Trajectory Planning and Collision Avoidance for an Autonomous Racing Car

Oran Keanly
21738394

Thesis presented in partial fulfilment of the requirements for the degree of Master of Engineering (Electronic) in the Faculty of Engineering at Stellenbosch University.

Supervisor: Prof J.A.A. Engelbrecht
Department of Electrical and Electronic Engineering

December 2023

UNIVERSITEIT·STELLENBOSCH·UNIVERSITY
jou kennisvennoot • your knowledge partner

Plagiaatverklaring / *Plagiarism Declaration*

1. Plagiaat is die oorneem en gebruik van die idees, materiaal en ander intellektuele eiendom van ander persone asof dit jou eie werk is.
   Plagiarism is the use of ideas, material and other intellectual property of another's work and to present is as my own.

2. Ek erken dat die pleeg van plagiaat 'n strafbare oortreding is aangesien dit 'n vorm van diefstal is.
   I agree that plagiarism is a punishable offence because it constitutes theft.

3. Ek verstaan ook dat direkte vertalings plagiaat is.
   I also understand that direct translations are plagiarism.

4. Dienooreenkomstig is alle aanhalings en bydraes vanuit enige bron (ingesluit die internet) volledig verwys (erken). Ek erken dat die woordelikse aanhaal van teks sonder aanhalingstekens (selfs al word die bron volledig erken) plagiaat is.
   Accordingly all quotations and contributions from any source whatsoever (including the internet) have been cited fully. I understand that the reproduction of text without quotation marks (even when the source is cited) is plagiarism

5. Ek verklaar dat die werk in hierdie skryfstuk vervat, behalwe waar anders aangedui, my eie oorspronklike werk is en dat ek dit nie vantevore in die geheel of gedeeltelik ingehandig het vir bepunting in hierdie module/werkstuk of 'n ander module/werkstuk nie.
   I declare that the work contained in this assignment, except where otherwise stated, is my original work and that I have not previously (in its entirety or in part) submitted it for grading in this module/assignment or another module/assignment.

21738394 Studentenommer / *Student number*	 Handtekening / *Signature*
OD Keanly Voorletters en van / *Initials and surname*	2023/12/08 Datum / *Date*

Acknowledgements

I would like to acknowledge several people who have made it possible for me to complete this project and get to this point in my life.
Firstly, I need to thank my supervisor Japie Engelbrecht. Without you, I could never have completed half of what I did. Thank you for your patience and your guidance. Just about every comment, thought or suggestion you gave me was pure gold.
Next, I would like to thank all of my colleagues at the Electronic Systems Laboratory. They have provided me with help and encouragement when I sorely needed it. The fun times at the lab were a highlight of these past two years. I would be remiss not to take this opportunity to apologise to ID for constantly stealing your sugar, Gert for stealing your coffee, and Merrick for describing you as my mate.
I would also like to thank my friends for all of your support. I most definitely owe a great deal of my sanity to you. All of you who have listened to me vent and cheered me up when I needed it, thank you. A special mention to Louise, who has helped me not just these past two years, but a great deal longer. Thank you for supporting me through everything. It means the world to me, and I'm more grateful than I could possibly express.
Finally, thank you to my family. My brother and sister for always being ready to chat with me and help me out in a pinch. My mother and father, who are most certainly going to take the lion's share of the credit, and well deservedly. My father for constantly being interested in my research, and my mother for always checking in on me. I wouldn't be half the man I am without your support, love, and encouragement.

Abstract

English

The autonomous racing problem is to traverse a race track in the fastest time possible without any human intervention. However, there are also safety concerns, as an autonomous race car must be able to react to unexpected obstacles on the track and safely navigate around such obstacles and continue racing. This thesis develops an autonomous driving system that makes use of a hierarchical motion planner. The global motion planner functions offline and generates the optimal motion plan for the entire race track. The local collision avoidance planner functions online and generates a motion plan for a limited horizon ahead of the autonomous car when an obstacle is detected in the path of the vehicle.

A dynamic bicycle model is presented and coordinated relative to a curvilinear track model. Then the global planner formulates the motion planning problem as an optimal control problem, using the dynamic bicycle model, to minimise the lap time of the car. A Stanley controller is used to execute the motion plan.

A collision prediction system is developed which uses LiDAR scan measurements to identify and cluster objects within the track. The car is then propagated forward in time and if a collision is detected then the local planner is activated. The local planner uses a kinematic bicycle model which is presented and coordinated relative to a curvilinear track model. The local planner that is developed in this thesis is an extension of the global planner that formulates the path planning problem as an optimal control problem for a limited horizon. However, the plan is constrained to free space in the track and minimises the deviation from the global path as well as the lap time.

The proposed autonomous racing system is tested on a variety of different tracks and track elements using the F1Tenth simulation environment. Tests were performed to analyse the planning, path-tracking, and collision-avoidance capabilities of the car.

The simulation results show that the proposed autonomous racing system is able to generate and execute a minimum-time global racing plan, and is able to detect and avoid unexpected obstacles on the race track using a local collision avoidance plan. A collision avoidance success rate of 90% was achieved for single obstacles. However, the success rate reduces as the number of obstacles increases.

Afrikaans

Die outonome resiesprobleem behels die voltooi van ’n resiesbaan in die kortste moontlike tyd sonder enige menslike intervensie. Veiligheid is egter ook belangrik, aangesien ’n outonome resiesmotor in staat moet wees om te reageer op onverwagte hindernisse op die baan en dit moet kan omseil en onverstoord aanhou met die resies. Hierdie tesis ontwerp ’n outonome bestuurstelsel wat gebruik maak van ’n hiërargiese bewegingsbeplanner. Die globale bewegingsbeplanner funksioneer aflyn en genereer die optimale bewegingsplan vir die volledige resiesbaan. Die plaaslike botsingsvermydingsbeplanner funksioneer in reële tyd en genereer ’n bewegingsplan vir ’n beperkte horison voor die outonome voertuig wanneer ’n hindernis in die pad van die voertuig waargeneem word.

’n Dinamiese fietsmodel word voorgestel en gekoördineer relatief tot binne ’n kurwilineêre baanmodel. Daarna formuleer die globale beplanner die bewegingsbeplanningsprobleem as ’n optimale beheerprobleem deur gebruik te maak van die dinamiese fietsmodel om die rondetyd van die motor te minimaliseer. ’n Stanley-beheerder word geïmplementeer om die bewegingsplan uit te voer.

’n Botsingsvoorspellingsisteem is ontwikkel wat gebruik maak van LiDAR-skanderingmetings om voorwerpe op die baan te identifiseer en te groepeer. Die motor word dan voorwaarts in tyd gepropageer, en indien ’n botsing waargeneem word, word die plaaslike beplanner geaktiveer. Die plaaslike beplanner maak gebruik van ’n kinematiese fietsmodel wat voorgestel en gekoördineer word relatief tot die kurwilineêre baanmodel. Die plaaslike beplanner wat in hierdie tesis ontwikkel is, is ’n uitbreiding van die globale beplanner wat die padbeplanningsprobleem formuleer as ’n optimale beheerprobleem vir ’n beperkte horison. Die beplanning is egter beperk tot die vrye spasie op die baan en minimeer afwyking van die globale pad, sowel as die rondetyd.

Die voorgestelde outonome resiesstelsel is getoets op ’n verskeidenheid van uiteenlopende bane en baanelemente deur gebruik te maak van die F1Tenth-simulasie-omgewing. Toetse is uitgevoer om die beplanning, padvolging, en botsingsvermydingsvermoëns van die motor te analiseer.

Die simulasie resultate wys dat die voorgestelde outonome resiesstelsel in staat is om minimum-tyd globale resiesplanne te genereer en uit te voer, en in staat is om onverwagte hindernisse op die resiesbaan waar te neem en te vermy deur gebruik te maak van ’n plaaslike botsingsvermyding plan. Die botsingsvermyding stelsel behaal ’n sukseskoers van 90% vir enkele hindernisse. Die sukseskoers daal egter soos wat die aantal hindernisse vermeerder.

Contents

Declaration i

Abstract iii

List of Figures ix

List of Tables xii

Nomenclature xiii

1. Introduction 1

1.1. Background . . . 1
1.2. Project Goal and Objectives . . . 1
1.3. Research Approach and Methodology . . . 2
1.4. Scope and Limitations . . . 3
1.5. Thesis Outline . . . 4

2. Literature Review 5

2.1. Racing . . . 5
2.2. Autonomous Racing . . . 5
2.3. Autonomous Racing System Designs . . . 7
2.4. Car Models . . . 8
2.4.1. Tyre models . . . 10
2.5. Localisation and Mapping . . . 11
2.5.1. SLAM . . . 12
2.5.2. Localisation . . . 13
2.6. Motion and Path Planning . . . 14
2.6.1. Global Planning . . . 14
2.6.2. Local Planning . . . 16
2.6.3. Alternate Obstacle Avoidance Methods . . . 17
2.7. Path Following . . . 17
2.8. Research Decisions . . . 19

3. System Overview 22

3.1. System Conceptualisation . . . 22
3.2. F1Tenth Platform . . . 23
3.2.1. The F1Tenth Vehicle . . . 23
3.2.2. The F1Tenth Stack . . . 23
3.3. Autonomous System . . . 24
3.3.1. High-level Overview . . . 24
3.3.2. Global Planner . . . 26
3.3.3. LiDAR Scan Processing . . . 27
3.3.4. Collision Predictor . . . 27
3.3.5. Local Planner . . . 27
3.3.6. Vehicle Control . . . 27
3.4. Simulator . . . 28
3.4.1. F1Tenth Simulator . . . 28
3.4.2. ROS Simulator . . . 28
3.5. Summary . . . 29

4. Mathematical Modelling and Conceptualization 30
4.1. Vehicle Modelling . . . 30
4.1.1. Kinematic Bicycle Model . . . 30
4.1.2. Dynamic Bicycle Model . . . 32
4.2. Track Model . . . 35
4.3. Curvilinear Coordinate System . . . 37
4.3.1. Curvilinear Bicycle Model . . . 38
4.3.2. Curvilinear to Cartesian Transformation . . . 39
4.3.3. Cartesian to Curvilinear Transformation . . . 40
4.4. Summary . . . 41

5. High-Level Motion Planning 42
5.1. Optimisation-based Path Planning . . . 42
5.2. Optimal Control . . . 43
5.2.1. Interior point optimisation method . . . 44
5.3. Global Path Planning . . . 44
5.3.1. System Model . . . 44
5.3.2. Optimisation setup and Constraints . . . 46
5.3.3. Objective function Design . . . 49
5.3.4. Problem Formulation and Optimisation . . . 50
5.4. System Implementation . . . 51
5.4.1. Example Output . . . 51
5.5. Testing . . . 53
5.5.1. Benchmark Tests . . . 53

5.5.2. Full Track Test . . . 62
5.6. Summary . . . 65

6. Trajectory Execution **66**
6.1. Trajectory Following . . . 66
6.1.1. Stanley Control . . . 66
6.1.2. Controller Tuning . . . 68
6.2. System Implementation . . . 69
6.3. State Estimation . . . 69
6.3.1. Particle Filter . . . 70
6.3.2. System Implementation . . . 71
6.4. Simulation Results . . . 72
6.4.1. Step Response . . . 72
6.4.2. Benchmark Tracks . . . 73
6.4.3. Circuit de Barcelona-Catalunya . . . 83
6.5. Summary . . . 85

7. Collision Avoidance **86**
7.1. Obstacle Detection . . . 86
7.1.1. LiDAR Point Cloud . . . 86
7.1.2. Obstacle Identification . . . 87
7.1.3. Obstacle Parameterisation . . . 88
7.2. Collision Prediction . . . 89
7.3. Collision Avoidance Planning . . . 91
7.3.1. System Model . . . 91
7.3.2. Track Formulation and Separation . . . 92
7.3.3. Optimisation Setup and Constraints . . . 93
7.3.4. Objective function design . . . 94
7.3.5. Problem Formulation and Optimisation . . . 95
7.4. System Implementation . . . 95
7.4.1. Example Output . . . 96
7.5. Simulation Results . . . 98
7.5.1. Straight . . . 98
7.5.2. Corner . . . 100
7.5.3. Hairpin . . . 103
7.5.4. Chicane . . . 105
7.5.5. Analysis of Results . . . 107
7.6. Monte Carlo Testing . . . 108
7.6.1. Columbia Track . . . 108
7.6.2. Circuit de Barcelona-Catalunya . . . 113

7.7. Summary 117

8. Conclusion **118**

8.1. Summary and Key Outcomes 118

8.2. Future Work 119

Bibliography **121**

List of Figures

2.1. Different standardised cars from the autonomous racing competitions . . . 6
2.2. A comparison of the different software pipelines for autonomous driving systems . . . 7
2.3. A comparison of the different vehicle models . . . 9

3.1. A block diagram of the complete autonomous racing system . . . 22
3.2. F1Tenth car equipped for autonomous racing . . . 23
3.3. Block diagram of the F1Tenth software stack . . . 24
3.4. Flow diagram of the autonomous racing system developed in this thesis . . 25
3.5. Block diagram of the autonomous racing system developed in this thesis . 26
3.6. Block diagram of the reference generator used in the autonomous racing system . . . 26

4.1. Kinematic bicycle model: v is the velocity, β is the slip angle, δ is the steering angle, θ is the orientation angle, L is the wheelbase length and l_r and l_f are the distances from the centre of gravity to the rear and front axle respectively. . . . 31
4.2. Dynamic bicycle model: v_y is the car's longitudinal velocity, v_x is the car's latitudinal velocity, r is the yaw rate of the vehicle, α_f is the slip angle of the front wheel, α_r is the slip angle of the rear wheel, $F_{y,F}$ is the lateral force acting on the front wheel, $F_{y,R}$ is the lateral force acting on the rear wheel, and F_x is the longitudinal force acting on the rear wheel . . . 32
4.3. Pacekja tyre model: lateral force vs slip angle for different tyre parameters. 34
4.4. A mapped track, with the centre line calculated from the map . . . 36
4.5. An illustration of the track model used in this thesis. . . . 37
4.6. Bicycle models formulated in a curvilinear coordinate system . . . 38
4.7. An illustration of the transformation from the curvilinear coordinate system to Cartesian coordinates . . . 40
4.8. An illustration of the transformation from the Cartesian coordinate system to curvilinear coordinates . . . 41

5.1. An illustration of the constraint that keeps that car within the track . . . 47
5.2. An illustration of the friction ellipse constraint . . . 48
5.3. Optimal global trajectory for a given racing track. . . . 51

5.4. The planned optimal state variables and inputs as a function of the in-track position. 52
5.4. The planned optimal state variables and inputs as a function of the in-track position. 53
5.5. Optimised trajectory for a straight line 54
5.6. The results, as a function of the in-track position, from the solved motion planning problem for a straight line 55
5.6. The results, as a function of the in-track position, from the solved motion planning problem for a straight line 55
5.7. Optimised trajectory for a corner 56
5.8. The results, as a function of the in-track position, from the solved motion planning problem for a corner 57
5.8. The results, as a function of the in-track position, from the solved motion planning problem for a corner 58
5.9. Optimised trajectory for a hairpin corner 59
5.10. The results, as a function of the in-track position, from the solved motion planning problem for a hairpin 60
5.11. Optimised trajectory for a chicane 61
5.12. The results, as a function of the in-track position, from the solved motion planning problem for a chicane 62
5.13. An illustration of the Circuit de Barcelona-Catalunya and the centre line of the race track. 63
5.14. Optimised trajectory for the Circuit de Barcelona-Catalunya 64
5.15. The results, as a function of the in-track position, from the solved motion planning problem for the Circuit de Barcelona-Catalunya 65

6.1. An illustration of the cross-track and heading error referenced to a planned trajectory 67
6.2. The results of the gain tuning for the Stanley controller 69
6.3. An illustration of the particle filter process [1] 70
6.4. Simulation step response 72
6.5. The simulation path and speed of the car for a straight 73
6.6. Simulation results for the straight line track 74
6.7. The simulation path and speed of the car for a corner 75
6.8. Simulation results for the corner 76
6.8. Simulation results for the corner 77
6.9. The simulation path and speed of the car for a hairpin 78
6.10. Simulation results for a hairpin 79
6.11. The simulation path and speed of the car for a chicane 80

6.12. Simulation results for a chicane 81
6.12. Simulation results for a chicane 82
6.13. The simulation path and speed of the car for the Circuit de Barcelona-Catalunya 83
6.14. Simulation results for the Circuit de Barcelona-Catalunya 84
6.14. Simulation results for the Circuit de Barcelona-Catalunya 85

7.1. An example of the output LiDAR scan. 87
7.2. An illustration of the DBSCAN algorithm [2]. 88
7.3. An example of an obstacle detected and clustered with the obstacle shadow depicted. 89
7.4. Illustration of trajectory propagation of the car. 90
7.5. Illustration of trajectory propagation of the car. 93
7.6. Illustration of trajectory propagation of the car. 93
7.7. Typical output from the local planner for a given track. 97
7.8. The planned state variables and inputs that are generated by the local planner, as a function of the in-track position. 97
7.8. The planned state variables and inputs that are generated by the local planner, as a function of the in-track position. 98
7.9. The obstacle avoidance detection and planning process for a straight 99
7.10. The obstacle avoidance execution for a straight 100
7.11. The obstacle avoidance detection and planning process for a corner 101
7.11. The obstacle avoidance detection and planning process for a corner 102
7.12. The obstacle avoidance execution for a corner 103
7.13. The obstacle avoidance detection and planning process for a hairpin 104
7.14. The obstacle avoidance execution for a hairpin 105
7.15. The obstacle avoidance detection and planning process for a chicane 106
7.16. The obstacle avoidance execution for a chicane 107
7.17. The Columbia track with a minimum lap-time plan 108
7.18. An illustration showing the probability distribution of the location of each obstacle. 109
7.19. Box and whisker plot of the lap times from the Monte Carlo simulations . 111
7.20. A histogram of the computation time of the local planner 112
7.21. Histogram showing the outlier planning times 113
7.22. The Circuit de Barcelona-Catalunya track with a minimum lap-time plan . 114
7.23. Histogram showing the lap times 115
7.24. Histogram showing the local planner computation time, with outliers removed 116
7.25. Histogram showing the outlying local planner computation times 116

List of Tables

2.1. A list of available sensors for each autonomous race car 12
2.2. Results from a study performed by Rokonuzzaman *et al.* [3] 19

3.1. A list of ROS topics used by the autonomous agent 29

6.1. A table showing the planned lap time and simulation lap time for each track element 82

7.1. A table showing the planned lap time and simulation lap time for each track element 108
7.2. A table showing the success rate, collision rate and stop rate of the Monte Carlo simulations 110
7.3. A table showing the success rate, collision rate and stop rate of the Monte Carlo simulations 112
7.4. A table showing the success rate, collision rate and stop rate of the Monte Carlo simulations 114
7.5. A table showing the success rate, collision rate and stop rate of the Monte Carlo simulations 115

Nomenclature

Variables and functions

θ	The orientation of the car with respect to the global X-axis.
μ	The orientation of the car with respect to the track.
δ	The steering angle of the car.
β	The kinematic slip angle of the car.
α	The dynamic slip angle of the car.
$W(s)$	The radius of the track as a function of the position along the centre line.
$R(s)$	The radius of curvature of the centre line as a function of the position along the centre line.
$K(s)$	The curvature of the centre line as a function of the position along the centre line.

Acronyms and abbreviations

LiDAR	Light Detection and Ranging
GPS	Global Positioning System
DoF	Degrees of freedom
SLAM	Simultaneous localisation and mapping
ROS	Robor Operating System
IMU	Inertial Measurement Units
MPC	Model Predictive Control
RRT	Rapidly-exploring random tree
LQR	Linear Quadratic Regulator
CoG	Centre of Gravity

Chapter 1

Introduction

1.1. Background

Automation is an increasingly popular and growing field in the 21st century. Computers are able to react to stimuli in a fraction of the time that humans can, and in many areas, robotics has advanced beyond human capabilities. These autonomous systems are typically comprised of a decision-making unit and a real-world actuator. However for autonomous systems to be useful, they need to be able to solve complex problems and react appropriately to environmental input.

Autonomous driving is a developing field where an autonomous system is designed to replace humans. However, a fully autonomous system must not require any human supervision and must be capable of making high-level decisions, and must be as safe as possible. Autonomous racing is a popular and useful platform with which to test autonomous driving systems due to the easily measurable metrics of racing, such as lap time, or successfully and safely navigating past an unexpected obstacle. Another reason why autonomous racing is a useful platform to test autonomous driving systems is that racing is inherently competitive. Thus any racing vehicle is driving at the limits of its capabilities and if an autonomous system can safely race competitively, then it is sure to be safe when the car is not at the limits of its handling capabilities. Scaled autonomous racing provides an accessible platform for autonomous control and planning, due to the relatively inexpensive hardware required for development. F1tenth racing cars are a good example, due to the standardized hardware and well-developed simulators.

1.2. Project Goal and Objectives

Racing is the simple goal of covering a set course in the fastest time possible. What this means for an autonomous racing platform is that racing cars must be high-performance vehicles capable of high speeds and must robustly operate at the limits of their handling capabilities. Additionally, an autonomous racing car must be able to safely navigate the environment and avoid obstacles within the set course. This means that autonomous

racing cars must be able to use sensor measurements in order to make complex decisions throughout the race. Commonly, scaled autonomous racing cars use a light detection and ranging (LiDAR) scanner to form a 2D scan of the immediate environment. This 2D scan can then be used for path planning, as well as localization via scan-matching, in order to navigate safely.

The challenges presented by autonomous racing are twofold. Firstly, the high-performance nature of racing demands a robust control architecture. Secondly, it is necessary that the race car can independently avoid danger in the form of unmapped physical obstacles or exceeding the limits of the car's capabilities.

The goal of this research project is to develop an autonomous racing system that can achieve an optimal lap time on a known race track, while safely avoiding unexpected obstacles in the track. Therefore, in order to achieve this, the project goals are broken down into the following objectives.

- Create a mathematical model that captures the dynamics of a car so that it can be used to design the racing plan.
- Determine the global racing plan for a race track.
- Develop a path-tracking controller to execute the planned trajectory.
- Develop a collision detection algorithm using the available sensors on the car.
- Develop a short-term planner that can generate collision-free trajectories in order to resolve detected collisions.

1.3. Research Approach and Methodology

This thesis approaches the challenges presented by autonomous racing by using classical optimization techniques to generate, and execute, a racing plan for an autonomous vehicle. Additionally, a path-tracking controller is used in order to execute the planned trajectory. Finally, an optimisation-based collision avoidance system is designed for short-term collision resolution.

The approach taken to some of the aspects mentioned is as follows are as follows:

- An autonomous racing system was designed to be implemented on a pre-existing scale racing car, with a LiDAR sensor available on the car. The system is designed to integrate with an existing software stack that abstracts hardware-level control.
- The autonomous system was designed to have a hierarchical motion planner, with a high-level global planner generating a complete plan for the entire track, and a low-level local planner generating a short-term plan for collision avoidance action.

- A Stanley controller was selected and implemented to track the planned trajectory and a particle filter was selected in order to localise the car based on LiDAR scan matching.
- An obstacle detection algorithm was developed that uses the output from a LiDAR scanner to generate a point cloud. Points that fall within the track are clustered with DBSCAN in order to identify individual objects.
- A collision prediction system was developed, which forward-propagates the position of the car in time up to a horizon limit, and determines if a collision will occur within that horizon limit.
- Tests of the autonomous system were performed in simulation, where the achieved lap time and path-tracking capabilities were analysed. Additionally, a Monte Carlo analysis was performed to test the robustness of the collision avoidance system.

1.4. Scope and Limitations

The scope of this research project includes:

- The software design of an autonomous racing system. The physical car design is outside the scope of the project.
- The autonomous racing system that is developed in this thesis is developed for use with the F1Tenth autonomous car and software stack. The low-level car control and driver integration are not included in this project.
- The designed autonomous system does not include any mapping or SLAM process, which is outside the scope of the project. Therefore, the autonomous racing system requires a pre-generated map of the racing environment.
- An open-source vision-based solution is used for localisation. No new method, or implementation, of localisation is developed in this thesis.
- The car only considers translational motion and yawing motion. Vertical motion, pitching motion, and rolling motion are not considered.
- The autonomous system is tested in simulation, using the F1Tenth simulator.

The limitations of this research project include:

- The LiDAR scanner on the car has a maximum reliable range of 30m and has a 270° field of vision. Obstacles that are not in the field of vision or within range will not be detected.

- The autonomous system developed in this thesis is not developed to accommodate dynamic obstacles. This includes the collision detection algorithm as well as the collision avoidance planning algorithm.

1.5. Thesis Outline

This thesis is split into eight chapters that present the different aspects of the research project. The layout of the thesis is as follows:

1. **Chapter 1** is an introduction to the research project, with a background to racing and autonomous racing. The project goals, objectives, and methodology are also presented in this chapter, as well as the scope and limitations of the research project.
2. **Chapter 2** is a review of previous racing literature. It focuses on different methods of modelling a car, planning techniques, methods of obstacle avoidance, and path-tracking methods.
3. **Chapter 3** gives an overview of all the components of the autonomous racing system developed in this thesis. Additionally, this chapter gives an overview of the F1Tenth platform and simulation environment used in this thesis.
4. **Chapter 4** presents the mathematical models that are used in this thesis. It presents the kinematic bicycle model and the dynamic bicycle model. The track model is also presented and the car models are reformulated into the track coordinate system.
5. **Chapter 5** presents the development of the global path planner. The chapter details the formulation of an optimal control problem, with the cost function and constraints, in order to generate a motion plan for the complete track. Additionally, the global planner is tested and the results are analysed.
6. **Chapter 6** shows the development and tuning of a Stanley controller in order to execute a planned trajectory. Additionally, a particle filter used for localisation is detailed. The results from the car executing a motion plan are also shown and analysed.
7. **Chapter 7** presents the development of the local path planner. An obstacle detection algorithm is developed, as well as a collision prediction system. This chapter formulates an optimal control problem in order to generate a local plan to avoid obstacles in the track. Additionally, the local planner is tested and the results are analysed.
8. **Chapter 8** summarises the thesis and discusses the findings and results of the research project. Finally, future work that can be done with the results of this project is discussed.

Chapter 2

Literature Review

This chapter first gives an introduction to motor racing and the factors which contribute to the successful performance of a manned racing car. Thereafter, previous literature applicable to autonomous racing is reviewed. The review first explores the common architectures of previous autonomous racing systems and different car models. Then the review moves on to different planning methods including both long-term planning and short-term planning as well as methods of obstacle avoidance. Finally, methods of executing planned trajectories are reviewed. This is followed by a discussion of the design decisions made using the information presented in the review.

2.1. Racing

Motor racing, as a motorsport, dates back to the late 19th century with the first recognized motor race occurring in 1894 [4]. Since its inception, racing has evolved into a fast-paced, precision-driven sport. This is due to advancements in technology, leading to higher racing speeds and lower error margins [5]. Success in racing is dependent on multiple factors, most notably the performance of the racing car, as well as the skill and decision-making ability of the driver. Specifically of import for a racing driver is how fast they can react to events on the track and the decisions that they make with regards to the racing line and how they position their car [6].

Autonomous racing adds a new element to racing, where success relies on the information available to the autonomous racing agent, the speed at which the agent can process the available information and the real-time decision-making capabilities of the car [7].

2.2. Autonomous Racing

There are many autonomous racing projects and competitions that have created standardized racing cars with specific physical characteristics, available sensors, and computation units. This includes F1Tenth [8], Formula Student Driverless [9], Indy Autonomous Challange [10], and Roborace [11] as shown in figure 2.1. Many of these projects include a

simulation environment that allows for the safe development and testing of autonomous racing systems. The F1Tenth simulator, Formula Student Driverless Simulator and Roborace Simulator all have multiple associated papers that have either tested the simulators or have made use of the simulators in support of the development of an autonomous racing system [12].

(a) F1Tenth car

(b) Formula Student Driverless car

(c) Indy Autonomous Challange car

(d) Roborace car

Figure 2.1: Different standardised cars from the autonomous racing competitions

Autonomous racing can be narrowed down to a definitive problem, which is to use sensor data and computation to generate control actions that will move a race car safely around a race track in as short a time as possible, subject to the physical constraints of the car and track [12]. This process has many different challenges: the sensor data is often either inaccurate, slow to process, or indirectly related to the required information. Additionally, the car must be modelled such that the physical handling limitations and capabilities are accurately identified. Finally, a racing plan must be generated such that the car can execute the required actions safely.

2.3. Autonomous Racing System Designs

There are multiple different approaches to designing the software for autonomous racing systems, each with a wide body of literature supporting it [12–17]. Three of the most prevalent software pipelines are shown in figure 2.2 [12].

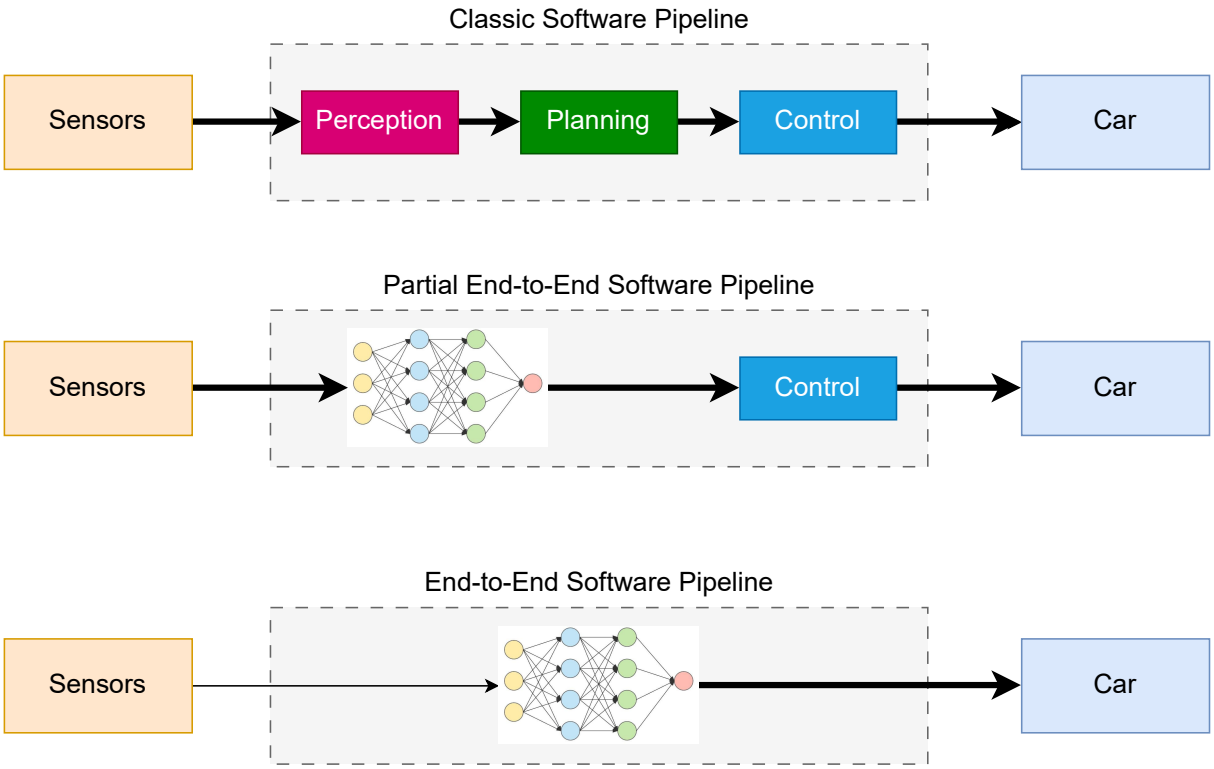


Figure 2.2: A comparison of the different software pipelines for autonomous driving systems

Tiaan *et al.* [13] present a classic autonomous racing system design, where the racing problem is broken up into 3 separate stages, perception, planning and control. The sensor data from a camera, LiDAR scanner, and GPS are used to localise the car and detect the environment around the car. This information, along with a pre-generated offline map, is used to create a motion plan for the car to execute. The car then executes the plan using a trajectory-tracking controller. The main advantage of the classic autonomous racing system is the modularity of the software pipeline. This allows for the development and testing of each individual component of the pipeline. One of the limitations of a classic system design is the inflexibility of the system. Classic pipelines may struggle to adapt to new or unexpected situations since the software component modules are designed for specific tasks and may not generalize well. This software design was successfully implemented both in simulation and on a real car by Kabzan *et al.* [18].

Anzalone *et al.* [15] present an end-to-end system design where machine learning is used for the complete process of driving, from an input image to an output control action. This

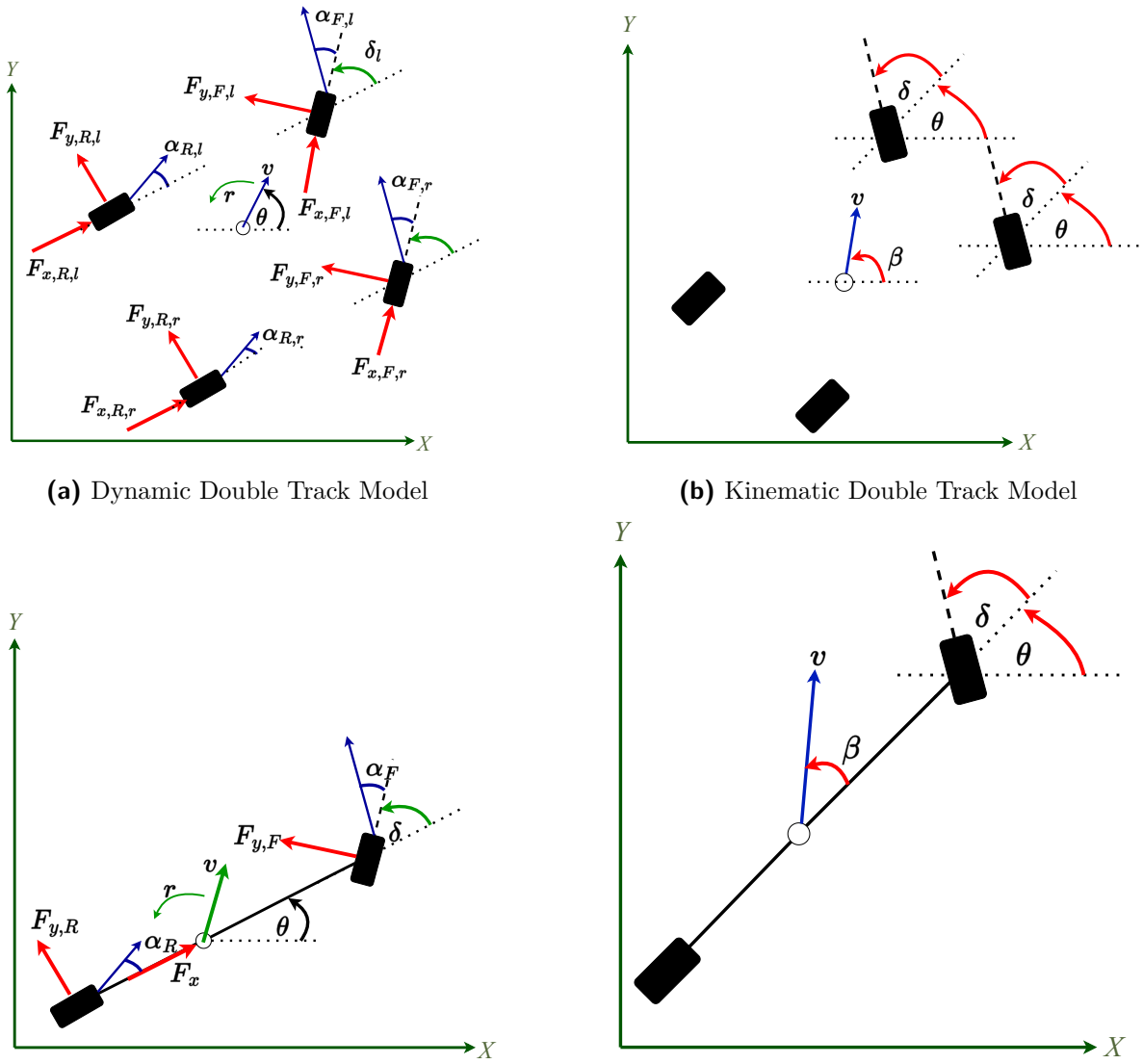
type of system architecture uses input data directly from onboard sensors and directly outputs a control action. The benefits of an end-to-end system design are the simplicity of the system architecture as well as the adaptability of the system. There are some limitations to an end-to-end system design: training a machine learning algorithm requires a large amount of training data and the reasoning for any given vehicle response is largely unknown [16].

A partial end-to-end system such as the one presented by Weiss *et al.* [17] uses machine learning techniques in order to process sensor data, specifically a LiDAR scan, and generate a reference trajectory. A pure pursuit controller is used to follow the reference trajectory. A partial end-to-end system design allows for a balance between the modularity of a classic system pipeline with the adaptability of an end-to-end system. However, the integration of the machine-learning component with classical software modules can be challenging. Additionally, the downsides of a full end-to-end system apply to a partial end-to-end system, specifically, the need for a large amount of training data as well as the lack of interpretability in the machine learning model's decisions.

2.4. Car Models

Modelling the car is one of the most important steps in developing an autonomous racing system. Whether one is using a classic system pipeline, an end-to-end system, or a partial end-to-end system, the accuracy of the vehicle model has a large effect on the racing ability of the system. However, the behaviour of a car is highly non-linear which leads to an inverse relationship between the accuracy of the model and the computational complexity of the model. Figure 2.3 shows the different vehicle models that are discussed in this section.

When modelling a car, each of the four tyres can be modelled individually and the effect that each wheel has on the car can be calculated and applied to the model, as is done by Novi *et al.* [19], Fors *et al.* [20] and Jin *et al.* [21]. This is often referred to as a double-track model. Alternatively, the bicycle model, or single-track model, can be used. The bicycle model regards the front two tyres of the car as a single individual tyre and, similarly, the back two tyres are modelled as an individual tyre. This is the most common method of modelling the car in autonomous racing studies [14, 22–24].



(a) Dynamic Double Track Model

(b) Kinematic Double Track Model

(c) Dynamic Bicycle Model

(d) Kinematic Bicycle Model

Figure 2.3: A comparison of the different vehicle models

Another important aspect of vehicle modelling is the equations describing the vehicle model. Again, there are largely two main different vehicle models that are used, namely the kinematic vehicle model and the dynamic vehicle model [25]. The kinematic vehicle model considers only the geometric relationship between each component in the vehicle model, such as the tyres and the centre of gravity, and does not consider the forces and torques that are involved in the vehicle dynamics [26, 27]. Polack *et al.* [26] performed a side-by-side comparison of a kinematic bicycle model with a double-track dynamic model, with 9 degrees of freedom, for the purposes of trajectory planning. The comparison was done in simulation, with each model parameterised with a constant steering angle, speed and wheel torque applied to the font wheels. It was observed that at higher speeds, and therefore higher lateral acceleration, the 9DoF model began to understeer due to the tyres beginning to slip. However, this was not observed using the kinematic bicycle

model, showing that while the kinematic bicycle model is a good model for low speeds, the accuracy of the model decreases at high speeds.

The dynamic vehicle model does consider the forces acting on the car and wheels when modelling the vehicle, leading to a more complex model but one that can capture complex vehicle phenomena. Jin *et al.* [21] present a dynamic double-track model that can capture the vehicle roll, pitch and yaw dynamics. However, in many studies vertical, roll and pitch motion of the vehicle is neglected and only the lateral, longitudinal and yaw motion is considered.

Kong *et al.* [25] performed a side-by-side comparison of a dynamic bicycle model using a linear tyre model, which is explored in the following section, and a kinematic bicycle model. In this comparison, vehicle states were recorded from a physical vehicle driving around a track. Then a series of simulations was performed where a model was initialized to a recorded state and a recorded input sequence was given to the model. Then the distance between the vehicle's actual location and the model predicted location was compared. This showed that the mean open-loop prediction error of the kinematic model is initially very similar to the dynamic model. However, after three 200ms timesteps, the mean open-loop prediction error of the kinematic model was greater than the dynamic model by 30%, and after four timesteps it was 48% higher than the dynamic model. From the research that has been presented, it can be concluded that the dynamic model is a more accurate but complex vehicle model compared to the kinematic model. Therefore, there is a trade-off between the two models of computation expense and accuracy. Therefore, the dynamic model is more appropriate when computation time is of lesser concern compared to the accuracy of the model. The kinematic is more appropriate when computation time is of a higher concern compared to the accuracy of the model.

2.4.1. Tyre models

When modelling the forces that act on the car, as is done in the dynamic vehicle models, one of the most important interactions to capture is the interaction between the tyres and the ground. There are several different ways to model this interaction using both linear models and non-linear models [21].

As presented by Jin *et al.* [21], a simple linear tyre model can be used to calculate the lateral forces acting on the tyres by using the cornering stiffness of the vehicle and the tyre slip angle. Similarly, the longitudinal force acting on the tyres can be derived by multiplying the tyre longitudinal stiffness with the tyre slip ratio. The linear tyre model is sufficiently accurate under normal conditions, however, when the car is at the limits of its handling, the tyre operates in a non-linear region, causing the linear tyre model to begin to fail.

There are non-linear tyre models which can capture the non-linearity of the tyre behaviour at the limits of the tyre handling capabilities. The LuGre tyre model is an example of such a non-linear model. It was originally presented by Canudas de Wit *et al.* [28] and modelled only the longitudinal force acting on the tyre. However, it was later improved and developed further to model combined longitudinal and lateral motion [29]. The LuGre model represents the frictional forces acting on the tyre using a dynamic elastoplastic friction model and, when applied to a tyre, gives 3 differential equations that describe the tyre-road contact forces and the aligning moment of the wheel. The LuGre tyre model is able to capture the non-linear and time-varying friction characteristics of tyres and is able to handle various different road conditions and tyre types. However, the LuGre model is computationally expensive and sensitive to parameter tuning of the model.

The Dugoff tyre model, originally presented in [30], is a non-linear tyre model that can calculate the longitudinal and lateral tyre force under slipping conditions using the slip ratio of the tyre, tyre side slip angle and the stiffness factors of the tyre. This tyre model does capture the non-linearity of the tyre under slipping conditions, however, the accuracy of the model decreases when the wheel slip ratio or the tyre slip angle grows large [31]. An enhanced Duggoff model was developed by Ding *et al.* [32] which introduces a friction correction factor to better model the tyre forces under large slip angles and wheel slip ratios.

Another non-linear tyre model is the Pacejka ”Magic Formula” tyre formula [33]. This tyre model is often described as the ”Magic Formula” as the equations describing the model are not based on a physical model but they still accurately describe the behaviour of the tyre. The Pacejka tyre model models the lateral tyre force as a function of the side slip angle and the longitudinal tyre force as a function of the longitudinal slip. The tyre model uses four different parameters that affect the function describing the tyre behaviour. These parameters are dependent on the tyre behaviour on a particular surface and are derived using experimental results. The Pacejka tyre model can represent a wide range of tyre behaviours, and tyre types, and is widely accepted to be one of the most accurate tyre models that are available [31]. However, the Pacejka tyre model is also computationally complex and sensitive to parameter tuning of the model.

