FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Autonomous Vehicle Racing - Learning Control

João Paulo de Jesus Marques Sampaio

FINAL VERSION

Mestrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Fernando Arménio da Costa Castro e Fontes

October 31, 2024

Resumo

As corridas autónomas apresentam um conjunto distinto de desafios no domínio da condução autónoma, exigindo o controlo preciso de um veículo que opera no limite das suas capacidades de desempenho em ambientes imprevisíveis e competitivos. Estes desafios caracterizam-se por altas velocidades, acelerações rápidas e a necessidade de uma tomada de decisões extremamente rápida.

Esta tese examina as potenciais aplicações da aprendizagem por reforço no contexto da condução autónoma, com especial destaque para um veículo à escala 1:43. O objetivo principal era facilitar a navegação do pequeno veículo numa pista de forma coerente e optimizada. A fase inicial desta investigação envolveu a implementação de um controlador proporcional-integral-derivativo (PID) para facilitar a navegação do veículo na região central da pista. Posteriormente, foi efectuada uma investigação sobre o algoritmo de perseguição Carrot, tendo sido feita uma comparação com o controlador PID. Isto permitiu uma análise dos méritos relativos e das limitações de ambas as abordagens. Além disso, foi utilizada a aprendizagem por imitação para replicar e substituir o comportamento do controlador PID. Adicionalmente, foi efectuada uma avaliação para determinar as vantagens e limitações destes dois métodos de controlo.

Finalmente, utilizámos o método de aprendizagem por reforço, especificamente o método de Otimização da Política Proximal (PPO). O algoritmo PPO, conhecido por estabilizar o processo de treino através de uma nova função objetiva que minimiza o desvio da política anterior, permitiu-nos melhorar gradualmente as decisões de condução do veículo. Através de várias iterações de treino, conseguimos não só guiar o veículo de forma consistente através da pista designada, mas também otimizar a sua trajetória, assegurando curvas mais suaves e uma melhor gestão global da velocidade.

Palavras-chave: Corrida autónoma, carros RC, aprendizagem por reforço, controlador PID, algoritmo Carrot Chase, aprendizagem por imitação, otimização de políticas proximais.

Abstract

Autonomous racing presents a distinctive set of challenges in the field of autonomous driving, requiring the precise control of a vehicle operating at the edges of its performance capabilities within unpredictable and competitive environments. These challenges are characterised by high speeds, rapid accelerations, and the need for lightning-fast decision-making.

This thesis examines the potential applications of reinforcement learning in the context of autonomous driving, with a particular focus on a 1:43 scale vehicle. The principal objective was to facilitate the navigation of the small vehicle around a track in a consistent and optimised manner. The initial stage of this research involved the implementation of a proportional-integral-derivative (PID) controller to facilitate the vehicle's navigation within the central region of the track. Subsequently, an investigation was conducted into the Carrot chase algorithm, with a comparison made to the PID controller. This allowed for an analysis of the relative merits and limitations of both approaches. Furthermore, imitation learning was employed to replicate and replace the behaviour of the PID controller. Additionally, an evaluation was conducted to assess the respective advantages and limitations of these two controller methods.

Finally, we employed the method of reinforcement learning, specifically the Proximal Policy Optimisation (PPO) method. The PPO algorithm, known for stabilising the training process through a novel objective function that minimises the deviation from the previous policy, allowed us to incrementally improve the vehicle's driving decisions. Through several iterations of training, we were able to not only consistently guide the vehicle through the designated track but also optimise its trajectory, ensuring smoother turns and better overall speed management.

Keywords: Autonomous Racing, RC cars, Reinforcement Learning, PID Controller, Carrot Chase Algorithm, Imitation Learning, Proximal Policy Optimisation.

Sustainable Development Goals (SDGs)

Autonomous racing involves vehicles driving without human intervention. To achieve this, these vehicles are equipped with advanced technologies such as artificial intelligence (AI), machine learning, and sensor integration. These technologies enable the vehicles to navigate and compete on race tracks.

This field is intimately connected with several United Nations Sustainable Development Goals (SDGs), particularly in this project, SDG 4 (Quality Education), SDG 9 (Industry, Innovation, and Infrastructure), and SDG 11 (Sustainable Cities and Communities). The table 1 below summarises each SDG, the specific targets in this work, and the performance indicators.

Table 1: Targets and Performance Indicators of SDGs.

