulink.

External Plant Model

The external plant model consists of a high-fidelity, nonlinear vehicle representation that captures detailed vehicle dynamics. The inputs to this plant model are the longitudinal acceleration command $a_{x,\text{cmd}}$ and the steering angle δ, both provided by the controller. Since the vehicle actuators operate based on normalized control inputs rather than direct acceleration values, the longitudinal acceleration is first converted into appropriate acceleration and braking commands using a Two-Dimensional (2D) lookup table. A lookup table is a modeling technique that uses precomputed arrays of data to map input values to output values, effectively approximating mathematical functions or system behavior [32]. In this case, the table maps the requested acceleration and vehicle speed to a corresponding normalized control signal ranging from -1 (full braking) to +1 (full acceleration). This signal is then separated into distinct acceleration and deceleration commands to match the input requirements of the vehicle actuators. This conversion ensures that the controller's acceleration outputs are translated into realistic actuator level signals for the vehicle model.

This conversion is necessary because the resulting normalized command serves as the input to the `Powertrain & Driveline` block. This block includes the `Combined Slip Wheel 2DOF` component [33], which allows the model to realistically represent the interaction between the wheels and the road surface under both acceleration and braking. As a result, this block calculates the total forces acting on the front and rear wheels F_{wf} and F_{wr}, which are essential inputs for the subsequent vehicle dynamics simulation.

The calculated front and rear wheel forces F_{wf} and F_{wr}, along with the steering angle command from the controller, and the vehicle's initial conditions, including positions $X_{v,i}$ and $Y_{v,i}$, and yaw

angle ψ_i, serve as inputs to the `Vehicle Body 3DOF Dual Track` block [34], which then simulates the vehicle's longitudinal, lateral, and yaw dynamics based on these inputs. Specifically, this block models the rigid-body motion of a two-axle vehicle without considering pitch, roll, or vertical dynamics. This simplification makes it suitable for automated driving studies simulating nonholonomic vehicle behavior.

The outputs of this block and ultimately of the vehicle external model include key vehicle states such as the positions of the vehicle X_v and Y_v, the longitudinal velocity V_x, lateral velocity V_y, yaw angle ψ, and yaw rate $\dot{\psi}$, which are used as feedback for the control system. These outputs complete the simulation of the high-fidelity vehicle dynamics, ensuring realistic feedback for controller performance evaluation.

4.2.3.3 Reference Trajectory Definition

In contrast to the previous steps, where the reference trajectories were synthetically generated either as a simple step profile (4.2.1) or designed scenarios from DSD (4.2.2), the reference trajectory in this case is derived directly from the path planned in Chapter 3. This path represents the trajectory computed by the path planner for the specific track layout.

In addition to the path itself, a velocity reference profile is provided via a 2D lookup table. This table was manually created for the specific track in use, with the intention of adjusting the vehicle's speed depending on its position along the path. For example, the table specifies higher target speeds on straight segments and lower speeds in curved sections to ensure stability and improve tracking performance.

This lookup table was constructed in an Excel spreadsheet and imported into MATLAB as a data file. During the simulation, the table provides the reference velocity to the controller based on the vehicle's current position along the track, as determined by the high-fidelity external plant model.

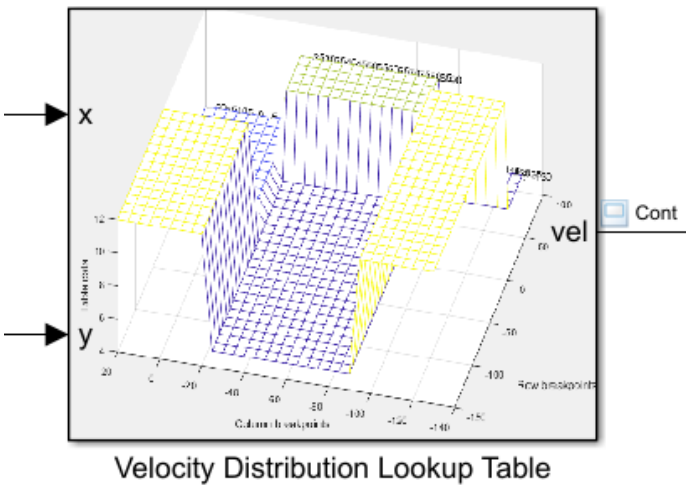


Figure 4.22: Simulink Lookup Table block.

This velocity reference serves as an important input to the controller, complementing the path position reference and enabling speed adaptation according to track geometry and driving strategy.

4.2.3.4 MPC Controller Design and Tuning

For this final step, tuning the model to suit our needs was critical. For that, the `Path Following Control System` block provided by [30] was fine-tuned. Its internal structure was left unmodified, but key controller parameters were tuned to suit the reference trajectory and velocity profile previously defined.

First, the initial longitudinal velocity of the vehicle model was adjusted to match the starting velocity specified in the reference velocity lookup table, ensuring consistency between the vehicle state and the controller's expectations. The sample time of the MPC controller was set to $T_s = 0.2\,\text{s}$, chosen to balance computational load with adequate prediction resolution along the track. The prediction and control horizons were tuned to $N_p = 25$ and $N_c = 5$, respectively. This configuration provided the controller with sufficient foresight to anticipate and respond to changes in the track geometry and target velocity profile.

Regarding constraints, the default limits provided by the example model were retained, as they aligned well with the vehicle's dynamic capabilities and the requirements of the scenario. Specifically, the steering angle was constrained to $\pm 0.78\,\text{rad}$, and the longitudinal acceleration was limited to $\pm 3\,\text{m/s}^2$. The weights in the cost function were also kept at their original values from the example model, as this configuration delivered satisfactory tracking performance without excessive control effort. These values included a velocity tracking weight of 1.5, a lateral deviation weight of 1.5, a weight of 0.5 on the change of longitudinal acceleration, and a weight of 2.0 on the change of steering angle.

With these tuning adjustments, the MPC controller was able to follow the reference trajectory generated in Chapter 3 effectively, adapting the vehicle's speed and lateral position according to the curvature and velocity profile of the track. Simulation results confirmed that the control actions remained smooth and all constraints were satisfied throughout the maneuver.

Figure 4.23 shows the Simulink schematic used. At the beginning of the schematic, the lookup table maps the reference velocity according to the vehicle's position on the track. In light pink, the controller block is shown, representing the Adaptive MPC block that uses the internal prediction model. In light blue, the vehicle model representing the external plant is visible, followed by various display options in light green, which serve to evaluate the controller's performance.