2.5. Localisation and Mapping

The autonomous racing projects and competitions mentioned in section 2.2 all have standardised cars and hardware, with limitations on the available sensors on the car that can be used for localisation and mapping. Table 2.1 shows the available sensors for each car [12].

Car	Available Sensors
F1Tenth	Monocular Camera Stereo Camera 2D Lidar
Formula Student Driverless	Monocular Camera Stereo Camera 2D Lidar 3D Lidar Radar GPS
Indy Autonomous Challenge	Monocular Camera(s) 3D Lidar(s) GPS
Roborace	Monocular Camera(s) 3D Lidar(s) Radar(s) GPS

Table 2.1: A list of available sensors for each autonomous race car

Localisation refers to determining the position and orientation of the car within the environment, in the particular case of racing the environment is the race track which may or may not have been previously mapped. Mapping refers to the process of creating a map of the race track, which is necessary to plan a path through the race track.

2.5.1. SLAM

If the race track has not been previously mapped, then the car must map the environment while driving, similarly to Gotlib *et al.* [34]. Mapping an unknown environment is a field of research in itself. Simultaneous localisation and mapping (SLAM) is a broad field of research that extends beyond autonomous racing. However, there are some common implementations that are used in autonomous racing research, most notably GraphSlam, EKF Slam and FastSLAM [35].

There are SLAM solutions that are available from using pre-built packages and libraries that are commonly used in autonomous racing research [36]. Gotlib *et al.* [34] uses a SLAM algorithm provided by Robot Operating System (ROS) called Cartographer [37]. Cartographer is a GraphSLAM algorithm that uses a LiDAR scan and IMU measurements to map and localise the vehicle. The algorithm uses LiDAR scan measurements and information from Inertial Measurement Units (IMUs) to perform pose estimation. Additionally, scan

matching algorithms, such as Iterative Closest Point (ICP), are used to align consecutive laser scans to determine the car's motion and loop closure detection techniques are utilised to correct mapping errors. The system represents the environment as a graph, where nodes are poses (positions and orientations of the car) and edges represent the transitions between poses. A graph optimization algorithm is used to globally optimize the map by minimizing the error in the relative poses between nodes. The map can then be represented by a 2D grid.

A common approach to racing in a large unknown environment is performing SLAM using cones as specific recognisable features. Cones are used to denote the borders of a track and the racing environment can then be mapped by detecting and localising the cones [38, 39]. The colour of the cones is used to differentiate between the left and right track boundaries. Le Large *et al.* [40] performed a comparison of different SLAM approaches for tracks denoted by cones. They compared GraphSlam and EKF SLAM by comparing the SLAM-generated maps to a ground truth map. This comparison showed that although both algorithms can achieve real-time solutions, GraphSLAM produced more accurate maps, at the cost of more computation power.

Valls *et al.* [41] used a FastSLAM algorithm in order to map and localise the track. FastSLAM uses a particle filter to represent the probability distribution of the potential orientations and positions of the car and generally assumes a feature-based map where the environment is represented by a set of distinct features. Valls *et al* uses cones that have been detected using a stereo camera as these features. A set of particles is generated, each of which independently represents a potential position and orientation of the car, as well as a map estimate. As the car moves, the particles are moved in a prediction step based on a model of the car using IMU measurements. Then the particles are updated using the detected features and LiDAR measurements to correct the pose of each particle and refine the map. Each particle has an associated weight, which denotes the probability of that particular particle. A weighted average of the particles gives a probable pose of the car and an associated map of the race track.

2.5.2. Localisation

Although many autonomous racing cars have GPS devices available, it is not always feasible to use a GPS for indoor localisation. Additionally, many autonomous racing researchers have developed visual-based localisation methods [12].

Woo *et al.* [35] present a number of different approaches to localisation for autonomous driving. The simplest and least accurate of which is dead reckoning, which is a method of localisation that estimates the current position of the vehicle based on a previously known position in conjunction with the speed and direction of the car. However, dead reckoning

can be inaccurate due to initial position errors, sensor errors, and model errors, such as vehicle slip. The associated errors of dead reckoning grow as the vehicle progresses and have no upper limit unless otherwise compensated. There are methods to compensate for the errors in dead reckoning, using other methods of localisation in order to correct the dead reckoned position and orientation. The Kalman filter [35] and its variants are examples of algorithms that use multiple sensors in order to most accurately propagate the position and orientation of the car.

Another common approach to localisation in autonomous racing is by using a particle filter [34, 42, 43] for LiDAR scan matching. This is a form of Monte Carlo localisation that works by generating a number of particles, with each particle representing a potential position and orientation of the car. Each particle has an associated weight which indicates the likelihood of that particle representing the true state of the vehicle. A motion model is used to propagate the particles and a sensor model is used to update the weights based on how likely each particle is to generate the LiDAR scan that the car is receiving at that point. Katwe *et al.* [42] implemented a particle filter with a GPS and IMU acting as a motion model to determine how to propagate the particles. This was then tested in simulation by comparing the results of the particle filter, to a reference ground truth. The mean absolute error of the particle filter in this experiment was 80cm. Stahl *et al.* [43] implemented a similar particle filter practically in a racing environment. The race car drove multiple laps around the track and the location of the vehicle was tracked using the particle filter, a GPS and a dynamics sensor. The lateral mean absolute lateral error of the particle filter was calculated to be 0.086m. Therefore when a LiDAR scanner is available and the map is known, a particle filter is an accurate method of localisation.

2.6. Motion and Path Planning

When considering planning in the context of autonomous racing, it is important to note the difference between global planning and local planning. Global planning provides the optimal path, or motion plan, for the entire race track whereas local planning provides the optimal path/motion plan for a limited horizon. Local planning is usually used for the purpose of avoiding obstacles on the race track while still providing a fast racing line for the car.

2.6.1. Global Planning

Global planning for autonomous racing has been researched in many ways [12]. One of the ways to roughly group the different methods of planning that have been presented is by racing strategy. In general, optimisation-based planning requires an objective function that defines the desirability of a path which determines the racing strategy. Machine learning

algorithms such as reinforcement learning or genetic algorithms require an evaluation function to gauge the performance of a particular agent which determines the racing strategy.

A natural racing strategy for global planning is minimising the lap time of the car. A popular approach to solving this type of planning problem is by formulating an optimal control problem (OCP). Metz *et al.* [44] present one of the earliest papers that uses this technique, where the objective function is the lap time, and the vehicle dynamics, friction constraints, and track limits are included as constraints. The OCP is solved by quasi-linearisation and formulating a two-point boundary problem. However, as optimisation algorithms and computing power have improved, more complex problems can be formulated and solved. Kelly *et al.* [45] similarly directly minimises the lap time of the car and solves the resulting non-linear OCP using Sequential Quadratic Programming. Vazquez *et al.* [46] formulates a minimum-time OCP by discretizing the space-based vehicle dynamics referenced to the centre line of the race track. The resultant problem is solved using an interior-point optimisation algorithm. A minimum-time trajectory can also be found without formulating the problem as an OCP. Perantoni *et al.* [47] represents the planned racing line using splines, and optimises the parameters of the splines to produce a minimum-lap time trajectory.

Another common racing strategy is to formulate a minimum curvature path throughout the race track. This is because a minimum curvature path would allow for the highest cornering speeds at a given maximum lateral acceleration, typically derived from the handling limits of the car. Heilmeier *et al.* [48] generates a minimum curvature trajectory through the track by formulating a quadratic optimisation problem. A velocity profile was then calculated for the calculated path using the acceleration limits of the car. Braghin *et al.* [49] generates a minimum curvature trajectory by breaking the race track up into discrete sections and then formulating the minimum curvature problem geometrically and minimising the curvature of each section. Cardamone *et al.* [50] formulates a minimum curvature trajectory by breaking up the track into several sections and using a genetic algorithm to find the minimum curvature path in each section. Similarly, Bevilacqua *et al.* [51] breaks up the track but uses particle swarm optimisation in order to calculate a minimum curvature path for each section.

Finally, another solution to global planning for autonomous racing is using machine learning solutions such as reinforcement learning. Remonda *et al.* [52] present four such solutions, comparing modifications to deep deterministic policy gradient (DDPG). The algorithms compared are Long Short-Term Memory (LSTM), Multi-step targets, Prioritized Experience Replay and window sampling. The reward given to the agent at each time step during the training of each agent is based on the curvature of the agent's trajectory, thus encouraging a minimum curvature trajectory. The lap times produced by each algorithm

are compared on known tracks. The prioritized experience replay produced the best time lap time of the machine learning approaches that were compared. All the models that were trained were able to generalise to unknown tracks, however, their performance suffered greatly due to the change. Lu *et al.* [53] also present a DDPG algorithm. However, they add prior knowledge about the vehicle dynamics to the learning process. This addition is compared to a standardised DDPG algorithm, and the results showed that the addition allowed the agent to finish racetracks that the standard DDPG algorithm cannot do successfully.

2.6.2. Local Planning

The main objective of local planning is to plan the motion of the car for a fixed horizon, avoiding any obstacles on the race track. In general, local planning is performed in order to deviate from a global plan in order to prevent a collision.

The most common approach to local planning, and also one of the most successful, is Model Predictive Control (MPC) [12]. MPC is an optimal control technique, but it can also be used as a local planning solution. MPC is a predictive control algorithm that optimises the vehicle's control inputs over a fixed time horizon for a desired racing trajectory. It can take into account the current state of the vehicle, vehicle dynamics, and track boundaries. By iteratively predicting the future states of the car and adjusting the control inputs, MPC enables the autonomous system to navigate a track dynamically and adapt to changes in the track environment. Typically the aim of implementing MPC is to reduce the deviation from an already planned path. Integrating obstacle avoidance into MPC involves adding a constraint to the optimisation formulation that prevents the planned path from entering the object space. Qian *et al.* [24] present an MPC-based trajectory planner that includes a constraint that forces the planned trajectory to avoid both dynamic and static obstacles. It is assumed that a global plan has already been generated and the cost function of the MPC formulation seeks to minimise the deviation of the speed of the car from the global planned speed while conforming to the vehicle dynamics and the environmental limits, and avoiding obstacles. Simulation results showed that the generated local plan successfully avoided obstacles on the track, while not deviating from the global plan unless necessary. Bulsara *et al.* [54] implemented a similar algorithm and tested it on an F1Tenth car. The practical results showed that although there was significantly more error than in the simulation experiments, the MPC local planner was successful in avoiding the detected obstacle with which it would have collided otherwise.

Another field of local planning methods is sampling-based methods. Sampling methods sample the free space around the current vehicle states and search for a target state. Once a goal is found, a motion plan is formulated to progress from the current state to the target state. Jeong *et al.* [55] combines the rapidly-exploring random tree (RRT*) algorithm

with a local steering algorithm which uses a dynamic bicycle model in order to generate viable trajectories along with the necessary control steering inputs for the trajectory. Feraco *et al.* [56] uses RRT to generate local trajectories which are then interpolated and smoothed using cubic splines. Arslan *et al.* [57] combined RRT with closed-loop prediction, allowing the local planner to handle complex or unstable vehicle dynamics. This led to an improvement in solution quality that guarantees asymptotic optimality.

2.6.3. Alternate Obstacle Avoidance Methods

Finally, there are other methods of obstacle avoidance which are not explicitly planning algorithms such as either rule-based approaches or end-to-end solutions. Sezer *et al.* [58] presented a method of obstacle avoidance called Follow the Gap. This algorithm finds the widest gap in the horizon and finds the centre of the gap. Then a gap angle is calculated, which is the angle between the car's current heading and a line from the car's current position to the centre of the gap. Finally, the gap angle is weighted against a goal angle, which is the angle between the car's current heading and a line from the car's current position to the goal point, based on the distance to the obstacle, a minimum safety distance and tuning parameter. The car then attempts to follow the resultant heading, which will update at every time step, leading the car to avoid the obstacle ahead of it by moving into the gaps between the obstacles. Evans *et al.* [59] present a hybrid racing architecture that uses a reinforcement learning agent. The hybrid architecture uses the current vehicle state and the output from the path-following controller (that tries to follow the long-term reference path) to modify the short-term path that the car follows. This system was compared to an optimisation-based local path planner, an end-to-end obstacle avoidance agent and the Follow the Gap method. The comparison was performed on a straight track with 4 randomly placed static objects on the track. 100 of these tests were performed and the results from the test were compared. The end-to-end solution produced the slowest race time and achieved a 99% success rate. The optimisation-based local planner produced the fastest race time and achieved a 100% success rate. The Follow the Gap method produced the second fastest race time and achieved a 99% success rate. Finally, the hybrid system produced the third fastest racing time and achieved a 94% success rate.

2.7. Path Following

Path following entails controlling the car to execute a defined trajectory. This is especially important in racing as the vehicle usually operates at high speeds and at the limits of its handling capabilities. The goal of path following is to reduce the lateral and heading errors of the car to stay as close to the reference as possible, as well as reduce the velocity error to be as fast as possible.

The most popular control strategy in autonomous racing is MPC. As explained in section 2.6.2, MPC works by iteratively predicting the future states of the vehicle over a finite time horizon and optimising a set of control inputs for a desired racing trajectory, formulating a finite time optimal control problem. In the context of path following for autonomous racing, the desired optimal control problem is to track the reference trajectory. This process is repeated at each time step, thereby forming a closed-loop system that can dynamically respond to the changes in the car and environment. MPC has several advantages, most notably the fact that the system can respond to changes in the environment before they impact the car due to the prediction step of the algorithm. Additionally, the optimal trajectory is updated constantly, thereby ensuring that the car is taking the optimal racing line. However, the main disadvantage associated with MPC is that it can be significantly computationally expensive which is dependent on the planning horizon. If the planning horizon is long, the computation time may become a significant challenge. Alternatively, if the planning horizon is short, the planned trajectory may not fully account for the future behaviour of the system and therefore make ill-advised control actions, leading to a sub-optimal outcome. Verschueren *et al.* [60] formulates a nonlinear model predictive control (NMPC) trajectory tracking controller and solves the resulting OCP using a sequential quadratic programming algorithm. Liniger *et al.* [14] formulates and solves a similar NMPC using a quadratic programming algorithm. Similarly, Kabzan *et al.* [22] formulate an NMPC path tracking controller, however, they solve it directly using a commercially available solver.

Another control strategy that is used is Linear Quadratic Regulator (LQR). LQR requires a linear model of the vehicle, where an optimisation problem is formulated to reduce the vehicle error over a period of time. The main goal of LQR is to find a suitable gain for the feedback control law [3]. Kim *et al.* [61] implemented an LQR control structure in order to generate a steering angle to minimise the lateral error. The autonomous system was tested in simulation with a maximum lateral error of 0.7m.

Pure Pursuit is a path-tracking controller that uses a geometric approach to generate the appropriate steering angle to reach a target point. Pure Pursuit works by locating the closest target point to the car and determining a lookahead which determines how far ahead the car plans a trajectory. Then the appropriate steering angle is calculated based on the geometry of the vehicle and the position of the target point [62]. An adaptive pure pursuit method was proposed by Sukhil *et al.* [63] that used a dynamic lookahead distance based on the output of an optimisation function to optimise vehicle velocity.

Stanley Control refers to the control algorithm created by Thrun *et al.* [64] which is based on a non-linear feedback function used to correct the cross-track error of the car. The basic implementation of the algorithm calculates the steering angle to track a given reference based on the heading error of the car and the cross-track error, which is loosely coupled to

the speed of the car.

Vivek *et al.* [65] performed a comparative study of Stanley, LQR and MPC controllers where the closed-loop system for each controller was implemented in a MATLAB simulation and given the same reference trajectory. In this experiment, MPC achieved an integral absolute error of 0.174m^2, Stanley control had an integral absolute error of 0.3922m^2 and finally LQR had an integral absolute error of 2.37m^2.

Rokonuzzaman *et al.* [3] also performed a review and evaluation of NMPC, LQR, Pure Pursuit and Stanley Control. A genetic algorithm was used to find the optimal tuning parameters for each controller. The controllers were tested in simulation, keeping the car at a constant speed and the same reference trajectory for each controller. The test was repeated for each controller, once without external disturbances and once with external disturbances. The external disturbance was generated by adding random noise to the output vehicle states at each timestep. The maximum lateral error and the orientation error for each controller are given in table 2.2.

	Maximum Lateral Error (m) Without Disturbance	**Maximum Lateral Error** (m) With Disturbance	**Maximum Orientation Error** (deg) Without Disturbance	**Maximum Orientation Error** (deg) With Disturbance
NMPC	0.002	0.09	0.05	0.6
LQR	0.005	0.055	0.01	0.05
Stanley Control	0.04	0.16	0.2	1.5
Pure Pursuit	0.03	0.05	0.15	1.7

Table 2.2: Results from a study performed by Rokonuzzaman *et al.* [3]

2.8. Research Decisions

The following key conclusions were drawn from the literature survey:

- There are a number of autonomous racing platforms that are available, however, the F1Tenth racing platform is the most accessible. This is because the car is cheaper and smaller than other autonomous racing cars.
- There are three main autonomous racing architectures: classical, partial end-to-end, and end-to-end. The classical architecture is modular and has multiple components in the software pipeline for each component of racing. Partial end-to-end systems use machine learning techniques to process sensor data and generate reference paths, but then use classical control techniques to follow the reference. End-to-end systems use machine learning techniques to complete the entire process of driving, raw sensor data is used as an input and a control action is received as the output.
- There are a variety of vehicle models with associated advantages and disadvantages. The double-track vehicle model treats each of the four wheels individually whereas the bicycle model treats the front two wheels and back two wheels each as a single

unit. The dynamic model is a more complex but accurate model that describes the interactions between the different forces acting on the car as opposed to the simpler kinematic model that considers only the geometric relationship between each vehicle component.

- There are a number of different tyre models that model the interaction between the car tyres and the road surface. A linear model is the simplest example, however more complex tyre models such as the LuGre model, enhanced Dugoff model, and Pacejka model capture the non-linearities of the tyre behaviour to varying degrees.
- Common localisation techniques in autonomous racing include the use of the Kalman filter and its variants (e.g. Extended Kalman filer) as well as LiDAR-based localisation, particularly the use of a particle filter.
- Methods of global path planning for a known track include formulating an optimisation problem that minimises the lap time of the car. Another method includes formulating a minimum curvature path throughout the race track. There are also machine learning solutions that can be used to determine the optimal path such as reinforcement learning.
- The surveyed methods of local planning and obstacle avoidance found that MPC was the most common and successful approach to local planning, however, sampling-based methods such as RRT* have also been successfully implemented. Other surveyed methods of obstacle avoidance include the use of the Follow the Gap algorithm and hybrid systems using machine learning techniques.
- The primary methods used for path following that were surveyed are MPC, LQR, Pure Pursuit and Stanley Control. The survey found that while MPC is the most successful, with the least error, they are complex to implement and require significant computation power whereas techniques such as Pure Pursuit or Stanley control are less effective, they are simpler to implement and less computationally intensive.

Based on the information gained from the literature survey, the following research decisions were made:

- A classic software pipeline will be used for autonomous racing system, due to its modularity and the fact that it does not require additional training data as opposed to partial end-to-end or end-to-end systems.
- A dynamic bicycle model with a Pacejka tyre model will be used for the global path planner when computation time is not of particular importance. This is due to the additional accuracy that is inherent to the dynamic model while making use of the simplicity of the bicycle model.

- A global planner that formulates the problem as a minimal-time optimal control problem is used to create a global racing plan because such a solution provides an optimal racing motion plan that directly minimises the lap time instead of indirectly maximising the velocity of the car or creating a path of least curvature. Such a solution can be considered the most optimal for racing purposes.
- A Stanley controller will be used for path tracking due to its simplicity of design and implementation
- A particle filter will be used for the localisation of the car from LIDAR measurements, due to the accuracy and lack of additional hardware necessary for its implementation
- A kinematic bicycle model will be used for the local path planner when computation time is of particular importance.
- The system will be tested using the F1Tenth simulator, and thus the autonomous racing system will be developed to integrate with an F1Tenth race car.

Chapter 3

System Overview

This chapter provides an overview of the autonomous racing system designed in this thesis. Firstly, the F1Tenth car and software stack are described as well as an overview of the integration of the car, software stack, and the developed autonomous racing software. Then the autonomous racing software is presented: a hierarchical motion planner comprising of an offline global planner and an online short-term planner, an obstacle detection and collision prediction algorithm, and a path tracking controller. Finally, the simulation environment, which is used for development and testing, is introduced.

3.1. System Conceptualisation

The autonomous racing system developed for this thesis is designed to be used with an F1Tenth race car in order to autonomously race on a pre-mapped race track. Figure 3.1 shows an overview of a complete autonomous racing project including the autonomous racing system developed in this thesis, the F1Tenth software stack, and the physical car. The autonomous racing system will use the sensor information that is available from the race car as well as information about the track that will have been generated offline, prior to the system startup, in order to generate the optimal safe racing plan. This plan is then executed by the race car.

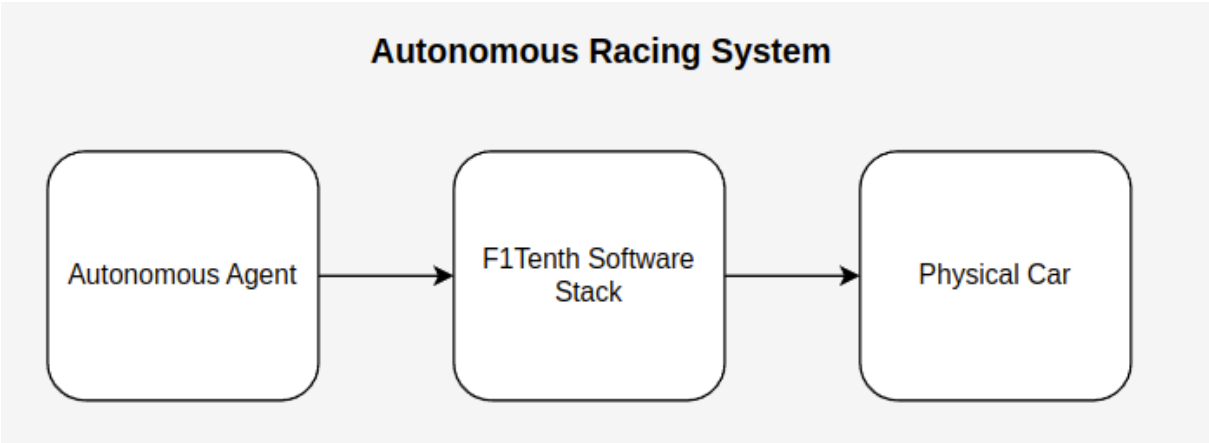


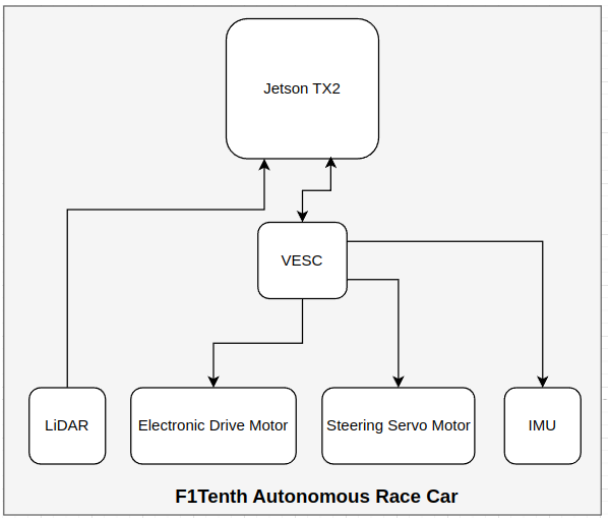
Figure 3.1: A block diagram of the complete autonomous racing system

3.2. F1Tenth Platform

This section describes the F1Tenth platform, which consists of the F1Tenth vehicle and the F1Tenth software stack.

3.2.1. The F1Tenth Vehicle

The standard F1Tenth racing vehicle, shown in figure 3.2 is a vehicle that is $1/10^{th}$ the size of normal F1 cars. The chassis of the F1Tenth car is a Traxxas Slash 4x4 Premium, with an electric drive motor to power the rear wheels and a servo motor to control the steering of the car. A light detection and ranging (LiDAR) sensor is included in the car, as well as an inertial measurement unit (IMU). This allows the vehicle to track the environment, as well as its own movement. The car is equipped with a Jetson TX2 computer that controls all the sensors on the car and computes any necessary path-planning algorithms. A virtual electronic speed controller (VESC) runs the low-level control system. It receives a steering angle and speed and controls the servo motor with pulse width modulation (PWM) and the drive motor with voltage modulation [66].



(a) Block diagram of an F1Tenth car

(b) An F1Tenth car

Figure 3.2: F1Tenth car equipped for autonomous racing

3.2.2. The F1Tenth Stack

The F1Tenth platform includes a Robot-Operating-System (ROS) open-source software stack that is a comprehensive framework designed to work with the F1Tenth car. ROS is a communication platform that uses independent nodes that communicate with each other through topics. These nodes are pieces of code that run simultaneously which allows for multiple tasks to be handled at once. Topics are communication streams through which nodes can publish information as well as subscribe to information that other nodes

publish on that topic. The F1Tenth software stack, illustrated in figure 3.3 provides the necessary tools, libraries and interfaces in order to deploy the F1Tenth car for use with an autonomous navigation system [67]. It includes components necessary for perception by interfacing with the LiDAR as well as the software necessary to interface with the ESC. The stack also includes a particle filter that can be used for the localisation of the car within the environment by using the LiDAR scan [68]. The F1Tenth stack is used for the physical implementation of the autonomous racing system on the F1Tenth car.

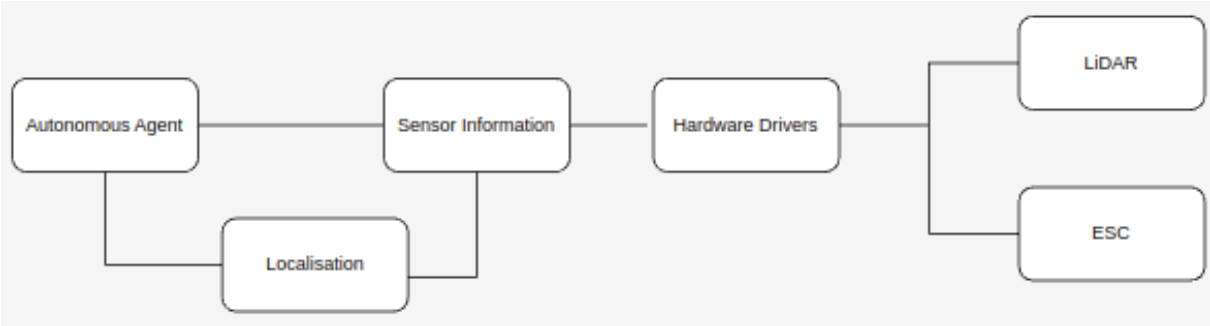


Figure 3.3: Block diagram of the F1Tenth software stack

3.3. Autonomous System

3.3.1. High-level Overview

The autonomous racing system uses a hierarchical motion planner in order to race competitively, while safely avoiding unexpected obstacles within the race track, using a classic software pipeline. This hierarchical motion planner plans an optimal long-term global motion plan for the entire race track offline, while a short-term local motion plan is generated if an unexpected obstacle is detected in the path of the vehicle in order to avoid a collision with that obstacle. The flow diagram in figure 3.4 describes the developed system.

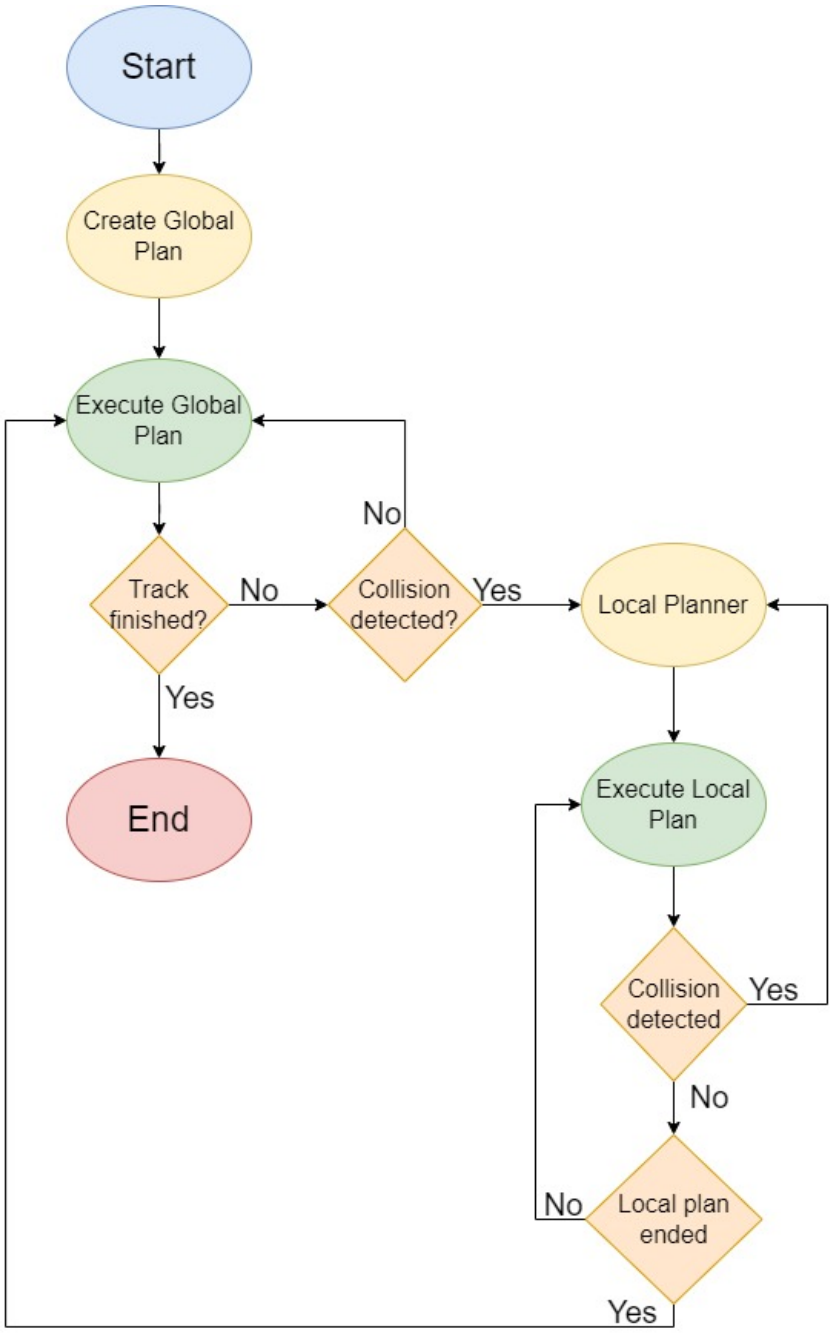


Figure 3.4: Flow diagram of the autonomous racing system developed in this thesis

Every time the location of the car is updated, the autonomous system uses the LiDAR scan in order to determine if any obstacles are on the race track. The point cloud from the LiDAR scan is transformed to the relevant coordinate framework and checked to see if any of the points are within the race track. If there are a significant number of points that are clustered together that are within the track boundary, then that point cluster is identified as an obstacle. Following this process, a collision prediction step is performed where the current position of the obstacle is propagated forward in time according to the current plan that the car is following. If, in this propagation, the car is in danger of colliding with

the obstacle, a new short-term local plan is generated to avoid the detected obstacle(s). However, the local plan is generated for a short horizon distance and once the car has successfully executed the local plan, the car will continue following the pre-planned global plan. Figures 3.5 and 3.6 show the components of the system and how they interact with each other.

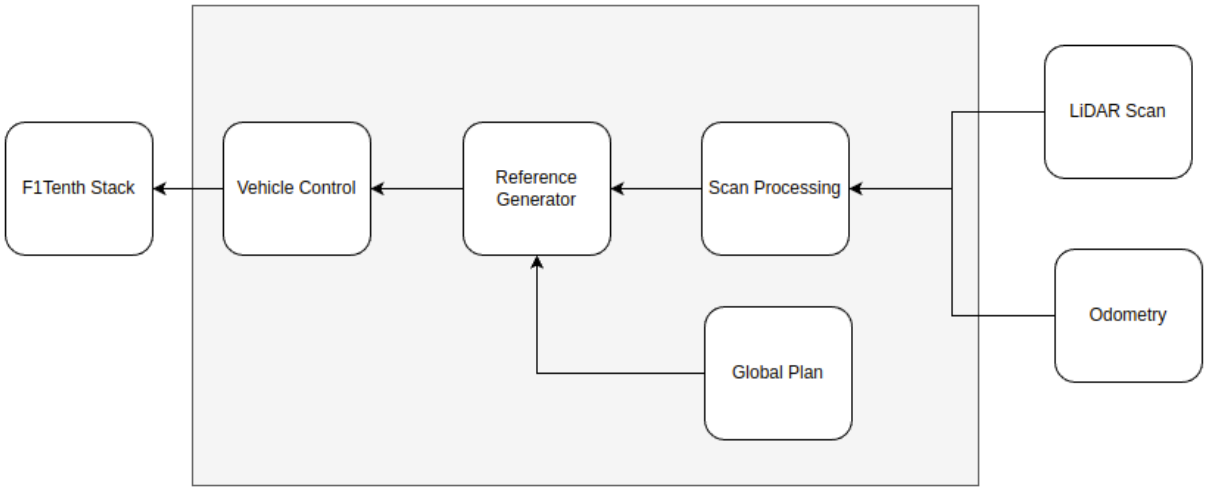


Figure 3.5: Block diagram of the autonomous racing system developed in this thesis

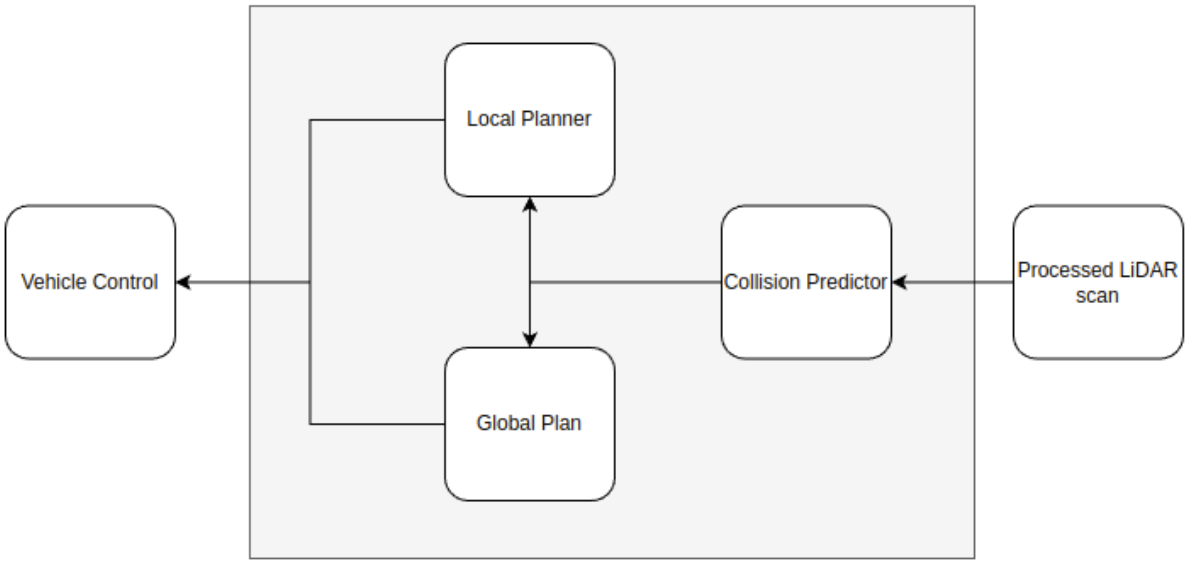


Figure 3.6: Block diagram of the reference generator used in the autonomous racing system

3.3.2. Global Planner

The high-level planner in the hierarchical motion controller is an offline process that uses a dynamic bicycle model to model the vehicle and calculate the optimal motion plan for the car in the pre-mapped race track. This is achieved by formulating the path planning

process as an optimal control problem and solving the resulting optimisation in order to minimise the time that it takes for the car to traverse the race track while accounting for the limitations of the dynamics of the car.

3.3.3. LiDAR Scan Processing

In order to detect any obstacles with which the car might collide, the LiDAR scan is converted to a point cloud expressed within the coordinate system of the race track and car. Following this, any points that are within the race track are isolated and clustered using DBSCAN (density-based spatial clustering of applications with noise) [69]. If there is a sufficient number of points that make up a cluster, then those points are treated as an obstacle within the track. The shape of the obstacle is approximated by a convex hull and from this, an area where the car cannot enter is determined.

3.3.4. Collision Predictor

If an obstacle(s) is detected using the LiDAR scan, a collision prediction step is used to determine if evasive action is required. The collision prediction step uses the current position and orientation of the car and, using the current reference trajectory that the car will follow, propagates the position and orientation of the car. A kinematic bicycle model is used to model the dynamics of the car, and a forward Euler integration is performed to propagate the position and orientation of the car forward a number of time steps up to a prediction time horizon. If at any time step in this process, the car would enter the space that is occupied by a detected obstacle, then the local planner is activated in order to avoid the obstacle and keep racing.

3.3.5. Local Planner

The low-level planner in the hierarchical motion controller is an online process that uses a kinematic bicycle model to model the vehicle and calculate the optimal motion plan for the car on the race track. This is achieved similarly to the global planner, solving an optimisation problem that models the path planning problem. However, in this optimisation formulation, constraints are added that restrict the car from entering the space that is occupied by any of the detected obstacles. Additionally, the deviation from the global plan is added to the cost function of this optimisation formulation, in order to minimise the corrective actions that the car has to take.

3.3.6. Vehicle Control

While the car is following the reference from either the global plan or the local plan, the speed reference is used directly from the motion planning results as the ESC on the car

has an internal speed control loop that uses the back EMF of the motor in order to have the car follow the given speed. However, a Stanley controller is used in order to generate a steering angle that will follow the generated path and take corrective action if necessary. The steering control is loosely coupled to the speed of the vehicle, which is useful for racing in order to minimise slipping of the car.

3.4. Simulator

3.4.1. F1Tenth Simulator

The testing platform is the open-source F1Tenth simulator [70]. It follows a similar structure and interface to the OpenAI Gym [71], which is a popular framework for developing and benchmarking reinforcement learning algorithms. The simulator was originally designed and built by a research group at the University of Pennsylvania [70] and has been used as a testing platform in other research [72]. At each timestep, the simulator takes an action that consists of a steering angle and speed reference, updates an internal vehicle model and returns a state which contains a LiDAR scan, position, orientation, velocity and steering angle. The simulator uses a 7-dimensional state single-track bicycle model as the internal vehicle model. The inputs are updated at a frequency of 10 Hz and the simulator is updated at a frequency of 100 Hz. The simulator is used to test the system developed in this thesis before it is deployed for use on the car.

3.4.2. ROS Simulator

The F1Tenth platform also includes a ROS simulator, which is used in order to run the Gym simulator and an autonomous racing agent as it would on the physical vehicle [73]. The ROS simulator runs the Gym simulator within a Docker container and an autonomous agent uses ROS topics to interact with the simulation, similarly to how it would interact with the software stack on the physical vehicle. This allows for an easier simulation-to-reality transfer as the ROS topics used in the ROS simulation and the physical vehicle are identical. The ROS simulator also allows for the implementation of the particle filter in real-time which provides an accurate simulation of the localisation quality available on the physical car.

Topic Name	Publisher	Message Type	Description
\drive	Autonomous Agent	AckermannDrive	Steering angle and speed reference that the car should follow
\scan	LiDAR	LaserScan	1080 ranges from the LiDAR
\pf_odom	Particle Filter	Odometry	A combination of Pose and Twist messages that describe the vehicle's position and orientation

Table 3.1: A list of ROS topics used by the autonomous agent

Table 3.1 lists the ROS topics used for controlling the vehicle. The autonomous agent publishes AckermannDrive messages to the \drive which the car will follow. The autonomous agent also subscribes to the \scan topic and the \pf_odom topic which contains the LaserScan message from the LiDAR and the Odometry message from the particle filter respectively. The particle filter also subscribes to the \scan topic in order to estimate the vehicle's location.

3.5. Summary

This chapter started by presenting an overview of the F1Tenth platform. The F1Tenth vehicle was presented with an overview of the hardware present on the physical car. This was followed by an overview of the F1Tenth software stack. Then an overview of the autonomous system developed in this thesis was presented. This included brief descriptions of the global planner, LiDAR scan processing, collision prediction, the local planner, and path-tracking. Finally, the chapter ended with a description of the F1Tenth Python simulator and ROS simulator.

Chapter 4

Mathematical Modelling and Conceptualization

This chapter presents the vehicle and track models that are used in this thesis. First, the kinematic bicycle model is presented as well as the state dynamics. Then, the dynamic bicycle model is presented and the state dynamics are described. Following this, the track model is presented and the vehicle models are reformulated in the track coordinate system. Finally, the coordinate system transforms are derived for the track coordinates and Cartesian coordinates.

4.1. Vehicle Modelling

4.1.1. Kinematic Bicycle Model

The kinematic bicycle model has been used in previous autonomous racing literature and is primarily used due to its simplicity [24]. The kinematic model is purely geometric, neglecting tyre slip and therefore loses accuracy as the vehicle speed increases and tends towards the tyre saturation point [26].

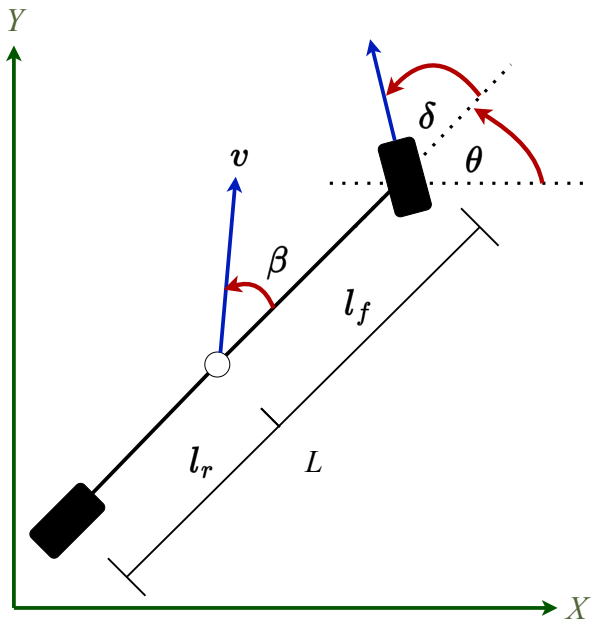


Figure 4.1: Kinematic bicycle model: v is the velocity, β is the slip angle, δ is the steering angle, θ is the orientation angle, L is the wheelbase length and l_r and l_f are the distances from the centre of gravity to the rear and front axle respectively.