SDG	Target	Contribution	Performance Indicators and Metrics
4	4.4	STEM Education and Innovation	Number of STEM graduates, participation in STEM programs, innovation index
	4.5	Educational Programs and Workshops	Number of workshops held, participant diversity, completion rates
	4.7	Research and Development	Publications and citations, R&D funding, partnerships with institutions
9	9.1	Smart Infrastructure	Investment in smart infrastructure, adoption rate of smart technologies
	9.4	Technological Advancements	Number of new technologies developed, tech adoption rate, R&D expenditure
	9.5	Sustainable Manufacturing	Emissions reduction, energy efficiency, use of sustainable materials
11	11.2	Urban Mobility Solutions	Public transport usage, reduction in commute times, coverage of mobility solutions
	11.6	Reduced Emissions	Air quality indices, emission reduction percentages, number of green spaces
	11.7	Safety Enhancements	Reduction in accident rates, number of safety measures implemented, public safety surveys
	11.3	Data Collection and Analysis	Number of data points collected, quality of data analysis, decision-making impact

Quality Education (SDG 4)

- **STEM Education and Innovation:** The development and integration of autonomous racing technologies promotes STEM (Science, Technology, Engineering, and Mathematics) education.

- **Educational Programs and Workshops:** Organisations engaged in autonomous racing frequently establish educational programmes and workshops for students, enhancing their technical skills and knowledge.
- **Research and Development:** The advancement of autonomous racing has the potential to stimulate research and development in a number of fields, including machine learning, sensor technologies, and software engineering, leading students to have the opportunity to engage in internships, research projects, and the resolution of real-world problems.

Industry, Innovation and Infrastructure (SDG 9)

- **Technological Advancements:** Autonomous racing stimulates innovation in vehicle technology, AI algorithms, and safety systems that can be applied to other industrial sectors, improving their efficiency.
- **Sustainable Manufacturing:** The development of autonomous vehicles emphasises sustainable manufacturing practices, leading to reducing the environmental impact of vehicle production, which aligns with the goals of sustainable industrialisation.
- **Smart Infrastructure:** Autonomous racing requires the development of advanced infrastructure, including smart roads, communication networks, and energy-efficient facilities. Investing in such infrastructure can have spillover effects on broader transportation systems and contribute to building resilient and sustainable infrastructure.

Sustainable Cities and Communities (SDG 11)

- **Urban Mobility Solutions:** Autonomous racing can test and improve technologies for urban mobility, making autonomous vehicles safer and more reliable for sustainable transportation systems.
- **Reduced Emissions:** Autonomous vehicles can be designed to optimise driving patterns and reduce energy consumption, leading to lower emissions, which can help create cleaner and healthier urban environments.
- **Safety Enhancements:** The rigorous testing of autonomous vehicles in racing scenarios helps to develop and refine safety features that can be integrated into public transportation systems, enhancing the overall safety of urban mobility.
- **Data Collection and Analysis:** The analysis of vast amounts of data from autonomous racing can improve urban planning, traffic management, flow optimisation, congestion reduction, and public transportation efficiency.

Acknowledgements

First and foremost, I want to express my deep gratitude to my supervisor, Fernando Fontes, for his unwavering support from the very beginning and his willingness to help me overcome any obstacles.

Additionally, I am thankful for the support and assistance provided by all members of the LSCOE (Laboratory of Systems, Control, Optimization, and Estimation) SYSTEC-ARISE located in Lab i202 DEEC-FEUP.

I would also like to acknowledge the support from FCT/MCTES (PIDDAC) through Projects 2022.02320.PTDC, UIDB/00147/2020UIDP/00147/2020 SYSTEC and LA/P/0112/2020 ARISE for providing the hardware, components, and other equipment for the autonomous racing platform and various lab equipment used in this work.

Lastly, I would like to extend my heartfelt appreciation to my mother, the rest of my family, and my friends for their unwavering support throughout this journey.

To all of you, thank you very much.

João Sampaio

"Out of clutter, find Simplicity.
From discord, find Harmony.
In the middle of difficulty lies Opportunity."

Albert Einstein

Contents

Sustainable Development Goals (SDGs) iii

Acknowledgements v

1 Introduction 1
- 1.1 Context . . . 1
- 1.2 Motivation . . . 2
- 1.3 Contributions . . . 3
- 1.4 Dissertation Structure . . . 4

2 Existing Platform and Related Models 5
- 2.1 General Contextualisation . . . 5
- 2.2 Components . . . 6
  - 2.2.1 Track . . . 6
  - 2.2.2 Cars . . . 7
  - 2.2.3 Choosing the camera . . . 9
- 2.3 Perception . . . 11
- 2.4 Digital Twin . . . 13

3 Background on Autonomous Racing and Reinforcement Learning 15
- 3.1 Introduction . . . 15
- 3.2 Control Methods in Autonomous Racing . . . 15
- 3.3 Model of the vehicle . . . 17
  - 3.3.1 Single-track (bicycle) model . . . 17
  - 3.3.2 Simplified Model of the vehicle . . . 19
- 3.4 Imitation learning . . . 19
  - 3.4.1 Application of Imitation leaning . . . 19
  - 3.4.2 Variations and extensions of Imitation learning . . . 20
- 3.5 Reinforcement learning . . . 20
  - 3.5.1 What is Reinforcement learning . . . 20
  - 3.5.2 System of Reinforcement learning . . . 20
  - 3.5.3 Practical example . . . 22
  - 3.5.4 Fundamental equations . . . 22
- 3.6 Model-free vs Model-based . . . 24
- 3.7 Choosing the best method . . . 27
  - 3.7.1 Proximal Policy Optimisation (PPO) . . . 28
  - 3.7.2 Trust Region Policy Optimisation (TRPO) . . . 28
  - 3.7.3 Deep Deterministic Policy Gradient (DDPG) . . . 29