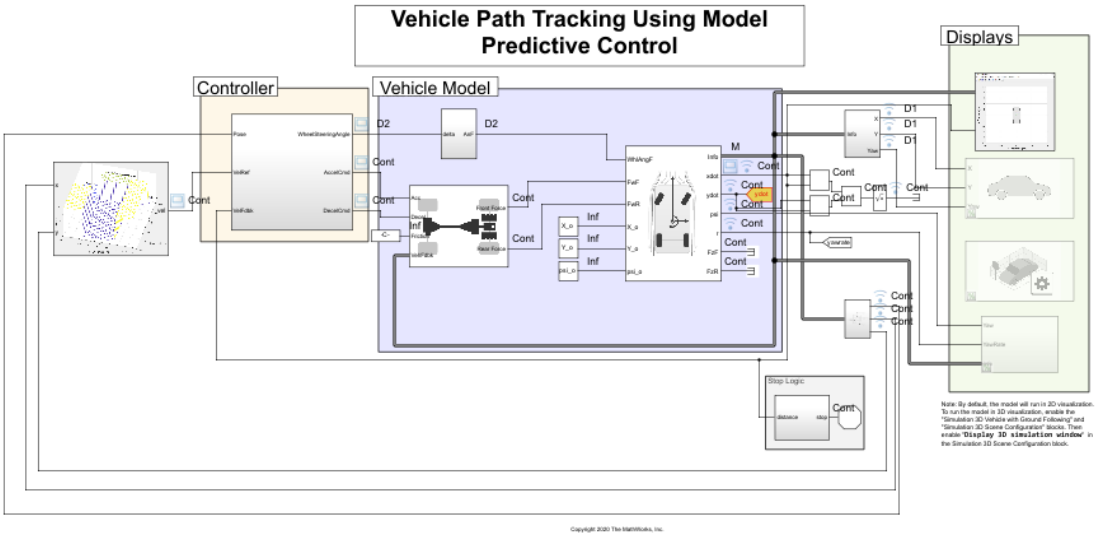


Figure 4.23: Simulink model used in Step 3 implementation [30].

The performance of this controller, applied to track the trajectories generated by the APF path planner described in Chapter 3, will be assessed in Chapter 5, where the complete integration of path planning and control is presented.

Chapter 5

Integration of Path Planning and Control

This chapter presents the integration of the hybrid approach with the path planning method using the APF, described in Chapter 3 and the MPC strategy discussed in Chapter 4, specifically the one reviewed in 4.2.3. The objective is to evaluate the controller's performance in accurately following the previously generated reference trajectories while ensuring dynamic feasibility and stability. Thus, various simulation results are provided to evaluate the system performance.

5.1 Test Scenarios

To evaluate the integrated path planning and control framework, two different simulation scenarios were tested. These scenarios are designed to assess the controller's performance under different reference trajectories and obstacle configurations, with a specific velocity profile for each.

- **Scenario 1: Simpler Track**
  In this scenario, the vehicle follows a trajectory shown in Figure 3.35 generated for a simpler track environment.
- **Scenario 2: Laboratory Track**
  In this scenario, the vehicle follows a trajectory on the laboratory track shown in Figure 3.33. This scenario serves as the main reference case because it tests the system's ability to follow a reference trajectory with obstacle avoidance in the laboratory track.

These scenarios collectively validate the ability of the controller to handle different paths and obstacle configurations.

5.2 Results and Discussion

This section presents and discusses the performance of the proposed controller in two distinct scenarios of increasing complexity. Scenario 1 (5.2.1) corresponds to a simpler reference trajectory,

while Scenario 2 (5.2.2) follows a reference trajectory projected in the laboratory track, which presents a greater challenge due to its sharper curves.

5.2.1 Scenario 1: Simpler Track

Figure 5.1 shows the reference trajectory (—) generated by the path planning algorithm compared to the trajectory followed by the vehicle (○). The vehicle successfully tracks the reference path with minor deviations.

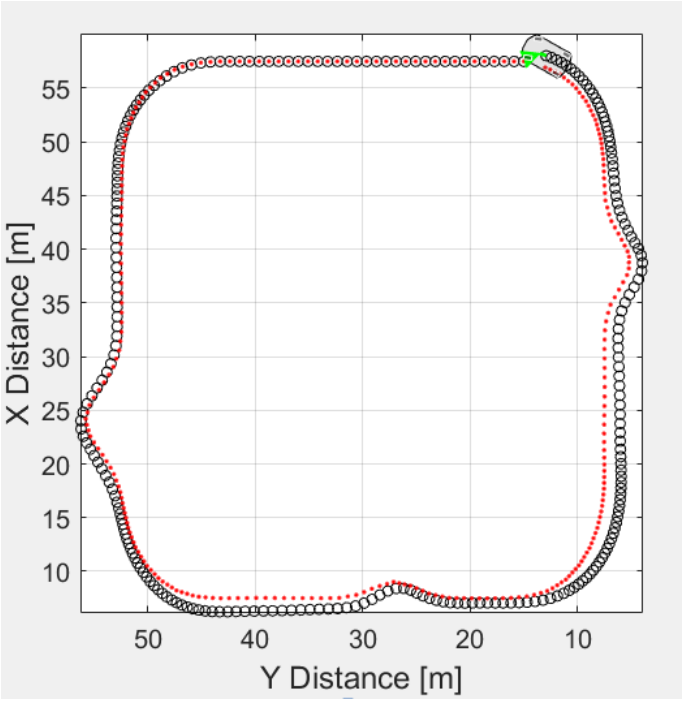


Figure 5.1: Path tracking: Vehicle position vs. Reference trajectory in Scenario 1.

The steering input δ applied by the Adaptive MPC controller to maintain this trajectory is shown in Figure 5.2.

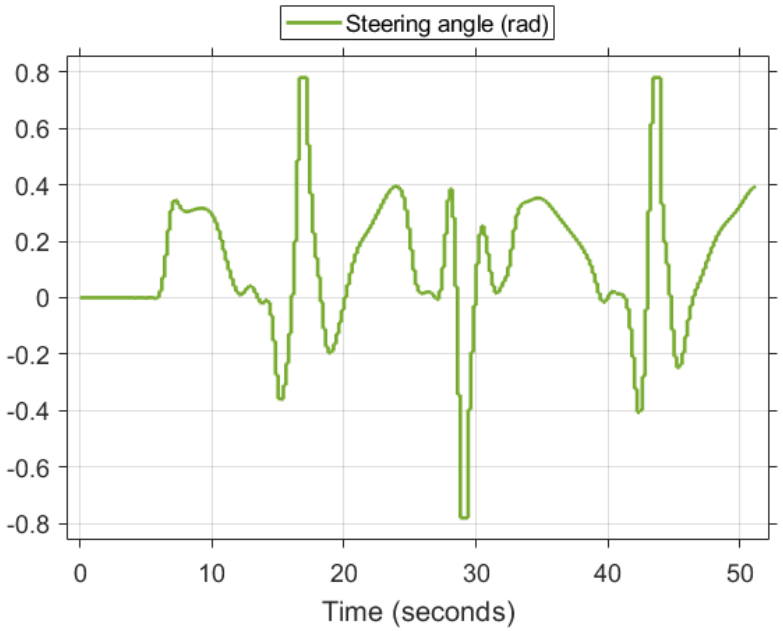


Figure 5.2: Steering angle δ applied by the controller in Scenario 1.

The longitudinal acceleration command $a_{x,\text{cmd}}$ generated by the controller is presented in Figure 5.3. As expected, higher peaks correspond to the straight segments of the track, while lower peaks occur in the curved sections where reduced velocity is required.