Figure 4.1 shows the kinematic model set in a Cartesian coordinate frame. The inputs to the kinematic model are the velocity v and the steering angle δ. The state of the vehicle is described by the position with X and Y coordinates as well as the orientation angle θ in the Cartesian coordinate system. The slip angle β describes the angle between the direction of the velocity at the centre of gravity (CoG) and the orientation of the vehicle. The slip angle is assumed to be related solely to the geometry of the car and the steering angle, neglecting any interaction between the tyres and the ground. The equations describing the vehicle dynamics are given by:

$$\begin{aligned}
\dot{X} &= v\cos(\beta+\theta) \\
\dot{Y} &= v\sin(\beta+\theta) \\
\dot{\theta} &= \frac{v}{L}\tan(\delta)\times\cos(\beta) \\
\beta &= \arctan(\frac{l_r\tan(\delta)}{L})
\end{aligned} \tag{4.1}$$

The X and Y coordinates are updated based on the velocity component in each direction, while the yaw rate is calculated using the steering angle and car parameters. More specifically, the relevant car parameter is the wheelbase L which is the sum of the distances from the CoG to the front and rear axles, l_f and l_r respectively. Despite being a less accurate model, the kinematic model is relatively simple and thus any computations that

use a kinematic model will be significantly faster than most vehicle models. Therefore in this thesis, the kinematic model is used when computation time is important.

4.1.2. Dynamic Bicycle Model

The dynamic bicycle model extends the kinematic model to include the relevant force laws that describe the motion of the car which is a more accurate representation of a real vehicle [23]. However, this is a significantly more complex model than the kinematic model. The car is modelled as a single rigid body with a mass m and rotational inertia I_z. At each of the wheels, there is a lateral force acting on the tyre as well as a longitudinal force acting on the rear tyre.

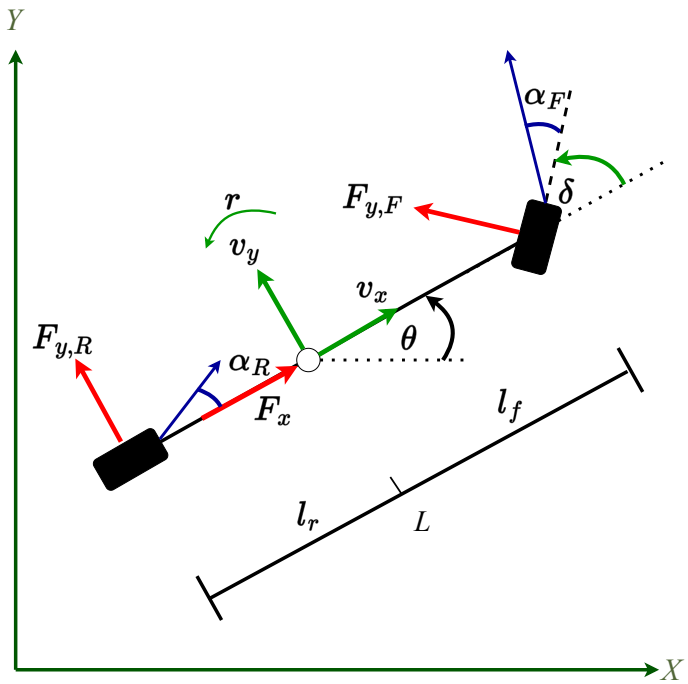


Figure 4.2: Dynamic bicycle model: v_y is the car's longitudinal velocity, v_x is the car's latitudinal velocity, r is the yaw rate of the vehicle, α_f is the slip angle of the front wheel, α_r is the slip angle of the rear wheel, $F_{y,F}$ is the lateral force acting on the front wheel, $F_{y,R}$ is the lateral force acting on the rear wheel, and F_x is the longitudinal force acting on the rear wheel

Figure 4.2 shows the dynamic model set in a Cartesian coordinate frame. The inputs to the dynamic model are a throttle command T and the steering angle δ. T is the normalised input command for the drive motor of the car and has the range $[-1, 1]$. The vehicle state is described by the X and Y coordinates and the orientation angle θ of the car in the Cartesian coordinate system. The state also includes the lateral and longitudinal velocity of the car, v_y and v_x, as well as the yaw rate of the car, r. In order to derive the differential equations of the model, the following simplifying assumptions are made:

- All vertical motion can be ignored. (all motion not in the plane of the car is

neglected.)

- Load changes can be ignored.
- Pitch and roll dynamics can be neglected.
- Combined slip can be neglected.
- All the drive train forces act at the CoG.
- The inputs to the vehicle model, throttle command and steering angle, can change instantaneously.

Using these assumptions, the equations that describe the vehicle dynamics are given by:

$$\begin{aligned}
\dot{X} &= v_x \cos(\theta) - v_y \sin(\theta) \\
\dot{Y} &= v_x \sin(\theta) + v_y \cos(\theta) \\
\dot{\theta} &= r \\
\dot{v_x} &= \frac{1}{m}(F_x - F_{y,F} \sin(\delta) + m v_y r) \\
\dot{v_y} &= \frac{1}{m}(F_{y,R} + F_{y,F} \cos(\delta) - m v_x r) \\
\dot{r} &= \frac{1}{I_z}(F_{y,F} l_f \cos(\delta) - F_{y,R} l_r)
\end{aligned} \tag{4.2}$$

The defining aspects of the dynamic model are the lateral tyre forces $F_{y,R}$ and $F_{y,F}$ and the longitudinal force F_x. The lateral tyre forces model the interaction between the car and the road. This is especially important when racing, as it is necessary to accurately model the vehicle at high speeds and at the limits of friction. In order to balance accuracy with complexity, the lateral tyre forces are modelled using a simplified Pacekja tyre model [33]. The simplified Pacejka tyre model models the lateral force F_y as a function of the side slip angle α as shown in figure 4.3, assuming a constant normal force acting on the tyre. This force curve is described using 3 parameters which are specific to each tyre and road conditions. Therefore a separate set of parameters is necessary for each wheel in the bicycle model. These parameters are the peak factor D, the stiffness factor B, and the shape factor C. The parameters B and C are dependent on the tyre while the parameter D is dependent on the friction force between the tyre and road surface.

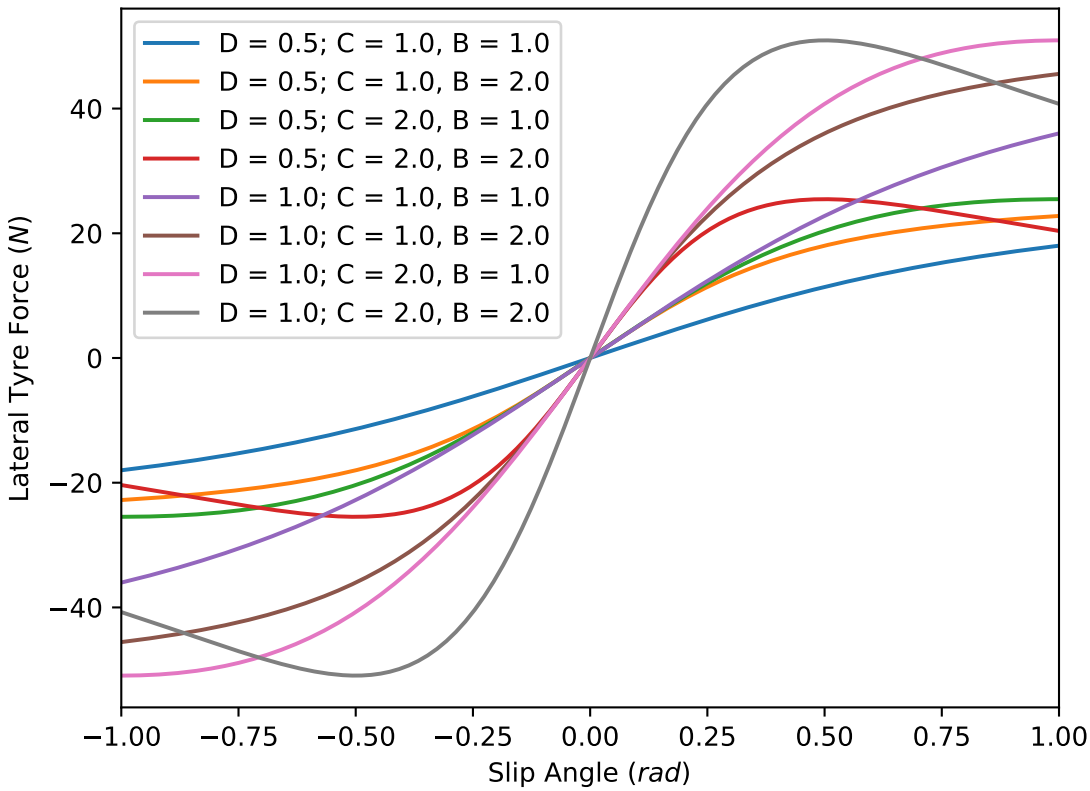


Figure 4.3: Pacekja tyre model: lateral force vs slip angle for different tyre parameters.

The equations describing the front and rear lateral forces using the simplified Pacekja model are shown in 4.3 and 4.4 respectively where $F_{N,r}$ and $F_{N,f}$ are the normal forces on the rear and front tyres [33].

$$F_{y,F} = F_{N,r} D_f \sin(C_f \arctan(B_f \alpha_f)) \tag{4.3}$$

$$F_{y,R} = F_{N,f} D_r \sin(C_r \arctan(B_r \alpha_r)) \tag{4.4}$$

The side slip angle of each tyre describes the angle between the direction of the velocity of the wheel and the orientation of the tyre. The slip angle is calculated using the geometric relationships of the wheel velocity vectors and the rigid body of the vehicle [74]. These derivations result in the slip equations below:

$$\alpha_f = -\arctan(\frac{l_f r + v_y}{v_x}) + \delta \tag{4.5}$$

$$\alpha_r = \arctan(\frac{l_r r - v_y}{v_x}) \tag{4.6}$$

The longitudinal force of the vehicle is modelled using a motor model for an electric DC motor as well as a simple friction model to represent the rolling resistance and the drag resistance of the vehicle [23]. The motor force F_M is a linear function of the throttle command T that uses a constant term C_m to represent the motor. The drag resistance of the car is modelled with a constant term C_{r0} while the drag force term is a function of the longitudinal velocity v_x as shown below:

$$\begin{aligned} F_x &= F_M - C_{r0} - C_{r2} v_x^2 \\ F_M &= C_m T \end{aligned} \tag{4.7}$$

The dynamic model is a more complicated model when compared to the kinematic model, though it is known to be more accurate [25]. However, this accuracy does come at the cost of additional complexity. Thus in this thesis, the dynamic model is used when computation time is not a great concern.

4.2. Track Model

In order to calculate a global racing plan, it is necessary to have a completely mapped racing track. Therefore, in this thesis, it is assumed that the race track has been previously mapped, and a map in the form of an image or bitmap is available. Figure 4.4 shows a track that has been mapped with the use of Cartographer [37], an open-source SLAM algorithm. The centre line of the race track can then be calculated, as shown in the figure 4.4.

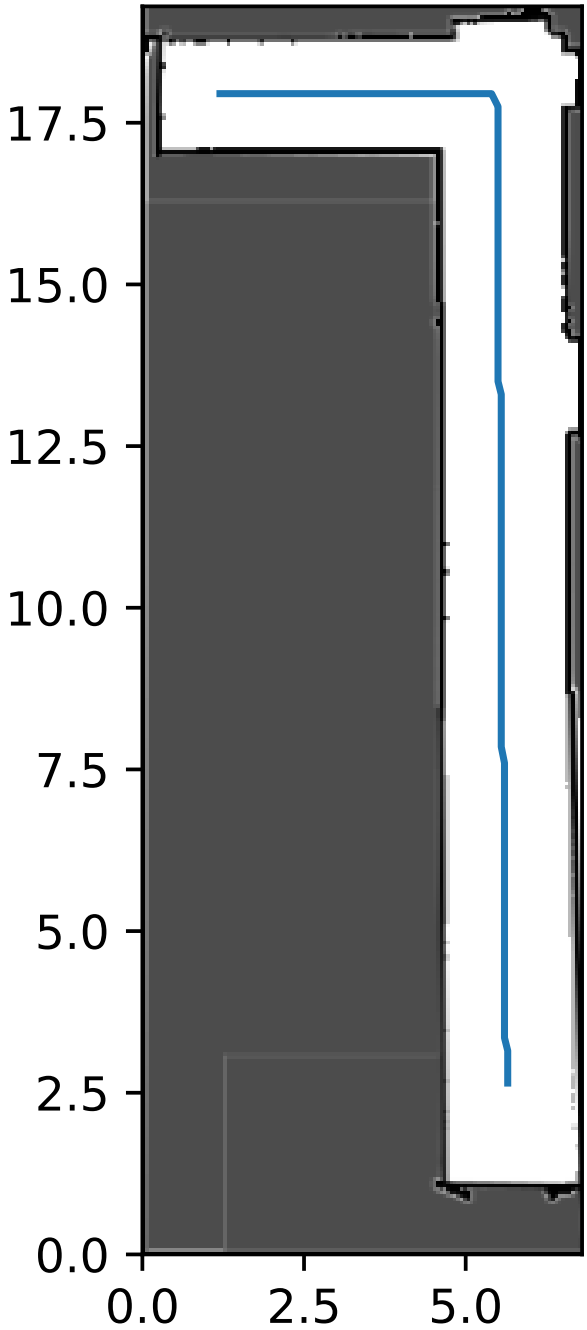


Figure 4.4: A mapped track, with the centre line calculated from the map

When modelling the race track in this thesis, a curvilinear line is used to represent the centre line of the track [75]. This centre line is parameterized with respect to s, where s is the distance along the centre line from the start of the track. At every point s on the centre line, there is an associated radius of the track $W(s)$ that designates the allowable racing area as well as the curvature of the centre line at that point $K(s)$. The radius of the track is the perpendicular distance from the centre line to the track perimeter, such that twice the radius of the track is equal to the total width of the track at any point. Figure 4.5 shows an illustration of the track model. The advantage of a curvilinear track

model is that it allows for a very simple track representation as well as allowing for the state of the vehicle model to be expressed with respect to the track itself rather than a reference axis system. When the track model is created, it is created within a Cartesian coordinate system to enable coordinate transforms between the curvilinear and Cartesian coordinate systems.

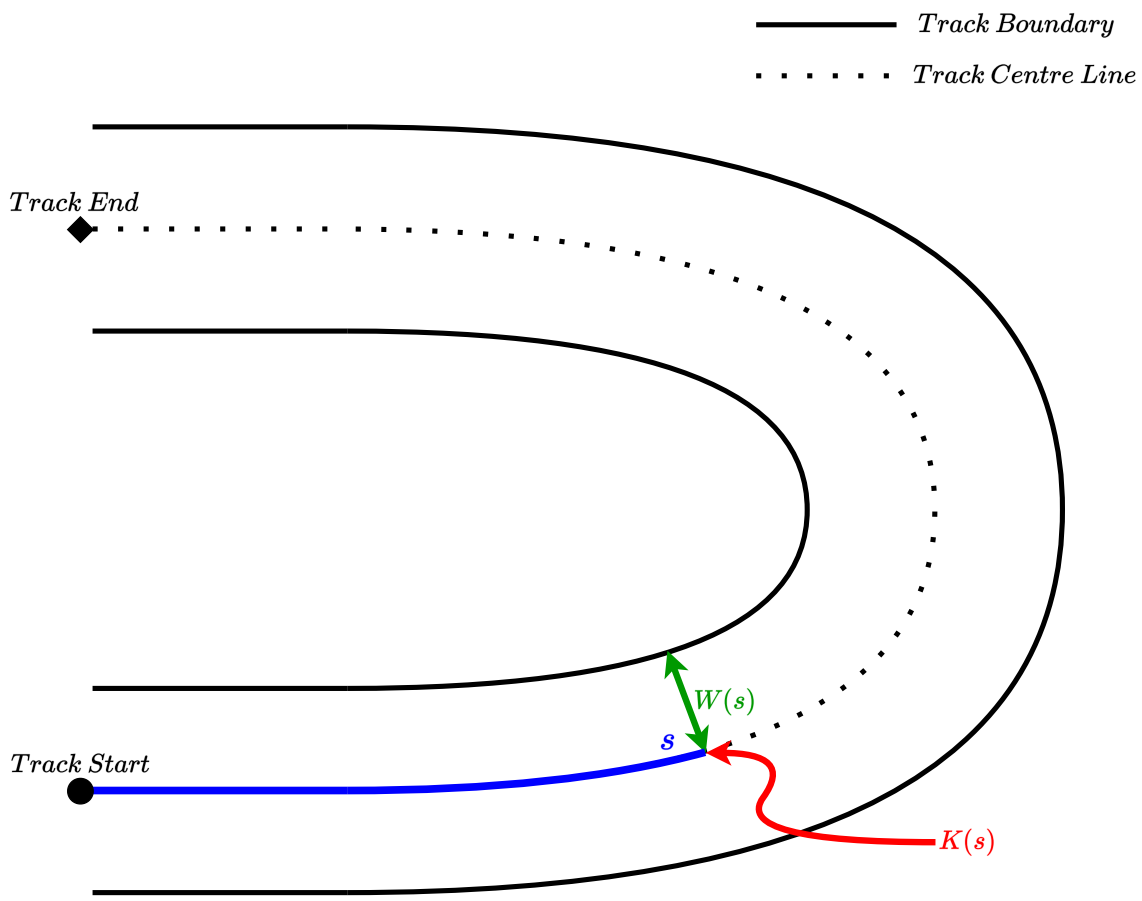
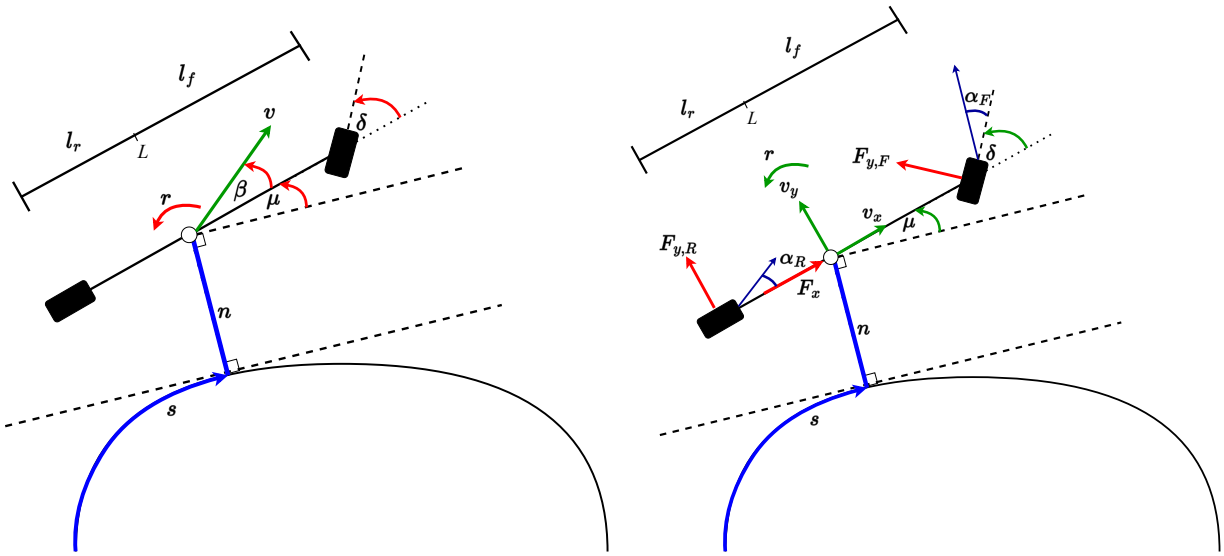


Figure 4.5: An illustration of the track model used in this thesis.

4.3. Curvilinear Coordinate System

In this section, the previously presented bicycle models are reformulated from the fixed Cartesian axis system to a curvilinear coordinate system of the track [46]. The bicycle models are reformulated to be referenced to the centre line of the track as opposed to a global coordinate system. This means that all the state information will be relative to the track as instead of the fixed-axis system.

4.3.1. Curvilinear Bicycle Model



(a) Curvilinear Kinematic Bicycle Model **(b)** Curvilinear Dynamic Bicycle Model

Figure 4.6: Bicycle models formulated in a curvilinear coordinate system

Figure 4.6 shows the kinematic and dynamic bicycles that have been reformulated in the curvilinear coordinate system. The only states that are influenced by this change in reference are position and orientation. The previous states X and Y that described the Cartesian coordinates of the car position are replaced with two states s and n which describe the progress along the centre line (in-track position) and the orthogonal distance away from the centre line (cross-track position). A positive cross-track position indicates that the car is to the left of the centre line while a negative cross-track position indicates that the car is to the right of the centre line. The state θ that represents the orientation of the car relative to the global axis system is replaced with a state μ that represents the orientation of the car relative to the track. This state describes the angle between the longitudinal axis of the car and the tangent of the centre line at the current in-track position s. The dynamics for these states are shown below:

$$
\begin{aligned}
\dot{s} &= \frac{v_x \cos(\mu) - v_y \sin(\mu)}{1 - nK(s)} \\
\dot{n} &= v_x \sin(\mu) + v_y \cos(\mu) \\
\dot{\mu} &= r - \dot{s}K(s)
\end{aligned}
\tag{4.8}
$$

The rest of the states for each model are not derived here, as they are not influenced by the change in the reference coordinate system.

Limitations

The bicycle model formulated in curvilinear coordinates has some limitations associated with the dynamic equation describing $\dot{s}$. When the term $1 - nK(s)$ is 0, there is a singularity. This occurs when $nK(s)$ is equal to 1. The curvature $K(s)$ is equivalent to $\frac{1}{R(s)}$ where $R(s)$ is the radius of curvature. Therefore, the term $1 - nK(s)$ is 0 when the cross-track distance n is equal to the radius of curvature $R(s)$. For the proposed system it is assumed that the track design will not allow such an occurrence and that the width of the track will limit the cross-track position such that it will never be equal to $R(s)$.

4.3.2. Curvilinear to Cartesian Transformation

The transformation from curvilinear coordinates, s and n, to Cartesian coordinates, X and Y, is given by:

$$\begin{aligned} x &= -n\sin(\gamma) + x_t \\ y &= n\cos(\gamma) + y_t \end{aligned} \tag{4.9}$$

where γ is the angle between the global X axis and the tangent of the centre line at the point s and ranges from $-180°$to $180°$. The points x_t and y_t are the Cartesian coordinates of the point s on the centre line, which can be obtained from the track model and are a function of s. Figure 4.7 shows an illustration of the transform shown in equation 4.9.

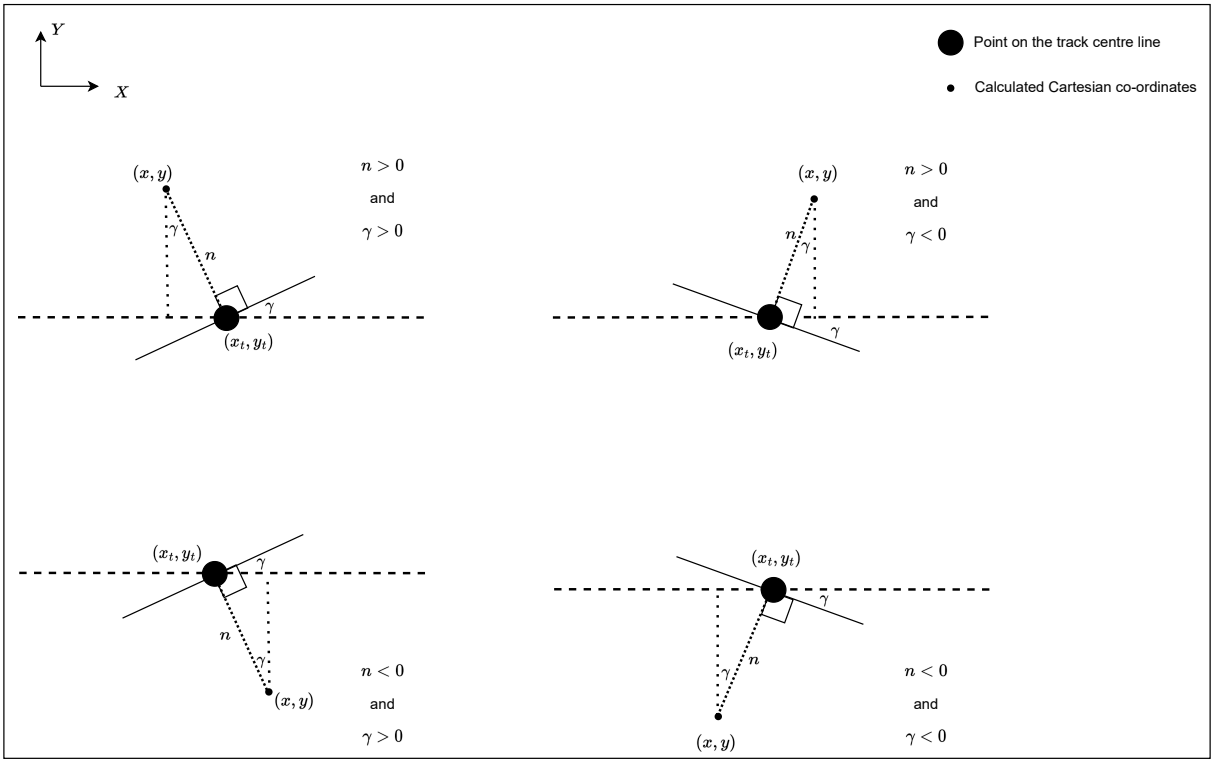


Figure 4.7: An illustration of the transformation from the curvilinear coordinate system to Cartesian coordinates

4.3.3. Cartesian to Curvilinear Transformation

Converting Cartesian coordinates to curvilinear coordinates starts with finding the line with the smallest Euclidean distance from the given Cartesian coordinate to the centre line. This line will be perpendicular to the centre line and thus represents the cross-track distance $|n|$, and the point of intersection between the line and centre line will represent the in-track distance s. However, the cross-track distance is unsigned and thus it is necessary to determine whether the given coordinate is to the left or right of the centre line. This is achieved by comparing the phase of the angle, θ, between the X axis the tangent of the centre line at the intersection point s and the angle ϕ where ϕ is calculated as:

$$\phi = \arctan 2(\frac{y - y_t}{x - x_t}) \tag{4.10}$$

By using complex number operations, the phase difference between θ and ϕ can be determined. If the phase is positive, that means that the angle ϕ is in the counterclockwise direction to the angle θ and therefore the cross-track position n is positive. If the phase is negative, then the angle ϕ is in the clockwise direction to the angle θ and the cross-track position n is negative. An illustration of the transform is shown in figure 4.8.

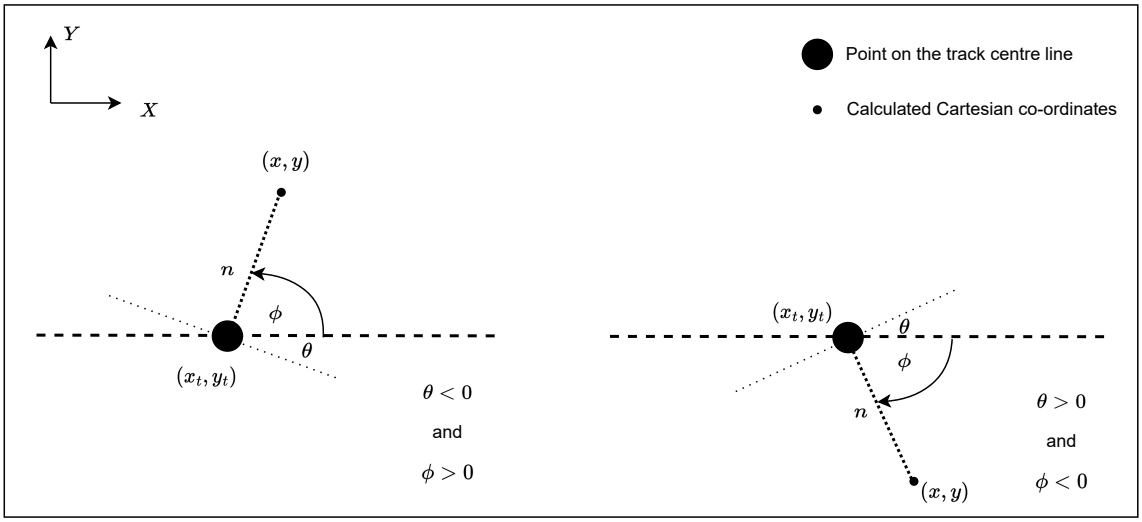


Figure 4.8: An illustration of the transformation from the Cartesian coordinate system to curvilinear coordinates

4.4. Summary

This chapter started by describing the kinematic bicycle model within a Cartesian coordinate system, including the vehicle dynamics and an explanation of the parameters of the model. Then the dynamic bicycle model was described within a Cartesian coordinate system, detailing the Pacejka tyre model that describes the lateral tyre forces. Following this, the track model was presented, specifically a curvilinear track model. Finally, the previously derived car models were reformulated within the track model, namely replacing the Cartesian coordinates with cross-track and in-track coordinates. The transformations between the coordinate systems were also described. The coordinate systems and vehicle models that were established in this chapter will be used as the basis for the global and local planners developed in this thesis.

Chapter 5

High-Level Motion Planning

This chapter presents the development of the global path planner. The chapter starts by discussing optimisation-based path planning, and then narrows it down to path planning via optimal control. The racing problem is then formulated as an optimal control problem, and is solved using an interior point optimisation method. The global planner is tested in simulation and the results are analysed.

5.1. Optimisation-based Path Planning

Optimisation-based path planning uses optimisation techniques to find an optimal, or near-optimal, path for a robot or vehicle to navigate from a starting point to a desired goal while considering various constraints and objectives. It involves formulating the path planning problem as an optimisation problem and solving it using mathematical optimisation algorithms, thereby leveraging the various available mathematical optimisation techniques such as $A*$, Rapidly Exploring Random Trees (RRT), Genetic Algorithms, or Probabilistic Roadmaps. The path planning problem is typically approached by formulating it as an optimization problem, wherein the decision variables represent waypoints or control inputs that steer the vehicle along the desired trajectory. The objective function quantifies the desirability of different paths, considering factors such as minimizing lap time, maximizing energy efficiency, or ensuring safety. Constraints are imposed to satisfy requirements like collision avoidance, vehicle dynamics, track limits, and racing rules.

The process of optimisation-based path planning typically involves the following steps: problem formulation, state and action space representation, objective function design, and finally solving the resultant optimisation problem. The problem formulation step is defined by specifying the start and goal locations as well as the environment, or map representation, and the constraints associated with the environment (if any). The state and action space representation is where the possible states available to the vehicle are defined mathematically. The objective function design defines the desirability of a path. This can range from minimizing a distance, or time, or maximizing efficiency. Finally, the optimisation algorithm is executed and the resultant solution will represent the optimal or

near-optimal path.

5.2. Optimal Control

Optimal control can also be used for path planning and shares a strong connection with optimisation-based path planning. Optimal control is a branch of control theory that focuses on determining the control inputs or actions that optimize a specific performance criterion for a dynamic system. It involves finding the control strategy that minimizes or maximizes a cost function while considering constraints imposed on the system. Both fields aim to find optimal or near-optimal solutions, but their perspectives differ in terms of problem formulation and emphasis. Both fields involve defining objectives, constraints, and system dynamics, and employing optimization algorithms to find solutions that optimize the defined criteria. Optimal control problems focus on determining the optimal control inputs that steer a dynamic system from an initial state to a desired final state while minimizing a cost function. This ultimately generates a motion plan, which contains the required control inputs that will move the vehicle to the final state in an optimal way. In contrast, optimization-based path planning centres around finding the optimal path or trajectory itself without necessarily considering the system dynamics of the vehicle or the required control inputs to the vehicle that will allow it to execute the planned trajectory.

Optimal control provides a framework to mathematically model the vehicle dynamics, incorporate system constraints, and design objective functions that capture racing performance, safety, and energy efficiency. The resulting optimal control problem can then be solved using numerical optimization methods, such as gradient-based algorithms, particle swarm optimisation, or other more sophisticated methods. Formulating the path planning problem as an optimal control problem allows for any control input constraints to be incorporated directly into the optimisation problem, as well as including the system dynamics directly in the path planning process which yields a more comprehensive motion plan as opposed to a simple trajectory

Motion planning using optimal control follows a similar process to optimisation-based path planning where the system dynamics are captured in a mathematical model. This model describes the relationship between the system's state variables, control inputs, and the environment. Constraints are then imposed to ensure that the planned trajectory adheres to the system limitations, implementation requirements and safety considerations. Following this, an objective function is designed to quantify the desirability of the system behaviour which is typically expressed as a cost function. Finally, the optimal control problem is formulated and solved with the objective of finding the control inputs that minimize the cost function while satisfying the system dynamics and constraints. The formulation considers the initial state, the desired final state or goal, and the time horizon

over which the optimization is performed.

5.2.1. Interior point optimisation method

The interior point method is an iterative algorithm that solves convex optimization problems by exploring the interior of the feasible region rather than traversing the boundary. Unlike traditional methods like the simplex algorithm, which moves along the constraint boundaries, the interior point method converges to the optimal solution by iteratively searching within the feasible region. It achieves this by introducing a barrier or logarithmic barrier function that penalizes violations of constraints and pushes the iterates toward the feasible interior.

The interior point method employs Newton's method to solve a sequence of barrier sub-problems, which are optimization problems with the objective of minimizing the barrier function while satisfying the constraints. At each iteration, it approximates the solution of the sub-problem using second-order information, such as the Hessian matrix, to improve convergence speed. By adjusting a parameter known as the barrier parameter, the algorithm strikes a balance between ensuring feasibility and approaching the optimal solution.

5.3. Global Path Planning

The global path planning problem is formulated as an optimal control problem and solved using an interior point optimisation method. This yields a motion plan that includes the necessary control inputs for the car to execute the global plan. The global path planning process used in this thesis uses the techniques presented by Vazquez *et al.* [46].

5.3.1. System Model

The system is modelled using a dynamic bicycle model set in the curvilinear coordinate system of the track, as described in section 4.3.1. The track is fully described by the curvature of the centre line $K(s)$ and the width of the track at each point $2W(s)$ and is parameterised by the centre line position s. The complete state information of the vehicle model is described by the in-track distance s, the cross-track distance n, the local heading mu, the longitudinal velocity v_x, the latitudinal velocity v_y and the yaw rate of the car r. The in-track distance s, the cross-track distance n, and the local heading μ are referenced to the centre line of the track. The inputs to the vehicle model are the normalised throttle

command T and the steering angle δ.

$$
\begin{aligned}
\dot{s} &= \frac{v_x \cos(\mu) - v_y \sin(\mu)}{1 - nK(s)} \\
\dot{n} &= v_x \sin(\mu) + v_y \cos(\mu) \\
\dot{\mu} &= r - \dot{s}K(s) \\
\dot{v_x} &= \frac{1}{m}(F_x - F_{y,F} \sin(\delta) + m v_y r) \\
\dot{v_y} &= \frac{1}{m}(F_{y,R} + F_{y,F} \cos(\delta) - m v_x r) \\
\dot{r} &= \frac{1}{I_z}(F_{y,F} l_f \cos(\delta) - F_{y,R} l_r)
\end{aligned}
\tag{5.1}
$$

The state dynamics of the vehicle model is given by $f_c^t(\mathbf{x}, \mathbf{u})$, where $\mathbf{x}$ is the state vector and $\mathbf{u}$ is the input vector consisting of the throttle input T and the steering angle δ. The subscript c denotes the continuous nature of the dynamics and the superscript t shows that this is a time-based state equation.

Time-based model vs Spatial-based model

The vehicle dynamics that have been described are formulated with respect to time. However, these vehicle dynamics are transformed to be dependent on a spatial element instead [75–77]. This transformation is described below:

$$
\begin{aligned}
\dot{\mathbf{x}} &= \frac{\partial \mathbf{x}}{\partial t} = \frac{\partial \mathbf{x}}{\partial s}\frac{\partial s}{\partial t} \\
\rightarrow \frac{\partial \mathbf{x}}{\partial s} &= \frac{1}{\dot{s}} f_c^t(\mathbf{x}, \mathbf{u}) = f_c^s(\mathbf{x}, \mathbf{u})
\end{aligned}
\tag{5.2}
$$

The state dynamic equations are shown in 5.2, where $f_c^s(\mathbf{x}, \mathbf{u})$ is the continuous spatial dependent state equations. In particular, the spatial element that the state equations are now dependent on is the in-track distance s. Due to this transformation, the state s becomes redundant and can be removed from vehicle state space thereby creating a reduced state space consisting only of n, μ, v_x, v_y, and r. The reduced state space is written as $\overline{\mathbf{x}}$ and thus the new state dynamics are given by the $f_c^s(\overline{\mathbf{x}}, \mathbf{u})$. This transformation and reduced state space are useful for the purposes of formulating the path planning optimisation problem, as the curvature of the track is no longer a function of the current vehicle state, which simplifies the formulation of the optimisation problem significantly. Additionally, formulating a minimal-time optimisation problem with time as an independent variable is inherently complex and often leads to a non-convex optimisation problem which

is particularly challenging to solve. However, this transformation lends itself to the formulation of a minimum-time cost function for the path planning problem where time is not directly considered and the spatial element is the independent variable. This can be seen below, where the time that it takes for the car to move from any given position $s = s_0$ to another position $s = s_T$ is calculated as:

$$Time = \int_{s_0}^{s_T} \frac{1}{\dot{s}} \, d\sigma \tag{5.3}$$

Therefore, the lap time, L, that it takes for the vehicle to travel from the beginning of the track to the end of the track, is the integral of $\frac{1}{\dot{s}}$ from the starting position to the end of the track, therefore providing a natural cost function for the path planning optimisation.

This spatial domain model does introduce a singularity when $\dot{s}$ is 0 which means that the car has stopped. It is also important to note that the use of the spatial domain model implies that when used for the control of the system, the inputs to the system should be given at a fixed sampling distance as opposed to a fixed sampling time [77].

Discretisation

Before formulating the optimisation problem, the continuous vehicle model and the racing track model that have been developed are discretized. Using the pre-mapped race track, cubic splines are used to create a continuous track centre line. The splines representing the centre line of the race track are discretized with a constant discretization distance Δs. The discretization distance Δs is chosen such that the total arc length is equal to $\Delta s(N+1)$, where N is the number of discretized points that are chosen for the optimisation problem.

The continuous spatial-based state equations $f_c^s(\overline{\mathbf{x}}, \mathbf{u})$ are also discretized with respect to the track centre line s, creating the discrete state equations $f_d^s(\overline{\mathbf{x}}, \mathbf{u})$. This is done using forward Euler integration using a spatial integration element. Specifically, Δs is used in the discretization of the vehicle model as shown below:

$$f_d^s(\overline{\mathbf{x}}_k, \mathbf{u}_k) = \overline{\mathbf{x}}_{k+1} = \overline{\mathbf{x}}_k + \Delta s f_c^s(\overline{\mathbf{x}}_k, \mathbf{u}_k) \tag{5.4}$$

5.3.2. Optimisation setup and Constraints

The necessary constraints are formulated in order to ensure that the car remains within the track limits as well as ensuring that the car stays within the limits of its handling capabilities.

Track Limits

The constraint that ensures that the car stays on the track is formulated considering the local heading of the car as well as the orthogonal distance away from the centre line. The constraint can be seen in equation 5.5

$$\begin{aligned} n + \frac{L_c}{2}\sin(|\mu|) + \frac{W_c}{2}\cos(\mu) \leq W(s) \\ -n + \frac{L_c}{2}\sin(|\mu|) + \frac{W_c}{2}\cos(\mu) \leq W(s) \end{aligned} \tag{5.5}$$

where the constraint uses the cross-track position n, the total length of the car L_c, the total width of the car W_c and the radius of the track $W(s)$ and is denoted by $\overline{\mathbf{x}} \in \mathcal{X}_{track}$. An illustration of the constraint is shown in figure 5.1.

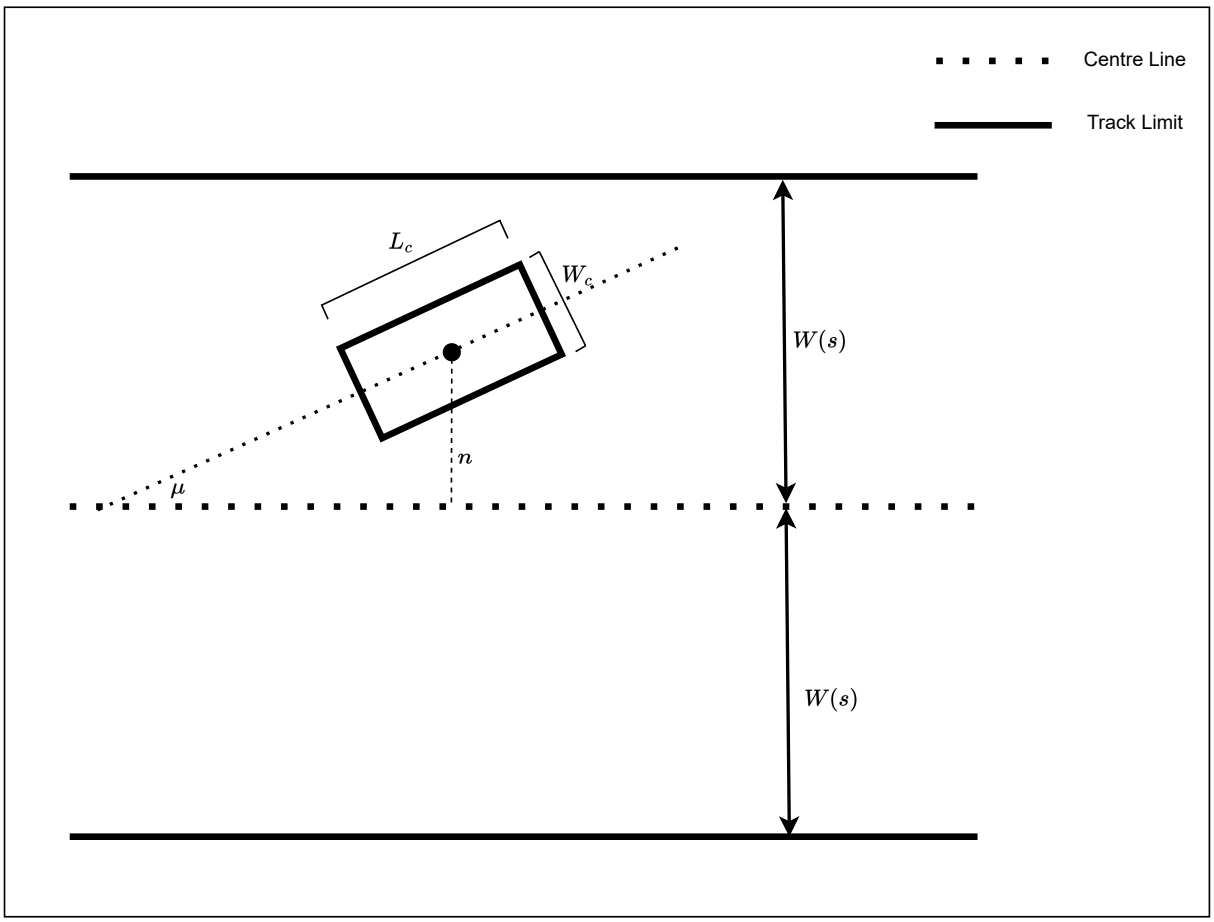


Figure 5.1: An illustration of the constraint that keeps that car within the track

Friction Constraints

The tyre model used in the dynamic bicycle model described in 4.1.2 that is used in the path planning optimisation does not consider combined tyre slip, and therefore combined

tyre forces. A friction ellipse constraint is implemented to limit the combined maximum tyre force, which is shown in equation 5.6.

$$
\begin{aligned}
(\rho_x \frac{F_M}{2})^2 + (F_{y,F})^2 \leq (\lambda D_F)^2 \\
(\rho_x \frac{F_M}{2})^2 + (F_{y,R})^2 \leq (\lambda D_R)^2
\end{aligned}
\tag{5.6}
$$

The constraint uses the longitudinal motor force F_M which is assumed to be applied equally between both the front and the back wheels. This assumption is based on the assumptions that were made when developing the vehicle model in section4.1.2. Although the assumption becomes less accurate when the car is undergoing extreme acceleration and deceleration, it is still valid for the purpose of constraining the maximum tyre forces. The shape of the friction ellipse is given by ρ_x, while the maximum tyre force is determined by λ. $F_{y,F}$ and $F_{y,R}$ are the front and rear lateral tyre forces and D is the peak Pacejka tyre parameter for each tyre. An illustration of the friction ellipse is shown in figure 5.2.