3.7.4 Deep Q Network (DQN) . . . 30
3.7.5 Methods comparison . . . 31

4 Path-Following Control in a Simplified Environment **32**
4.1 Simplified Model . . . 32
4.2 PID controller . . . 32
4.2.1 With an unknown environment . . . 33
4.2.2 With a known environment . . . 35
4.2.3 Tuning of the PID . . . 36
4.3 Carrot chase algorithm . . . 37
4.4 Comparisons between the PID and the Carrot chase algorithm . . . 39
4.5 Imitation learning . . . 40
4.6 Comparisons between the PID and the Imitation learning . . . 43

5 Planning and Control with Higher Fidelity Models **45**
5.1 Model used . . . 45
5.2 PID controller . . . 45
5.2.1 Results . . . 48
5.3 Reinforcement Learning . . . 49
5.3.1 State Space and Control Inputs . . . 49
5.3.2 Reward function . . . 50
5.3.3 Training process . . . 51
5.3.4 Behaviour parameters . . . 52
5.3.5 Results . . . 53

6 Conclusions and Future Work **58**
6.1 Conclusion . . . 58
6.2 Future work . . . 59

References **60**

List of Figures

1.1 Example of a Roborace car [1]. . . . 2
1.2 Example of an IAC car [2]. . . . 2

2.1 Overview of the system [3]. . . . 6
2.2 Track used for the tests. . . . 6
2.3 Cars used for the tests. . . . 7
2.4 Original interior of the car. . . . 7
2.5 Microcontroller [4]. . . . 8
2.6 Steering and Motor Drivers PCB. . . . 8
2.7 Schematic of the PCB circuit. . . . 9
2.8 GoPro Hero 10. . . . 10
2.9 ArUco markers. . . . 11
2.10 Red car detected. . . . 12
2.11 Blue car detected. . . . 12
2.12 Pink car detected. . . . 12
2.13 Finished physical track model. . . . 13
2.14 Finished physical car model. . . . 13
2.15 Top view camera. . . . 14
2.16 Race view camera. . . . 14

3.1 Model of the vehicle [5]. . . . 18
3.2 Model of the vehicle simplified. . . . 19
3.3 System of Reinforcement learning [6]. . . . 22
3.4 Taxonomy of RL algorithms [7]. . . . 27

4.1 Representation of the car with the lines to get the interception with the borders of the track. . . . 33
4.2 Calculation of the perpendicular distance to the car. . . . 34
4.3 Representation of the trajectory mapped. . . . 34
4.4 Representation of the closest point to the trajectory. . . . 35
4.5 Guide line with 20 points of distance between the two endpoints. . . . 36
4.6 Guide line with 80 points of distance between the two endpoints. . . . 36
4.7 Result of the PID controller with a guide line with 20 points of distance between the two endpoints. . . . 36
4.8 Result of the PID controller with a guide line with 80 points of distance between the two endpoints. . . . 36
4.9 Representation of the carrot chase algorithm. . . . 38
4.10 Representation of the angle Φ. . . . 38
4.11 Result of the carrot chase algorithm. . . . 39

4.12 Diagram of the neural network. 41
4.13 Path using Imitation Learning with a state size of 6. 42
4.14 Path using Imitation Learning with a state size of 12. 42
4.15 Path using Imitation Learning with a state size of 20. 43
4.16 Path using Imitation Learning. 43

5.1 Track marked with the checkpoints. 46
5.2 Track marked with the trajectory. 46
5.3 Representation of the method to find the closest point of the trajectory in a straight. 47
5.4 Representation of the method to find the closest point of the trajectory in a curve. 47
5.5 Path using the updated PID in the new environment. 48
5.6 Difference in angle between the car and the trajectory. 50
5.7 Representation of the referential used in Unity. 50
5.8 Environment_Cumulative Reward. 54
5.9 Environment_Episode Length. 54
5.10 Losses_Policy Loss. 55
5.11 Losses_Value Loss. 55
5.12 Policy_Beta. 55
5.13 Policy_Epsilon. 56
5.14 Policy_Learning Rate. 56
5.15 Policy_Entropy. 56
5.16 Policy_Extrinsic Value Estimate. 57

List of Tables

1 Targets and Performance Indicators of SDGs. iii

3.1 Advantages and Disadvantages of the Model-Free algorithms. 25
3.2 Advantages and Disadvantages of the Model-Based algorithms. 26