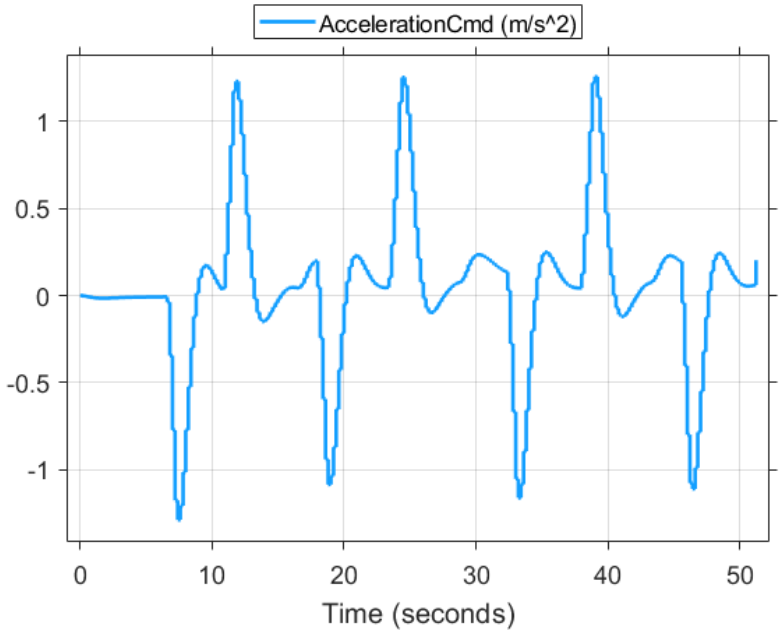


Figure 5.3: Longitudinal acceleration command $a_{x,\text{cmd}}$ applied by the controller in Scenario 1.

Figures 5.4 and 5.5 illustrate the excellent tracking of the vehicle's longitudinal and lateral positions, denoted by X_v and Y_v, respectively.

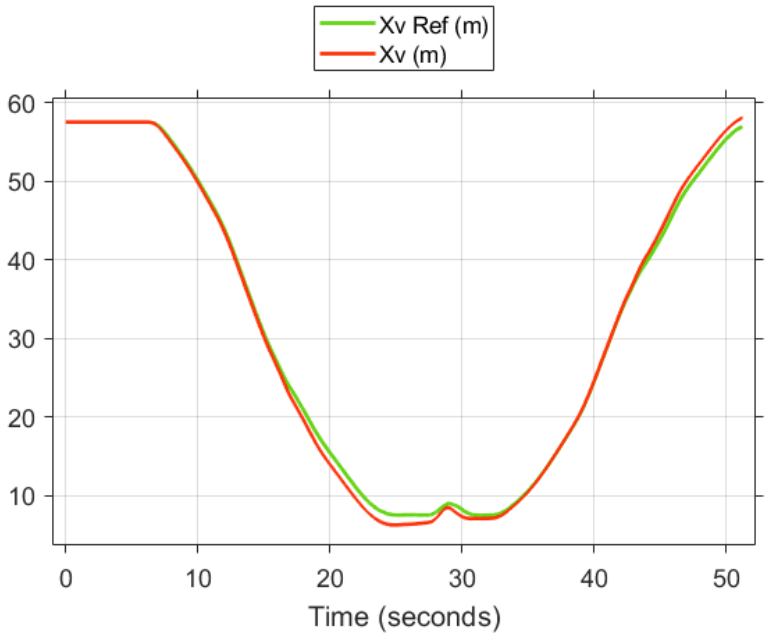


Figure 5.4: Longitudinal position tracking: X_v vs. $X_{v,\text{ref}}$ in Scenario 1.

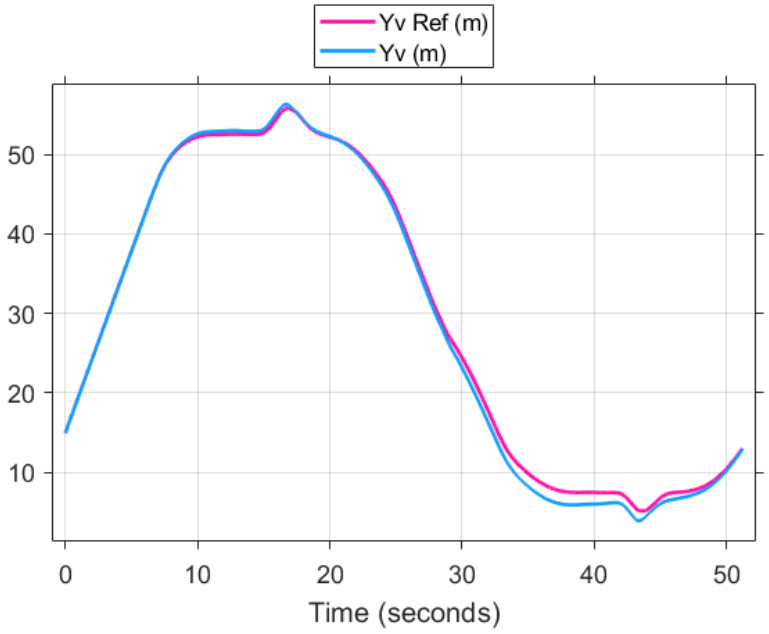


Figure 5.5: Lateral position tracking: Y_v vs. $Y_{v,\text{ref}}$ in Scenario 1.

The vehicle velocity V, defined as the norm of the velocity vector, $V = \sqrt{V_x^2 + V_y^2}$, closely follows the reference velocity profile, as seen in Figure 5.6. The vehicle is commanded to maintain

a speed of approximately 4.5 m/s on straight sections and reduce to 3 m/s when navigating curves.

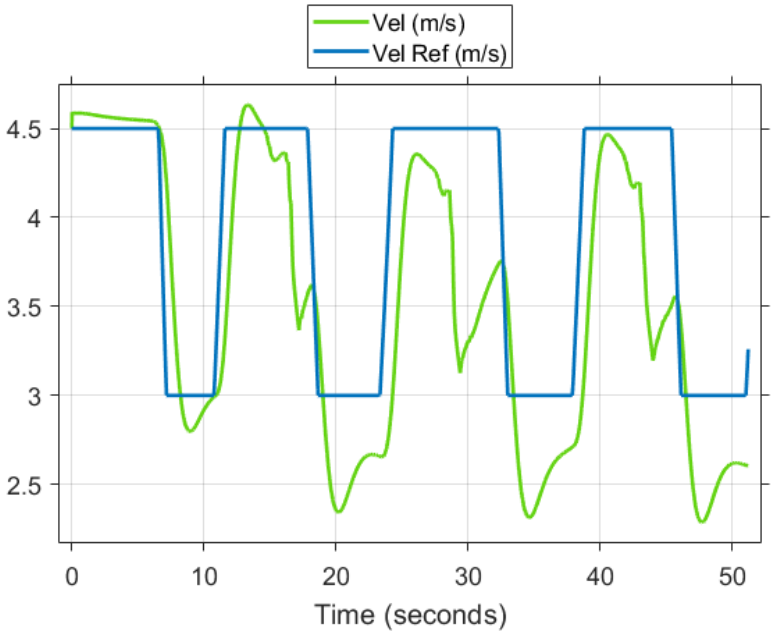


Figure 5.6: Velocity tracking: V vs. V_{ref} in Scenario 1.