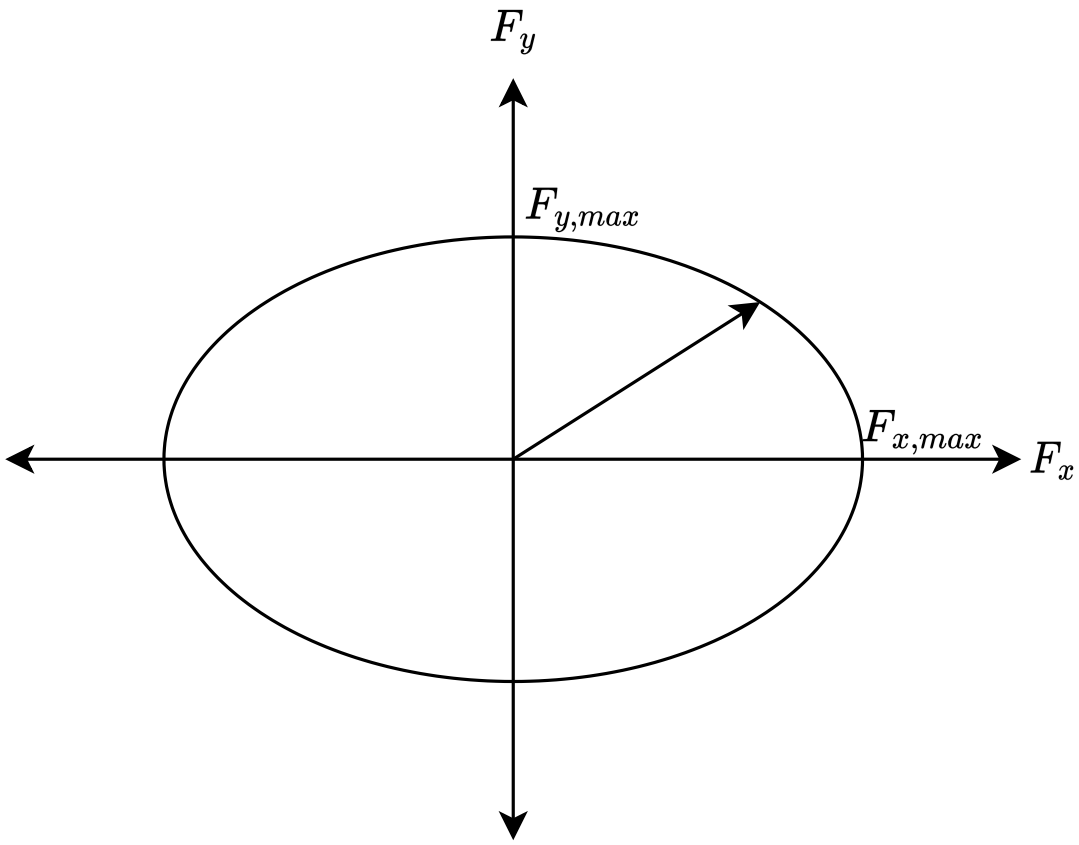


Figure 5.2: An illustration of the friction ellipse constraint

This constraint limits the total force that the tyre will experience at any given point. The longitudinal tyre force is produced by the motor where the motor force F_M is assumed to be split evenly between both the front and back wheels. ρ_x determines the shape of the friction ellipse, determining how much force can be applied in the longitudinal direction. λ determines the total combined force that can act on the tyres. This friction constraint is denoted by $\overline{\mathbf{x}} \in \mathcal{X}_{friction}$

Input Constraints

Both of the inputs to the vehicle mode must be constrained to the limits of the real car. The throttle command is constrained to be within the range of -1 and 1 in equation 5.7

$$-1 \leq T \leq 1 \tag{5.7}$$

The steering angle is similarly constrained in equation 5.8. This constraint limits the steering angle to be within the minimum and maximum steering angles that are possible on the car.

$$\delta_{Min} \leq \delta \leq \delta_{Max} \tag{5.8}$$

This set of constraints on the input is denoted by $\mathbf{u} \in \mathcal{A}$

5.3.3. Objective function Design

The cost function aims to minimise the lap time L which is shown in equation 5.9

$$L = \sum_{k=0}^{N} \frac{\Delta s}{\dot{s_k}} \tag{5.9}$$

where Δs is the discretisation distance and $\dot{s_k}$ is the time-based derivative for the in-track position at the in-track position k.

Additionally, it is useful to add some regularisation terms to the cost function, in order to elicit certain behaviour from the system. A penalisation term for the inputs, described in equation 5.10 is added to the cost function in order to minimise erratic changes to the throttle and steering inputs. In the penalisation term Q, $\mathbf{u}$ is the input matrix and R is a diagonal matrix with weights on the diagonal for each input.

$$Q(\mathbf{u}_k) = \mathbf{u}_k^T R \mathbf{u}_k \tag{5.10}$$

The cost function also includes a term that penalises the difference between the dynamic

and the kinematic slip angle as shown in equation 5.11

$$B(\overline{\mathbf{x}}_k, \mathbf{u}_k) = (B(\overline{\mathbf{x}}_k)_{dyn} - B(\mathbf{u}_k)_{kin})^2 \tag{5.11}$$

The kinematic slip angle is a purely geometric parameter, shown in 5.12 that describes the angle between the direction in which the vehicle is pointing and the velocity of the vehicle. When the lateral acceleration of the vehicle increases, the tyre behaviour becomes more dynamic and the kinematic slip angle becomes less accurate.

$$B(\mathbf{u}_k)_{kin} = \arctan(\frac{\delta_k l_R}{(l_R + l_F)}) \tag{5.12}$$

where δ_k is the steering angle at the in-track position k, l_R is the distance from the CoG to the rear wheelbase, and l_f is the distance from the CoG to the front wheelbase.

The dynamic slip angle is a more accurate representation of the tyre slip angle that takes the vehicle velocity into account as shown in 5.13.

$$B(\overline{\mathbf{x}}_k)_{dyn} = \arctan(\frac{v_{y,k}}{v_{x,k}}) \tag{5.13}$$

where $v_{y,k}$ and $v_{x,k}$ are the lateral and longitudinal velocity of the car at the in-track position k.

Minimising the difference in the dynamic slip angle and kinematic slip angle improves the stability of the vehicle by making the tyres operate closer to their optimal conditions. By aligning the slip angles the lateral forces on the tyres are minimised which reduces the chances of slipping or skidding.

5.3.4. Problem Formulation and Optimisation

The full optimal control problem is shown in equation 5.14 which includes the cost function and the constraints that have been presented.

$$\begin{aligned}
\min_{\overline{\mathbf{x}},\mathbf{u}} \quad & \sum_{k=0}^{N} \frac{\Delta s}{\dot{s}} + Q(\mathbf{u}_k) + B(\overline{\mathbf{x}}_k, \mathbf{u}_k) \\
\text{s.t.} \quad & \overline{\mathbf{x}}_{k+1} = f_d^s(\overline{\mathbf{x}}, \mathbf{u}) \\
& \overline{\mathbf{x}} \in \mathcal{X}_{track} \\
& \overline{\mathbf{x}} \in \mathcal{X}_{friction} \\
& \mathbf{u} \in \mathcal{A}
\end{aligned} \tag{5.14}$$

5.4. System Implementation

This optimal control problem is defined in Python using Pyomo [78, 79] and solved offline using IPOPT [80]. The initial conditions of the vehicle are set and fixed in the optimisation problem. However, given that the spatial-based vehicle model fails when the car is stationary, if the car starts from rest, a small value is used as the initial condition for the longitudinal velocity. The car will use the motion planning results and attempt to execute the plan throughout the track.

5.4.1. Example Output

The optimisation process will find the states and inputs of the vehicle model, which are a function of s, that minimises the cost function while adhering to the imposed constraints. An example of the output from the global path planner is shown in figure 5.3 and figure 5.4. The planned path is shown on the race track with the centre line in figure 5.3 and the vehicle states and inputs are shown in figure 5.4.

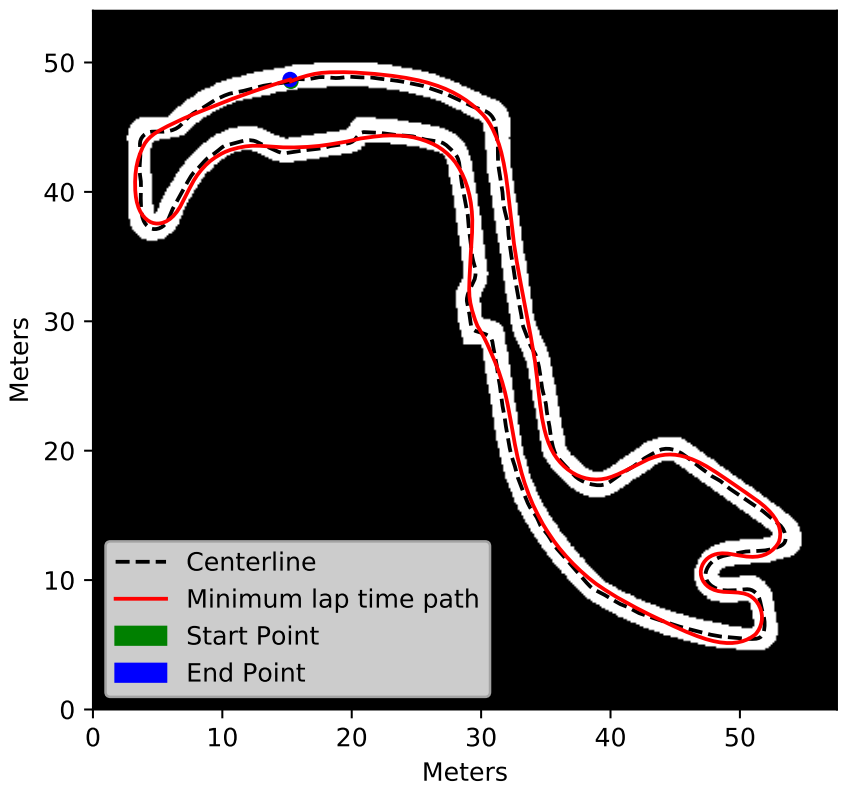
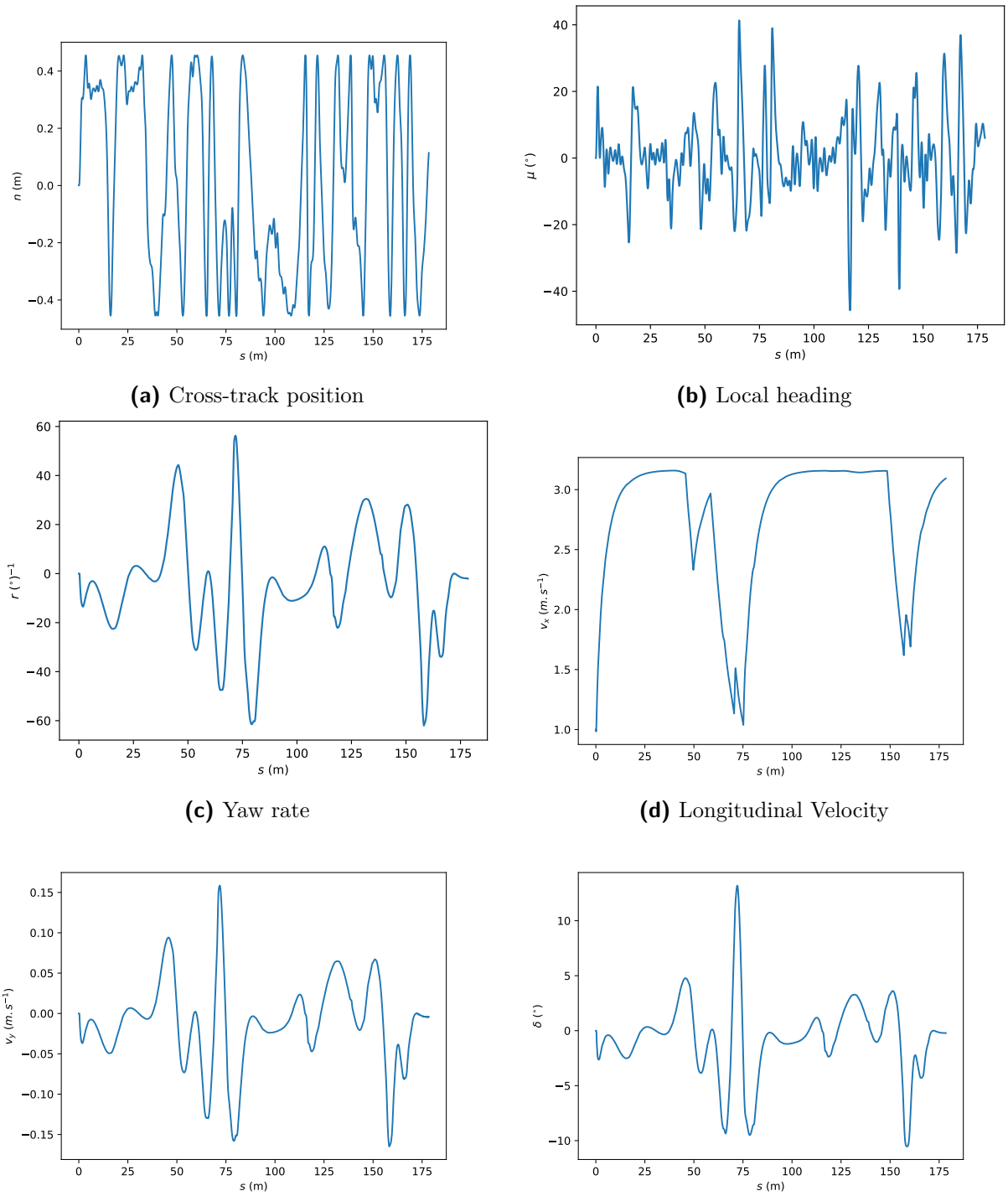


Figure 5.3: Optimal global trajectory for a given racing track.



(a) Cross-track position

(b) Local heading

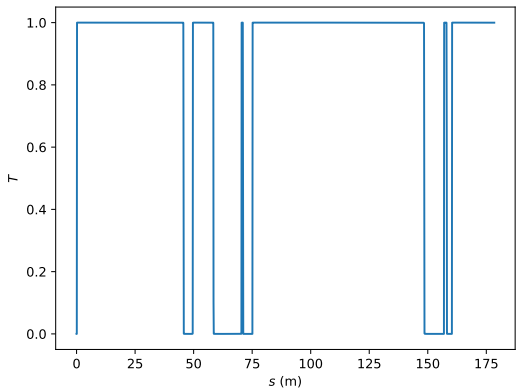
(c) Yaw rate

(d) Longitudinal Velocity

(e) Lateral Velocity

(f) Steering Angle

Figure 5.4: The planned optimal state variables and inputs as a function of the in-track position.



(g) Normalised throttle Command

Figure 5.4: The planned optimal state variables and inputs as a function of the in-track position.

5.5. Testing

The motion planner was tested on different tracks in order to ensure that the constraints that were placed on the planner were obeyed and that the results of the motion planner were consistent with the expected response.

5.5.1. Benchmark Tests

The motion planning process was tested on individual track elements that make up typical racetracks. The motion planner was tested on a straight, a simple corner, a hairpin corner and a chicane. Most race tracks can be broken down into these 4 components and thus it is useful to see the results of the motion planner on the individual track components when other track elements are not considered in the motion planning problem. The model was consistent for each benchmark track and no parameters were changed during or after a test.

Straight Line

The motion planner was first tested using a straight-line track as shown in figure 5.5. The width of the track is constant throughout the entire track and is 1.8m, meaning that the radius of the track is 0.9m. The track is 50m long and was discretised with a discretisation distance of 0.1m.

The results of the motion planner are shown in figures 5.5 and 5.6. Figure 5.5 shows the path that was planned through the track, and figure 5.6 shows some selected variables from the motion planner. The initial conditions of the car were set to 0.

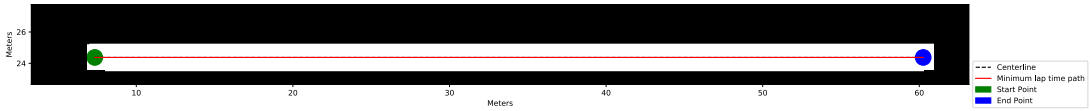
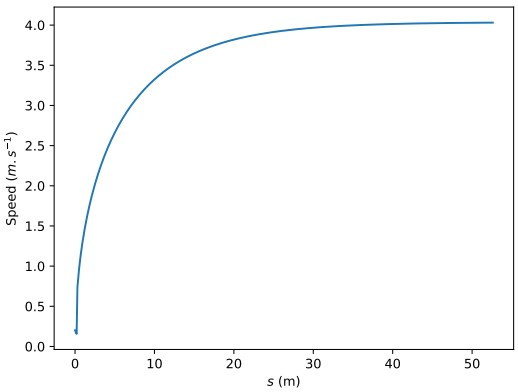
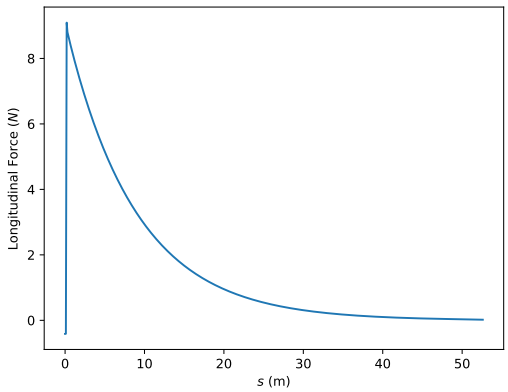


Figure 5.5: Optimised trajectory for a straight line

Figure 5.6a shows the planned speed of the car increases as the car accelerates from its near-stationary initial condition, and then tends exponentially towards its maximum speed with a steady state value of 4.03ms^{-1}. This is achieved through the planned throttle command T, shown in figure 5.6f, which increases instantly to a maximum value that the car friction parameters will allow for and stays constant for the entire straight line. The effect of the constant throttle command is represented in the longitudinal force that the car would experience as shown in figure 5.6b, where the longitudinal force that would act on the car immediately rises to a peak when the car would accelerate and then as the speed increases, the force of the air pushing against the car would also increase until the car reaches a steady state. Figures 5.6c and 5.6d show the forces that would act on the front and rear tyre respectively, as well as the elliptical constraint that was placed on the total tyre force that each tyre could accommodate while operating under optimal conditions. It is worth noting that the throttle command does not reach the maximum value that it can reach, meaning that the car could potentially go faster, however, figures 5.6c and 5.6d show both of the tyre forces that would act on the tyre are at the limits of the friction ellipse, meaning that although the motor could exert more longitudinal force on the car, this would, in turn, cause the forces acting on the tyres to increase and therefore likely start slipping on the racing surface. Both of these friction ellipses also show that there is no planned lateral force acting on either of the tyres, which is expected as the optimal racing line for a straight line, is simply just to continue straight and accelerate. Figure 5.6e shows that the planned steering command stays 0 for the entire straight, as expected, meaning the planned path of the car does not deviate from the centre line throughout the straight as seen in figure 5.5.

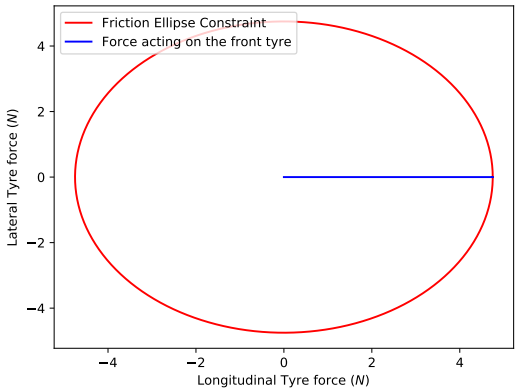


(a) Speed

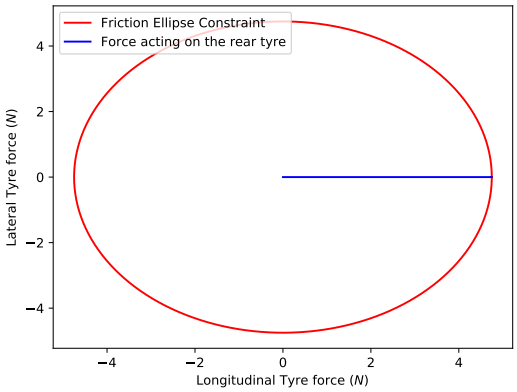


(b) Longitudinal force acting on the car

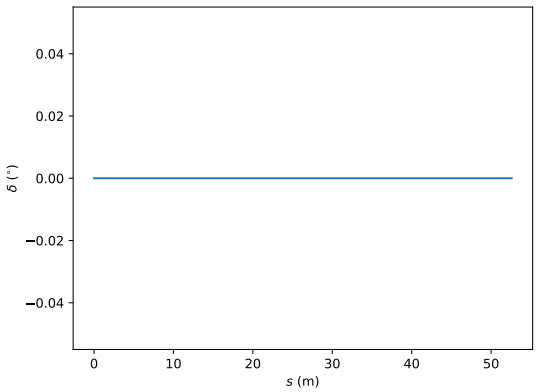
Figure 5.6: The results, as a function of the in-track position, from the solved motion planning problem for a straight line



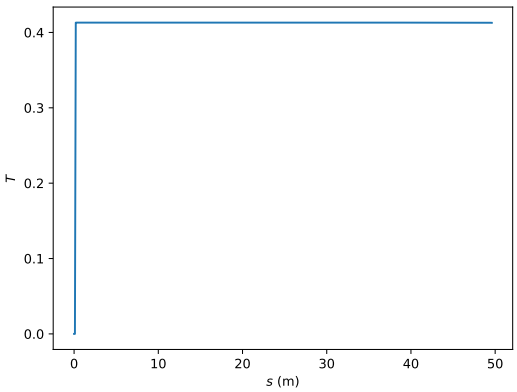
(c) Forces acting on the front tyre of the vehicle



(d) Forces acting on the rear tyre of the vehicle



(e) Steering angle



(f) Normalised throttle command

Figure 5.6: The results, as a function of the in-track position, from the solved motion planning problem for a straight line

Corner

The motion planner was tested on a corner shown in figure 5.7. The track radius is 0.9m throughout the track and the total length of the track centre line is 37m and is discretised with a discretisation distance of 0.1m.

The results of the motion planner are shown in figures 5.7 and 5.8. Figure 5.7 shows the path that was planned through the track, and figure 5.8 shows some selected variables from the motion planner. The car was started on the centre line, with initial conditions equal to the terminal conditions from the straight-line track motion plan. These conditions are a longitudinal velocity of 4.036ms^{-1} and a throttle command of 0.413. The initial conditions of the rest of the states and inputs were set to 0.

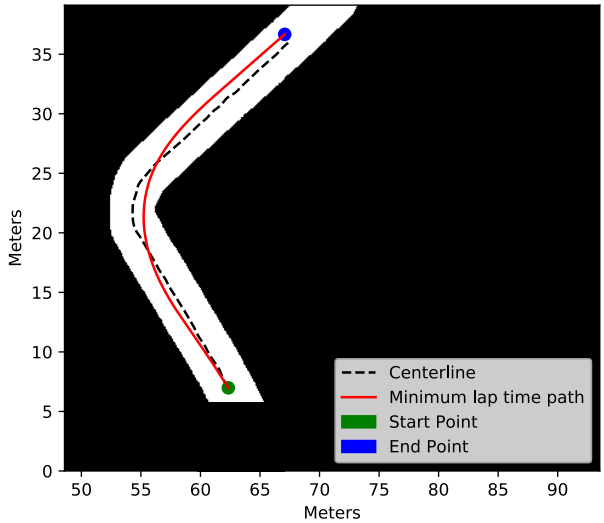
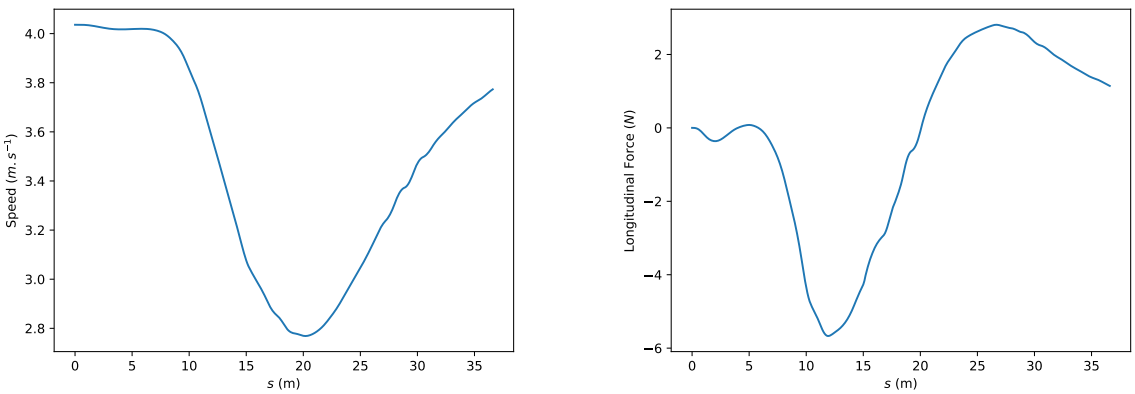


Figure 5.7: Optimised trajectory for a corner

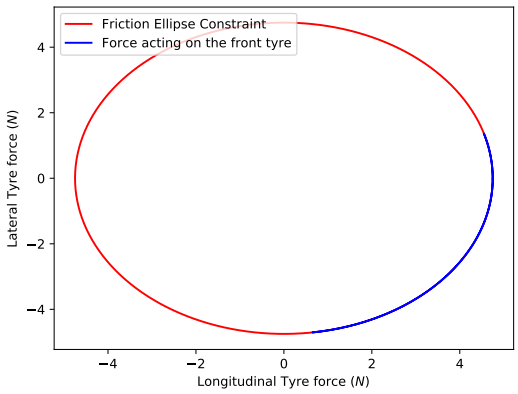
Figure 5.8a shows the planned speed of the car as a function of s. The car would maintain its speed initially as it progresses through the track, however, once the car reaches the corner, the planned speed of the car begins to decrease. This is caused by the planned throttle command, seen in figure 5.8f. The motion plan does not actively apply a braking force, as seen in figure 5.8f, but rather the planned throttle command decreases such that the car would slow down to the point that it can sufficiently navigate the corner without slipping. This decrease in throttle command is reflected in the longitudinal force that would act on the car as shown in figure 5.8b. The motion plan results show that once the car had successfully manoeuvred around the corner, the throttle command would begin to increase and eventually reach a steady state after the car had fully rounded the corner. This increase in the planned throttle command correspondingly leads to an increase in the longitudinal force that would act on the vehicle. When the throttle command reaches

a steady state, the longitudinal force acting on the vehicle begins to decline, tending to a steady state longitudinal vehicle force of 0N. The speed of the car also tends to a steady state value, similar to the results of the motion planner for a straight line. The longitudinal force acting on the car is not directly related to the acceleration of the car as the steering angle of the car changes throughout the car's planned motion throughout the track, shown in figure 5.8e, which causes additional lateral forces to act on the car. These lateral forces make the car gain a lateral acceleration which leads to a change in the car's cross-track position as shown in figure 5.5, which shows the planned position of the car throughout the track. The planned trajectory shows the car moving slightly to the outside of the corner initially, and then moving across the centre line in order to get as close as possible to the inside edge of the track. This planned path allows the car to maintain as much speed as the friction constraints allow for, as well as decreasing the distance that the car would have to travel. The planned forces that would act on the front and rear tyres are shown in figures 5.8c and 5.8d, and correspond to the rest of the results, as the longitudinal tyre forces start at the constrained maximum values with no lateral tyre force. However, when the car would make its way around the corner, the longitudinal tyre force would decrease and the lateral tyre force would increase until it reaches a maximum at the apex of the corner. The front tyre forces traverse the edge of the friction ellipse constraints as the longitudinal tyre force decreases and the lateral tyre force increases. The plan shows that once the car manoeuvres through the apex of the corner, the front tyre forces again traverse the edge of the friction ellipse as the lateral tyre force decreases and the longitudinal tyre force increases. The rear tyre forces behave similarly, however, the rear lateral tyre forces do not reach the same peak as the front lateral tyre forces.

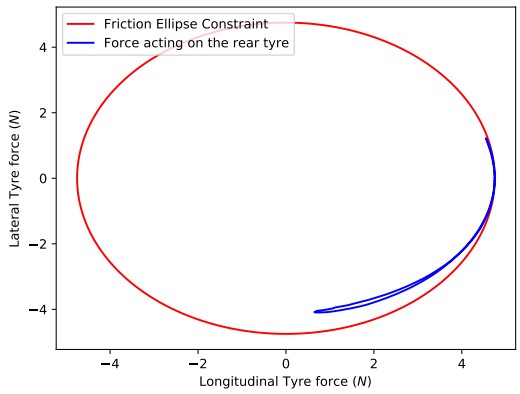


(a) Speed **(b)** Longitudinal force acting on the car

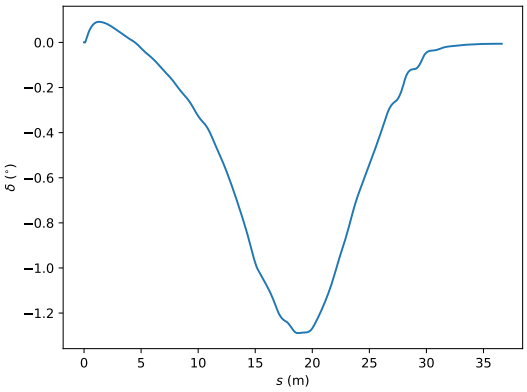
Figure 5.8: The results, as a function of the in-track position, from the solved motion planning problem for a corner



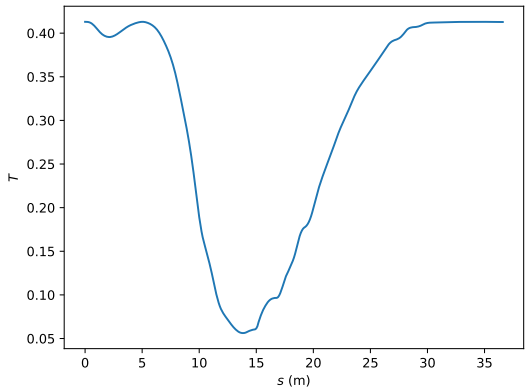
(c) Forces acting on the front tyre of the vehicle



(d) Forces acting on the rear tyre of the vehicle



(e) Steering angle



(f) Normalised throttle command

Figure 5.8: The results, as a function of the in-track position, from the solved motion planning problem for a corner

Hairpin

The motion planner was then tested on a hairpin shown in 5.9. The track radius is 0.9m throughout the track and the total length of the track centre line is 37m and is discretised with a discretisation distance of 0.1m

The results of the motion planner are shown in figures 5.9 and 5.10. Figure 5.9 shows the path that was planned through the track, and figure 5.10 shows some selected variables from the motion planner. Similarly to the previous motion planning test on the corner, the car was started on the centre line, with initial conditions equal to the terminal conditions from the straight-line track motion plan. These conditions are a longitudinal velocity of 4.036ms^{-1} and a throttle command of 0.413. The initial conditions of the rest of the states and inputs are set to 0.

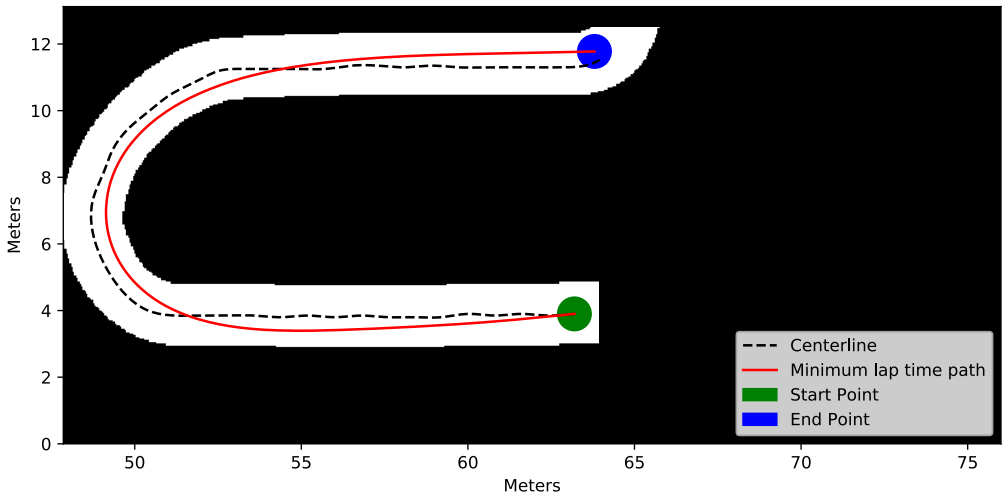
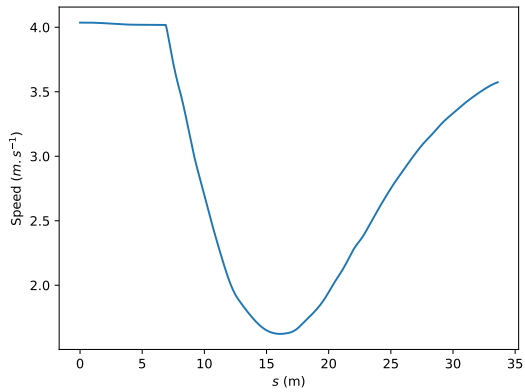
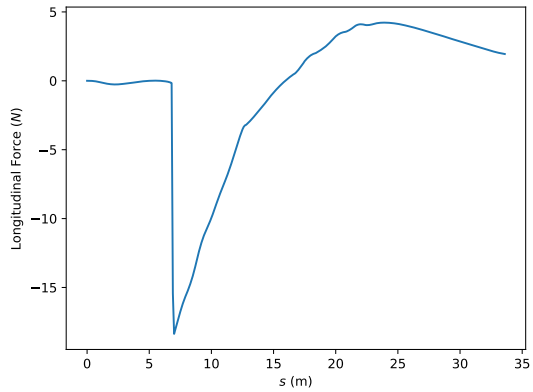


Figure 5.9: Optimised trajectory for a hairpin corner

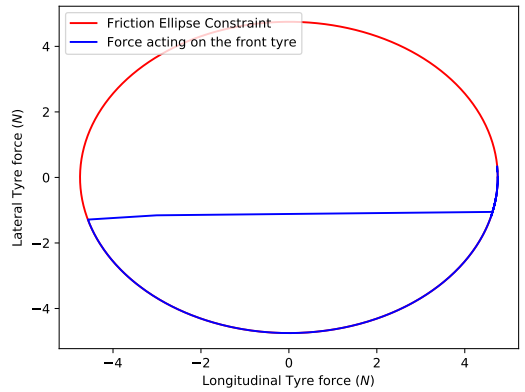
The results of the motion planner that was tested on the hairpin were similar to the results of the motion planning results for the corner. However, figure 5.10f shows that the planned throttle command decreases to a negative value meaning that the car plans to actively brake before the hairpin in order to slow down. This can be seen in figure 5.10b which shows the longitudinal force acting on the car. The planned throttle command decreases to -0.4 in one sampling instant, and in that instant, the longitudinal force acting on the car also decreases to a minimum. This causes a faster deceleration of the car, as shown in figure 5.10a. Additionally, the car's planned speed decreases to a slower speed compared to the minimum speed of the motion plan for the corner. Figure 5.10e shows that the planned steering angle decreases as the car starts to slow down and the car would begin to manoeuvre around the corner. When the car reaches the apex of the corner, where the speed and the steering angle of the car are at a minimum, the motion plan shows that the car begins to accelerate and the speed increases. The planned steering angle also begins to correct back to a steady state of $0°$. The tyre forces also behave similarly to the results from the motion plan for the corner, however, the motion plan for the hairpin has a negative longitudinal tyre force on the front and rear tyres due to the braking action. The tyre forces remain within the friction ellipse constraint as expected.



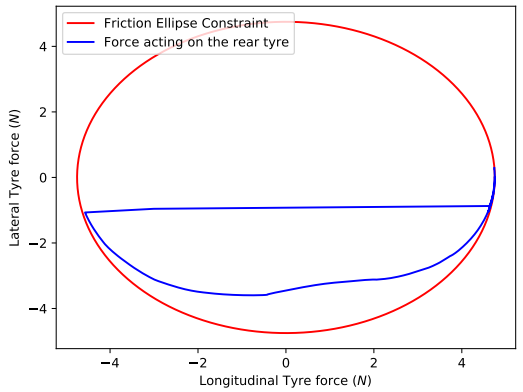
(a) Speed



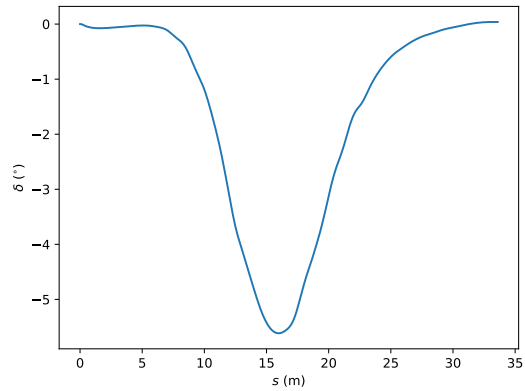
(b) Longitudinal force acting on the car



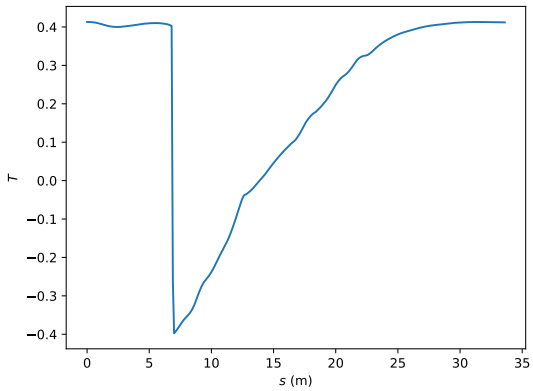
(c) Forces acting on the front tyre of the vehicle



(d) Forces acting on the front tyre of the vehicle



(e) Steering angle



(f) Normalised throttle command

Figure 5.10: The results, as a function of the in-track position, from the solved motion planning problem for a hairpin

Chicane

Finally, the motion planner was tested on a chicane, as shown in figure 5.11. The track radius is 0.9m throughout the track, the total length of the track centre line is 37m, and the track is discretised with a discretisation distance of 0.1m

The results of the motion planner are shown in figures 5.11 and 5.12. Figure 5.11 shows the path that was planned through the track, and figure 5.12 shows some selected variables from the motion planner. Similar to the previous motion planning tests on the corner and hairpin, the car was started on the centre line, with initial conditions equal to the terminal conditions from the straight-line track motion plan. These conditions are a longitudinal velocity of 4.036ms^{-1} and a throttle command of 0.413. The initial conditions of the rest of the states and inputs are set to 0.

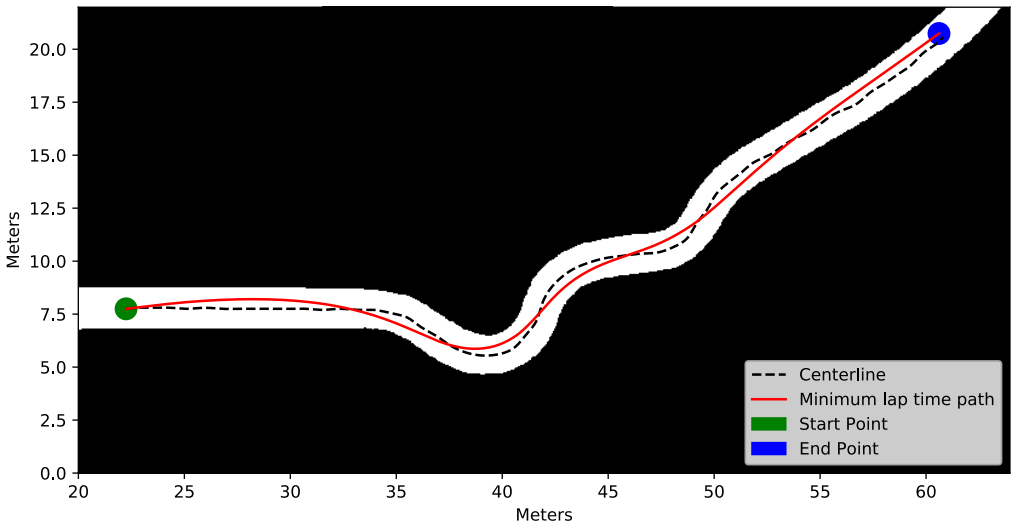
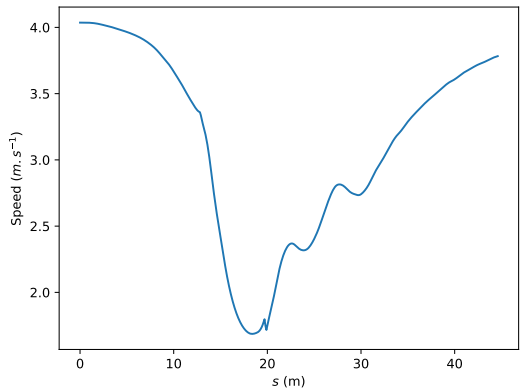
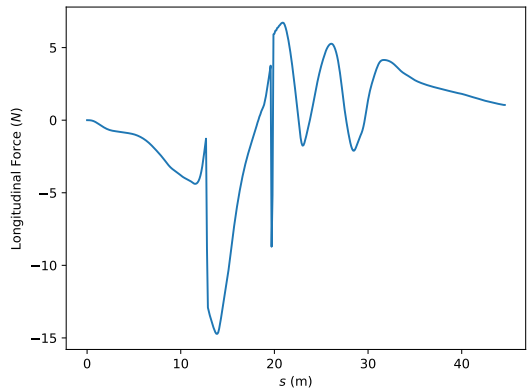


Figure 5.11: Optimised trajectory for a chicane

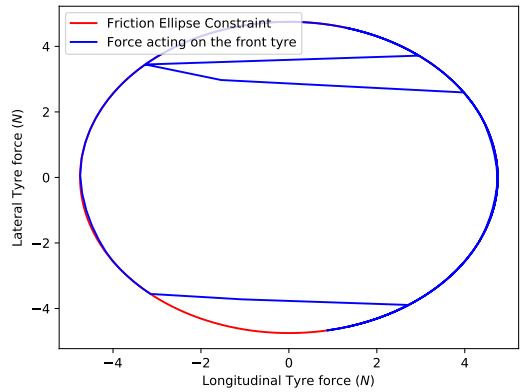
Figure 5.11 shows that the planned trajectory for the car cuts close to the right of the track at the beginning of the chicane and then keeps quite close to the centre line, with minor adjustments in order to manoeuvre around the bends in the track. Figure 5.12e shows the planned steering angle from the motion planner that would execute the planned trajectory. The planned speed also decreases along the planned trajectory when the car enters the chicane as shown in figure 5.12a. The planned throttle command decreases initially, which would actively apply a braking force to the car. The planned throttle command then increases as the car progresses through the chicane. However, the planned throttle command decreases 3 times while the car is progressing through the chicane. This corresponds with the planned speed of the car, which increases while the car is progressing through the chicane, however, the planned speed does decrease slightly at 3 points along the trajectory. This allows the tyre forces, shown in figures 5.12c and 5.12d, to stay within the tyre friction ellipse constraint that is imposed on the motion planner.