Abbreviations and Symbols

AI	Artificial Intelligence
AIL	Adversarial Imitation Learning
BLE	Bluetooth Low Energy
CTE	Cross-Track Error
DDPG	Deep Deterministic Policy Gradient
DQN	Deep Q Network
EKF	Extended Kalman Filter
HER	Hindsight Experience Replay
I2A	Imagination-Augmented Agents
IAC	Indy Autonomous Challenge
IIL	Interactive Imitation Learning
IL	Imitation Learning
IRL	Inverse Reinforcement Learning
MBMF	Magnitude Bounded Matrix Factorisation
MBVE	Model-Based Value Expansion
PCB	Printed Circuit Board
PID	Proportional-Integral-Derivative
PPO	Proximal Policy Optimisation
QR-DQN	Quantile Regression Deep Q Network
RC	Radio Controller
RL	Reinforcement Learning
SAC	Soft Actor-Critic
SDG	Sustainable Development Goal
STEM	Science, Technology, Engineering, and Mathematics
TD3	Twin Delayed Deep Deterministic
TRPO	Trust Region Policy Optimisation

α	slip sideways angle
δ	steering angle
θ	heading angle
v	velocity
R	reward
G	discounted return
γ	discount factor
S	state
A	action
π	policy
V	state value function
Q	action value function
ϕ	angle perpendicular to the trajectory
s	progress made into the track
μ	difference in angle between the car and the reference trajectory direction
v_x	velocity in the x-axes
v_z	velocity in the z-axes
w	angular velocity
V_{max}	maximum velocity
V_{xmax}	maximum velocity in the x-axes
V_{zmax}	maximum velocity in the z-axes
δ_{max}	maximum steering angle

Chapter 1

Introduction

1.1 Context

The rise of autonomous vehicles represents a significant shift in the automotive landscape, moving beyond conventional transportation to include new frontiers such as competitive racing. Autonomous racing, where self-driving cars compete on professional tracks, showcases the cutting-edge fusion of artificial intelligence, advanced engineering, and high-performance automotive technology. This domain not only exemplifies the latest advancements in autonomous driving systems but also serves as a critical testing ground for these technologies under extreme conditions. The racing environment, characterised by high speeds, rapid decision-making, and complex manoeuvres, presents unique challenges that push the boundaries of current autonomous vehicle capabilities.

Autonomous racing is distinct from traditional automotive challenges due to its emphasis on real-time adaptability and performance under pressure. Unlike standard driving, racing requires vehicles to make split-second decisions in highly dynamic environments. This setting provides an unparalleled opportunity to test and refine the algorithms that power autonomous systems. The insights gained from these high-intensity scenarios can be translated to improve the safety and efficiency of autonomous vehicles in everyday traffic conditions. By pushing the limits in a controlled, competitive environment, researchers can better understand the potential and limitations of current technologies.

Autonomous racing also plays a crucial role in promoting technological innovation. Competitive events such as the Roborace series and the Indy Autonomous Challenge (IAC) bring together a diverse array of participants, including leading research institutions, technology companies, and automotive manufacturers. These competitions create a fertile ground for collaboration and the exchange of ideas, driving rapid advancements in the field. The innovations developed through these challenges often extend beyond the racetrack, influencing broader applications in autonomous vehicle technology and related industries. Figure 1.1 and Figure 1.2 can serve as references for car design in these competitions.

Figure 1.1: Example of a Roborace car [1].

Figure 1.2: Example of an IAC car [2].

1.2 Motivation

The dissertation is part of the Autonomous Racing project, developed at LSCOE - Laboratory for Systems Control, Optimization and Estimation of SYSTEC-ARISE in DEEC - FEUP. This is the first year of this project and will lay the groundwork for future thesis proposals related to this topic.

The primary motivation behind autonomous racing is the advancement and refinement of artificial intelligence (AI) and machine learning algorithms in high-stakes scenarios. Central to this is the use of reinforcement learning (RL), a subset of machine learning where algorithms learn to make decisions by receiving feedback from their actions in the environment. In the context of autonomous racing, RL enables vehicles to improve their performance through continuous learning and adaptation.

Racing conditions demand sophisticated real-time data processing and adaptive algorithms, driving innovation and accelerating the development of robust, efficient, and safe autonomous systems. By tackling the complex challenges posed by high-speed racing, engineers and researchers can develop technologies that enhance the performance and reliability of autonomous vehicles in various contexts, from urban environments to long-haul transportation.

Reinforcement learning plays a critical role in this process by allowing autonomous racing cars to learn optimal driving strategies through trial and error. This involves simulating numerous racing scenarios where the autonomous system receives rewards for successful manoeuvres and penalties for mistakes. Over time, the system refines its strategies to maximise its performance on the track. This continuous learning approach is essential for handling the unpredictable and fast-paced nature of racing, where predefined rules and static programming fall short. The ability to adapt to new situations in real time makes RL a powerful tool in the advancement of autonomous driving technologies.