Figures 5.7 and 5.8 present the tracking performance for the yaw angle ψ and the yaw rate $\dot{\psi}$, respectively. It is important to note that the yaw angle ψ continuously increases and exceeds 2π radians, as the wrap angle option in Simulink was intentionally disabled to avoid discontinuities in the plotted signals.

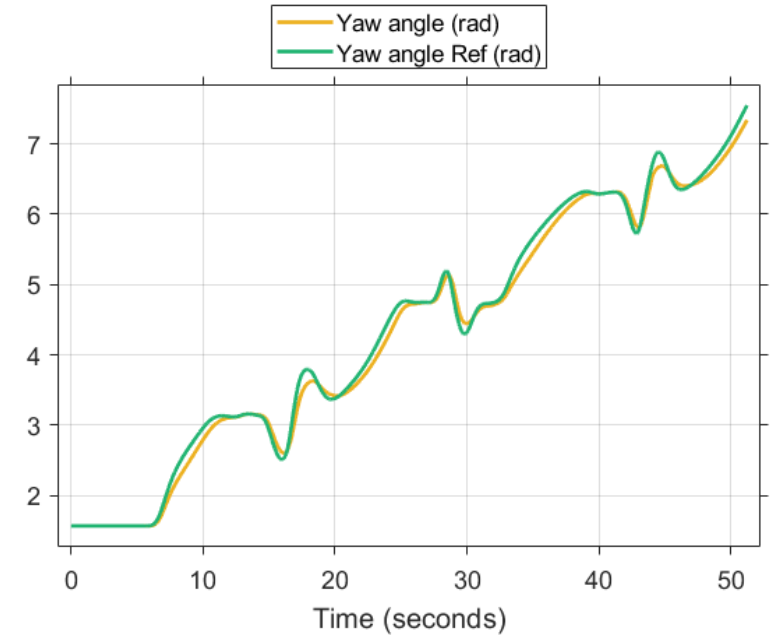


Figure 5.7: Yaw angle tracking: ψ vs. ψ_{ref} in Scenario 1.

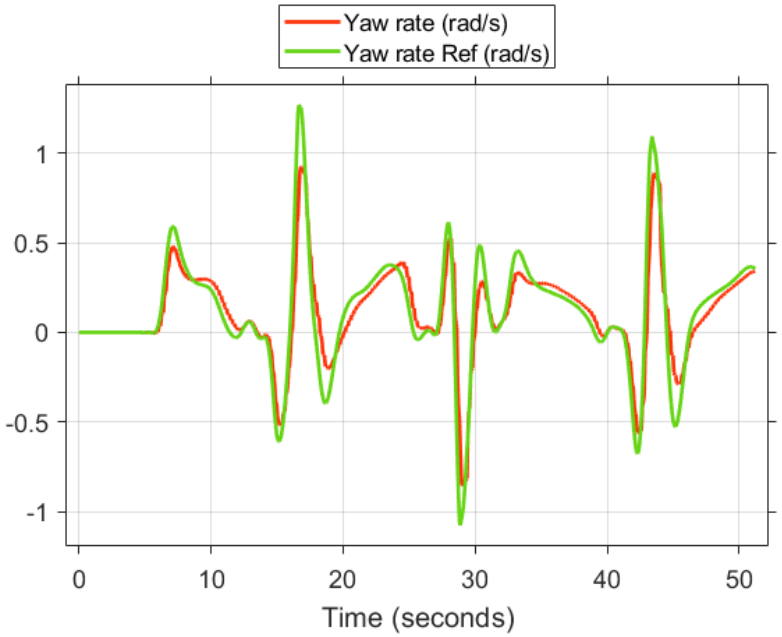


Figure 5.8: Yaw rate tracking: $\dot{\psi}$ vs. $\dot{\psi}_{\text{ref}}$ in Scenario 1.

To quantitatively evaluate lateral deviation, the normalized lateral tracking error percentage is calculated as:

$$\text{Lateral Tracking Error (\%)} = \frac{|Y_{v,\text{ref}} - Y_v|}{\max(Y_{v,\text{ref}})} \times 100 \tag{5.1}$$

where $Y_{v,\text{ref}}$ is the reference lateral position, Y_v denotes the actual lateral position of the vehicle, and $\max(Y_{v,\text{ref}})$ is the maximum reference lateral position value along the track.

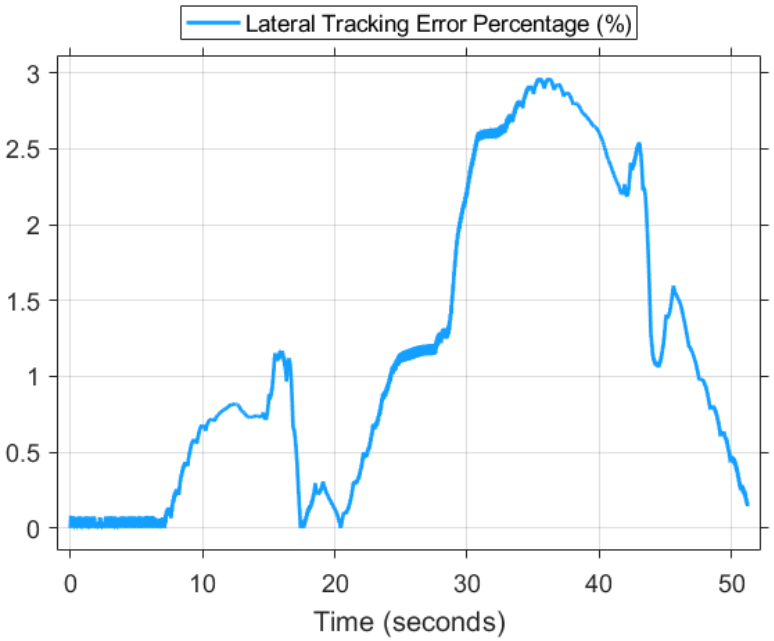


Figure 5.9: Lateral tracking error (%) in Scenario 1.

The lateral error shows a slight increase towards the end of the trajectory, however, this trend reverses as the error begins to decrease again, likely due to the Adaptive MPC controller responding and compensating for the accumulated deviation.

5.2.2 Scenario 2: Laboratory Track

Figure 5.10 shows the reference trajectory (—) generated by the path planning algorithm compared to the trajectory followed by the vehicle (○). The vehicle follows the desired path with good accuracy, although slightly larger deviations are observed compared to Scenario 1 due to the increased complexity of the trajectory.