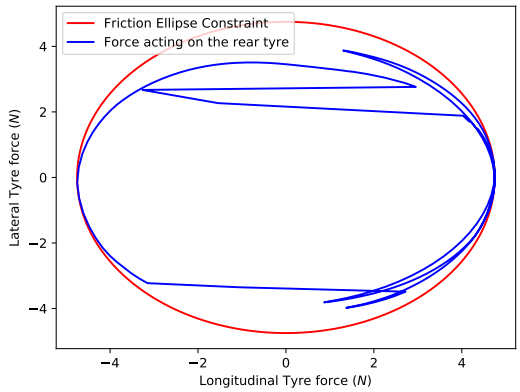
(a) Speed



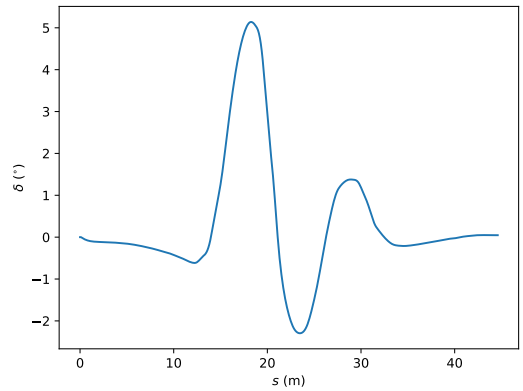
(b) Longitudinal force acting on the car



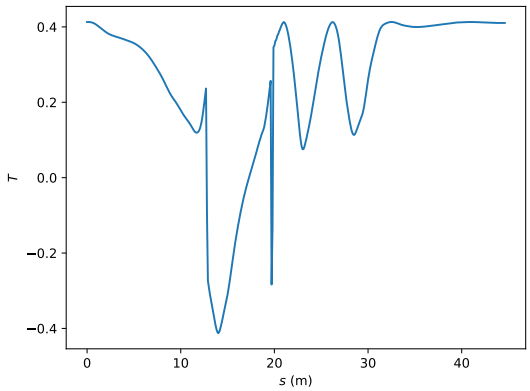
(c) Forces acting on the front tyre of the vehicle



(d) Forces acting on the rear tyre of the vehicle



(e) Steering angle



(f) Normalised throttle command

Figure 5.12: The results, as a function of the in-track position, from the solved motion planning problem for a chicane

5.5.2. Full Track Test

The motion planner was then tested on a full track, the Circuit de Barcelona-Catalunya, which is shown in figure 5.13. This track contains all four track elements that were previously tested with the motion planner, and therefore this test will show the results of

the motion planner for a combination of all these elements. The initial condition for the longitudinal velocity of the car was set to 1ms^{-1}. The initial conditions of the rest of the states and inputs were set to 0.

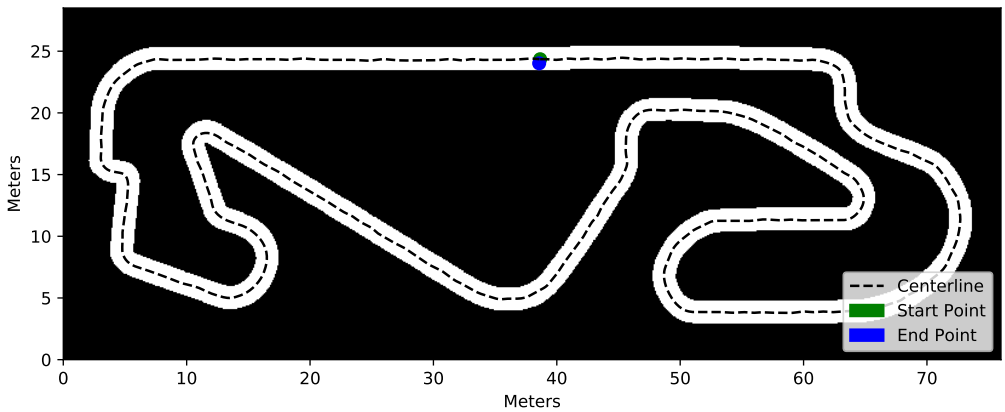


Figure 5.13: An illustration of the Circuit de Barcelona-Catalunya and the centre line of the race track.

The results of the motion plan are shown in figure 5.14 and 5.15. Figure 5.14 shows the planned trajectory on the racing track, while figure 5.15 shows some selected variables from the motion planner. These results show that although there are similar elements in this racing track as the previous benchmark tests, the results are not the same, as the upcoming track elements have an impact on the optimal motion plan for the part of the track for which the car may be planning. The motion planner calculates the optimal motion plan for the entire track and thus the effect of how one track element leads into another is seen in this test.

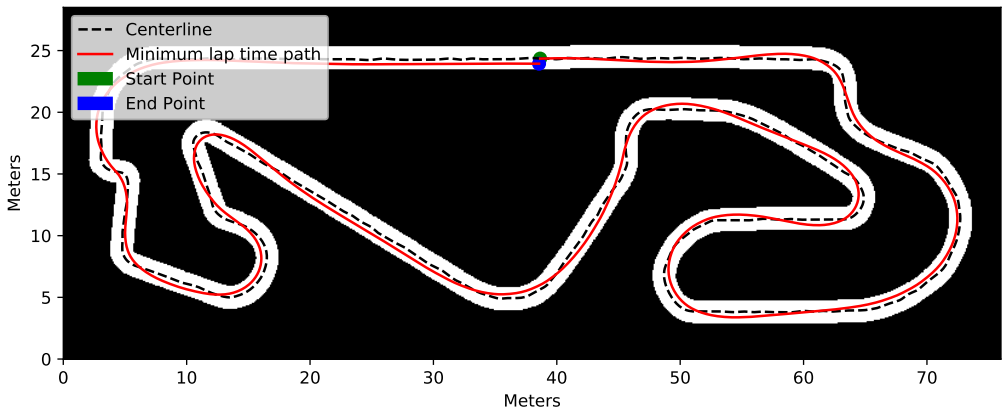
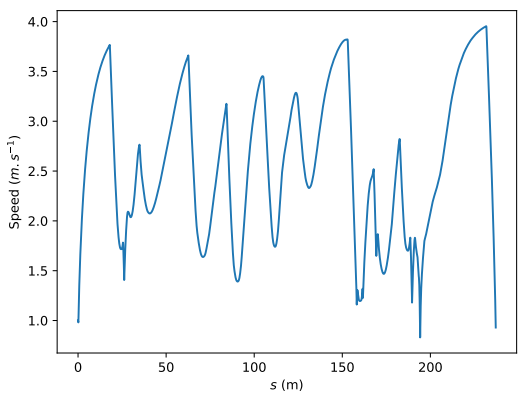
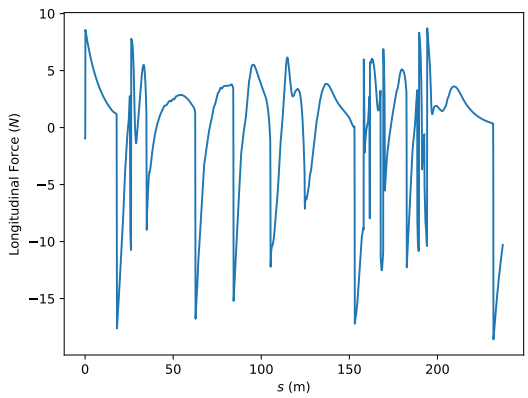


Figure 5.14: Optimised trajectory for the Circuit de Barcelona-Catalunya

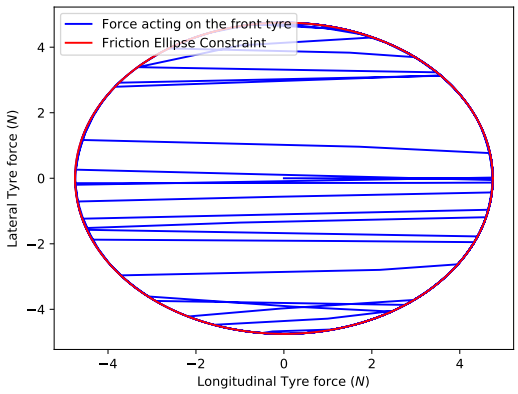
The motion plan results in figure 5.15 shows the car accelerating where it can, and then braking in order to reduce the speed of the car in order to successfully navigate a corner, chicane, or hairpin. The tyre force stays within the elliptical constraints that are imposed, and figure 5.14 shows that the planned trajectory stays within the bounds of the track at all times.



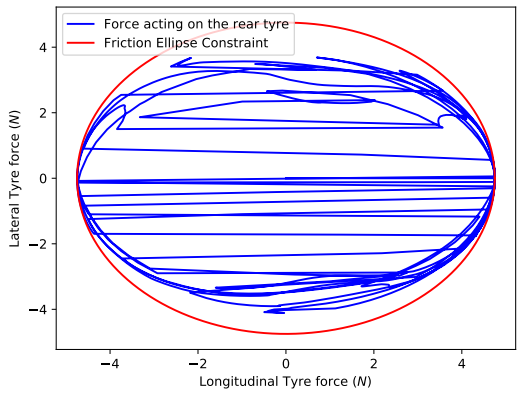
(a) Speed



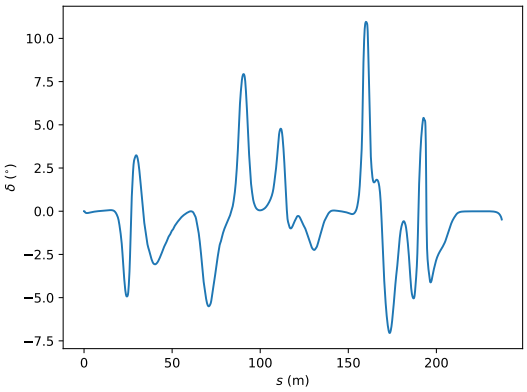
(b) Longitudinal force acting on the car



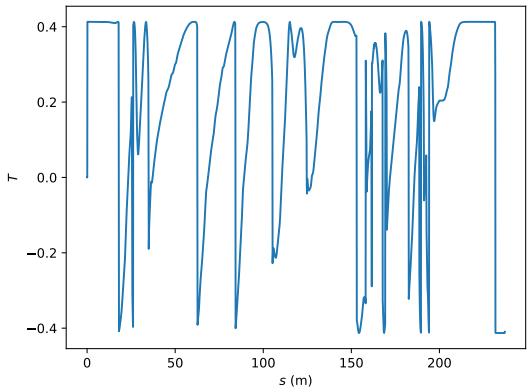
(c) Forces acting on the front tyre of the vehicle



(d) Forces acting on the rear tyre of the vehicle



(e) Steering angle



(f) Normalised throttle command

Figure 5.15: The results, as a function of the in-track position, from the solved motion planning problem for the Circuit de Barcelona-Catalunya

5.6. Summary

This chapter presented the global planner developed and used for this research project. This chapter started with an explanation of optimisation-based path planning. Then, a system model was formulated using the dynamic bicycle model. This model was then transformed from a time-based model into a spatial-based model and discretised. Following this, the constraints implemented in the optimisation formulation were presented, including track limits, friction constraints, and input constraints. An objective function was then defined and the path planning problem was presented was full. The global planner was then tested on various individual track elements as well as on a full track.

Chapter 6

Trajectory Execution

This chapter presents the development of the trajectory execution system that controls the car to follow the reference trajectories generated by the global path planner and the collision avoidance function. First, the Stanley path-tracking controller is described. Following this, a particle filter is presented which is used for localisation of the car. Finally, the results of the car executing the planned trajectory are presented and analysed.

6.1. Trajectory Following

In order for the car to accurately follow the trajectory planned by the hierarchical motion planner, a path-tracking controller is used to minimise the lateral error and heading error while following the reference trajectory.

6.1.1. Stanley Control

The Stanley controller is a widely used control law in the field of autonomous racing that enables a vehicle to navigate along a desired racing trajectory. It was developed by Sebastian Thrun and his team for Stanford University's autonomous vehicle project [64].

Overview

The Stanley controller calculates steering commands that guide a vehicle along a predefined trajectory. Unlike traditional controllers that focus on regulating speed or position, the Stanley controller prioritises lateral control of the car, to ensure accurate tracking of a reference path.

Error Computation

The Stanley controller computes the cross-track error E_d which is the distance between the vehicle's current position and the reference trajectory. Thrun *et al.* use the cross-track error from the front axle to the reference path [64], however, in this thesis, the cross-track error is calculated referenced to the CoG of the car as shown in figure 6.1. The heading

error E_h is calculated as the difference between the vehicle's current orientation and the orientation of the trajectory at the current position. These errors are the basis for generating the steering command.

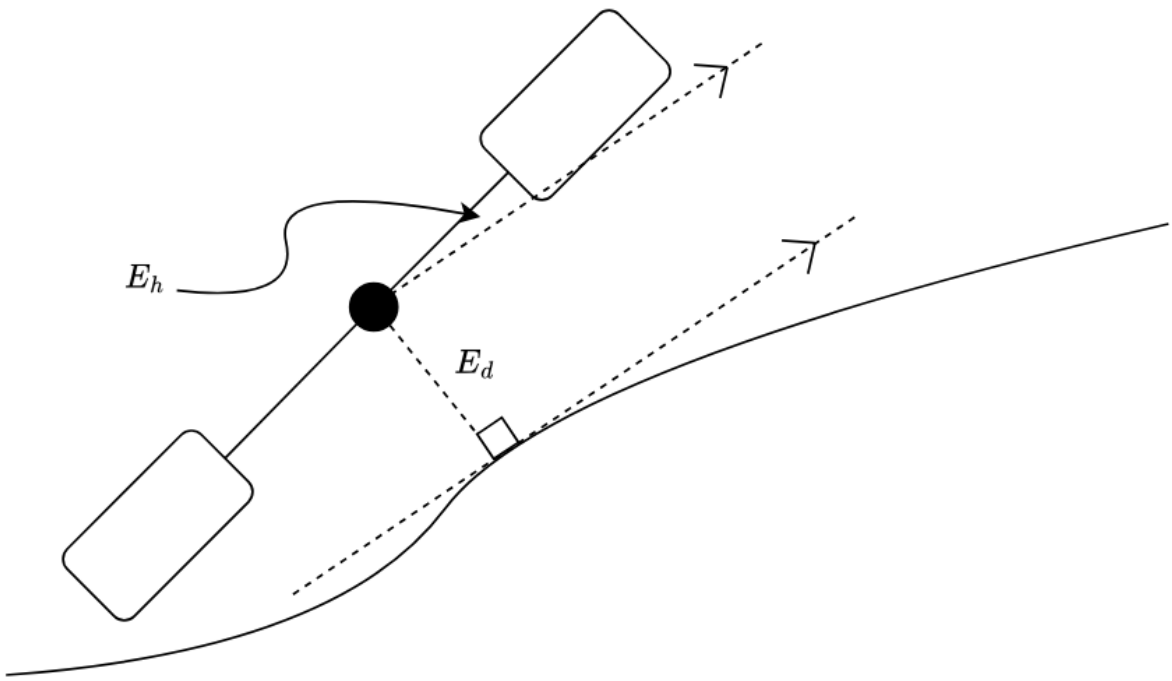


Figure 6.1: An illustration of the cross-track and heading error referenced to a planned trajectory

Control Law

The control law of the Stanley controller combines both lateral error and heading error to generate a steering command. The steering command consists of two components:

1. **Proportional lateral control**: The proportional term is responsible for correcting the lateral error, aiming to minimise the cross-track error. It computes a steering angle based on E_d. The control law is shown in equation 6.1 where K is a tuning parameter that determines the responsiveness of the controller and v is the vehicle's speed. This loosely couples the steering control to the speed of the vehicle.

$$\delta_d = \arctan(\frac{KE_d}{v}) \tag{6.1}$$

2. **Heading Error Correction**: The heading error is used to correct the steering angle computed by the proportional control. This correction aligns the vehicle's

orientation with the tangent of the reference trajectory at the current position.

$$\delta_h = E_h \tag{6.2}$$

These terms are added together to form a steering angle that will follow the reference trajectory as shown in equation 6.3.

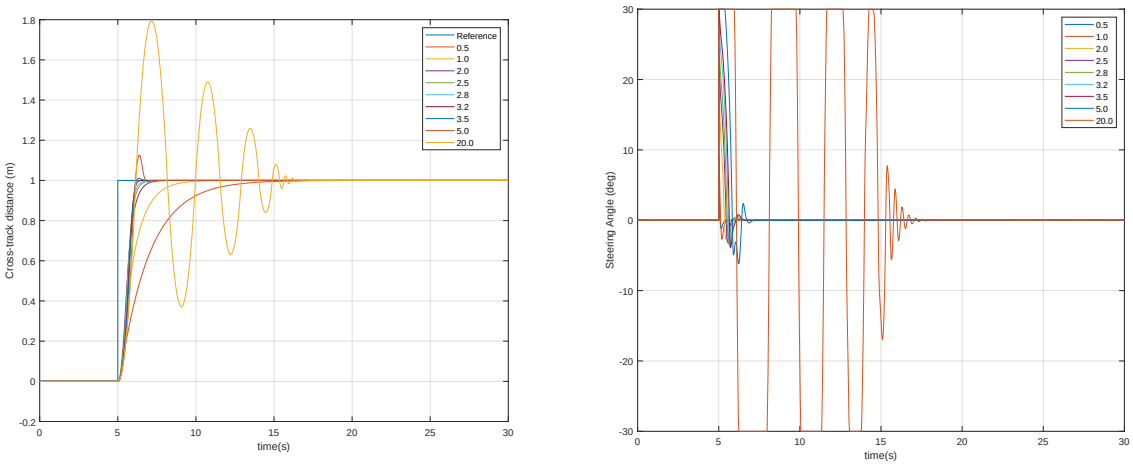
$$\delta_{ST} = \delta_h + \delta_d \tag{6.3}$$

However, the steering angle in equation 6.3 must be bounded to the steering limits of the car. Therefore, the full control law of the path following controller is given by equation 6.4

$$\delta = \begin{cases} \delta_{MIN} & \text{if } \delta_{ST} \leq \delta_{MIN} \\ \delta_{ST} & \text{if } \delta_{MAX} \geq \delta_{ST} \geq \delta_{MIN} \\ \delta_{MAX} & \text{if } \delta_{ST} \geq \delta_{MAX} \end{cases} \tag{6.4}$$

6.1.2. Controller Tuning

The gain K that is used in the lateral control of the Stanley controller in 6.1 was tuned in order to achieve an optimal response. The tuning process was done by simulating the closed-loop system with a reference cross-track distance and various different gains of the Stanley controller. The system model that was used in the simulation is shown in section 4.1.2 and is the same vehicle model that is used in the global planner in chapter 5. By introducing a step in the reference cross-track distance that the car should follow, the step response of the closed-loop system can be observed and changed by manipulating the gain of the controller. The speed of the vehicle was fixed at the maximum speed that the global planner would output. Figure 6.2 shows the response of the system for different values for K. Figure 6.2b shows the steering angle produced by the path tracking controller, and figure 6.2a shows the closed-loop steep responses for the cross-track position of the car. It is important to note that overshoot is a highly undesirable characteristic in the context of racing, as the planned reference trajectory of the vehicle will always remain within the bounds of the track. However, if there is an overshoot present in the system response, this may lead to the car crashing as the overshoot in the cross-track position may cause the vehicle to impact the side of the track. Therefore, the desired closed-loop system response is a fast rise time with minimal overshoot. Based on these requirements and the results in figure 6.2, a gain of 3.2 was selected for the Stanley controller.



(a) **(b)**

Figure 6.2: The results of the gain tuning for the Stanley controller

6.2. System Implementation

The Stanley controller uses the reference motion plan in order to determine the reference heading and the reference cross-track. The reference motion plan is dependent on the car's position, s. Thus the Stanley control steering angle is dependent on the current position s of the car and is not time-dependent. Using the references and the current position and orientation of the car, the appropriate Stanley control steering angle is calculated and used as the control input for the car. The velocity command of the car is obtained directly from the output of the motion planner. This is due to the fact that the global planner output is limited by the dynamics of the vehicle, and thus requires no additional correction. The car hardware includes a VESC, which controls the speed of the car given a reference speed and thus there is no need for another control loop for speed control.

6.3. State Estimation

This section gives an overview of how localisation is performed on a physical F1Tenth car, as well as how the localisation process is integrated with the autonomous racing system.

6.3.1. Particle Filter

Particle filters are a type of Monte Carlo Localisation (MCL) that provides a probabilistic framework for estimating the state of a moving object, such as an autonomous race car, based on sensor measurements and motion models. This section gives an overview of the utilisation of the particle filter within the context of the F1Tenth autonomous racing platform, available on the F1Tenth GitHub repository [68].

Overview

A particle filter operates by representing the possibilities of a system's state using a set of particles. Each particle represents a potential state hypothesis. These particles are propagated through time using a motion model, and their weights are updated based on how well their predicted sensor measurements match the actual sensor measurements. Over time, particles with higher weights will converge towards the true state of the system, providing an estimate of the system's state even in the presence of noise and uncertainties. An illustration of this process is shown in figure 6.3.

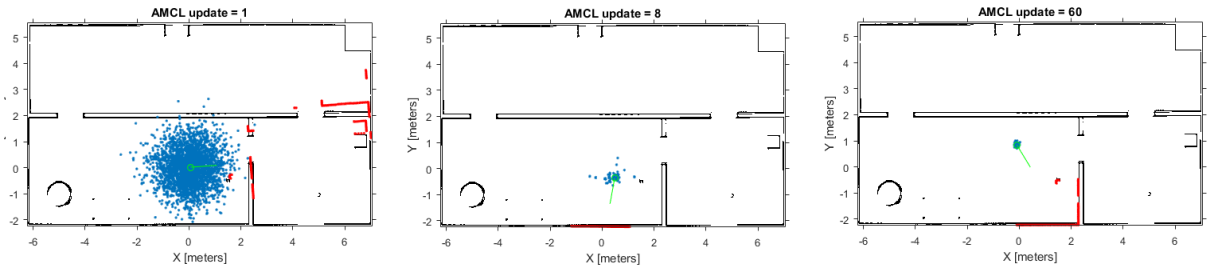


Figure 6.3: An illustration of the particle filter process [1]

F1Tenth Implemenatation

The F1Tenth GitHub repository offers a comprehensive implementation of a particle filter tailored for the specific requirements of autonomous racing. This implementation combines a motion model that captures the kinematics of the race car and a sensor model that incorporates sensor measurements from LiDAR scans and odometry data. The particle filter's workflow can be summarised as follows:

1. **Initialisation**: The particle filter begins with an initial set of particles distributed around the estimated initial pose of the race car.

2. **Motion Update**: The particles are propagated using a motion model that predicts the new state of each particle based on the previous state and control inputs, specifically the velocity and angular velocity. The motion model also takes noise into

account when predicting the state of the particles.

$$
\begin{aligned}
x_{t+1}^i &= x_t^i + \Delta t \cdot v_t \cdot \cos(\theta_t) + \text{noise} \\
y_{t+1}^i &= y_t^i + \Delta t \cdot v_t \cdot \sin(\theta_t) + \text{noise} \\
\theta_{t+1}^i &= \theta_t^i + \Delta t \cdot r_t + \text{noise}
\end{aligned}
\tag{6.5}
$$

Equation 6.5 shows the prediction equations for the state of particle i where (x_t^i, y_t^i, θ_t^i) are the state variables (position and orientation) at time t for particle i. v_t and w_t are the velocity and the angular velocity of the car at time t. The noise represents the added uncertainty in the car dynamics and is represented by a Gaussian distribution

3. **Weight Update**: Using the particles' new predicted positions, the sensor model calculates the likelihood of the sensor measurements given the predicted positions of each particle: This step assigns weights to each particle based on how well its predicted measurements match the actual sensor data.

4. **Resampling**: To ensure that particles representing more probable states are retained and less probable ones are discarded, a resampling step is performed. Particles with higher weights have a higher chance of being duplicated, while particles with lower weights are more likely to be discarded. This process effectively refocuses the particle cloud on the regions of the state space that are more likely to represent the true state of the race car.

5. **Estimation**: The estimated state of the race car can be obtained from the weighted average of the particle distribution. This provides the most likely pose and orientation of the car.

6.3.2. System Implementation

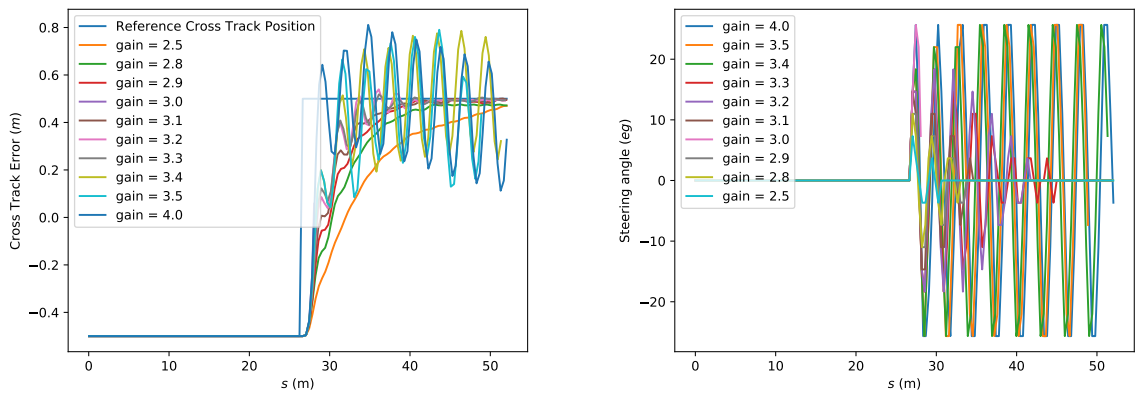
The particle filter is used when the racing system is implemented on the physical vehicle. It is implemented using ROS and is independent of the autonomous racing system. The resulting position and orientation from the particle filter are used by the autonomous racing system as the true position and orientation of the car at that particular time. Thus, all the operations of the autonomous racing system that require the use of the position or orientation of the car use the results of the particle filter. The results of the particle filter are given as Cartesian coordinates and an orientation angle relative to the initial starting pose. This is then transformed into the cross-track and in-track positions by using the transform detailed in section 4.3.

6.4. Simulation Results

The autonomous racing system was tested in simulation using the F1Tenth simulator [70]. The global planner described in chapter 5 was used to generate the optimal trajectory for the vehicle and the Stanley control law described in section 6.1 was used to execute the planned trajectory. The speed reference that is generated by the global planner as a function of the car's in-track position is used as the reference speed for the simulation. The steering angle, Cartesian position coordinates, and orientation of the car are obtained from the simulator at a frequency of 10Hz. A LiDAR scan is also obtained from the simulator at a frequency of 10Hz.

6.4.1. Step Response

The step response of the car was recorded using the F1Tenth simulator, with various different gains for the Stanley controller, in order to confirm that the system response was as expected. The speed of the vehicle was fixed at the maximum speed that the global planner would output, 4.03ms^{-1}. The step response is shown in figure 6.4. Figure 6.4b shows the steering angle of the car as a function of s and figure 6.4a shows the cross-track position of the car as a function of s. The figure shows that the system response is more volatile than the model response in figure 6.2 and that there are small oscillations in the cross-track position of the vehicle as it reaches the steady state reference position. This is because the open-loop simulation did not consider combined tyre slip. When higher gains are used for the Stanley controller, the system becomes unstable and the oscillations grow as the car travels. The system is much more sensitive to changes in the gain of the Stanley controller, and thus it is necessary to retune the gain in order to ensure that the car does not become unstable and potentially crash. Based on the results in figure 6.4, a new Stanley controller gain of 2.9 was selected.



(a) Simulation cross-track position

(b) Simulation steering angle

Figure 6.4: Simulation step response

6.4.2. Benchmark Tracks

The trajectory-following system was tested on the benchmark tracks that the motion plan planner was tested on in section 5.5 in order to determine how well the car can follow a given reference motion plan.

Straight

The trajectory execution was initially tested on a straight, with the states of the car initialised to 0, except for the longitudinal velocity which was set to a small initial value (as a speed of 0ms^{-1} causes an error for the global planner). The car was started at the beginning of the track and set to execute the motion plan generated by the global planner. The simulation results of the car executing the motion plan for the straight-line track are shown in figures 6.5 and 6.6. Figure 6.5 shows the executed trajectory of the car, as well as the speed of the vehicle along the path that it travelled through the track.

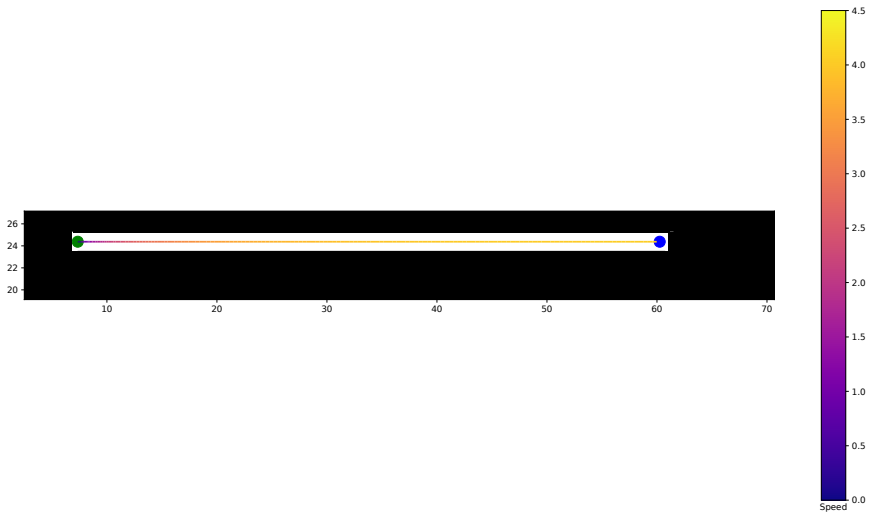
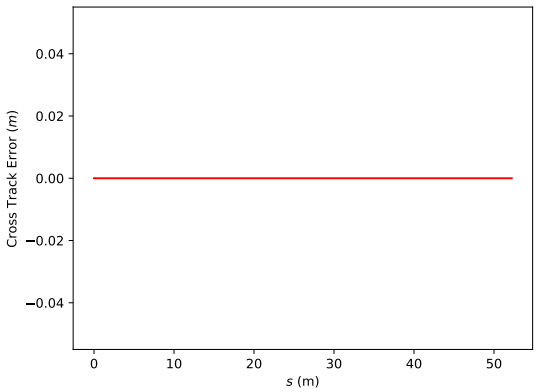
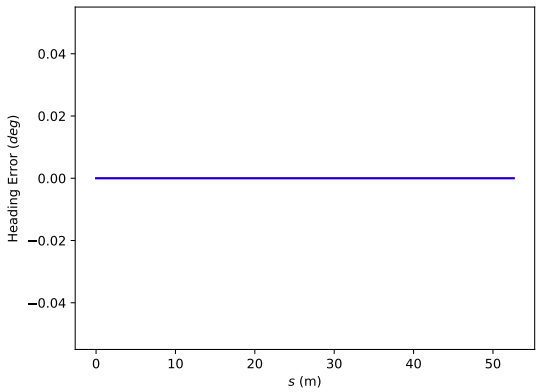


Figure 6.5: The simulation path and speed of the car for a straight

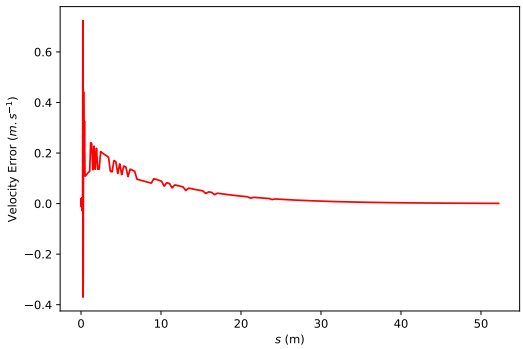
The results shown in figure 6.6 show that for straight track the car executes the motion plan without any cross-track error or heading error. The steering angle follows the motion plan, which is a constant steering angle of 0°, and does not change throughout the track. Figure 6.6c shows the speed error of the simulation, showing that when the car initially accelerates, the error increases drastically at the start of the track and the difference in the planned speed and simulation speed varies between 0.68ms^{-1} and -0.23ms^{-1}. However, this error soon reduces, as the car's speed increases, as shown in 6.6d. The speed error decreases as the speed tends to a steady state.



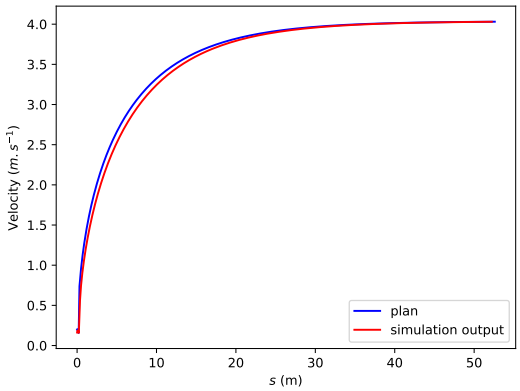
(a) The cross-track error of the simulation as a function of s



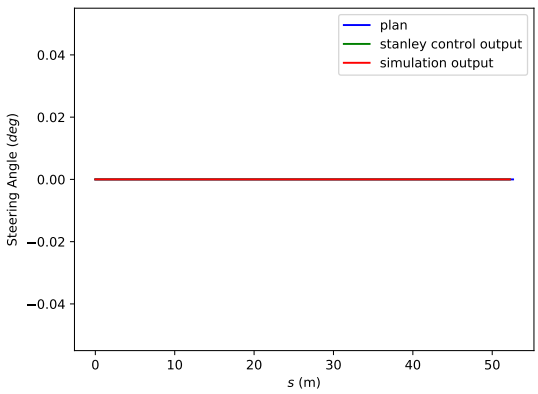
(b) The heading error of the simulation as a function of s



(c) The speed error of the simulation as a function of s



(d) The planned speed and simulation speed as a function of s



(e) The planned steering angle, Stanley control steering angle and the simulation steering angle as a function of s

Figure 6.6: Simulation results for the straight line track

Corner

The trajectory execution was then tested on a corner, with the states of the car initialised to 0, except for the longitudinal velocity which was initialised to a steady state value of 4.03ms^{-1}. The car was started at the beginning of the track and set to execute the motion plan generated by the global planner. The simulation results of the car executing the motion plan for the corner are shown in figures 6.7 and 6.8. Figure 6.7 shows the executed trajectory of the car, as well as the speed of the vehicle along the path that it travelled through the track.

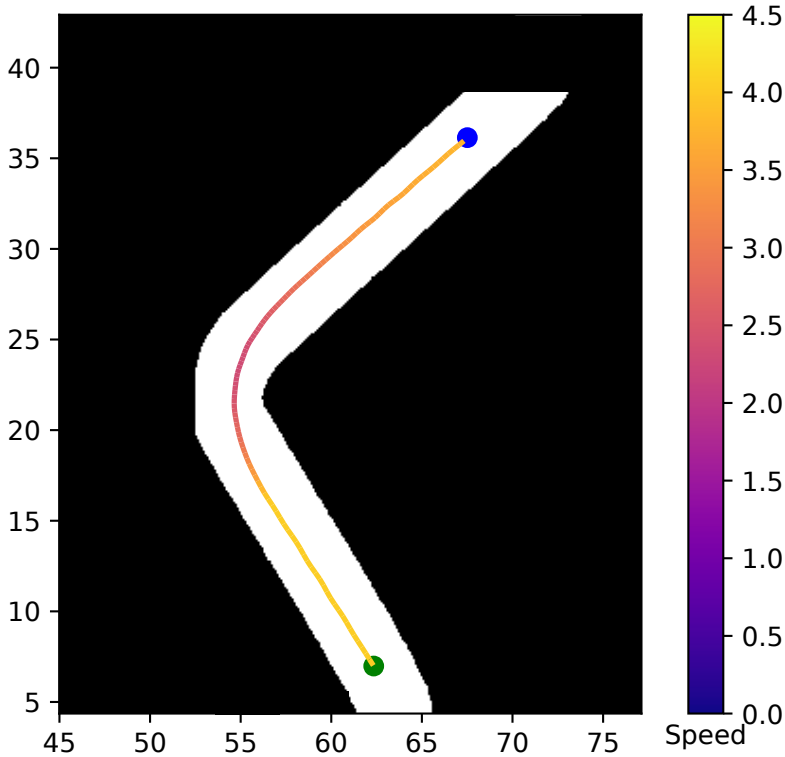
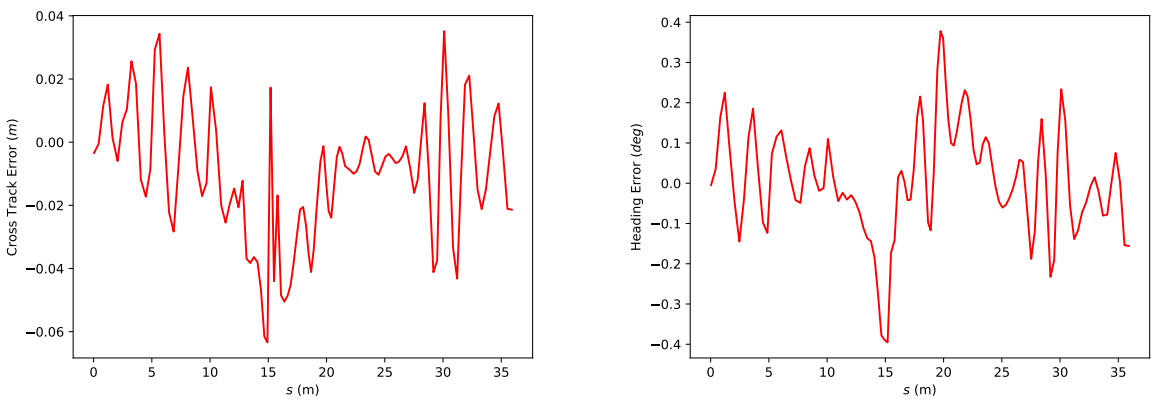


Figure 6.7: The simulation path and speed of the car for a corner

Figure 6.8a shows the cross-track error of the car, which is initially very small and varies from 0.035m to −0.025m. The error does get larger when the car begins to turn the corner, reaching a maximum error of −0.062m. Once the car has rounded the corner and is travelling in a straight line, the cross-track error again begins to vary between 0.036m to −0.045m. This shows that when the car is travelling in a straight line, and there is a disturbance such as a corner that causes a cross-track error, the car is able to continue executing the planned trajectory and remain within an acceptable error margin compared to the planned position. The heading error is shown in figure 6.8b and is consistently very small, within an error margin of ±0.4°. The heading error reaches a peak at the

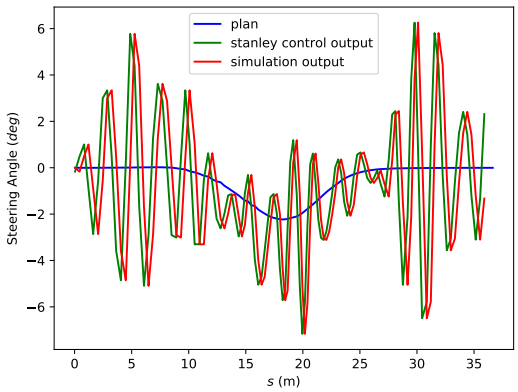
same time as the cross-track error, specifically when the car begins turning into the corner. The steering angle, shown in figure 6.8c, has a small delay between the steering angle commanded by the Stanley controller and the actual steering angle executed by the vehicle. This delay is a spatial delay, which corresponds to the sampling time of the simulation and is the distance that the car travels along the centre line in that sampling instance. This means that at higher speeds there is a bigger delay and at lower speeds, there is a smaller delay. This delay likely contributes to the cross-track and heading error, although the errors are within an acceptable margin. The steering angle of the simulation is significantly larger than the planned steering angle and the simulation steering angle has many abrupt changes compared to the smoother changes of the plan. Although this may seem as though the steering angle is underdamped, given that the full steering angle range is approximately $-30°$to $30°$, the steering angle changes are well within the capabilities of the car and amount to minor adjustments in order to minimise the cross-track and heading error. The changes in steering angle do not lead to any significant swerving in the vehicle. The speed and the speed error of the car are shown in figure 6.8e and figure 6.8d. This shows that although the simulation starts at a slightly slower speed than planned the simulation speed increases exponentially to the correct speed and the steady state error is 0ms^{-1}. When the speed is disturbed and the car slows down the absolute error increases, however, the error does tend to 0 as the car accelerates to a steady state speed. The same delay that is present in the steering angle results is also present in the speed, which largely contributes to the speed error that is present in the results.