Furthermore, the competitive nature of autonomous racing events fosters a spirit of innovation and excellence. High-stakes competitions, such as those organised by the Roborace series and the Indy Autonomous Challenge, bring together top-tier research institutions, technology companies, and automotive manufacturers. This environment encourages collaboration and rivalry, which in turn drives rapid technological advancements. The intense competitive atmosphere pushes partic-

ipants to explore novel approaches, optimise their technologies, and strive for breakthroughs that could revolutionise the field of autonomous driving.

In addition to technological advancements, autonomous racing plays a crucial role in shaping public perception and acceptance of self-driving technologies. High-profile races featuring driverless cars can captivate public interest and demonstrate the reliability and capabilities of autonomous systems in a controlled, thrilling setting. These events provide a platform to showcase the potential of autonomous technology, helping to build trust and confidence among the general public. Successful demonstrations in the racing arena can alleviate concerns about safety and performance, paving the way for broader acceptance and integration of autonomous vehicles into everyday life.

Ultimately, autonomous racing is not merely about competition; it is a driving force for technological progress and societal acceptance of autonomous driving innovations. By addressing complex technical challenges, fostering innovation through competition, and shaping public perception, autonomous racing plays a pivotal role in the ongoing evolution of autonomous driving technology. This multifaceted impact underscores the importance of autonomous racing as a critical component of the future of transportation.

1.3 Contributions

This thesis makes several key contributions to the field of control strategies in autonomous racing. First, it compiles and synthesises existing literature on the subject, providing a comprehensive overview of the theoretical foundations essential for understanding the research. The development and implementation of a proportional-integral-derivative (PID) controller is detailed, demonstrating how it enables the vehicle to autonomously navigate the centre of the race course with optimised speed and precision.

Next, the thesis explores the Carrot Chase algorithm, offering a comparative analysis with the PID controller to highlight their respective advantages and limitations. This comparative study informs the decision-making process regarding controller selection.

Furthermore, the research introduces imitation learning techniques to replicate the PID controller's performance, presenting a novel approach to replacing the traditional PID controller. The strengths and weaknesses of both the PID and imitation learning methods are evaluated to provide a thorough understanding of their applicability.

Finally, the thesis implements reinforcement learning, specifically Proximal Policy Optimisation (PPO), to enhance the vehicle's autonomous navigation capabilities. This section emphasises real-time decision-making to balance speed and safety, showcasing the advanced application of PPO in autonomous racing scenarios.

1.4 Dissertation Structure

This dissertation is structured into 5 chapters in addition to the introduction.

Chapter 2 presents a general contextualisation of the system as a whole, followed by an examination of the components employed in this study. Subsequently, this chapter presents and synthesises the contributions and findings developed by the other project team members.

Chapter 3 establishes the foundation for the entire thesis, providing the necessary background knowledge to understand this dissertation. We first discuss the control methods used in previous works on autonomous vehicles. Then, we introduce the model that will be used throughout this thesis and explain the concept of Imitation Learning, along with its applications and variants. Following that, we delve into the concept of Reinforcement Learning, providing a practical example of an RL system and presenting its fundamental equations. Then, we explore the two types of models, discussing their advantages and disadvantages. Finally, we conclude the chapter by comparing and choosing the best method for this work from the available candidates.

In Chapter 4, the focus shifts to the practical application of theories in a first simulation environment. It describes the development of two implementations of a PID controller, one with a known environment and the other without. Furthermore, this chapter introduces the "carrot chase" algorithm, intending to explore the reactive controllers further. The adoption of imitation learning, which aims to replace the PID controller, represents a significant shift in methodology, trying to bridge the gap between the traditional controllers and those based on neural networks.

Chapter 5 progresses into a more complex simulation environment, reflecting a more realistic scenario. It outlines the modifications implemented to the PID controller, consequently of the updated environment dynamics. The results from this updated approach are discussed, providing valuable insights into its efficacy. Moreover, this chapter discusses the implementation of the reinforcement learning method PPO and its outcomes.

Finally, Chapter 6 concludes this dissertation by reflecting on the findings and contributions of the study and providing a conclusive summary, sparking curiosity by suggesting ways for future research to build upon the work.

Chapter 2

Existing Platform and Related Models

This chapter will provide a comprehensive contextualisation of the system as a whole, followed by an examination of the components employed in this study. The following section will present a detailed account of the assembly process employed in the construction of the track. Subsequently, the modifications made to the vehicle in order for it to communicate with the system will be discussed. We will then proceed to examine the rationale behind the selection of the camera and its specifications. After that, we will analyse and synthesise the contributions and findings of other project models that were developed by the other members of the project team. While that work is not the core research presented in this dissertation, an understanding of the efforts and outcomes is valuable for gaining a comprehensive understanding of the overall project. We first explore the contributions related to the perception aspect of the project and then turn our attention to the digital twin component.