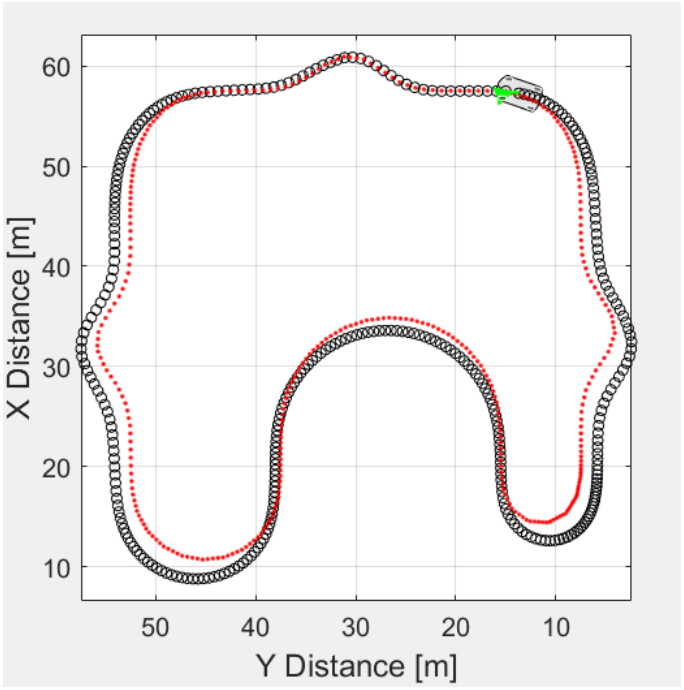


Figure 5.10: Path tracking: Vehicle position vs. Reference trajectory in Scenario 2.

The steering input δ applied by the Adaptive MPC controller to perform these maneuvers is shown in Figure 5.11.

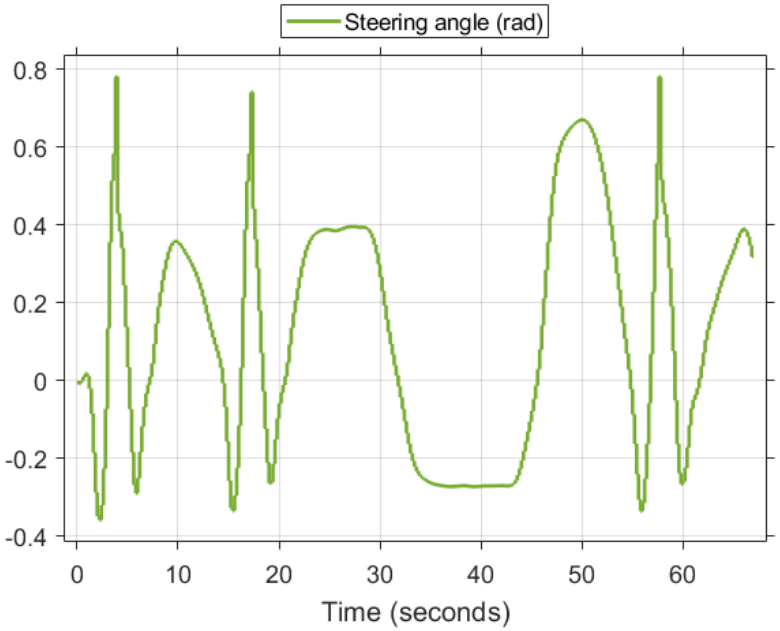


Figure 5.11: Steering angle δ applied by the controller in Scenario 2.

The longitudinal acceleration command $a_{x,\text{cmd}}$ generated by the controller is presented in Figure 5.12. As expected, the controller accelerates in the straights and decelerates in the curves.

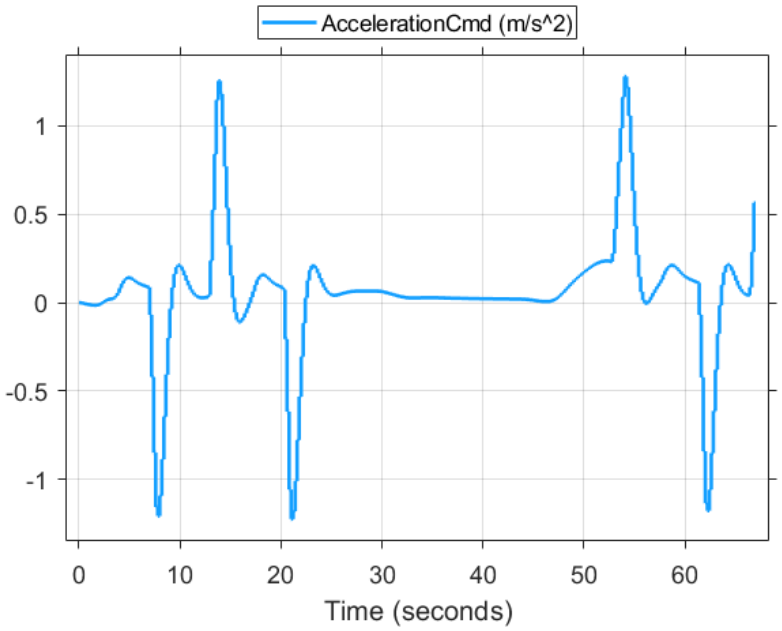


Figure 5.12: Longitudinal acceleration command $a_{x,\text{cmd}}$ applied by the controller in Scenario 2.

Figures 5.13 and 5.14 demonstrate satisfactory tracking of the vehicle's longitudinal and lateral positions, denoted by X_v and Y_v, respectively. The actual positions closely match the reference values, confirming the controller's reliability also in this scenario.

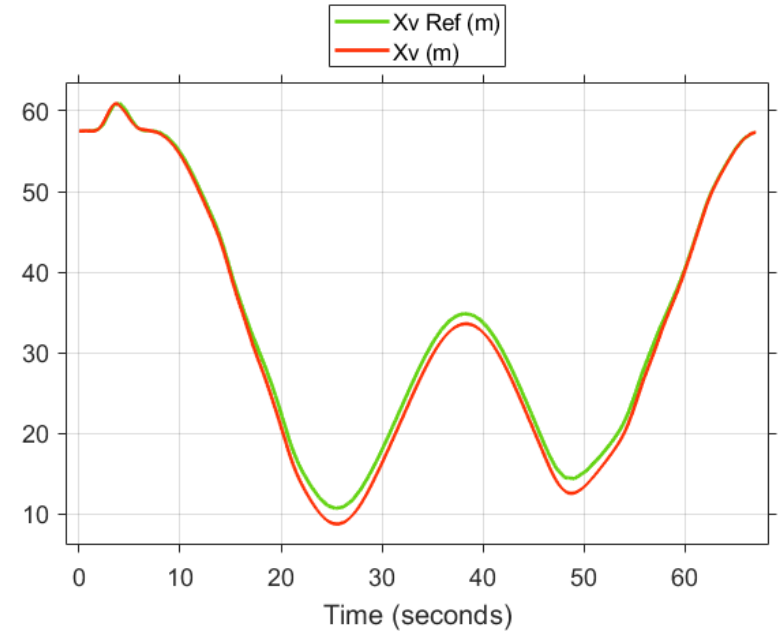


Figure 5.13: Longitudinal position tracking: X_v vs. $X_{v,\text{ref}}$ in Scenario 2.