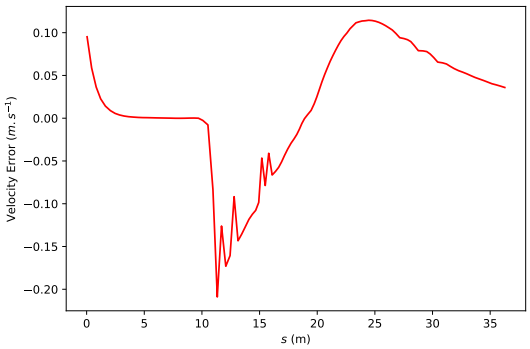
(a) The cross-track error of the simulation as a function of s

(b) The heading error of the simulation as a function of s

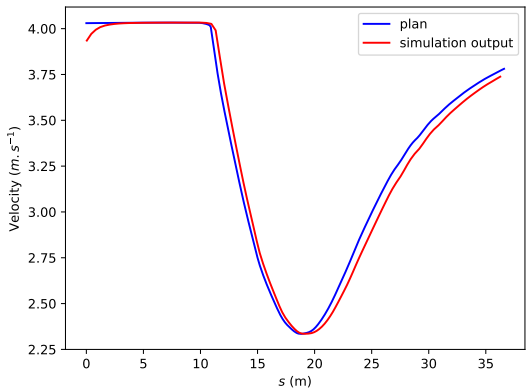
Figure 6.8: Simulation results for the corner



(c) The planned steering angle, Stanley control steering angle and the simulation steering angle as a function of s



(d) The speed error of the simulation as a function of s



(e) The planned speed and simulation speed as a function of s

Figure 6.8: Simulation results for the corner

Hairpin

The trajectory execution was next tested on a hairpin, with the states of the car initialised to 0, except for the longitudinal velocity which was initialised to a steady state value of 4.03ms^{-1}. The car was started at the beginning of the track and set to execute the motion plan generated by the global planner. The simulation results of the car executing the motion plan for the hairpin are shown in figures 6.9 and 6.10. Figure 6.9 shows the executed trajectory of the car, as well as the speed of the vehicle along the path that it travelled through the track.

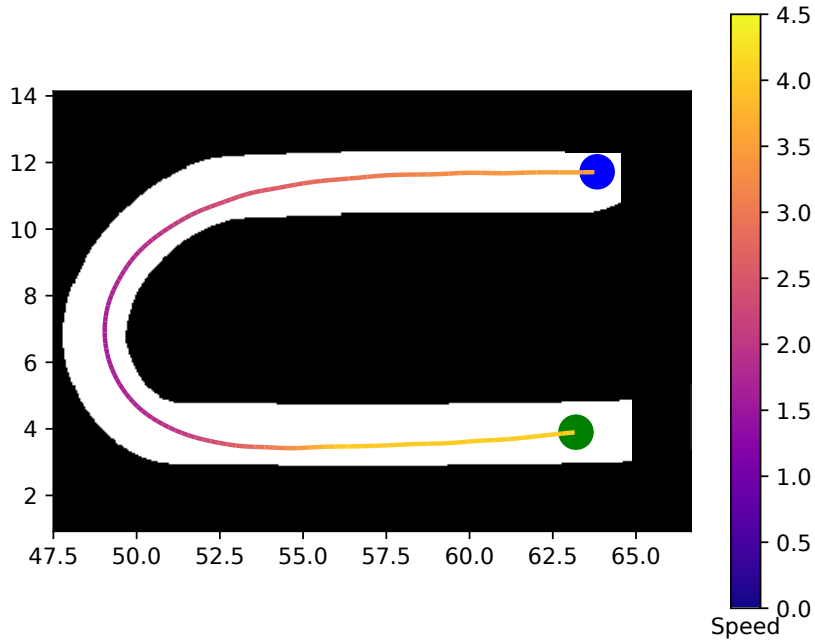
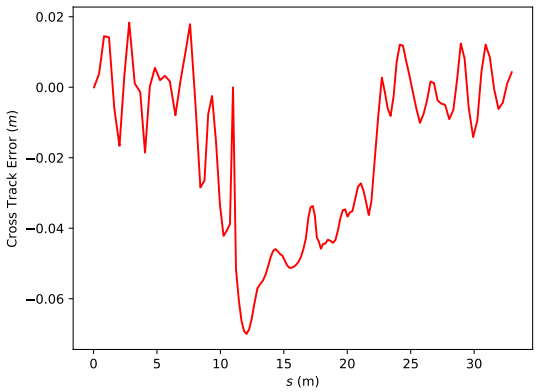
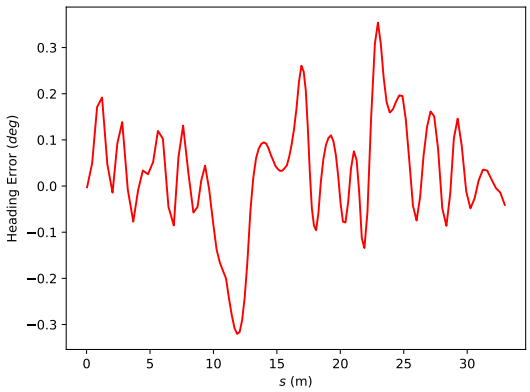


Figure 6.9: The simulation path and speed of the car for a hairpin

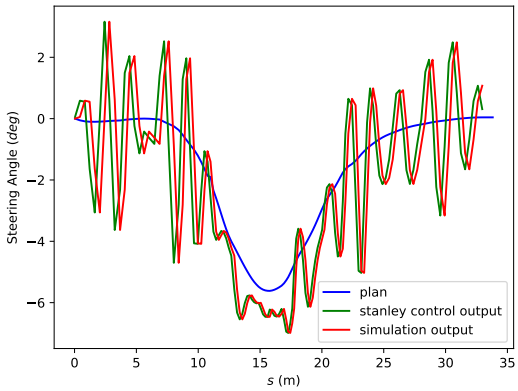
The cross-track error of the simulation of the car executing the motion plan for the hairpin track is shown in 6.10a. This shows that the cross-track error is initially between 0.02m and −0.02m, however, this error becomes greater as the car reaches the apex of the corner, reaching a value of −0.065m. The cross-track error then reduces to less than 0.02m as the car makes its way around the corner and starts driving straight again. The heading error of the car is shown in figure 6.10b, which shows that the heading error is consistently within the range of −0.4°to 0.4°which is well within an acceptable error margin. Similarly to the results of the corner simulation, the steering angle of the simulation for the hairpin, shown in figure 6.10c, also has the same delay between the steering angle input and the simulation. The steering angle also contains many abrupt changes with the purpose of minimising the cross-track error and the heading error of the car and is in actuality minor adjustments to the steering angle of the car. The speed and the speed error of the car are shown in 6.10e and 6.10d. These results are similar to the simulation results for the corner track, with a steady-state error of 0ms^{-1}, and when the speed is disturbed by the car slowing down, the absolute speed error increases to a maximum of 0.24ms^{-1}.



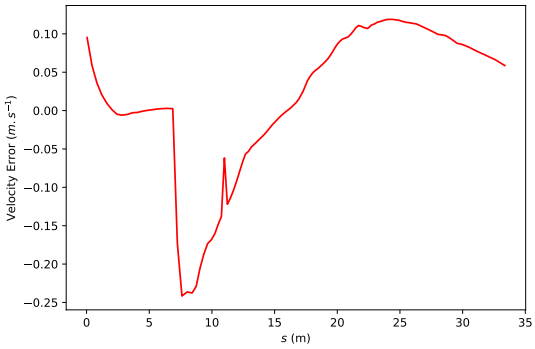
(a) The cross-track error of the simulation as a function of s



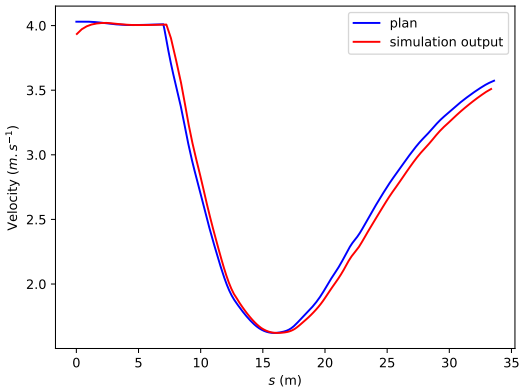
(b) The heading error of the simulation as a function of s



(c) The planned steering angle, Stanley control steering angle and the simulation steering angle as a function of s



(d) The speed error of the simulation as a function of s



(e) The planned speed and simulation speed as a function of s

Figure 6.10: Simulation results for a hairpin

Chicane

Finally, the trajectory execution was tested on a chicane, with the states of the car initialised to 0, except for the longitudinal velocity which was initialised to a steady state value of 4.03ms^{-1}. The car was started at the beginning of the track and set to execute the motion plan generated by the global planner. The simulation results of the car executing the motion plan for the chicane are shown in figures 6.11 and 6.12. Figure 6.11 shows the executed trajectory of the car, as well as the speed of the vehicle along the path that it travelled through the track.

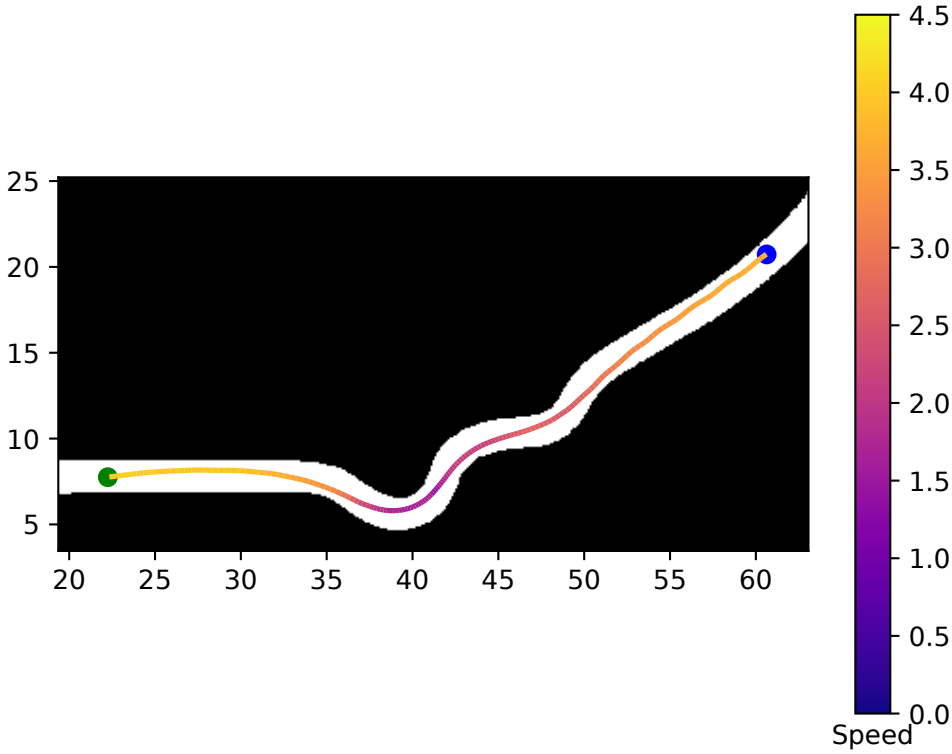
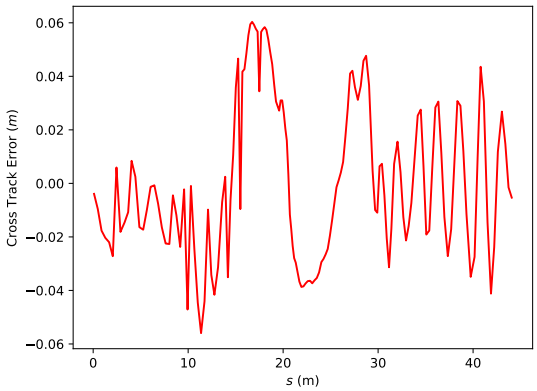


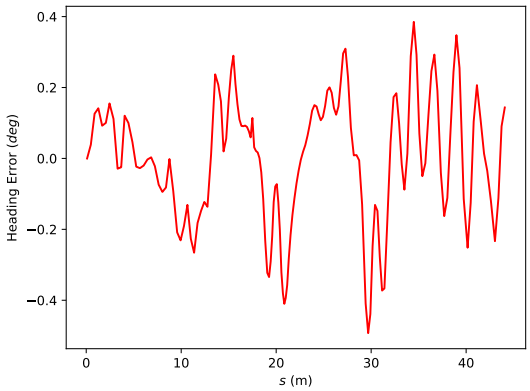
Figure 6.11: The simulation path and speed of the car for a chicane

The simulation results of the car executing the motion plan for the chicane in figures 6.12a and 6.12b show that the cross-track error and the heading error are respectively very small, with the absolute cross-track error reaching a maximum of 0.06m and the absolute heading error reaching a maximum of 0.49°. The maximum cross-track error occurs at the apex of the first turn of the chicane when the steering angle, shown in 6.12c, is at a maximum. Other rises in the cross-track error occur at the same points where the steering angle forms peaks, showing that at the apex of each turn, the car either oversteers or understeers very slightly. However, the cross-track error is very small and within an acceptable error margin. The speed of the simulation that is shown in figure 6.12e shows

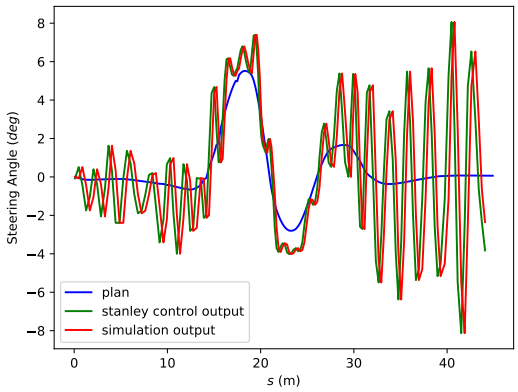
that when the car is accelerating and decelerating in quick succession, the delay between the speed reference and the simulation speed has a greater impact and produces a larger speed error. This is evident in figure 6.12d where the peaks of speed error occur when the car accelerates or decelerates.



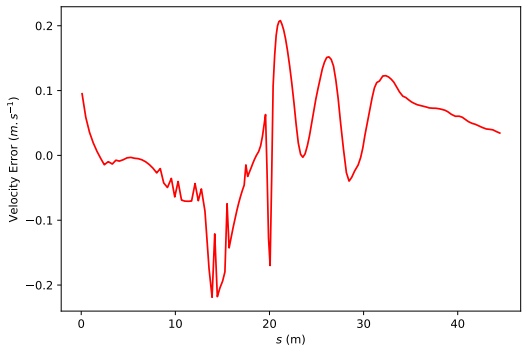
(a) The cross-track error of the simulation as a function of s



(b) The heading error of the simulation as a function of s

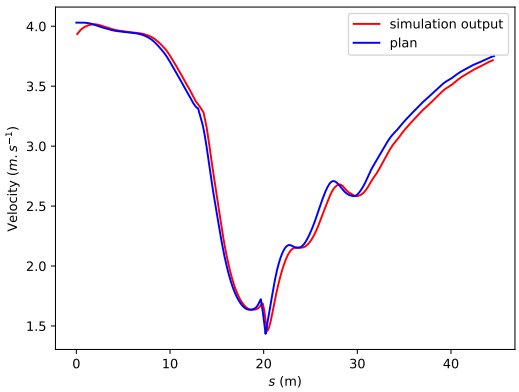


(c) The planned steering angle, Stanley control steering angle and the simulation steering angle as a function of s



(d) The speed error of the simulation as a function of s

Figure 6.12: Simulation results for a chicane



(e) The planned speed and simulation speed as a function of s

Figure 6.12: Simulation results for a chicane

Analysis of results

The results of the car executing the motion plan for each of the individual track elements show that the Stanley controller can accurately control the car to minimise the lateral error and the heading error of the car while maintaining the stability of the car. The maximum lateral error of the car through the tests was 0.07m while the maximum heading error was less than 0.06°. Table 6.1 shows the planned lap times for each track element as well as the lap times of the simulation.

	Planned Lap Time	Executed Lap Time
Straight	16.82	17.31
Corner	11.30	11.21
Hairpin	13.14	12.61
Chicane	16.10	15.51

Table 6.1: A table showing the planned lap time and simulation lap time for each track element

Comparing these results shows that the simulation lap times are very similar to the planned lap times, with a maximum error of less than 4%. It is also interesting to note that apart from the straight-line track, the simulation results are consistently faster than the planned results. This suggests that the decrease in lap time occurs in the cornering actions of the car. Although the difference between planned and simulated lap times is small, the consistency of this result suggests that the global planner is pessimistic about the car's handling capabilities. This, however, is not a negative attribute as the car is consistently slower in braking and reducing speed than planned and does not reach the minimum planned speed before beginning to accelerate, meaning that the car experiences a slightly higher amount of lateral force than planned.

6.4.3. Circuit de Barcelona-Catalunya

The racing system was tested on the complete racing track that includes all of the track elements that were represented in the previous benchmark tracks. Testing the execution of a motion plan on a full circuit shows how well the car can handle multiple track elements and accelerations and decelerations. The states of the car were initialised to 0, except for the longitudinal velocity which was initialised with a value of 1.0ms^{-1}. The car was started at the beginning of the track and set to execute the motion plan generated by the global planner. The results of the simulation of the car racing on the Circuit de Barcelona-Catalunya are shown in figure 6.13 and figure 6.14

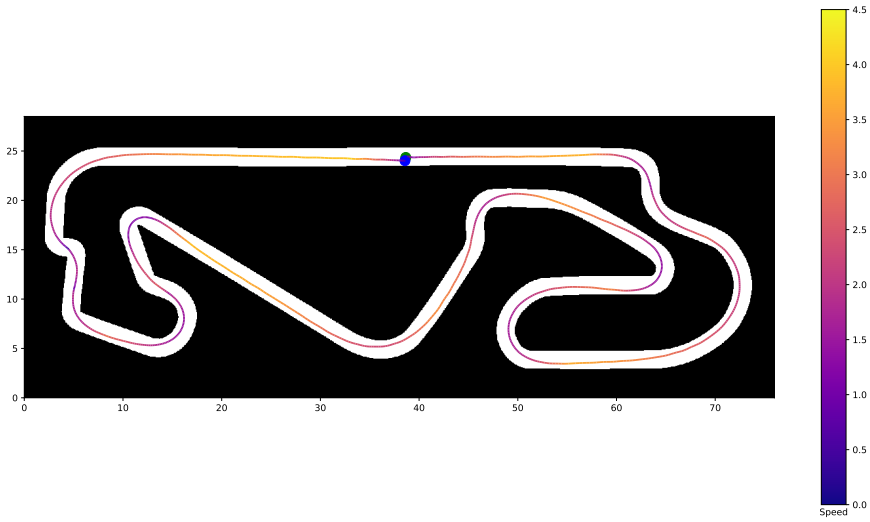
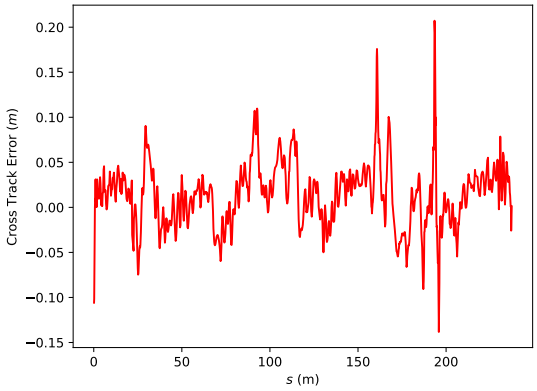


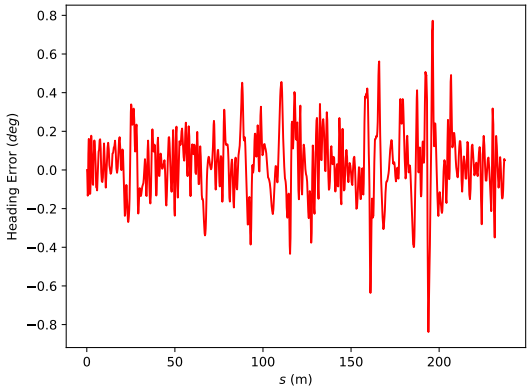
Figure 6.13: The simulation path and speed of the car for the Circuit de Barcelona-Catalunya

The cross-track error results in figure 6.14a show a larger cross-track error than any of the benchmark tests. The cross-track error peaks occur at the apex of the turns in the track when the steering angle is at a peak as well. The car starts 0.1m away from the planned initial position, which causes a large initial steering angle in order to correct the position of the car, however following the initial large spike, the steering angle follows the trend of the planned steering angle with many minor adjustments in order to minimise the cross-track error and the heading error. Examining the cross-track error as well as the simulation path in figure 6.13 shows that the car is understeering slightly in some of the corners. This leads to the car moving off the optimal trajectory and having to correct its position. The heading error is consistently small and within the range of $-0.8°$to $0.8°$. The speed error, in figure 6.14d, shows that the error tends to 0ms^{-1}. However, as expected, when the car accelerates or decelerates the magnitude of the error increases. However, comparing the speed error to a side-by-side comparison of the planned speed and simulation speed

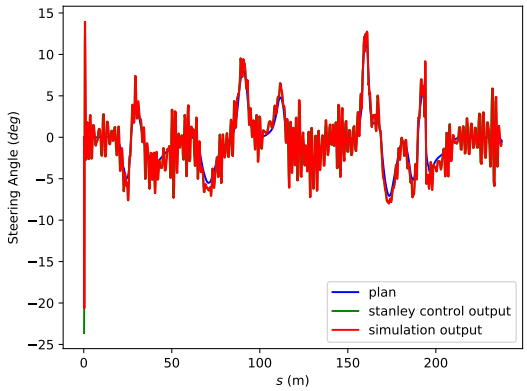
in figure 6.14e shows that when the car accelerates and decelerates in quick succession, the magnitude of the speed error increases drastically to a maximum error of -0.85ms^{-1}. This peak occurs at the same point that the maximum cross-track error and heading error occur, suggesting that velocity error negatively affects the cross-track and heading correction given that the Stanley controller is loosely coupled to the speed of the vehicle and that the speed control of the vehicle is not coupled to the other states of the vehicle.



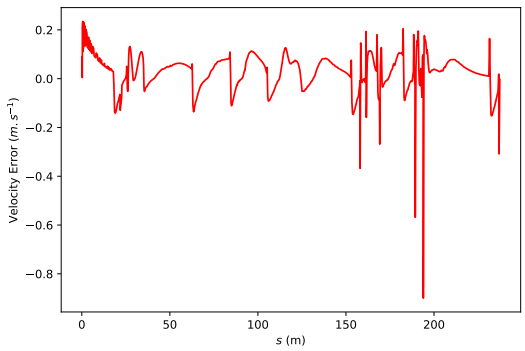
(a) The cross-track error of the simulation as a function of s



(b) The heading error of the simulation as a function of s

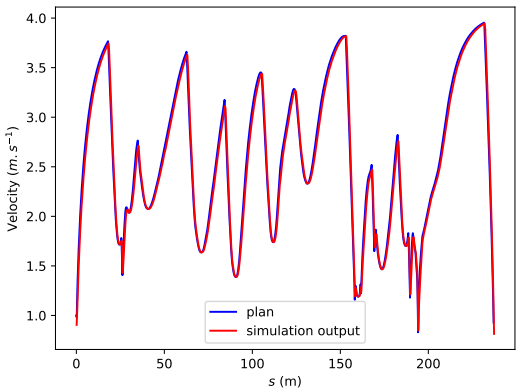


(c) The planned steering angle, Stanley control steering angle and the simulation steering angle as a function of s



(d) The speed error of the simulation as a function of s

Figure 6.14: Simulation results for the Circuit de Barcelona-Catalunya



(e) The planned speed and simulation speed as a function of s

Figure 6.14: Simulation results for the Circuit de Barcelona-Catalunya

It is interesting to note that, similar to the tests performed on the individual track elements, the simulation lap time result is slightly faster than planned. The planned lap time of the generated motion plan is 99.11s while the simulation lap time is 102.41s. Although not a large difference, it is consistent with the lap time results of the previous tests.

6.5. Summary

This chapter started by presenting a path-tracking Stanley controller which generates a steering angle command which is calculated based on the heading error of the car as well as a cross-track error correction factor. Then a particle filter was presented which is used for localisation via LiDAR scan matching. Finally, the execution of the planned trajectory was tested in simulation and analysed.

Chapter 7

Collision Avoidance

This chapter presents the collision avoidance system developed for this thesis. Firstly, an obstacle detection algorithm is presented which uses a LiDAR scan in order to localise unexpected obstacles within the track. Then, a collision prediction process is shown, which leads to the development of a local motion planner. The local planner, formulated as an optimal control problem, is presented and tested, and the results are analysed.

7.1. Obstacle Detection

7.1.1. LiDAR Point Cloud

Obstacles are detected using the laser scan from the LiDAR sensor. The laser scan generates a 2D point cloud with an origin at the LiDAR sensor using the formula shown below:

$$\begin{aligned} x_i &= d_i \cos(\alpha_i + \theta_t) + x_t \\ y_i &= d_i \sin(\alpha_i + \theta_t) + y_t \end{aligned} \tag{7.1}$$

where x_i and y_i are the points of the point cloud, d_i is the range given by the lidar scanner at the angle αi, θ_t is the orientation of the car at the time that the scan was recorded, and x_t and y_t are the cartesian coordinates of the car. αi is the angle of each beam projected by the LiDAR scanner and is dependent on the field of view of the LiDAR scanner, as well as the number of beams that the LiDAR scanner uses.

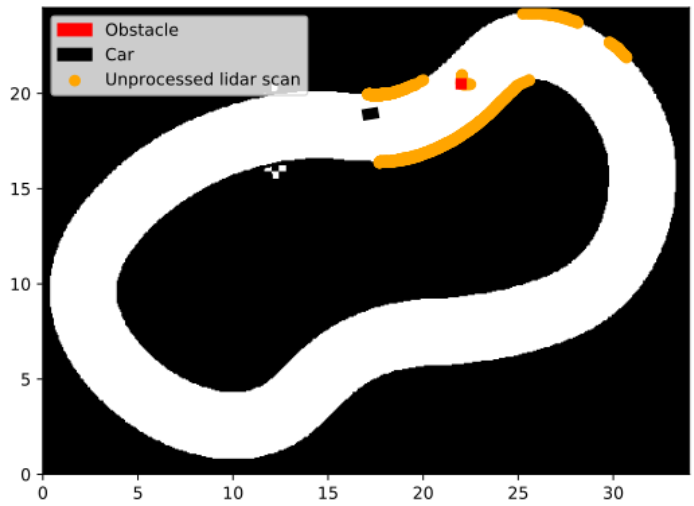


Figure 7.1: An example of the output LiDAR scan.

The generated cartesian point cloud is then transformed into the coordinate system of the track using the transformation in section 4.3.3 which transforms cartesian coordinates, x and y, to in-track and cross-track position, s and n. This allows for easy distinction between the track boundary and any obstacles within the track, by comparing the cross-track position n of each point in the point cloud to the track boundary at the in-track position s of that point. If the cross-track position of a point indicates that it is within the track, then that point may belong to an obstacle that is within the track.

7.1.2. Obstacle Identification

Any points that do fall within the track are then clustered using DBSCAN (Density-Based Spatial Clustering of Applications with Noise) [69] in order to separate and identify all the individual objects that are within the track.

DBSCAN

DBSCAN is a clustering algorithm that is well-suited to identifying clusters of data points in spatial data. It was first introduced by Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu in 1996 [69]. It relies on the density of data points to identify clusters and noise. It classifies points as core points, border points, or noise based on the number of neighbours within a specified radius, ϵ, and a minimum neighbour threshold, B. An illustration of the process of the algorithm is shown in figure 7.2.

The algorithm works by choosing an arbitrary starting data point and calculating how many neighbouring points fall within the radius ϵ. If the number of neighbours within

the radius is greater or equal to the minimum neighbour threshold, the data point is then classified as a core point. Core points initiate a cluster which is then expanded by recursively visiting its neighbours and marking them as part of the same cluster. This process continues until no more core points can be reached within that cluster. Other points within the cluster are classified as border points. Border points are points that are within the radius of a core point but do not have enough neighbours to be considered core points themselves and are assigned to the cluster of their corresponding core point. This is repeated until all the points in the data set have been visited and their neighbours calculated. The points which do not belong to a cluster at the end of the algorithm are classified as noise points. Noise points are points that are neither core points nor border points and are not assigned to any cluster.

The advantage of DBSCAN is that it can effectively handle datasets with noise and outliers, which are common in real-world applications. It is also less sensitive to the initial choice of parameters compared to some other clustering algorithms. Additionally, DBSCAN is capable of discovering clusters of arbitrary shapes and sizes. This is useful for clustering the resultant point cloud from the lidar scan as there is no typical or assumed shape of an object in the race track.

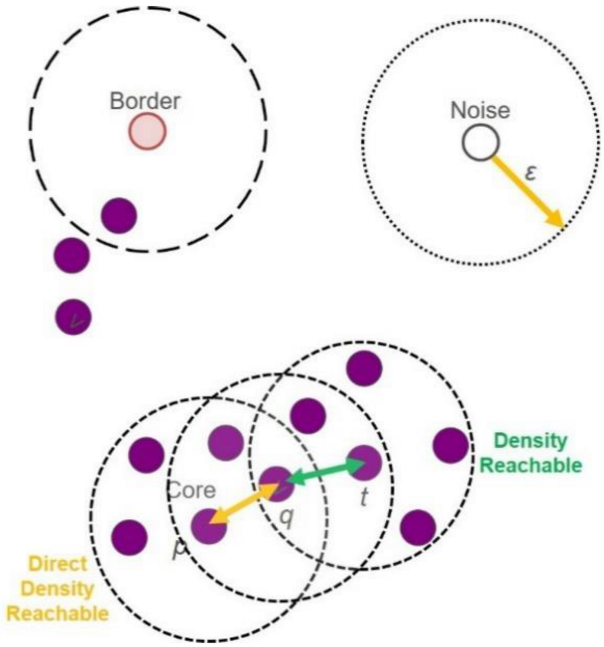


Figure 7.2: An illustration of the DBSCAN algorithm [2].

7.1.3. Obstacle Parameterisation

Any obstacles that are identified are each made up of clustered points that lie within the track instead of on the track boundary, as shown in figure 7.3. A convex hull is then fitted to the set of points that make up an identified obstacle in order to define a closed area in

the track that is occupied. A collision prediction step can then be performed which will determine whether the car will enter the space that the obstacle occupies. If the car were to enter this space, a collision would occur, and thus it is necessary for the car to take evasive action.

The presence of an object in the track will add additional uncertainty about the track ahead, as the obstacle will block the view of the LiDAR scan. Thus there will be a section of the track where there is uncertainty whether it is occupied or not. This uncertain section of the track is represented by the shadow of the identified object, which is defined by the widest LiDAR points that are assigned to the obstacle and the immediate points on the track boundary on either side of the obstacle, as shown in figure 7.3. The shadow of an object can impact the desirability of the evasive trajectory as it has an effect on the potential safety of the trajectory. An obstacle's shadow can potentially cause a dead end, as is the case in figure 7.3, which means that if the evasive trajectory were to cause the car to enter the unknown space and that space is occupied, the car will get stuck. However, there may be known unoccupied space which is guaranteed to be safe that the car can use for the evasive trajectory. Due to these potential circumstances, the shadow of each identified object is also calculated in addition to the object itself.

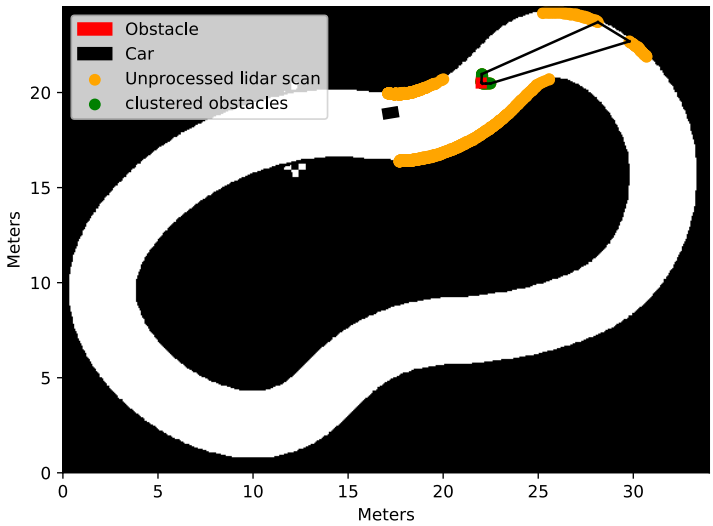


Figure 7.3: An example of an obstacle detected and clustered with the obstacle shadow depicted.

7.2. Collision Prediction

If an obstacle is identified then a collision prediction step is performed in order to determine whether the car needs to take evasive action to avoid colliding with the detected obstacle. This process is done by propagating the current position of the car forward in time, according to the intended motion plan that the car is currently executing. The car is

propagated forward in discrete time steps up to a maximum lookahead time of T, with the number of integration steps dependent on the size of the timesteps. The lookahead time of the vehicle determines how far in advance the car will plan to avoid an obstacle and therefore it is advantageous to have a longer lookahead time. However, a higher lookahead time will lead to more integration steps and therefore a longer computation time which may affect the performance of the system. Therefore when considering the lookahead time of the car it is necessary to take into account the top speed of the car as well as the complexity of the racing track in order to find the optimal time.

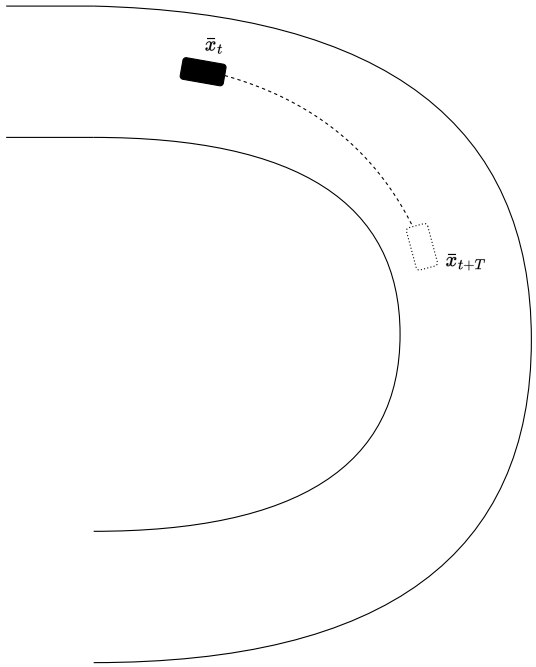


Figure 7.4: Illustration of trajectory propagation of the car.

The trajectory propagation makes use of the kinematic bicycle model, derived in section 4.1.1, as the model does not require complex calculations and therefore the propagation process is computationally less expensive and strikes a good balance between accuracy and computation speed.

The propagation of the car's trajectory uses the intended motion plan that the car is currently executing, either the pre-planned global plan or a local collision avoidance plan. This means that the system may detect a collision with a newly identified object while the car is busy expecting a collision avoidance trajectory for a previously identified object. In such a case, a new local plan is required in order to successfully avoid the previously identified obstacles as well as any new obstacles that are identified.

If the car is busy executing a local collision avoidance plan and a trajectory propagation process is started, the propagation of the car may extend beyond the horizon limit of the local plan. In such a case, it is assumed that the car will then proceed to continue executing

the global plan. The trajectory propagation process will then continue to propagate the car's position using the global plan until the maximum lookahead time is reached.

It is important to note that the collision prediction process is triggered at every sampling instant if an obstacle is detected on the track, regardless of whether or not the local planner has already generated a collision-free plan for the horizon track. This allows the car to react to new information, whether that is the presence of new obstacles, a change in an obstacle's location, or the car failing to successfully execute a local plan.

7.3. Collision Avoidance Planning

Once an object has been identified and the collision prediction process determines that a collision would occur if evasive action is not taken, a local planner calculates a short-term motion plan in order to safely avoid the obstacle. The most common approach to local path planning is the use of model predictive control (MPC) to generate a motion plan for the car to execute [12]. However, MPC is very computationally expensive and requires one to formulate and solve an optimisation problem in real-time at every timestep [3]. However, the local planner is used in this thesis only when the collision prediction process triggers it and the plan is then executed by the path-tracking controller.

The local planner uses a similar optimisation process as the global planner in section 5.3, but introduces new constraints to the optimisation process as well as new elements to the cost function of the path planning process. This is to ensure that the car does not move into the occupied space, as well as to encourage minimum deviation of the car from the planned path. Additionally, due to the need for this to occur in real time, the simpler kinematic bicycle model is used in the optimisation process as opposed to the more complex dynamic bicycle model. This reduces the computation power needed for the motion planning process and allows for a real-time solution. The local path planner is only used to generate a motion plan that will avoid any detected obstacles in the track, which means it is only necessary to plan from the car's current in-track position up to a horizon planning distance. The horizon distance is determined by the range of the LiDAR scanner as well as the maximum speed of the car. The horizon planning distance is used to formulate a horizon track that will be used in the local path planning process. The horizon track is a subsection of the complete racing track that starts at the car's current in-track position with a total centre-line arc length equal to the horizon planning distance.

7.3.1. System Model

The system is modelled using a kinematic bicycle model set in a curvilinear coordinate system described in 4.3.1. The vehicle model is referenced to the track centre line, which

is described in 4.2. The track is fully described by the curvature of the centre line $K(s)$ and the width of the track at each point $2W(s)$. The complete state information of the vehicle model is described by the in-track distance s, the cross-track distance n, and the local heading μ, referenced to the centre line of the track. The inputs to the vehicle model are the speed v and the steering angle δ.

$$\begin{aligned}
\dot{s} &= \frac{v_x \cos(\mu) - v_y \sin(\mu)}{1 - nK(s)} \\
\dot{n} &= v_x \sin(\mu) + v_y \cos(\mu) \\
\dot{\mu} &= \dot{\theta} - \dot{s}K(s) \\
\dot{\theta} &= \frac{v}{L} \tan(\delta) \times \cos(\beta) \\
\beta &= \arctan(\frac{l_r \tan(\delta)}{L}) \\
v_x &= v \cos(\beta) \\
v_y &= v \sin(\beta)
\end{aligned} \tag{7.2}$$

The state dynamics of the vehicle model are shown in 7.2 and is given by $g_c^t(\mathbf{x}, \mathbf{u})$, where $\mathbf{x}$ is the state vector and $\mathbf{u}$ is the input vector consisting of the speed v and the steering angle δ. Similarly to the global path planning process, the time-based dynamics of the system are transformed to spatial-based dynamics using the transformation shown in equation 5.2. This allows the state s to be dropped from the state space, leading to the reduced state space $\overline{\mathbf{x}}$. The state equations are then discretised with respect to the centre line with a discretisation distance of Δs in the same manner as in the global path planning process in section 5.3. This leads to discrete spatial-based system dynamics that are given by:

$$g_d^s(\overline{\mathbf{x}}_k, \mathbf{u}_k) = \overline{\mathbf{x}}_{k+1} = \overline{\mathbf{x}}_k + \Delta s g_c^s(\overline{\mathbf{x}}_k, \mathbf{u}_k) \tag{7.3}$$

7.3.2. Track Formulation and Separation

The direct solution to the collision avoidance problem of preventing the car from entering occupied space on the track would be to add an additional constraint to the optimisation formulation. This ideal constraint would limit the cross-track position of the car from entering the space occupied by an obstacle at every position along the centre line that the obstacle occupies. Figure 7.5 shows how this constraint would limit the allowable track space for the horizon track that the car would be able to enter if such a constraint were

implemented.

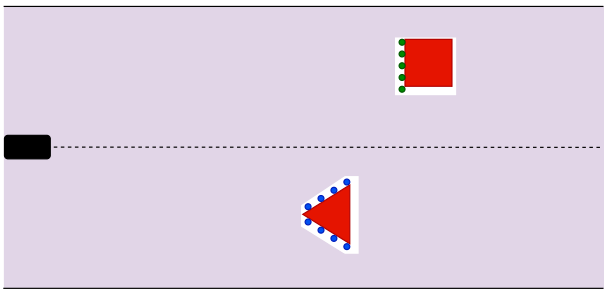


Figure 7.5: Illustration of trajectory propagation of the car.

However, implementing such a constraint would introduce a discontinuity in the otherwise continuous range of values that the state n could take on. Formulating and solving an optimisation problem that contains such a discontinuity is not feasible in real time with the computation power available on the physical car.

Thus in order to avoid introducing discontinuities to the optimisation problem, but also avoiding any additional limitation on the values on the cross-track position that are not due to any obstacles, the allowable range of n is split over multiple continuous ranges and a separate optimisation problem is formulated and solved for each scenario. Figure 7.6 shows an example of such a case where the horizon track is separated into multiple cases. When the horizon track is separated into different cases, the limits of the cross-track position for each individual case are given by $O_L(s)$ and $O_R(s)$ for the left limit and right limit respectively

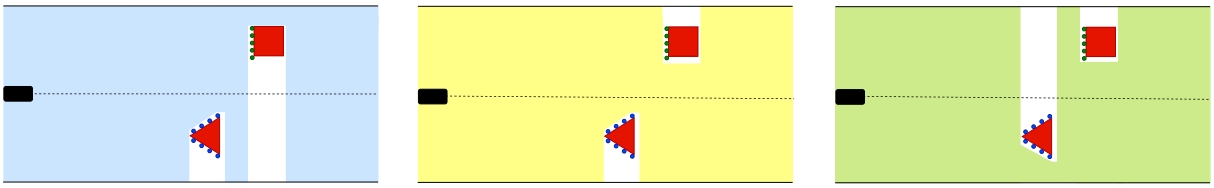


Figure 7.6: Illustration of trajectory propagation of the car.

7.3.3. Optimisation Setup and Constraints

The constraints for the optimisation problem are formulated in order to ensure that the car stays within the track limits as well as ensuring that the car does not enter any occupied space in the track.