2.1 General Contextualisation

The entire system is segmented into three distinct dissertations. The perception system employs a camera to identify all cars on the track, estimating their positions, orientations, and velocities through information gleaned from the camera. This data is then integrated using a Kalman filter for non-linear systems, specifically an Extended Kalman Filter (EKF). The updated information is subsequently transmitted via a UDP protocol. Each car is equipped with a microcontroller featuring an ESP32-S2FN4R2 WIFI IC.

The control system is responsible for computing the required control inputs for the cars, which are then transmitted via Bluetooth to the respective embedded cars. To enhance trajectory performance on the track, reinforcement learning methods are employed to facilitate learning and improvement.

The concluding segment of this project is the Digital Twin, serving as a tool for visually representing real-world systems. This tool enables developers to optimise their systems remotely, and concurrently, it can be directly linked to the physical system.

This project drew inspiration from the research conducted by the Automatic Control Laboratory at ETH Zurich [3]. Figure 2.1 provides an overview of the entire system, illustrating the interconnection of its components.

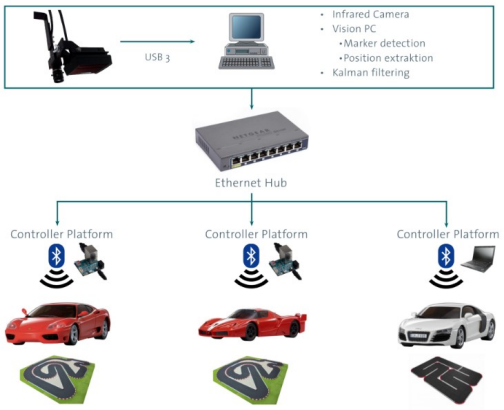

Figure 2.1: Overview of the system [3].

2.2 Components

2.2.1 Track

The track used for this project was a 2x2m RCP track. However, the order did not bring enough pieces to form the desired layout, so we had to improvise and modify the original design to fit the materials available. Our solution was to fill a square of 2×2m and then mark the boundaries with tape to create the desired trajectory. Although the result was not the most appealing visually, in terms of functionality, it worked as intended. This result can be seen in Figure 2.2.

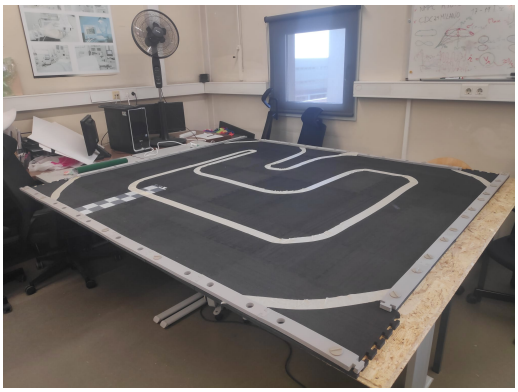

Figure 2.2: Track used for the tests.

2.2.2 Cars

The 1:43 scale Radio Controller (RC) cars utilised in this research, as shown in Figure 2.3, were originally equipped with a basic circuit board that only enabled manual control via a handheld controller. To enhance the autonomous capabilities of these vehicles, we replaced the original circuit board with two components: an ESP32 S2 Mini microcontroller and a custom-designed control board.

The ESP32 S2 Mini is a powerful, low-power microcontroller with integrated Wi-Fi and Bluetooth functionality that will serve as the vehicle's central processing unit. Additionally, the custom-designed control board will be responsible for managing power distribution to the car's motors and handling the steering control.

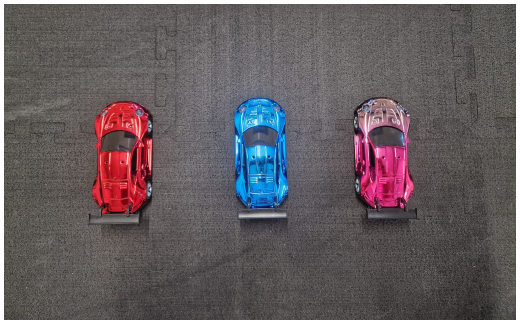

Figure 2.3: Cars used for the tests.

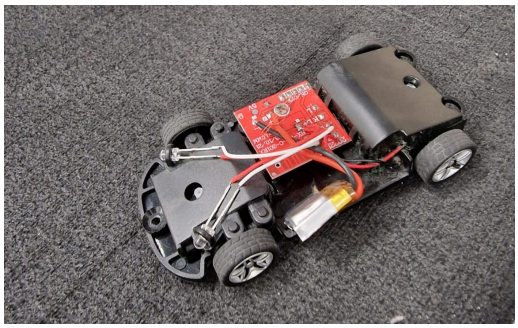

Figure 2.4: Original interior of the car.