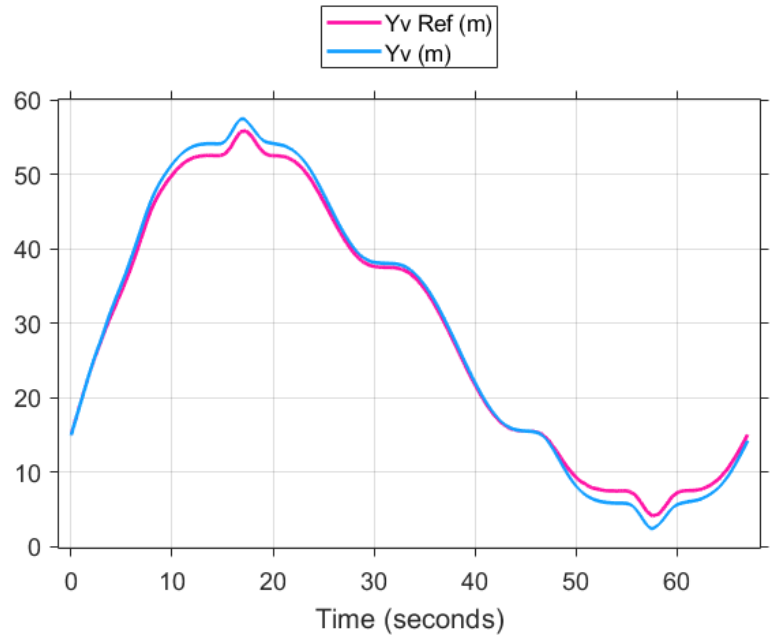


Figure 5.14: Lateral position tracking: Y_v vs. $Y_{v,\text{ref}}$ in Scenario 2.

Figure 5.15 presents the vehicle velocity magnitude V compared to the reference velocity profile. The vehicle accurately follows the predefined speed targets: 4.5 m/s on straight segments and 3 m/s in the curved sections, as dictated by the lookup table.

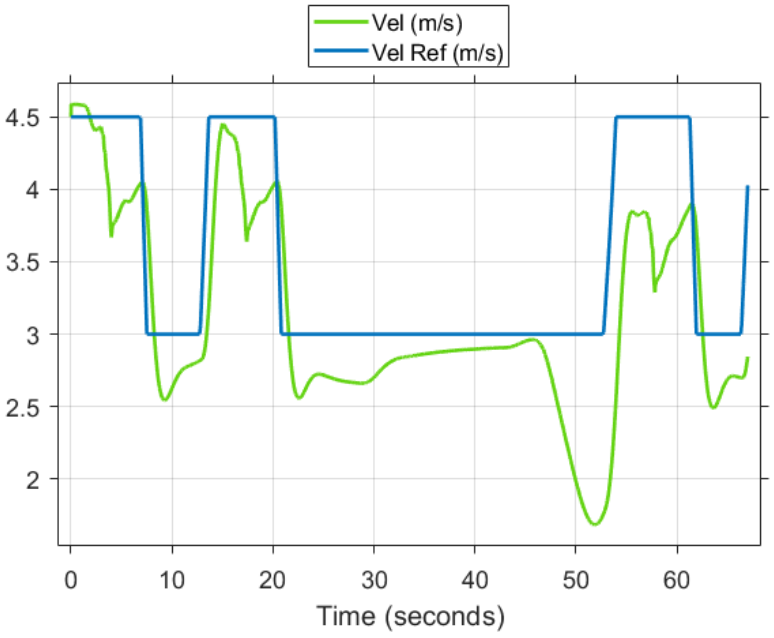


Figure 5.15: Velocity tracking: V vs. V_{ref} in Scenario 2.

Figures 5.16 and 5.17 show the yaw angle ψ and the yaw rate $\dot{\psi}$ tracking performance, respectively. As in Scenario 1, the yaw angle ψ exceeds 2π radians since the wrap angle option in Simulink was intentionally disabled to prevent discontinuities in the plot.

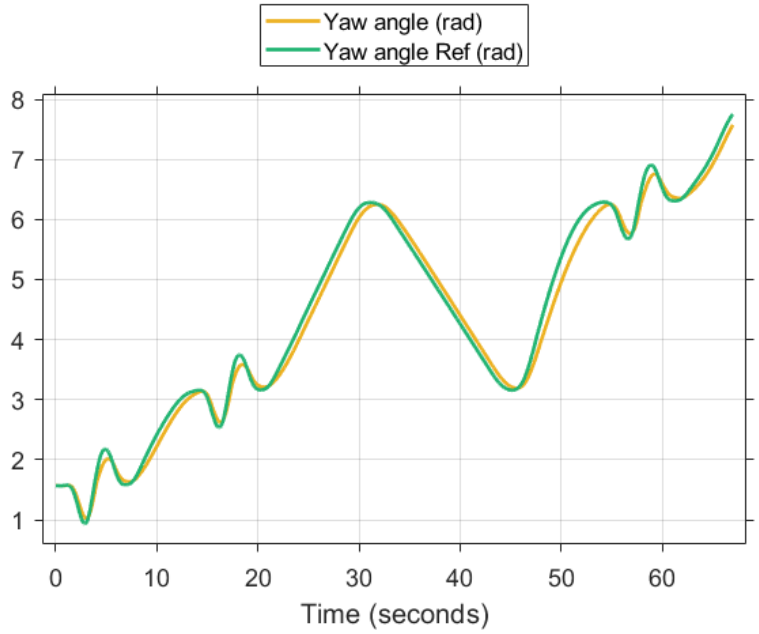


Figure 5.16: Yaw angle tracking: ψ vs. ψ_{ref} in Scenario 2.

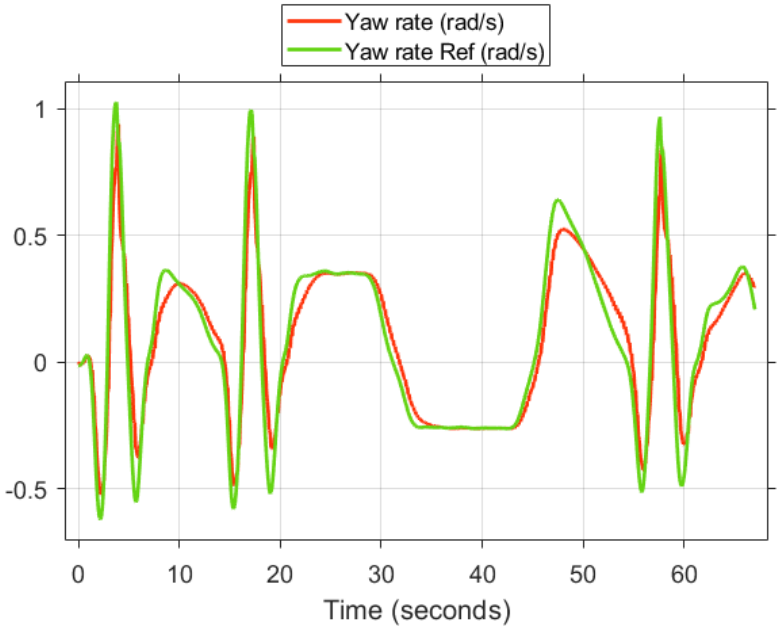


Figure 5.17: Yaw rate tracking: $\dot{\psi}$ vs. $\dot{\psi}_{\text{ref}}$ in Scenario 2.