Track Limits

The constraint that limits the range of the cross-track position is similar to the global planner's constraint which constrains the car to stay within the track limits. However,

instead of using the track limits of the entire track, a range of free space of the horizon track is used. The constraint is given in equation 7.4 and denoted by $\overline{\mathbf{x}} \in \mathcal{X}_{free}$. This constraint is more restrictive compared to the constraint that limits the car to stay within the boundaries of the track, and thus it is not necessary to have a separate constraint in order to limit the car to the track boundaries.

$$\begin{aligned} n + \frac{L_c}{2}\sin(|\mu|) + \frac{W_c}{2}\cos(\mu) \leq O_L(s) \\ n - \frac{L_c}{2}\sin(|\mu|) - \frac{W_c}{2}\cos(\mu) \geq O_R(s) \end{aligned} \tag{7.4}$$

Input Constraints

The steering angle is constrained in the same manner as the global planner to limit the steering angle to be within the minimum and maximum steering angles that are possible on the car and is given in equation 7.5

$$\delta_{Min} \leq \delta \leq \delta_{Max} \tag{7.5}$$

Due to the simplicity of the kinematic model and the emergency nature of the collision avoidance plan, the speed input to the system is kept constant for the motion plan. This will prevent rapid accelerations and decelerations while the collision avoidance plan is being executed and reduce the chance of the car exceeding the limits of its handling capabilities. The speed of the car is set constant at the speed that the car is travelling in the instant that the planning step occurs, v_t as shown in equation 7.6

$$v_k = v_0 = v_t \tag{7.6}$$

The constraints that are imposed on the inputs to the system model are denoted by $\mathbf{u} \in \mathcal{Y}$

7.3.4. Objective function design

The cost function aims to minimise the time that it takes for the car to progress through the horizon track in a similar manner to the global planner, shown in equation 7.7.

$$L = \sum_{k=0}^{N} \frac{\Delta s}{\dot{s}} \tag{7.7}$$

Additionally, similar to the global planner, the difference between the dynamic and the kinematic slip angle is penalised as shown in equation 7.8

$$B(\overline{\mathbf{x}}_k, \mathbf{u}_k) = (B(\overline{\mathbf{x}}_k)_{dyn} - B(\mathbf{u}_k)_{kin})^2 \tag{7.8}$$

The local planner has an additional term in the cost function that the global planner does not have and that is a term that penalises the deviation from the pre-planned global path. This penalisation term penalises the difference between the cross-track position that the local planner outputs and the cross-track position of the optimal cross-track position that the global planner outputs as shown in equation 7.9.

$$D(\overline{\mathbf{x}}_k) = C(n_{optimal} - n_k)^2 \tag{7.9}$$

This constraint encourages the results of the motion planner to deviate as minimally as possible from the optimal plan in order to avoid any obstacles in the track.

7.3.5. Problem Formulation and Optimisation

The full optimal motion planning problem is shown in equation 7.10 which includes the cost function and constraints that have been presented.

$$\begin{aligned} \min_{\overline{\mathbf{x}},\mathbf{u}} \quad & \sum_{k=0}^{N} \frac{\Delta s}{\dot{s}} + D(\mathbf{u}_k) + B(\overline{\mathbf{x}}_k, \mathbf{u}_k) \\ \text{s.t.} \quad & \overline{\mathbf{x}}_{k+1} = g_d^s(\overline{\mathbf{x}}, \mathbf{u}) \\ & \overline{\mathbf{x}} \in \mathcal{X}_{free} \\ & \mathbf{u} \in \mathcal{Y} \end{aligned} \tag{7.10}$$

7.4. System Implementation

This optimal control problem is formulated and solved in a similar manner to the global path planner using Pyomo [78, 79] and solved online using IPOPT [80]. The initial conditions of the vehicle are set and fixed in the optimisation problem based on the state of the vehicle when the collision avoidance planning is activated.

Each case of the horizon track is ordered by the smallest width within the horizon track. If the minimum width in a horizon track is smaller than the width of the car, that case

is discarded as there is no viable solution to the path planning problem for that specific horizon track. The optimal control problem is formulated and solved for each separate viable case of the horizon track. Each solution represents a motion plan that will avoid the detected obstacle(s). Motion plans that do not cause the car to enter the shadow of an object (uncertain space) are preferable solutions. Following that, the solution with the lowest resultant cost function is the solution that is used as the motion plan result for the collision avoidance plan.

The car will then use the motion plan results and execute the plan until it is either successfully executed and the car reverts to executing the global plan or a new local plan is generated.

It is possible that a solution is not found within an acceptable time limit, or that no viable solution exists. In such a case the failure to resolve the collision is noted and the car will continue to execute the global plan. If a collision is detected and it is not successfully resolved 5 times in a row, then the car will brake and come to a stop as it is assumed that there is no safe way for the car to continue racing.

7.4.1. Example Output

The local planner generates the optimal state trajectory and associated control, as a function of the in-track position s, which minimises the cost function while adhering to the imposed constraints. An example output from the local planner is shown in figure 7.7 and figure 7.8, which shows the generated local trajectory, and the vehicle states and inputs, respectively.

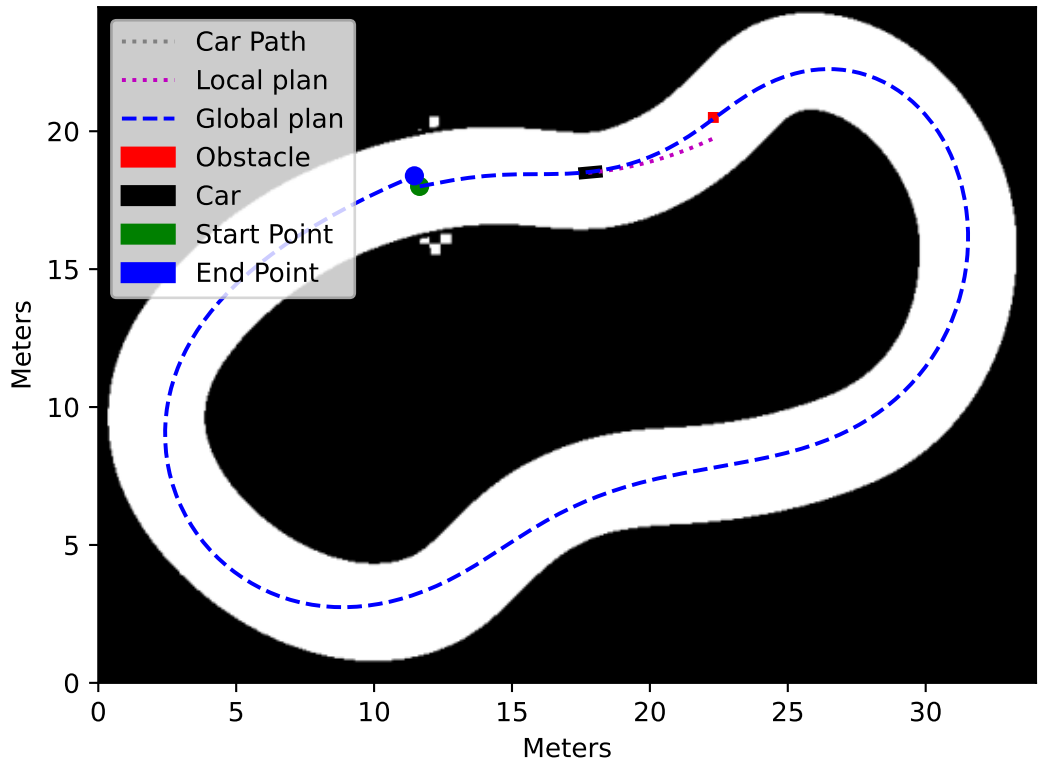
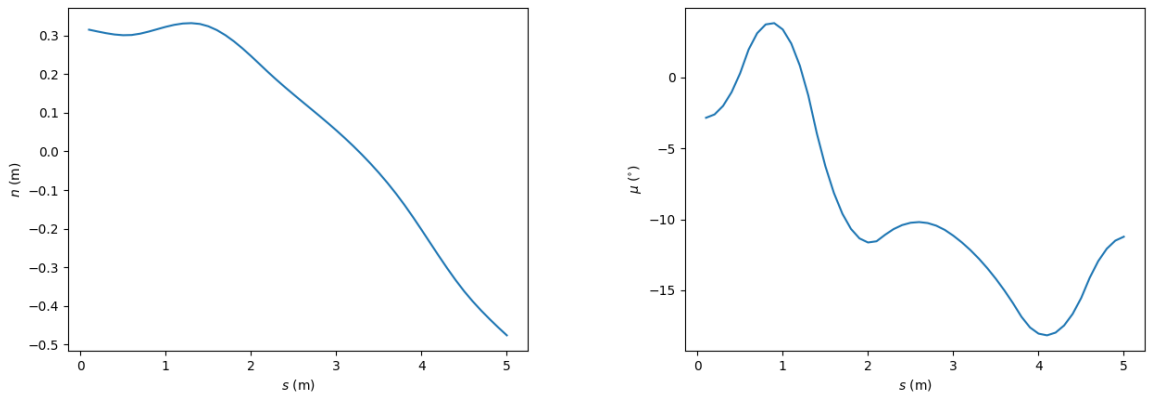
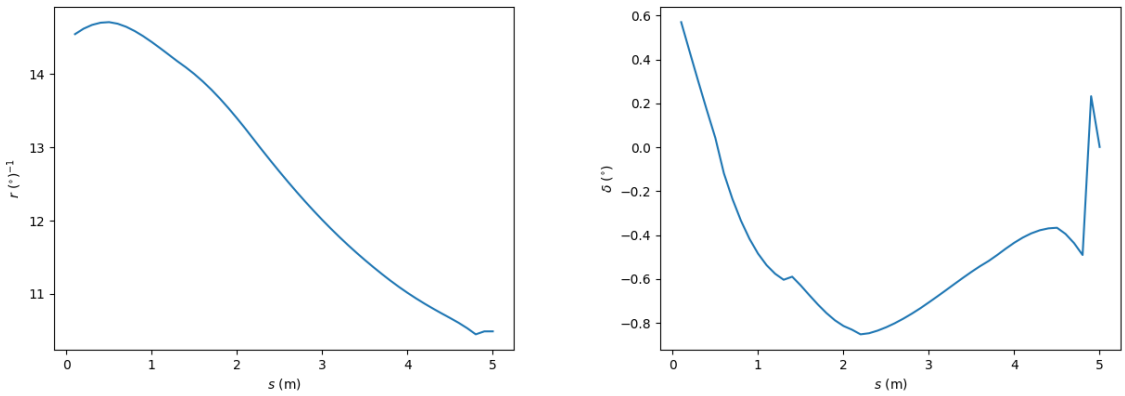


Figure 7.7: Typical output from the local planner for a given track.



(a) Cross-track position **(b)** Local heading

Figure 7.8: The planned state variables and inputs that are generated by the local planner, as a function of the in-track position.



(c) Yaw rate **(d)** Steering angle

Figure 7.8: The planned state variables and inputs that are generated by the local planner, as a function of the in-track position.

7.5. Simulation Results

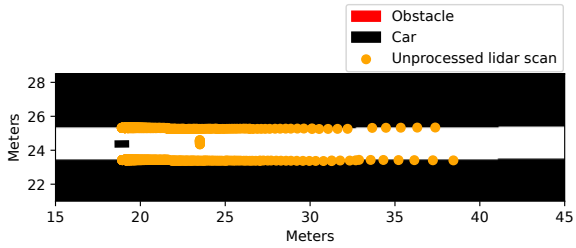
The obstacle avoidance system was tested using the F1Tenth simulator [70]. An obstacle was placed on each benchmark track, such that it overlapped with the global reference trajectory, and would cause a collision. The results of the lap were recorded with the position and orientation of the car obtained from the simulator at a frequency of 10Hz. The car was initialised on the starting point, aligned with the track with a steady-state longitudinal velocity of 4.03ms^{-1}.

7.5.1. Straight

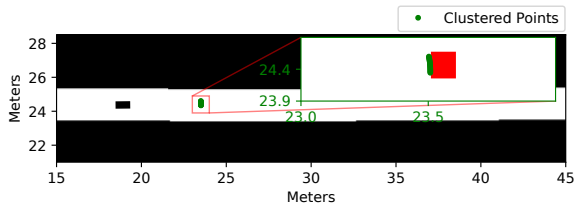
The obstacle avoidance system was tested on a straight, the results of which are shown in figures 7.9 and 7.10. Figure 7.9 shows the process of obstacle detection, collision prediction, and collision avoidance planning for the car, while figure 7.10 shows the car executing the local plan and rejoining the global track.

The raw point cloud from the LiDAR scan is shown in figure 7.9a, which is then processed such that the points that are within the track limits are identified and clustered. Those points are shown in figure 7.9b. As expected, because there is only one obstacle in the track, there is only one cluster of points identified within the track. When the cluster is identified, the collision prediction process is started and is repeated at every sampling instant until there are no obstacles detected within the track. Figure 7.9c shows when the collision prediction process activated the local plan, while figures 7.9d and 7.9e show the potential horizon tracks that represent possible space that the car can move into, to avoid the obstacle. These figures show that there are 2 possible gaps that the car can move into, in order to avoid the detected obstacle. Therefore, the local planner attempts to a path

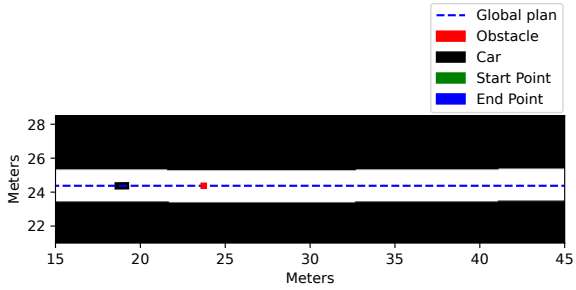
for each horizon track representation, and the most optimal result is chosen as the output of the local planner. This plan is shown in figure 7.9f, which is a short-term trajectory, that the car will attempt to execute in order to avoid the obstacle.



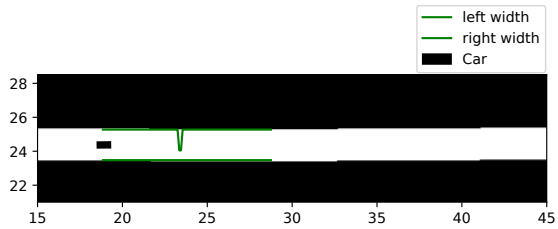
(a) Unprocessed LiDAR scan



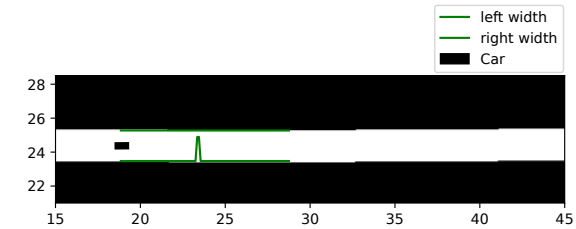
(b) Clustred points detected on the track



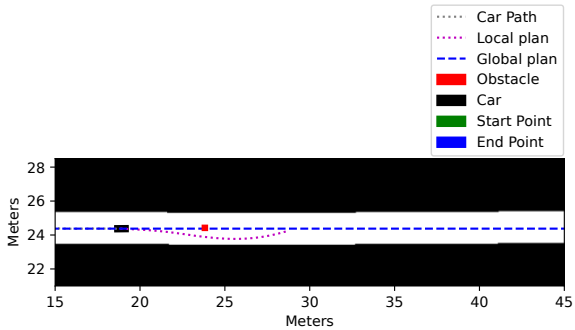
(c) Collision detected with identified Obstacle



(d) A potential horizon track, which represents unoccupied space that the car can freely enter.



(e) A potential horizon track, which represents unoccupied space that the car can freely enter.

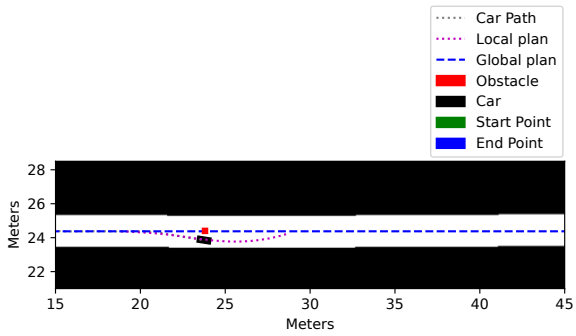


(f) Local Plan created to avoid the detected obstacle

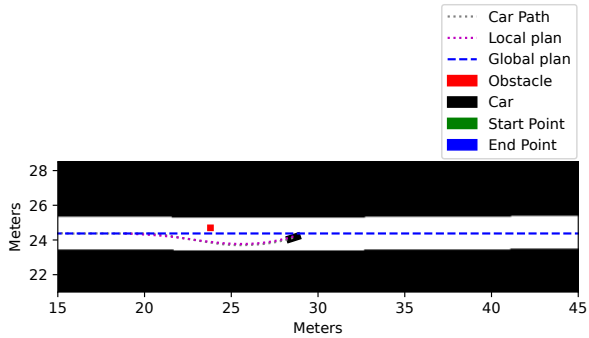
Figure 7.9: The obstacle avoidance detection and planning process for a straight

Once the car has received a local plan, it attempts to execute the local plan, as shown in figure 7.10a. Once the car successfully executes the local plan, shown in figures 7.10b and

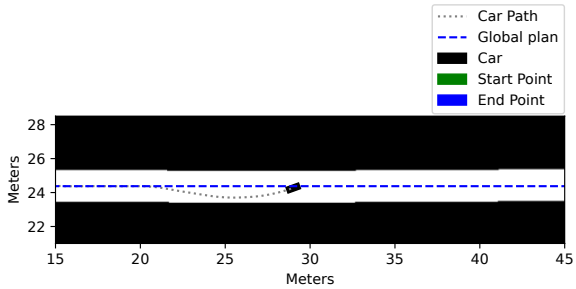
7.10c, the car will begin to execute the global plan once more. As figure 7.10c shows, the car may finish the local plan, but will often have a difference in heading or cross-track position compared to the global plan. In such a case the Stanley controller will correct the difference, and over time the car will rejoin the global plan as shown in figure 7.10d



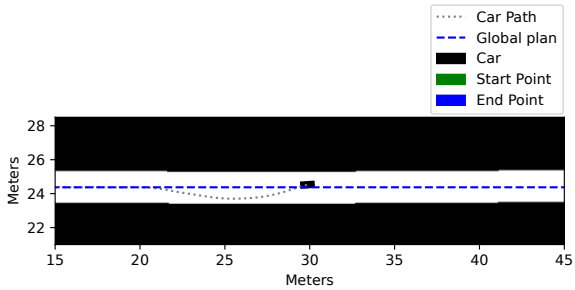
(a) Car executing the created local plan



(b) The end of the local plan



(c) The car begins executing the global plan again



(d) The control loop of the system brings the car back to the global reference

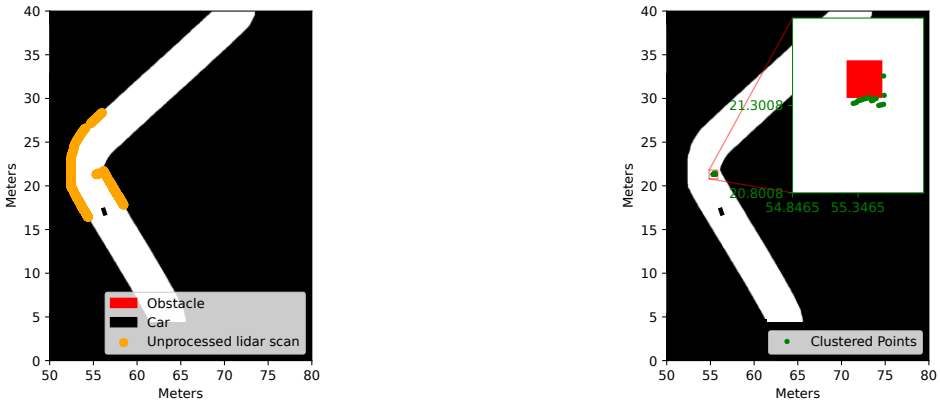
Figure 7.10: The obstacle avoidance execution for a straight

7.5.2. Corner

The obstacle avoidance system was then tested on a corner, the results of which are shown in figures 7.11 and 7.12. Figure 7.11 shows the process of obstacle detection, collision prediction, and collision avoidance planning for the car, while figure 7.12 shows the car executing the local plan and rejoining the global track.

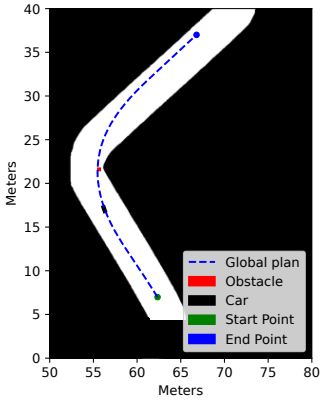
Similar to the collision avoidance process that was shown for the straight, the raw point cloud from the LiDAR scan, in figure 7.11a, is processed and points within the track boundaries are identified and clustered, which is shown in figure 7.11b. Interestingly, unlike the previous example on the straight, figure 7.11b shows the noise that is present within the lidar scan, as the identified points that represent the obstacle do not align exactly with the edges obstacle. However, the general shape of the obstacle is still clearly

visible, and the biggest difference between the location of an identified object point and the true location of the point is less than 0.05m. Thus, the obstacle detection system can still accurately localise the obstacle and determine the free space that the car can safely enter, which is shown in figures 7.11d and 7.11e. In this particular instance, there is only one horizon track representation that contains a viable collision-free trajectory, particularly the representation shown in figure 7.11e. This is because the smallest gap is the track representation in figure 7.11d is smaller than the width of the car, therefore that track representation is discarded. Therefore, the local planner only considers the track representation in figure 7.11e. The output of the local planner is shown in figure 7.11f, which as expected is contained to that track representation in figure 7.11e. This plan successfully avoids the obstacle, however by examining the plan, the effect that including the deviation from the global plan in the cost function is also clear. The local plan deviates a minimal amount from the global plan, with the largest cross-track difference being less than 0.25m. Additionally, when examining the generated local plan, it is evident that it is not the trajectory that would be generated if the local planner were to only minimise the time that it takes for the car to progress through the horizon track.

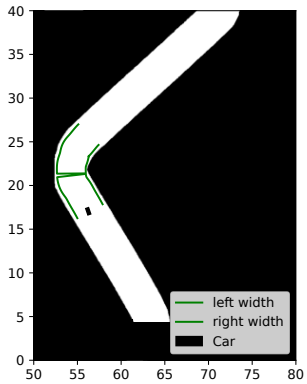


(a) Unprocessed LiDAR scan **(b)** Clustred points detected on the track

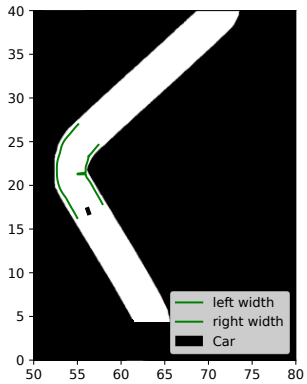
Figure 7.11: The obstacle avoidance detection and planning process for a corner



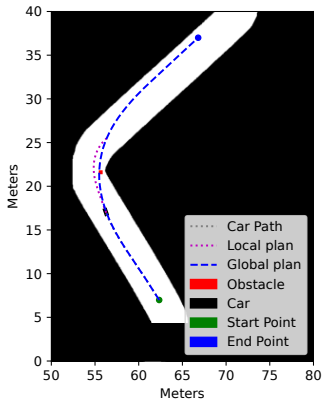
(c) Collision detected with identified Obstacle



(d) A potential horizon track, which represents unoccupied space that the car can freely enter.



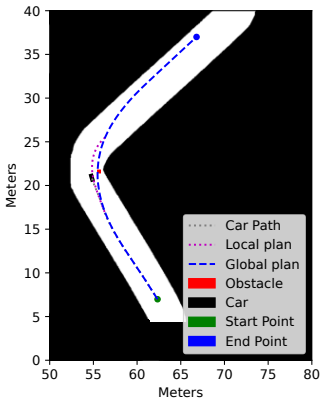
(e) A potential horizon track, which represents unoccupied space that the car can freely enter.



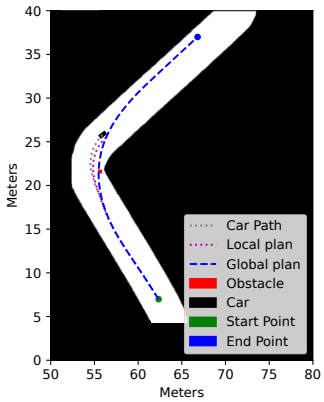
(f) Local Plan created to avoid the detected obstacle

Figure 7.11: The obstacle avoidance detection and planning process for a corner

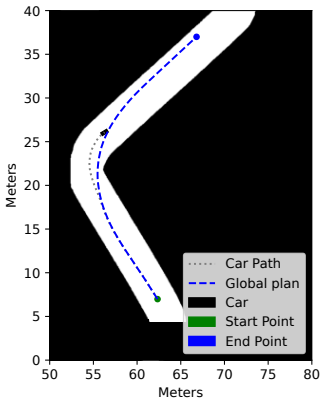
Figure 7.12a shows the car executing the generated local plan and figure 7.12b shows the car finishing the local plan. These figures show that the car did not execute the plan perfectly as there is a peak cross-track error of 0.1m, which is higher than the peak cross-track error that was obtained from the test performed in section 6.4.2. This is because the local planner uses the less accurate kinematic model as opposed to the more accurate, but computationally intensive, dynamic model. Therefore the trajectory generated by the local planner may be significantly more challenging for the car to execute, and in fact, may result in a motion plan that is not practically possible to implement. Figure 7.12c shows the car once it has successfully completed the local track, and it can be seen that the car has a different heading when compared to the global plan. This is corrected by the Stanley controller, as seen in figure 7.12d, which controls the car to rejoin the global plan with minimal cross-track and heading errors.



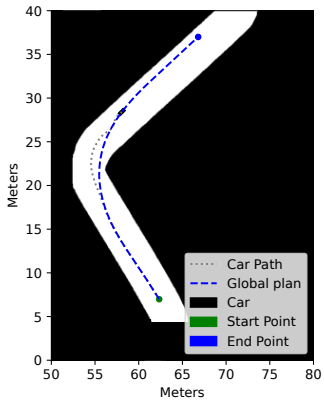
(a) Car executing the created local plan



(b) The end of the local plan



(c) The car begins executing the global plan again



(d) The control loop of the system brings the car back to the global reference

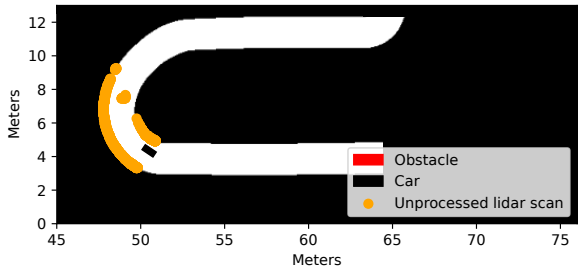
Figure 7.12: The obstacle avoidance execution for a corner

7.5.3. Hairpin

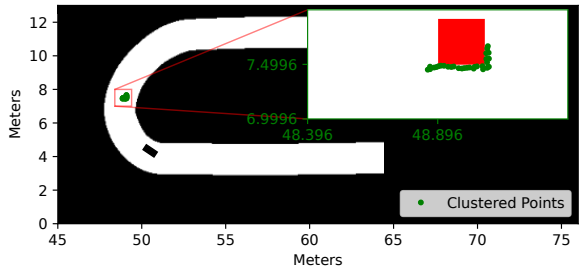
The obstacle avoidance system was then tested on a hairpin, the results of which are shown in figures 7.13 and 7.14. Figure 7.13 shows the process of obstacle detection, collision prediction, and collision avoidance planning for the car, while figure 7.14 shows the car executing the local plan and rejoining the global track.

The process of the collision avoidance system is demonstrated in figure 7.13, with figures 7.13a and 7.13b showing the process by which the obstacle in the track is detected. In this particular case, when examining the clustered points in figure 7.13b it is evident that there are more points in the track than what would be expected for an obstacle this size. This is again indicative of the noise associated with the LiDAR scan results. In this particular case, the additional points cause the obstacle to appear wider compared to the true size of the obstacle. This can be seen in figures 7.13d and 7.13e which shows the free space that the car has identified in the horizon track. In this case, the widths of the potential horizon tracks are smaller than the true representation of the horizon track. However, one of the representations, specifically the one shown in figure 7.13e is still wide enough to contain a

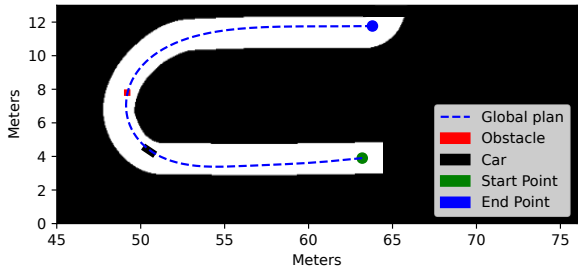
viable safe trajectory, as shown in figure 7.13f. The other track representation is discarded due to the fact that the space it represents is not large enough to accommodate the car.



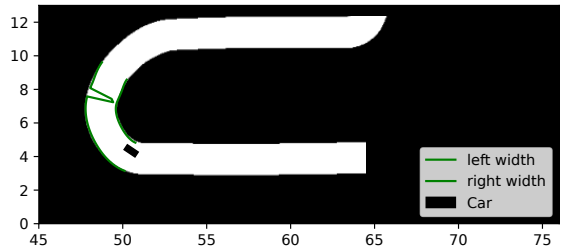
(a) Unprocessed LiDAR scan



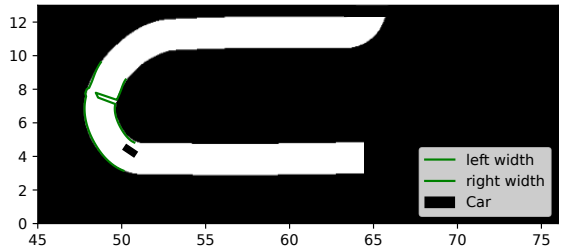
(b) Clustred points detected on the track



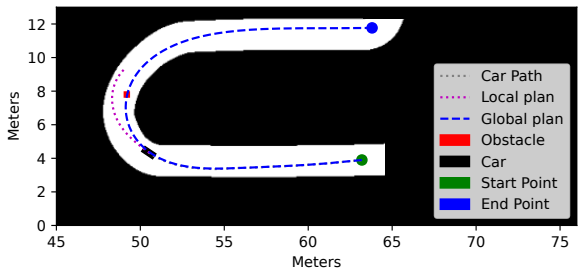
(c) Collision detected with identified Obstacle



(d) A potential horizon track, which represents unoccupied space that the car can freely enter.



(e) A potential horizon track, which represents unoccupied space that the car can freely enter.

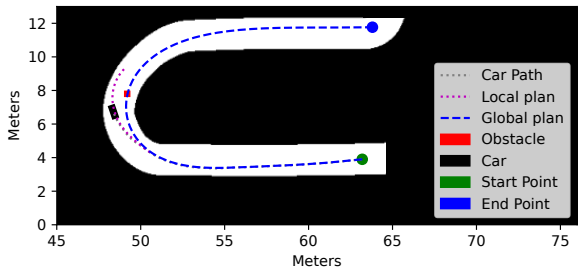


(f) Local Plan created to avoid the detected obstacle

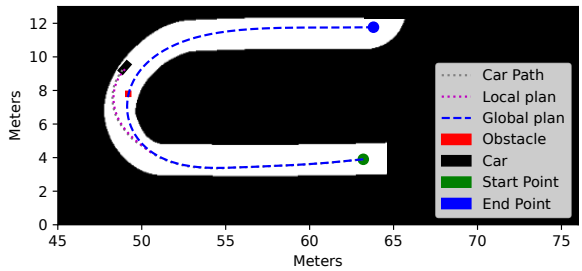
Figure 7.13: The obstacle avoidance detection and planning process for a hairpin

Figure 7.14 shows the car executing the local plan and rejoining the global plan. Figure 7.14a shows the car while it is executing the local plan and figure 7.14b when it has successfully completed the local plan. The car is able to complete the plan with a peak

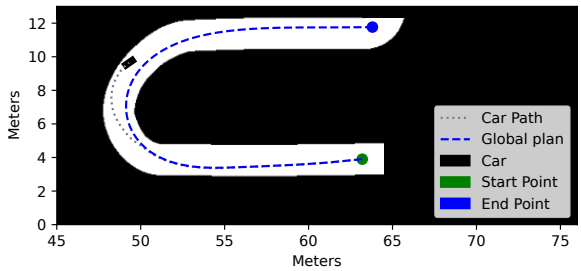
cross-track error of less than 0.05m. However, when the car has executed the local plan and begins to execute the global plan once more, figure 7.14c shows that the car is out of position and there is a significant cross-track error when the car's cross-track position is compared to the global reference. However, as shown in figure 7.14d, the Stanley controller is able to safely control the car to rejoin the global reference.



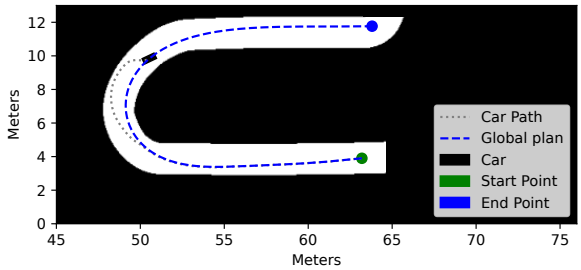
(a) Car executing the created local plan



(b) The end of the local plan



(c) The car begins executing the global plan again



(d) The control loop of the system brings the car back to the global reference

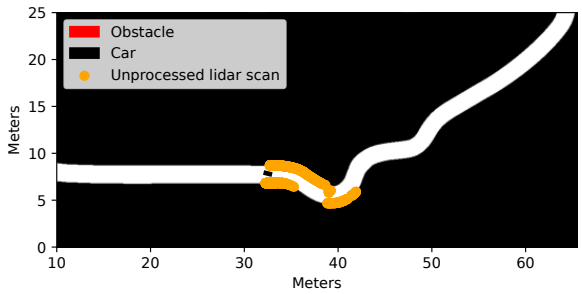
Figure 7.14: The obstacle avoidance execution for a hairpin

7.5.4. Chicane

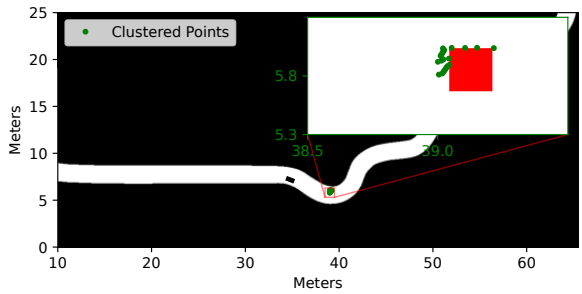
Finally, the obstacle avoidance system was tested on a chicane, the results of which are shown in figures 7.15 and 7.16. Figure 7.15 shows the process of obstacle detection, collision prediction, and collision avoidance planning for the car, while figure 7.16 shows the car executing the local plan and rejoining the global track.

Figures 7.15a and 7.15b show how the obstacle is detected. In this case, figure 7.15b shows that the points that represent the obstacle do not align with the obstacle shape, which is due to the noise associated with the LiDAR scan. Additionally, this is an example of a case where the car is not in a position where the LiDAR scan can detect the full obstacle. However, once the empty space in the horizon track is detected, (shown in figures 7.15d

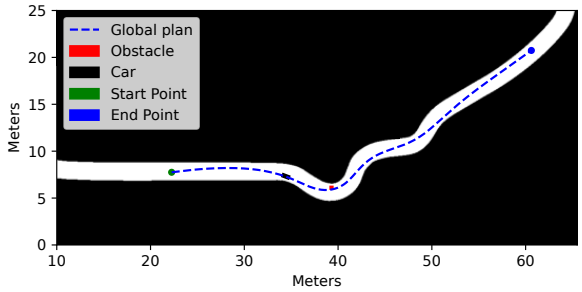
and 7.15e), a local plan is generated. Although the car did not have full knowledge of the size and shape of the obstacle when generating the local plan, it is clear in figure 7.15f that the plan avoids the obstacle in its entirety. Thus while the obstacle is executing the local plan the collision predictor will not be triggered again, despite the initial lack of information about the obstacle.



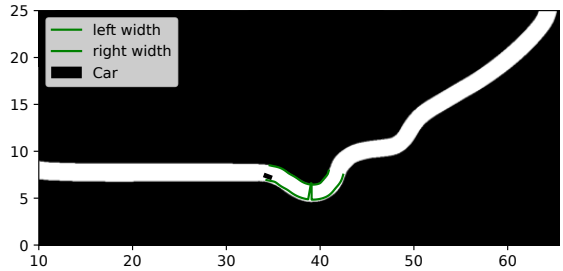
(a) Unprocessed LiDAR scan



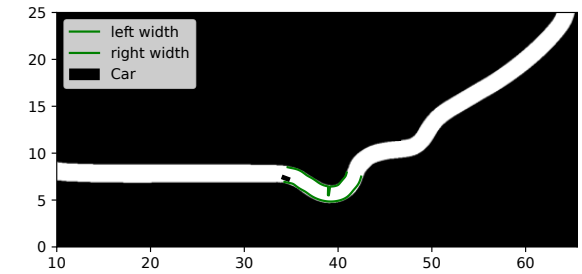
(b) Clustred points detected on the track



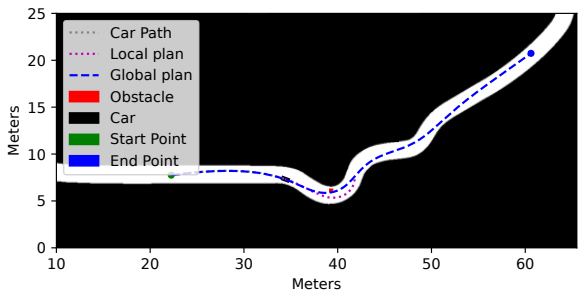
(c) Collision detected with identified Obstacle



(d) A potential horizon track, which represents unoccupied space that the car can freely enter.



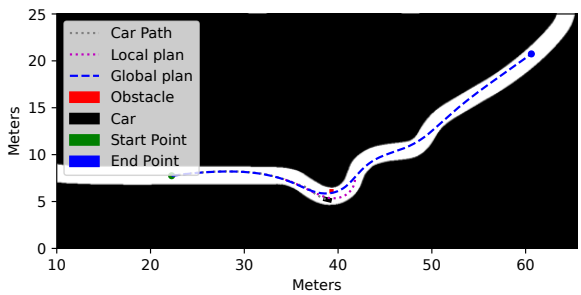
(e) A potential horizon track, which represents unoccupied space that the car can freely enter.



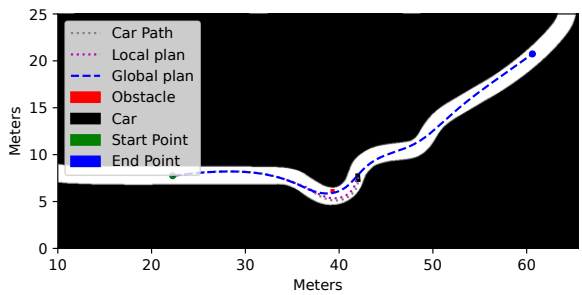
(f) Local Plan created to avoid the detected obstacle

Figure 7.15: The obstacle avoidance detection and planning process for a chicane

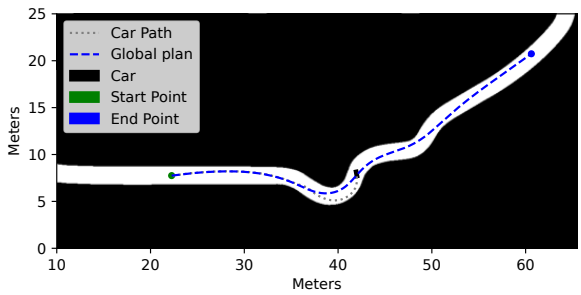
Figure 7.16a shows that while the car is executing the local plan, there is a slight cross-track error. This is especially evident when comparing the executed trajectory of the car and the planned local trajectory of the car in figure 7.16b. There is a peak cross-track error of 0.07m which does bring the car quite close to the track boundary. However, as shown in figure 7.16c, the car successfully executes the local plan. However, in this case, once the car has finished executing the local plan, there is a significant difference between the heading of the car and the global reference heading. Therefore, as shown in figure 7.16d, the Stanley controller causes the car to take extreme corrective measures to rejoin the global reference. However, the car does rejoin the global reference safely and continues executing the global plan.



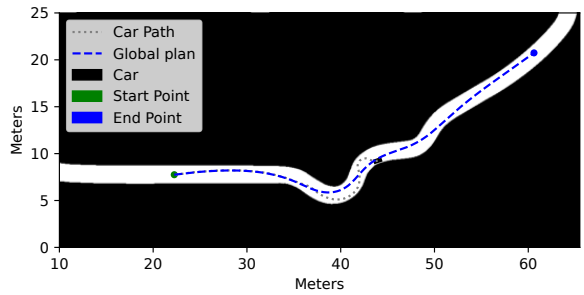
(a) Car executing the created local plan



(b) The end of the local plan



(c) The car begins executing the global plan again



(d) The control loop of the system brings the car back to the global reference

Figure 7.16: The obstacle avoidance execution for a chicane

7.5.5. Analysis of Results

The results in section 7.5 show that the obstacle avoidance system can successfully detect an obstacle, predict a collision, and plan a collision-free plan around detected obstacles. Table 7.1 shows a comparison of the planned lap times, the simulation lap time with an obstacle, and the simulation lap time without an obstacle. The lap time when the car is

forced to take avoidance action is, as expected, consistently higher than the lap time when the car can simply execute the global plan.

	Planned Lap Time	**Simulation Lap Time** with obstacle	**Simulation Lap Time** without obstacle
Straight	16.82	17.56	17.31
Corner	11.30	11.62	11.21
Hairpin	13.14	13.07	12.61
Chicane	16.10	16.13	15.51

Table 7.1: A table showing the planned lap time and simulation lap time for each track element

7.6. Monte Carlo Testing

In order to test the robustness of the collision avoidance system, a series of Monte Carlo tests were performed, using the F1Tenth simulator [70]. A Monte Carlo analysis tests how well the system performs in various scenarios.

7.6.1. Columbia Track

A series of Monte Carlo tests were performed using the Columbia track, as shown in figure 7.17.