2.2.2.1 ESP32 S2 Mini

The ESP32 S2 Mini microcontroller, shown in Figure 2.5, is the crucial component that enables the RC car to operate autonomously. Serving as the central processing unit, it handles sensor data processing and decision-making, generating the necessary control signals for the car's motors and steering mechanism.

The ESP32 S2 Mini is a robust, low-power microprocessor that offers a variety of features suitable for this application. It is equipped with a high-performance single-core Xtensa 32-bit LX7 microprocessor that can operate at up to 240 MHz, providing ample computational power to handle the complex algorithms and real-time decision-making required for autonomous racing. Additionally, the microcontroller includes extensive connectivity options, such as Wi-Fi and Bluetooth Low Energy (BLE) support, allowing it to communicate directly with the custom-designed control board connected to the car's motors and steering [4].

Finally, the ESP32 S2 Mini is designed with power efficiency in mind, making it an ideal choice for battery-powered applications like autonomous RC cars. Its low-power modes and advanced power management features help to extend the vehicle's life and ensure reliable operation.

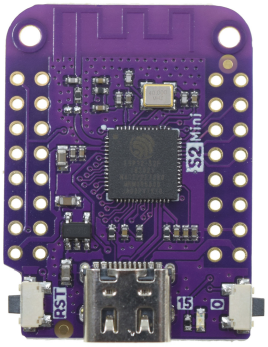

Figure 2.5: Microcontroller [4].

2.2.2.2 Steering and Motor Drivers

The customised printed circuit board (PCB) for the steering and motor drivers of a radio controller car, shown in Figure 2.6, incorporates several essential components to ensure efficient and reliable operation. We will proceed to explain the underlying principles of the key functional blocks, including the voltage regulator, motor control, and electromagnetic steering control. The schismatic of these blocks can be seen in Figure 2.7.

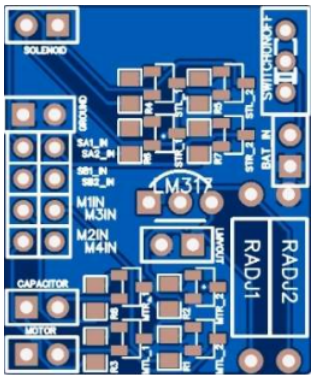

Figure 2.6: Steering and Motor Drivers PCB.

The circuit board is powered by a battery connected to the 'BAT_IN' terminal. A linear voltage regulator, the LM317, provides a stable and adjustable output voltage to power the subsequent motor and steering control circuits. The regulator's output voltage is set using two resistors, R_{adj1} and R_{adj2}, according to this formula:

$$V_{out} = V_{ref} \cdot \left(1 + \frac{R_{adj1}}{R_{adj2}}\right) + I_{adj} \cdot R_{adj1} \tag{2.1}$$

where V_{ref} is the reference voltage (typically 1.25V for LM317), and I_{adj} is the adjustment pin current. Capacitors are used at the input and output of the regulator to filter noise and ensure stability. This regulated voltage, called LMVOUT, supplies power to the motor and steering control sections.

The motor control circuit interfaces with four input signals, M1IN, M2IN, M3IN, and M4IN, which are processed through a network of resistors and a capacitor to provide smooth driving signals. These signals are then directed to the motor terminals, MTR_1 and MTR_2, to control the motor's operation, such as forward, reverse, and braking.

The steering mechanism is controlled by an electromagnetic solenoid actuator. The steering control circuit receives four input signals, SA1_IN, SA2_IN, SB1_IN, and SB2_IN, which are processed through a network of resistors and applied to the solenoid terminals, STR_1 and STR_2. The current induced in the solenoid generates a magnetic field that causes linear motion, which is linked to the steering mechanism, enabling directional control.

This PCB is connected to the ESP32 S2 Mini microcontroller, which generates the motor and steering control signals based on user commands received wirelessly, and transmits them to the PCB to drive the motor and control the steering.