To quantitatively evaluate the lateral deviation, the normalized lateral tracking error percentage

is calculated using the equation 5.1, where $Y_{v,\text{ref}}$ denotes the reference lateral position, Y_v the actual lateral position of the vehicle, and $\max(Y_{v,\text{ref}})$ the maximum reference lateral position value along the track.

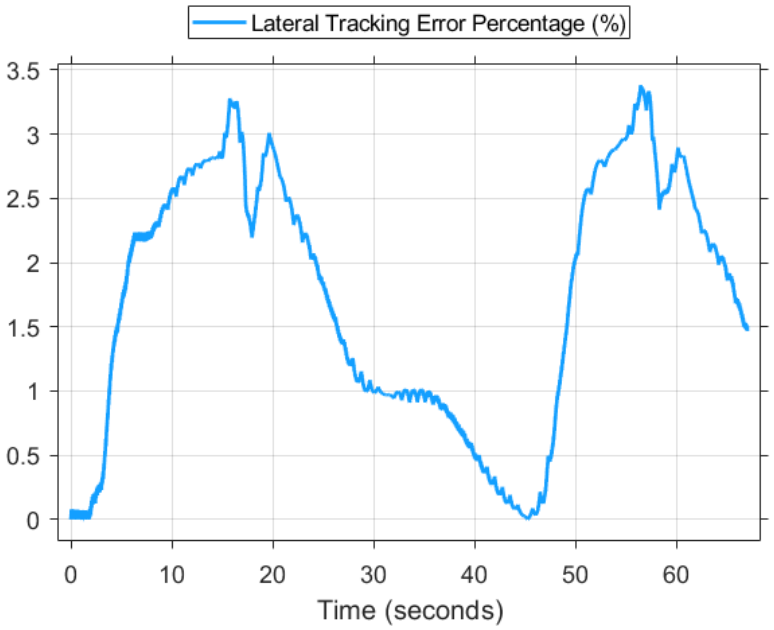


Figure 5.18: Lateral tracking error (%) in Scenario 2.

In Scenario 2, the lateral tracking error slightly increases, exceeding 3%, compared to Scenario 1 (5.2.1), where it remained below this threshold. The maximum error reaches approximately 3.4%, which can be attributed to the more complex and dynamic nature of the laboratory track trajectory compared to the simpler track. Despite this, the controller still demonstrates great overall tracking accuracy, proving its capability to accurately follow challenging, collision-free trajectories.

Chapter 6

Conclusions and Future Work

In this study, we presented the development and simulation of a hybrid collision avoidance system for autonomous racing vehicles, combining APF with MPC. The proposed method was designed to generate collision-free reference trajectories across a variety of racing scenarios and track conditions. The work focused on the integration of these two techniques into a simulation framework, and evaluated their interaction under different scenarios, laying the groundwork for further development and real-world implementation.

The combination of APF and MPC leverages the complementary strengths of both techniques, APF provides a computationally efficient way to generate reactive, collision-free trajectories, while MPC enables predictive control that respects vehicle dynamics and ensures smooth path following. The proposed method was tested across different simulated track layouts and obstacle configurations, allowing the evaluation of its ability to generate feasible trajectories and assess the controller's capacity to follow them with reliability.

The simulation results confirm the successful integration of the APF path planning algorithm with the Adaptive MPC controller, enabling robust and reliable path tracking in both test scenarios. While the controller maintained good performance throughout, a slight reduction in tracking accuracy was observed in the second scenario. This can be attributed to the increased curvature and variability of the laboratory track, which posed more demanding dynamic challenges compared to the simpler reference path. Despite this, the overall system remained stable and effective, validating the suitability of the proposed approach for real-time autonomous navigation tasks.

Building on these promising simulation results, several important avenues for future work have been identified to advance the system toward practical deployment and enhanced performance. These include:

1. **Integration with the Perception System**
   The current system uses track and obstacle data defined by the user in the simulation. To enhance its real-world applicability, the perception module must be integrated to provide real-time obstacle detection and environment mapping using the camera present in the laboratory. This would enable the APF planner to dynamically adjust the reference trajectory based on real-time data, making the system reactive to changes in the environment.

2. **Adaptation of the MPC Vehicle Model for Laboratory-Scale Cars**
   The current MPC controller uses a plant model based on a full-scale vehicle. However, the laboratory setup involves 1:43 scale RC cars, whose dynamics, velocity limits, and response characteristics differ significantly. Future work should involve adapting the plant model to reflect the behavior of these smaller cars, ensuring that the controller respects their physical constraints, thereby ensuring meaningful and safe simulations that reflect the limitations and capabilities of the actual test platform.

3. **Hardware Integration**
   The control and planning algorithms are currently simulated on a high-level computing environment (MATLAB/Simulink). For practical deployment, the entire control system needs to be implemented on embedded hardware platforms, such as the ESP32 S2 Mini and the custom PCB used to control the laboratory race cars. This step requires optimization of the computational load, real-time execution testing, and validation under laboratory conditions, ensuring that the algorithms meet timing and resource constraints inherent to embedded systems.

The outlined future steps have the potential to push the limits of autonomous racing technology, bringing the system closer to real-world application and experimentation.

References

[1] United Nations. *Take Action for the Sustainable Development Goals*. en-US. URL: https://www.un.org/sustainabledevelopment/sustainable-development-goals/ (visited on 05/04/2025).

[2] United Nations. *Sustainable Development*. en. Publisher: United Nations. URL: https://www.un.org/en/teach/SDGs (visited on 05/04/2025).

[3] Johannes Betz et al. "Autonomous Vehicles on the Edge: A Survey on Autonomous Vehicle Racing". en. In: *IEEE Open Journal of Intelligent Transportation Systems* 3 (2022), pp. 458–488. ISSN: 2687-7813. DOI: 10.1109/OJITS.2022.3181510. URL: https://ieeexplore.ieee.org/document/9790832/ (visited on 10/25/2024).

[4] Joseph Funke et al. "Collision Avoidance and Stabilization for Autonomous Vehicles in Emergency Scenarios". en. In: *IEEE Transactions on Control Systems Technology* 25.4 (July 2017), pp. 1204–1216. ISSN: 1063-6536, 1558-0865. DOI: 10.1109/TCST.2016.2599783. URL: http://ieeexplore.ieee.org/document/7585053/ (visited on 01/12/2025).

[5] Jie Ji et al. "Path Planning and Tracking for Vehicle Collision Avoidance Based on Model Predictive Control With Multiconstraints". en. In: *IEEE Transactions on Vehicular Technology* 66.2 (Feb. 2017), pp. 952–964. ISSN: 0018-9545, 1939-9359. DOI: 10.1109/TVT.2016.2555853. URL: http://ieeexplore.ieee.org/document/7458179/ (visited on 10/25/2024).