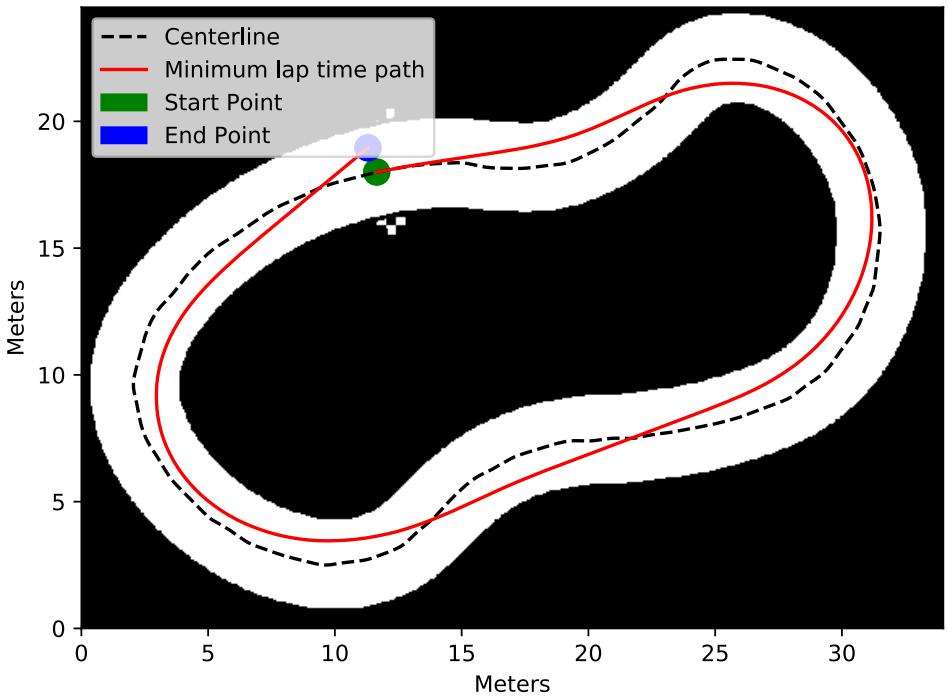
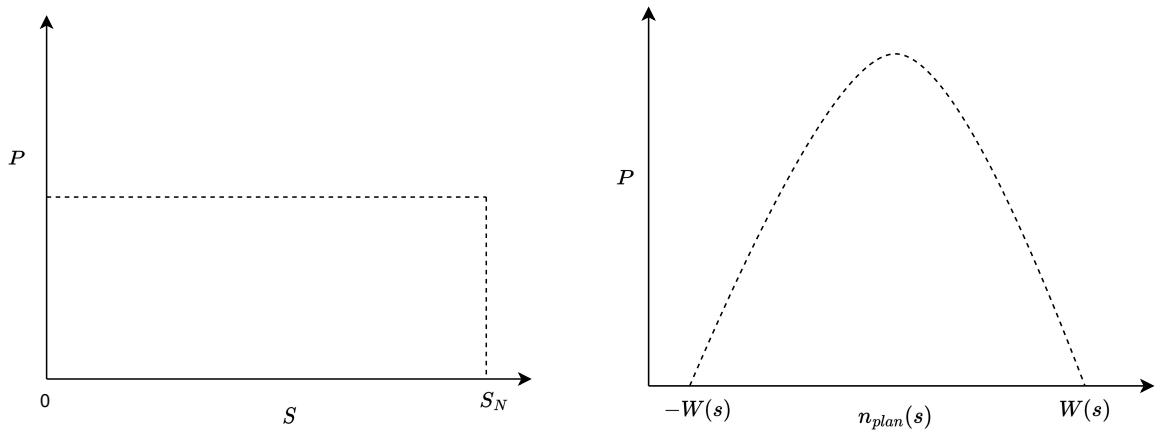


Figure 7.17: The Columbia track with a minimum lap-time plan

Setup

Each Monte Carlo test that was performed consisted of 1000 laps, where the car was initialised at the start of the track and set to execute the global plan as shown in figure 7.17. A number of random obstacles were placed on the track and the car was monitored to determine whether the car would collide with the unexpected obstacles or successfully avoid the obstacles. When the car had either finished the lap or crashed, the result was noted along with the lap time. Then the car and track were reinitialised and new random obstacles were generated within the track to replace the previous obstacles. The initial conditions of the car were kept constant throughout the entire Monte Carlo analysis. The car was initialised with a longitudinal velocity of 1ms^{-1} while the rest of the states were initialised to 0. A lookahead time of 4s was used for the collision prediction process, and a horizon planning distance of 5m was used.

The size and shape of the obstacles placed on the track were kept constant as 0.25m^2 squares. The location of obstacles placed on the track was placed according to probability distributions illustrated in figure 7.18. The obstacle location across the track is given by a Gaussian-like distribution with a mean that was centred along the planned cross-track position of the car. The probability of the object's location along the arc length of the planned racing line was given by a uniform probability distribution. If a lap occurred where that car did not have to take any avoidance action,



(a) An illustration of the probability distribution of the object location along the length of the track

(b) An illustration of the probability distribution of the object location across the width of the track

Figure 7.18: An illustration showing the probability distribution of the location of each obstacle.

Three Monte Carlo tests were performed, each consisting of 1000 individual laps around the track. The tests were characterised by the number of obstacles that were placed on the track. The first test was performed with one obstacle in the track, the second test with two obstacles and the final test with three obstacles. The lap times of each successful lap were recorded, as well as the number of times that the car successfully completed a lap,

the number of times that the car collided with the obstacle (or track boundary), as well as the number of times that the car stopped due to exceeding the number of permissible failed local plan (as detailed in section 7.4).

Results

The results of the three Monte Carlo tests are presented in this section. Table 7.2 shows the success rate, collision rate, and stop rate for each test.

Number of Obstacles in the Track	Success Rate (%)	Collision Rate (%)	Safety Stop Rate (%)
0	100%	0%	0%
1	96.598%	0.320%	3.082%
2	87.230%	2.128%	10.642%
3	70.409%	4.494%	25.097

Table 7.2: A table showing the success rate, collision rate and stop rate of the Monte Carlo simulations

The results in table 7.2 show that the success rate of the car safely finishing a lap decreases as the number of obstacles increases. The number of times that the car crashes increases, as well as the number of times that the car stops due to safety reasons, as the number of obstacles increases. The majority of the time when the car does not successfully finish a lap, it stops safely before it would experience a collision. However, there are times when the car does experience a collision with either the track boundaries or an obstacle in the track. The main explanation for the collisions is the fact that the local planner does not take the dynamics of the car into account. Additionally, the local planner maintains the speed of the vehicle at the moment the plan is made until the local plan is successfully executed. This means that depending on the location of the obstacle, the layout of the track and the speed that the vehicle was travelling when the local plan was made, the car may exceed the limits of its handling capabilities when attempting to execute the local plan. If the car does exceed the limits of its handling capabilities, it will start to slip and will be extremely difficult to control. This loss of control is the leading cause of the increase in collisions as the car cannot properly execute the local motion plan and the car crashes while attempting to execute it.

Figure 7.19 shows a box and whiskers plot comparing the lap times of the successful laps for the tests that were performed. It is important to note that the time that it takes for the car to complete a lap without any obstacle avoidance manoeuvres is 29.2s.

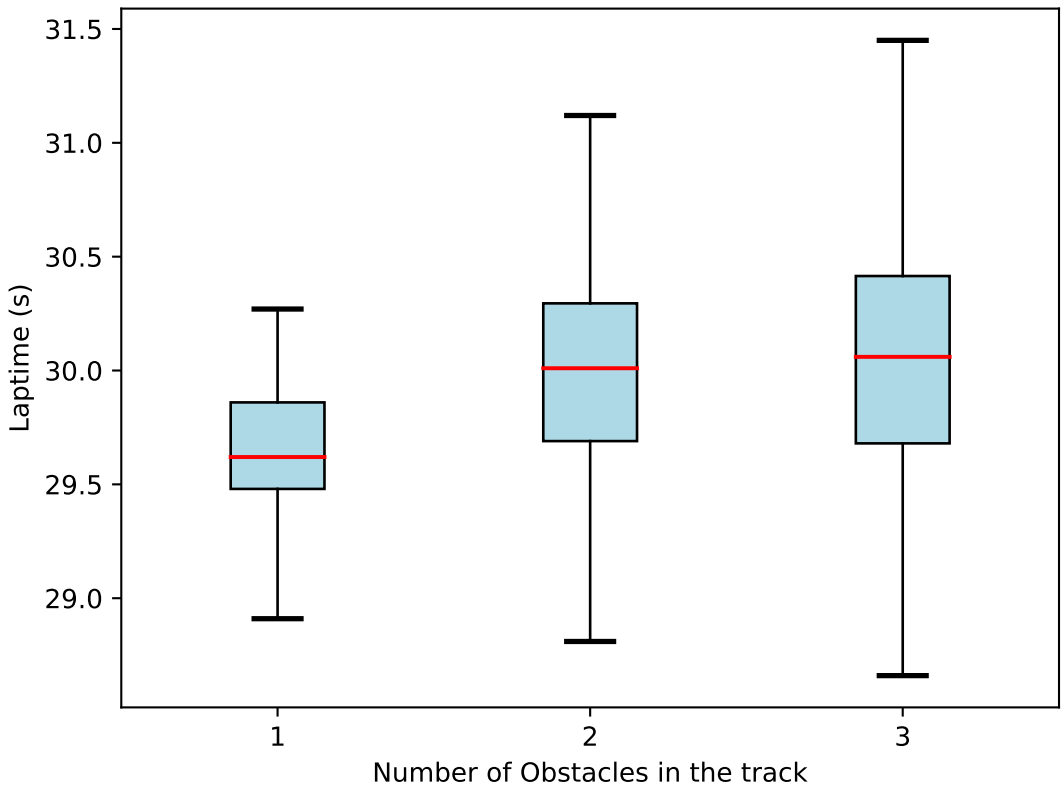


Figure 7.19: Box and whisker plot of the lap times from the Monte Carlo simulations

Figure 7.19 shows that the time that it takes for the car to complete a lap tends to increase when more obstacles are placed on the track. This is expected, as the car will be required to deviate from the optimal motion plan in order to avoid the obstacles, and thus lose time in the process. However, an interesting result from the test shows that there are cases when the lap time decreases while the car is avoiding an obstacle compared to the lap time when the car simply executes the global optimal motion plan. This is due to the same reason that the car might crash during the execution of the local plan, specifically the fact that the local planner generates a plan with a constant speed that does not consider the dynamics of the vehicle. This can lead to cases where the car begins to slip and become difficult to control. Although this is the main reason why the car might crash, in some instances, this may lead to a shorter lap time as the car may begin to slip due to excess speed, while not crashing thus finishing the lap in a shorter time, albeit in a dangerous fashion. Given that this behaviour from the car is just as likely to cause a collision as not, when this occurs it can be treated as a failure. A more stringent definition of a successful lap yields results as shown in table 7.3

Number of Obstacles in the Track	Success Rate (%)	Collision Rate (%)	Safety Stop Rate (%)
0	100%	0%	0%
1	90.386%	6.532%	3.082%
2	80.481%	8.877%	10.642%
3	63.423%	11.480%	25.097

Table 7.3: A table showing the success rate, collision rate and stop rate of the Monte Carlo simulations

The failure rate of the autonomous racing system increases significantly in all three tests, as expected. However, the increase in failure rate is fairly consistent, although it does increase slightly more as each obstacle is added to the track.

Another interesting result shown in figure 7.19 is the increase in variance in lap times as the number of obstacles increases, as well as the fact that the variance is largely symmetrical for each test. This is largely because, in general, the more obstacles that are placed on the track, the more local plans are generated. Therefore, the speed of the car will stay constant for a longer amount of time. Whether in that time, without the presence of an obstacle, the car would be accelerating, or decelerating depends on the track and the location of the obstacle. If, in the time that the speed of the car stays constant, the car would be accelerating then the lap time of the car will increase more than it would have due to the deviation from the optimal path. Otherwise, if the car would be decelerating during that period, the lap time will be shorter than planned, if the lap is succesfully executed.

Finally, the results shown in figures 7.20 and 7.21 show the computation time of the local planner. Figure 7.20 shows the results with outliers removed, while figure 7.21 shows those outliers.

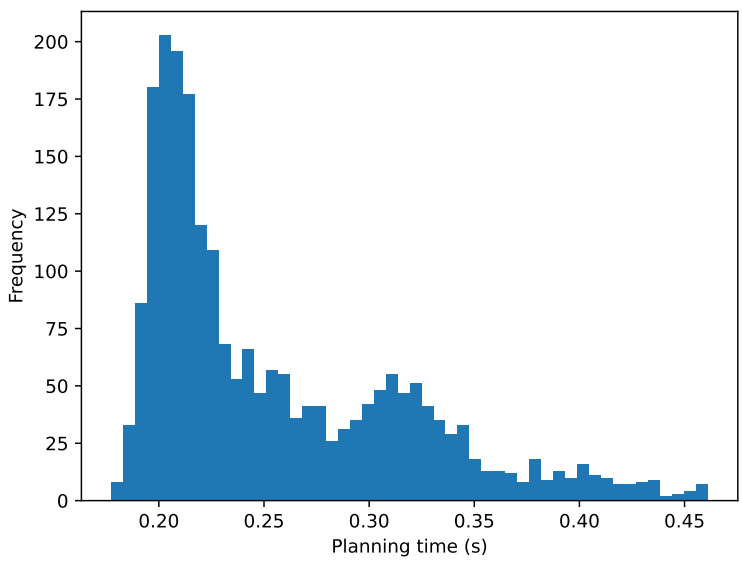


Figure 7.20: A histogram of the computation time of the local planner

The vast majority of times, the local planner generates a motion plan in less than 0.25s, with an average planning time of 0.24s and a range of 0.16s to 0.46s (excluding outliers). The planning time is dependent on a variety of factors but the factors that influence the planning time that are part of the autonomous racing system are the initial conditions of the car, the planning horizon distance and the complexity of the track. It is worth noting that the planning time will be heavily dependent on the hardware and that for these tests, the planning process took place on a desktop PC equipped with an Intel Core i7-11700 processor running at 2.50 GHz, 16 GB of DDR4 RAM, and an NVIDIA Quadro P400 GPU with 2 GB GDDR5 VRAM. The software environment will also play a role in the planning time, and for these tests, the planning process was implemented in Python 3.8.5 and executed on a system running Ubuntu 20.04 using Pyomo and IPOPT.

The outliers shown in figure 7.21 are a minimal few, extending from 0.5s to 2.7s, that do not significantly impact the overall effectiveness of the collision avoidance system. This is due to the limit that can be placed on the computation time of the local planner to ensure a solution is generated in an acceptable time frame, or that a specific instance is treated as a failed plan.

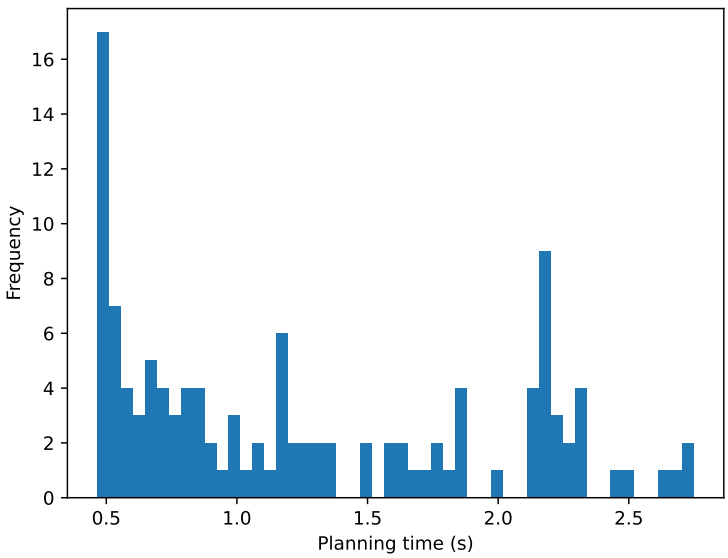


Figure 7.21: Histogram showing the outlier planning times

7.6.2. Circuit de Barcelona-Catalunya

A Monte Carlo test was also performed on the Circuit de Barcelona-Catalunya as shown in figure 7.22

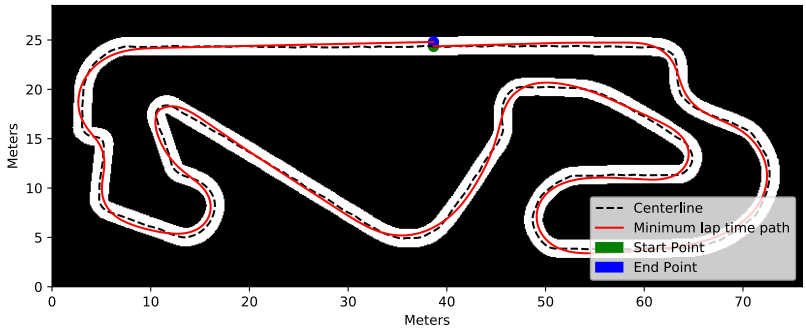


Figure 7.22: The Circuit de Barcelona-Catalunya track with a minimum lap-time plan

Setup

The test was set up in a similar manner as described in section 7.6.1, using a similar probability distribution in order to generate random obstacles in the track, initialisation process and test methodology. The same lookahead time for the collision prediction system was used, as well as the same horizon planning distance. However, this Monte Carlo test consisted of 100 laps with 2 obstacles in the track and the shape and size of the obstacles were fixed as a 0.09m^2 square.

Results

Table 7.4 shows the success rate, collision rate and stop rate for the test.

Success Rate (%)	Collision Rate (%)	Safety Stop Rate (%)
84%	2%	14%

Table 7.4: A table showing the success rate, collision rate and stop rate of the Monte Carlo simulations

The results show that the car was largely able to race safely through the track, and safely avoid any obstacles. However, as in the previous tests, there were times when the car either crashed or stopped due to safety concerns. Again, similarly to the previous tests, the main reason why the car crashes is the fact that the local planner does not take the car dynamics into account when generating a collision-free plan. However, as the results show, this is a minority case, even for a complex track.

Figure 7.23 shows the lap times, from the Monte Carlo test. The lap time when the obstacle does not take any collision avoidance action is 97.3s. As seen in the other tests, in general, when the car is forced to take collision avoidance action, the lap time tends to

increase. However, there are cases when this can lead to a shorter time, although this is a trade-off with the ability to control the car reliably. Similarly to the previous tests, we can classify these cases as a failure, which leads to the results shown in table 7.5.

Success Rate (%)	Collision Rate (%)	Safety Stop Rate (%)
77%	9%	14%

Table 7.5: A table showing the success rate, collision rate and stop rate of the Monte Carlo simulations

An interesting thing to note is that the results of this test on this track closely resemble the results of the tests performed on the Columbia track in the previous section. The consistency between the results shows that the autonomous racing system can successfully generalise to a more complex track.

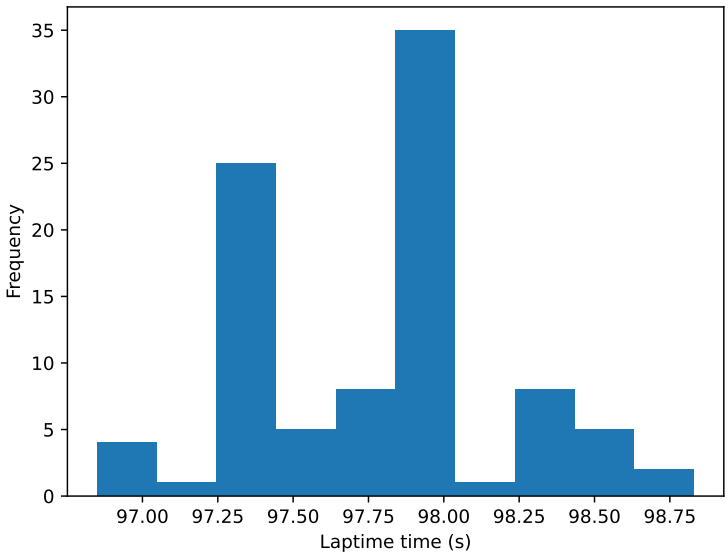


Figure 7.23: Histogram showing the lap times

Figures 7.24 and 7.25 show the computation time of the local planner which was implemented on the same hardware and in the same software environment as the previous tests. Figure 7.24 shows the computation time of the local planner with the outline removed, with an average time of 0.34s. This is slightly higher than the previous tests, however, the track is more complex, which explains the increase in computation time. The outliers are shown in figure 7.25 and range from 0.76s to 1.9s. Unlike the average computation time of 0.34s, the outlying computation times are not viable for real-time racing. However, this can be amended by decreasing the allowable time for the local planner to solve before the planning is treated as unsuccessful, although this may end up increasing the number of safety stops that the system performs.

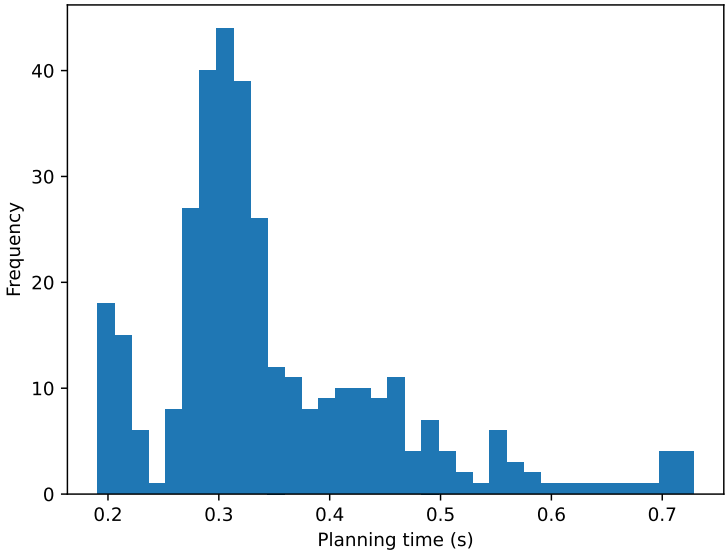


Figure 7.24: Histogram showing the local planner computation time, with outliers removed

Something to note about these results, is that the number of plans that are made in the test exceeds the minimum number of viable plans, 1 plan per obstacle in each lap, which would total 200. This shows that even if a local plan is made and executed, if a collision is detected again, whether due to the car's inability to accurately follow the plan or new information being made available to the car, a new plan is made and thus the car will adjust to the new conditions.

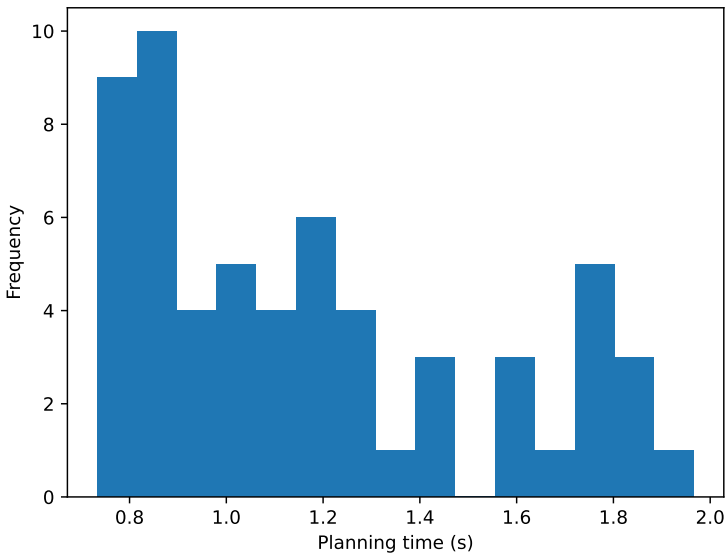


Figure 7.25: Histogram showing the outlying local planner computation times

7.7. Summary

This chapter detailed the obstacle avoidance process used by the autonomous racing system developed in this thesis. The chapter started by detailing the obstacle detection algorithm by clustering LiDAR points that are within the track boundary. Then a collision prediction process was presented, whereby the car is propagated forward in time to determine whether a collision will occur within a time horizon. Then a local planner was presented which formulates an optimal control problem, using a kinematic bicycle model, in order to generate a motion plan that avoids detected obstacles by formulating additional constraints that prevent the car from entering occupied space. The obstacle avoidance process was then tested in simulation and the results were analysed accordingly.

Chapter 8

Conclusion

8.1. Summary and Key Outcomes

This research project aimed to create an autonomous racing system that is capable of racing competitively and avoiding unexpected obstacles while racing. To accomplish this, a hierarchical motion planner was developed using both a dynamic bicycle model and a kinematic model.

The high-level motion planner was generated by formulating and solving an optimal control problem that accounted for the vehicle dynamics by using the dynamic bicycle model and solving a minimal-lap time optimisation problem. The output of the high-level motion planner showed that the resultant motion plan obeyed the vehicle dynamics and tyre constraints while generating a motion plan that was at the limits of the vehicle's handling capabilities. The global planner demonstrated that it could successfully generalise to different tracks and track elements and provide a minimal-laptime motion plan for each.

A path-tracking controller was developed and tuned to the response of the car while a particle filter was implemented for vehicle localisation. The execution of the global plan was tested on a variety of track elements as well as a complete track and it was demonstrated that the car could successfully execute the motion plan that was generated. The maximum lateral error of the car when executing a motion plan even was 0.2m even while operating at the limits of its handling capabilities. The maximum heading error of the car was less than 0.8° which is well below any meaningful standard. The car was shown to be able to complete racing circuits in a similar, or less, amount of time than was planned for, with a consistent error margin within 4%.

An obstacle detection algorithm was developed which could detect unexpected obstacles within the track, differentiating between obstacles and track borders. A collision prediction process was then presented which would determine if a collision would occur between the car and any detected obstacles. The collision prediction process triggers the low-level motion planner in order to create a collision-free local plan that successfully navigates around any obstacles while continuing to race.

The local plan is generated by formulating an optimal control problem, using a kinematic bicycle model, and incorporates additional constraints to ensure that the car avoids the obstacles that have been detected. The obstacle avoidance process was tested on different track elements, and the car was successfully able to avoid the obstacles placed on the track and safely rejoin the global plan.

A Monte Carlo analysis was performed using the full autonomous racing system developed in the thesis. The analysis showed that the autonomous car could reliably detect and avoid collision with unexpected obstacles, with a maximum collision rate of 4.94% and a minimum successful finish rate of 70.409%. The rest of the time, the system was unable to calculate a viable safe plan to continue racing, However, the car was brought safely to a stop before a collision occurred. Thus an autonomous racing system was successfully developed in this thesis, one that can successfully generate a racing plan for a known track, accurately execute the plan, and successfully avoid unexpected obstacles within the track, while avoiding having to formulate and solve an optimisation problem at every timestep.

8.2. Future Work

- The local planner could be improved by having it account for the dynamics of the car, using a more accurate vehicle model while avoiding compromising the real-time capabilities of the local planner.
- The collision prediction system can be improved by having it account for the dynamics of detected obstacles. A simple example of such an implementation is to have the autonomous system remember previously detected obstacles and their locations. If the location of an obstacle changes in the next sampling instance then that object can be assumed to be a dynamic obstacle. The behaviour of the obstacle can be modelled based on its previous history and the future positions of that obstacle can be predicted based on its behaviour.
- The local planner can also be improved to account for dynamic obstacles. The local planner is spatial-based and can easily account for static obstacles. However, this can be extended to dynamic obstacles by creating a model that propagates the obstacle forward in time, and then maps the position of the car at each timestep in the propagation back onto the track, while propagating the car forward as well (similar to the suggestion for the improved collision prediction process). The result of this would yield a map within the track of where the obstacle will be in the future and thus the local planner can plan to avoid the spaces which will be occupied in the future.
- Finally, an interesting concept to consider expanding this research to is overtaking

manoeuvres. When a competitor race car is detected, a possible method of generating an overtaking plan would be to create a minimum-curvature trajectory for a short horizon period ahead of the car. This would not generate an optimal racing line, but would instead give a racing line that may force the opposing car to take avoidance action itself. This would compromise the times of both cars, but allow the overtaking car a chance to pull ahead by forcing the opposing car to take a less optimal racing line.

Bibliography

[1] "Localize turtlebot using monte carlo localization algorithm - matlab amp; simulink - mathworks américa latina." [Online]. Available: https://la.mathworks.com/help/nav/ug/localize-turtlebot-using-monte-carlo-localization.html

[2] R. Davidson, "Dbscan algorithm from scratch in python," Oct 2020. [Online]. Available: https://scrunts23.medium.com/dbscan-algorithm-from-scratch-in-python-475b82e0571c

[3] M. Rokonuzzaman, N. Mohajer, S. Nahavandi, and S. Mohamed, "Review and performance evaluation of path tracking controllers of autonomous vehicles," *IET Intelligent Transport Systems*, vol. 15, 03 2021.

[4] D. Nye, "The 1894 paris-rouen trial, the very first competitive motorsport event," Jun 2019. [Online]. Available: https://www.goodwood.com/grr/race/historic/2019/6/the-1894-paris-rouen-trial-the-race-that-wasnt-a-race/

[5] M. Kew, "The 10 biggest innovations in formula 1 history: Active suspension, halo, fan car and more," Apr 2022. [Online]. Available: https://www.autosport.com/f1/news/the-10-biggest-innovations-in-formula-1-history-active-suspension-halo-fan-car-more/10125979/

[6] C. Smith, *Drive to win: The Essential Guide to Race Driving.* Carroll Smith Consulting, Inc., 1996.

[7] A. Jiang, "Research on the development of autonomous race cars and impact on self-driving cars," *Journal of Physics: Conference Series*, vol. 1824, no. 1, p. 012009, 2021.

[8] [Online]. Available: https://f1tenth.org/

[9] [Online]. Available: https://www.fsaeonline.com/

[10] [Online]. Available: https://www.indyautonomouschallenge.com/

[11] [Online]. Available: https://www.roborace.com/

[12] J. Betz, H. Zheng, A. Liniger, U. Rosolia, P. Karle, M. Behl, V. Krovi, and R. Mangharam, "Autonomous vehicles on the edge: A survey on autonomous

vehicle racing," *IEEE Open Journal of Intelligent Transportation Systems*, vol. 3, pp. 458–488, 2022. [Online]. Available: https://doi.org/10.1109%2Fojits.2022.3181510

[13] h. TIAN, J. NI, and J. HU, "Autonomous driving system design for formula student driverless racecar," in *2018 IEEE Intelligent Vehicles Symposium (IV)*, 2018, pp. 1–6.

[14] A. Liniger, A. Domahidi, and M. Morari, "Optimization-based autonomous racing of 1:43 scale rc cars," *Optimal Control Applications and Methods*, vol. 36, pp. 628–647, 09 2015.

[15] K. Güçkıran and B. Bolat, "Autonomous car racing in simulation environment using deep reinforcement learning," in *2019 Innovations in Intelligent Systems and Applications Conference (ASYU)*, 2019, pp. 1–6.

[16] J. Culley, S. Garlick, E. Gil Esteller, P. Georgiev, I. Fursa, I. Vander Sluis, P. Ball, and A. Bradley, "System design for a driverless autonomous racing vehicle," in *2020 12th International Symposium on Communication Systems, Networks and Digital Signal Processing (CSNDSP)*, 2020, pp. 1–6.

[17] T. Weiss and M. Behl, "Deepracing: Parameterized trajectories for autonomous racing," *CoRR*, vol. abs/2005.05178, 2020. [Online]. Available: https://arxiv.org/abs/2005.05178

[18] J. Kabzan, L. Hewing, A. Liniger, and M. N. Zeilinger, "Learning-based model predictive control for autonomous racing," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3363–3370, 2019.

[19] T. Novi, A. Liniger, R. Capitani, and C. Annicchiarico, "Real-time control for at-limit handling driving on a predefined path," *Vehicle System Dynamics*, vol. 58, pp. 1–30, 04 2019.

[20] V. Fors, B. Olofsson, and L. Nielsen, "Formulation and interpretation of optimal braking and steering patterns towards autonomous safety-critical manoeuvres," *Vehicle System Dynamics*, vol. 57, pp. 1–18, 11 2018.

[21] X. Jin, G. Yin, and N. Chen, "Advanced estimation techniques for vehicle system dynamic state: A survey," *Sensors*, vol. 19, p. 4289, 10 2019.

[22] J. Kabzan, M. Valls, V. Reijgwart, H. Hendrikx, C. Ehmke, M. Prajapat, A. Bühler, N. Gosala, M. Gupta, R. Sivanesan, A. Dhall, E. Chisari, N. Karnchanachari, S. Brits, M. Dangel, I. Sa, R. Dube, A. Gawel, M. Pfeiffer, and R. Siegwart, "Amz driverless: The full autonomous racing system," 05 2019.

[23] A. Liniger, "Path planning and control for autonomous racing," Ph.D. dissertation, ETH Zurich, 2018. [Online]. Available: https://doi.org/10.3929/ethz-b-000302942

[24] X. Qian, A. de La Fortelle, and F. Moutarde, "A hierarchical model predictive control framework for on-road formation control of autonomous vehicles," in *2016 IEEE Intelligent Vehicles Symposium (IV)*, 2016, pp. 376–381.

[25] J. Kong, M. Pfeiffer, G. Schildbach, and F. Borrelli, "Kinematic and dynamic vehicle models for autonomous driving control design," 06 2015, pp. 1094–1099.

[26] P. Polack, F. Altché, B. Novel, and A. de La Fortelle, "The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles?" pp. 812–818, 06 2017.

[27] G. Ishigami, K. Nagatani, and K. Yoshida, "Path following control with slip compensation on loose soil for exploration rover," 11 2006, pp. 5552 – 5557.

[28] C. Canudas de Wit and P. Tsiotras, "Dynamic tire friction models for vehicle traction control," in *Proceedings of the 38th IEEE Conference on Decision and Control (Cat. No.99CH36304)*, vol. 4, 1999, pp. 3746–3751 vol.4.

[29] C. C.-d.-W. E. Velenis, P. Tsiotras and M. Sorine, "Dynamic tyre friction models for combined longitudinal and lateral vehicle motion," *Vehicle System Dynamics*, vol. 43, no. 1, pp. 3–29, 2005. [Online]. Available: https://doi.org/10.1080/00423110412331290464

[30] H. Dugoff, P. S. Fancher, and L. A. Segel, "An analysis of tire traction properties and their influence on vehicle dynamic performance," 1970. [Online]. Available: https://api.semanticscholar.org/CorpusID:108727350

[31] M. Bian, L. Chen, Y. Luo, and K. Li, "A dynamic model for tire/road friction estimation under combined longitudinal/lateral slip situation," vol. 1, 04 2014.

[32] N. Ding and S. Taheri, "A modified dugoff tire model for combined-slip forces," *Tire Science and Technology*, vol. 38, pp. 228–244, 09 2010.

[33] E. Bakker, L. Nyborg, and H. B. Pacejka, "Tyre modelling for use in vehicle dynamics studies," *SAE Transactions*, vol. 96, pp. 190–204, 1987. [Online]. Available: http://www.jstor.org/stable/44470677

[34] A. Gotlib, K. Łukojć, and M. Szczygielski, "Localization-based software architecture for 1:10 scale autonomous car," in *2019 International Interdisciplinary PhD Workshop (IIPhDW)*, 2019, pp. 7–11.

[35] A. Woo, B. Fidan, W. Melek, S. Zekavat, and R. Buehrer, "Localization for autonomous driving," pp. 1051–1087, 01 2019.

[36] F. Nobis, J. Betz, L. Hermansdorfer, and M. Lienkamp, "Autonomous racing: A comparison of slam algorithms for large scale outdoor environments," 02 2019, pp. 82–89.

[37] "Cartographer ros integration." [Online]. Available: https://google-cartographer-ros.readthedocs.io/en/latest/

[38] N. B. Gosala, A. Bühler, M. Prajapat, C. Ehmke, M. Gupta, R. Sivanesan, A. Gawel, M. Pfeiffer, M. Bürki, I. Sa, R. Dubé, and R. Y. Siegwart, "Redundant perception and state estimation for reliable autonomous racing," *2019 International Conference on Robotics and Automation (ICRA)*, pp. 6561–6567, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:52844316

[39] L. Andresen, A. Brandemuehl, A. Honger, B. Kuan, N. Vodisch, H. Blum, V. Reijgwart, L. Bernreiter, L. Schaupp, J. Chung, M. Bürki, M. Oswald, R. Siegwart, and A. Gawel, "Accurate mapping and planning for autonomous racing," 10 2020, pp. 4743–4749.

[40] N. Large, F. Bieder, and M. Lauer, "Comparison of different slam approaches for a driverless race car," *tm - Technisches Messen*, vol. 88, 03 2021.

[41] M. Valls, H. Hendrikx, V. Reijgwart, F. Meier, I. Sa, R. Dube, A. Gawel, M. Bürki, and R. Siegwart, "Design of an autonomous racecar: Perception, state estimation and system integration," 05 2018, pp. 2048–2055.

[42] S. Katwe, N. Iyer, M. Khan, M. Peters, and M. Mahale, "Particle filter based localization of autonomous vehicle," in *2021 2nd Global Conference for Advancement in Technology (GCAT)*, 2021, pp. 1–6.

[43] T. Stahl, A. Wischnewski, J. Betz, and M. Lienkamp, "Ros-based localization of a race vehicle at high-speed using lidar," *E3S Web of Conferences*, vol. 95, p. 04002, 01 2019.

[44] D. Metz and D. Williams, "Near time-optimal control of racing vehicles," *Automatica*, vol. 25, no. 6, pp. 841–857, 1989. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0005109889900526

[45] D. Kelly and R. Sharp, "Time-optimal control of the race car: A numerical method to emulate the ideal driver," *Vehicle System Dynamics*, vol. 48, pp. 1461–1474, 12 2010.

[46] J. Vazquez, M. Bruhlmeier, A. Liniger, A. Rupenyan-Vasileva, and J. Lygeros, "Optimization-based hierarchical motion planning for autonomous racing," 10 2020, pp. 2397–2403.

[47] G. Perantoni and D. J. Limebeer, "Optimal control for a formula one car with variable parameters," *Vehicle System Dynamics*, vol. 52, no. 5, pp. 653–678, 2014. [Online]. Available: https://doi.org/10.1080/00423114.2014.889315

[48] A. Heilmeier, A. Wischnewski, L. Hermansdorfer, J. Betz, M. Lienkamp, and B. Lohmann, "Minimum curvature trajectory planning and control for an autonomous race car," *Vehicle System Dynamics*, vol. 58, pp. 1497 – 1527, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:197448385

[49] F. Braghin, F. Cheli, S. Melzi, and E. Sabbioni, "Race driver model," *Computers & Structures*, vol. 86, no. 13, pp. 1503–1516, 2008, structural Optimization. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0045794908000163

[50] L. Cardamone, D. Loiacono, P. L. Lanzi, and A. P. Bardelli, "Searching for the optimal racing line using genetic algorithms," in *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*, 2010, pp. 388–394.

[51] M. Bevilacqua, A. Tsourdos, and A. Starr, "Particle swarm for path planning in a racing circuit simulation," in *2017 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, 2017, pp. 1–6.

[52] A. Remonda, S. Krebs, E. Veas, G. Luzhnica, and R. Kern, "Formula rl: Deep reinforcement learning for autonomous racing using telemetry data," *ArXiv*, vol. abs/2104.11106, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:201724736

[53] Z. Lu, C. Zhang, H. Zhang, Z. Wang, C. Huang, and Y. Ji, "Deep reinforcement learning based autonomous racing car control with priori knowledge," in *2021 China Automation Congress (CAC)*, 2021, pp. 2241–2246.

[54] A. Bulsara, A. Raman, S. Kamarajugadda, M. Schmid, and V. Krovi, "Obstacle avoidance using model predictive control: An implementation and validation study using scaled vehicles," 04 2020.

[55] J. hwan Jeon, R. V. Cowlagi, S. C. Peters, S. Karaman, E. Frazzoli, P. Tsiotras, and K. Iagnemma, "Optimal motion planning with the half-car dynamical model for autonomous high-speed driving," in *2013 American Control Conference*, 2013, pp. 188–193.

[56] S. Feraco, S. Luciani, A. Bonfitto, N. Amati, and A. Tonoli, "A local trajectory planning and control method for autonomous vehicles based on the rrt algorithm," in *2020 AEIT International Conference of Electrical and Electronic Technologies for Automotive (AEIT AUTOMOTIVE)*, 2020, pp. 1–6.

[57] O. Arslan, K. Berntorp, and P. Tsiotras, "Sampling-based algorithms for optimal motion planning using closed-loop prediction," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 4991–4996.

[58] V. Sezer and M. Gokasan, "A novel obstacle avoidance algorithm: "follow the gap method"," *Robotics and Autonomous Systems*, vol. 60, p. 1123–1134, 09 2012.

[59] B. Evans, H. A. Engelbrecht, and H. W. Jordaan, "Learning the subsystem of local planning for autonomous racing," in *2021 20th International Conference on Advanced Robotics (ICAR)*, 2021, pp. 601–606.

[60] R. Verschueren, M. Zanon, R. Quirynen, and M. Diehl, "Time-optimal race car driving using an online exact hessian based nonlinear mpc algorithm," in *2016 European Control Conference (ECC)*, 2016, pp. 141–147.

[61] D. Kim, J. Kang, and K. Yi, "Control strategy for high-speed autonomous driving in structured road," in *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, 2011, pp. 186–191.

[62] M. Samuel, M. Hussein, and M. Binti, "A review of some pure-pursuit based path tracking techniques for control of autonomous vehicle," *International Journal of Computer Applications*, vol. 135, pp. 35–38, 02 2016.

[63] V. Sukhil and M. Behl, "Adaptive lookahead pure-pursuit for autonomous racing," *ArXivorg*. [Online]. Available: https://par.nsf.gov/biblio/10320139

[64] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L.-E. Jendrossek, C. Koelen, and P. Mahoney, "Stanley: The robot that won the darpa grand challenge." *J. Field Robotics*, vol. 23, pp. 661–692, 01 2006.

[65] V. K and M. Sheta, "A comparative study of stanley, lqr and mpc controllers for path tracking application (adas/ad)," 06 2019, pp. 67–674.

[66] [Online]. Available: https://f1tenth.org/build

[67] f1tenth, "Drivers and system level code for the f1tenth vehicles." [Online]. Available: https://github.com/f1tenth/f1tenth_system

[68] ——, "A fast particle filter localization algorithm for the mit racecar." [Online]. Available: https://github.com/f1tenth/particle_filter

[69] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *kdd*, vol. 96, no. 34, 1996, pp. 226–231.

[70] M. O'Kelly, H. Zheng, D. Karthik, and R. Mangharam, "F1tenth: An open-source evaluation environment for continuous control and reinforcement learning," in *NeurIPS 2019 Competition and Demonstration Track*. PMLR, 2020, pp. 77–89.

[71] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[72] H. Zheng, J. Betz, and R. Mangharam, "Gradient-free multi-domain optimization for autonomous systems," 2022.

[73] f1tenth, "Containerized ros communication bridge for f1tenth gym environment." [Online]. Available: https://github.com/f1tenth/f1tenth_gym_ros

[74] M. Abe and W. Manning, "Chapter 3 - fundamentals of vehicle dynamics," in *Vehicle Handling Dynamics*, M. Abe and W. Manning, Eds. Oxford: Butterworth-Heinemann, 2009, pp. 47–118. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9781856177498000039

[75] A. Buyval, A. Gabdullin, R. Mustafin, and I. Shimchik, "Deriving overtaking strategy from nonlinear model predictive control for a race car," 09 2017, pp. 2623–2628.

[76] F. Kehrle, J. V. Frasch, C. Kirches, and S. Sager, "Optimal control of formula 1 race cars in a vdrift based virtual environment," *IFAC Proceedings Volumes*, vol. 44, no. 1, p. 11907–11912, 2011.

[77] Y. Gao, A. Gray, J. V. Frasch, T. Lin, E. H. Tseng, J. K. Hedrick, and F. Borrelli, "Spatial predictive control for agile semi-autonomous ground vehicles," 2012. [Online]. Available: https://api.semanticscholar.org/CorpusID:29661260

[78] M. L. Bynum, G. A. Hackebeil, W. E. Hart, C. D. Laird, B. L. Nicholson, J. D. Siirola, J.-P. Watson, and D. L. Woodruff, *Pyomo–optimization modeling in python*, 3rd ed. Springer Science & Business Media, 2021, vol. 67.

[79] W. E. Hart, J.-P. Watson, and D. L. Woodruff, "Pyomo: modeling and solving mathematical programs in python," *Mathematical Programming Computation*, vol. 3, no. 3, pp. 219–260, 2011.

[80] A. Wächter and L. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical Programming*, vol. 106, pp. 25–27, 2006.