[6] Xiaohua Song et al. "A Review of the Motion Planning and Control Methods for Automated Vehicles". en. In: *Sensors* 23.13 (July 2023), p. 6140. ISSN: 1424-8220. DOI: 10.3390/s23136140. URL: https://www.mdpi.com/1424-8220/23/13/6140 (visited on 10/25/2024).

[7] Shih-Lin Lin and Bo-Chen Lin. "Enhancing Safety in Autonomous Vehicle Navigation: An Optimized Path Planning Approach Leveraging Model Predictive Control". en. In: *Computers, Materials & Continua* 80.3 (2024), pp. 3555–3572. ISSN: 1546-2226. DOI: 10.32604/cmc.2024.055456. URL: https://www.techscience.com/cmc/v80n3/57921 (visited on 10/25/2024).

[8] Benjamin Evans, Herman A. Engelbrecht, and Hendrik W. Jordaan. "Reward Signal Design for Autonomous Racing". en. In: *2021 20th International Conference on Advanced Robotics (ICAR)*. Ljubljana, Slovenia: IEEE, Dec. 2021, pp. 455–460. ISBN: 978-1-6654-3684-7. DOI: 10.1109/ICAR53236.2021.9659438. URL: https://ieeexplore.ieee.org/document/9659438/ (visited on 11/02/2024).

[9] Howie Choset et al. "Potential Functions". In: *Principles of Robot Motion: Theory, Algorithms, and Implementations*. 2005, pp. 77–106.

[10] Ryuzo Hayashi et al. "Autonomous collision avoidance system by combined control of steering and braking using geometrically optimised vehicular trajectory". en. In: *Vehicle System Dynamics* 50.sup1 (Jan. 2012), pp. 151–168. ISSN: 0042-3114, 1744-5159. DOI: 10.1080/00423114.2012.672748. URL: http://www.tandfonline.com/doi/abs/10.1080/00423114.2012.672748 (visited on 01/12/2025).

[11] Duarte Lima Silva. "A Digital Twin Framework for Autonomous Racing". Supervisor: Luís Tiago de Freixo Ramos Paiva. MSc Thesis. Porto, Portugal: Faculdade de Engenharia da Universidade do Porto, 2025.

[12] *Driving Scenario Designer - Design driving scenarios, configure sensors, and generate synthetic data - MATLAB*. en. URL: https://www.mathworks.com/help/releases/R2024b/driving/ref/drivingscenariodesigner-app.html (visited on 05/15/2025).

[13] *Automated Driving Toolbox Documentation*. en. URL: https://www.mathworks.com/help/driving/index.html (visited on 06/15/2025).

[14] *linspace - Generate linearly spaced vector - MATLAB*. en. URL: https://www.mathworks.com/help/matlab/ref/double.linspace.html (visited on 06/19/2025).

[15] *unique - Unique values - MATLAB*. en. URL: https://www.mathworks.com/help/matlab/ref/double.unique.html (visited on 06/21/2025).

[16] *meshgrid - 2-D and 3-D grids - MATLAB*. en. URL: https://www.mathworks.com/help/matlab/ref/meshgrid.html (visited on 06/21/2025).

[17] *flip - Flip order of elements - MATLAB*. en. URL: https://www.mathworks.com/help/matlab/ref/flip.html (visited on 06/20/2025).

[18] Eric W. Weisstein. *"Ellipse."* From MathWorld–A Wolfram Web Resource. URL: https://mathworld.wolfram.com/Ellipse.html (visited on 06/18/2025).

[19] Lars Grüne and Jürgen Pannek. *Nonlinear Model Predictive Control: Theory and Algorithms*. Jan. 2017. ISBN: 978-3-319-46024-6. DOI: 10.1007/978-3-319-46024-6.

[20] MathWorks. *Three Ways to Speed Up Model Predictive Controllers*. en. URL: https://www.mathworks.com/videos/series/understanding-model-predictive-control.html (visited on 05/05/2025).

[21] Luís Tiago de Freixo Ramos Paiva. "Numerical Methods for Optimal Control and Model Predictive Control". PhD Thesis. Universidade do Porto (Portugal), 2014. URL: https://search.proquest.com/openview/b5ce0ec1d37854f9620d8d2eef1e4eb8/1?pq-origsite=gscholar&cbl=2026366 (visited on 06/26/2025).

[22] J Rawlings and D.Q. Mayne. *Model Predictive Control: Theory and Design*. Jan. 2009.

[23] MathWorks. *Understanding Model Predictive Control*. URL: https://www.youtube.com/playlist?list=PLn8PRpmsu08ozoeoXgxPSBKLyd4YEHww8.

[24] Eric Hillsberg. *Model Predictive Control (MPC) virtual lab*. 2025. URL: https://github.com/MathWorks-Teaching-Resources/Model-Predictive-Control-virtual-lab (visited on 05/04/2025).

[25] *ss - State-space model - MATLAB*. en. URL: https://www.mathworks.com/help/control/ref/ss.html (visited on 06/26/2025).

[26] *MPC Designer - Design and simulate model predictive controllers - MATLAB*. en. URL: https://www.mathworks.com/help/mpc/ref/mpcdesigner-app.html (visited on 05/15/2025).

[27] *Simulation Data Inspector - Inspect and compare data and simulation results to validate and iterate model designs - MATLAB*. en. URL: https://www.mathworks.com/help/releases/R2024b/simulink/slref/simulationdatainspector.html (visited on 05/15/2025).

[28] *Sensor Simulation and Virtual Scene Design with the Driving Scenario Designer App, Part 1*. en. URL: https://www.mathworks.com/videos/driving-scenario-designer-1529302116471.html (visited on 05/16/2025).

[29] *Sensor Simulation and Virtual Scene Design with the Driving Scenario Designer App, Part 2*. en. URL: https://www.mathworks.com/videos/sensor-simulation-and-virtual-scene-design-with-the-driving-scenario-designer-app-part-2-1612441664992.html (visited on 05/16/2025).

[30] MathWorks Student Competitions Team. *Vehicle Path Tracking Using Model Predictive Control (MPC)*. 2025. URL: https://github.com/mathworks/vehicle-model-predictive-control (visited on 06/12/2025).

[31] *Path Following Control System - Simulate path-following control using adaptive model predictive controller - Simulink*. en. URL: https://www.mathworks.com/help/releases/R2024b/mpc/ref/pathfollowingcontrolsystem.html (visited on 06/13/2025).

[32] *Lookup Tables - MATLAB & Simulink*. en. URL: https://www.mathworks.com/help/simulink/lookup-tables.html (visited on 06/13/2025).

[33] *Combined Slip Wheel 2DOF - Combined slip 2DOF wheel with disc, drum, or mapped brake - Simulink*. en. URL: https://www.mathworks.com/help/vdynblks/ref/combinedslipwheel2dof.html (visited on 06/13/2025).

[34] *Vehicle Body 3DOF - 3DOF rigid vehicle body to calculate longitudinal, lateral, and yaw motion - Simulink*. en. URL: https://www.mathworks.com/help/vdynblks/ref/vehiclebody3dof.html (visited on 06/13/2025).