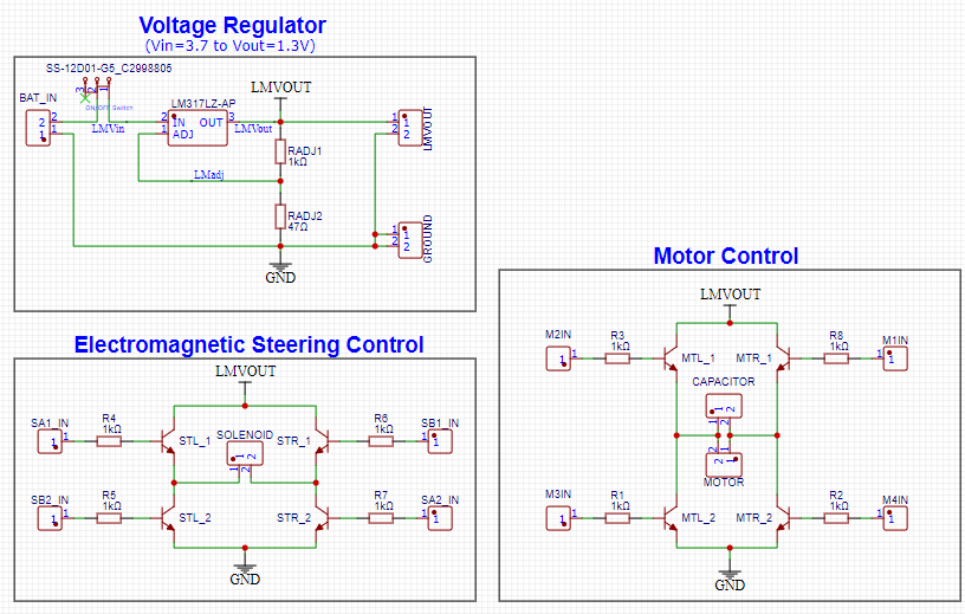

Figure 2.7: Schematic of the PCB circuit.

2.2.3 Choosing the camera

In order to reliably detect the moving car, we needed to invest in a high-quality camera that could capture clear, consistent footage even at high speeds. After carefully researching our options within our limited budget, we narrowed down the selection to four promising candidates, each with its own unique strengths and capabilities.

- **The Arducam 120fps 12MP UVC Camera Module** stood out for its impressive frame rates, making it well-suited for tracking fast-moving objects. With its ability to capture up to 120 frames per second, this camera could freeze the car's motion with clarity, allowing us to analyse its movements in detail.

- **The Drift Ghost XL Pro 4K** was an action camera known for its robust construction and stabilisation capabilities, making it an ideal choice for capturing smooth, steady footage of the car's movements. Its advanced stabilisation algorithms could help compensate for any camera shake or vibrations, providing us with high-quality video that would be easy to analyse.
- **The Walksnail Avatar HD Pro Kit 1080P 120fps** offered a balance of affordability and performance, with high frame rates that could freeze the car's motion without exceeding the budget limit. This option could be particularly appealing if our budget was tighter, as it would still provide us with the necessary capabilities to track the moving vehicle effectively.
- **The GoPro HERO10** offers cutting-edge image quality, advanced stabilisation, and a reputation for reliability in demanding environments. As a well-established brand in the action camera market, the GoPro HERO10 could provide us with a proven solution that consistently delivers high-quality footage, even in the most challenging conditions.

Given our limited budget and the need for a versatile camera that could be used in various projects, we carefully evaluated the pros and cons of each option. After thorough consideration, we concluded that the GoPro HERO10, presented in Figure 2.8, was the best choice due to its advanced features, reliability, and value within our budget.

Figure 2.8: GoPro Hero 10.

2.3 Perception

The initial stage of this process involves calibrating the selected camera in order to capture the track and the vehicle accurately. This is done in order to ensure the generation of precise measurements and reliable data for the subsequent analysis. The Pinhole Camera Model was employed in this calibration, which describes the relationship between the three-dimensional world and the two-dimensional image plane. This model allows for the estimation of the camera's intrinsic parameters, including the focal length, principal point, and distortion coefficients.

It was of the utmost importance to accurately calibrate the camera in order to eliminate all types of distortions, namely radial and tangential distortions, which could significantly impact the accuracy of the measurements obtained from the video footage.

After the calibration process was successfully completed, the system was fully equipped to move forward with its primary functions: capturing and identifying vehicles as well as monitoring their progression along the predefined track. The project's initial phase involved utilising ArUco markers, as the ones shown in Figure 2.9, which were strategically placed on the vehicles' rooftops. This method was chosen for its accuracy in facilitating the identification of individual vehicles amidst a moving traffic flow. The ArUco markers, known for their distinctive patterns and ease of detection by computer vision algorithms, seemed like an ideal choice at the outset.

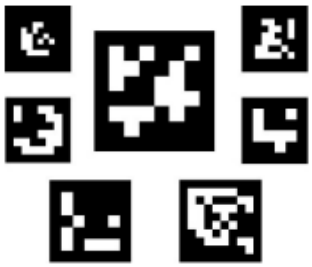

Figure 2.9: ArUco markers.

However, as the project progressed, it became apparent that relying solely on ArUco markers presented significant drawbacks. The primary issue was the substantial amount of processing power required to detect and decode the markers in real time. This processing bottleneck resulted in slower system responses, impacting the overall efficiency of the vehicle tracking process. Recognising the need for a more efficient method, the team embarked on the development of an alternative strategy.

The new approach shifted away from the reliance on ArUco markers towards the implementation of a sophisticated colour filtering technique. By applying a colour filter, the system could effectively isolate the vehicles from their background environment, as seen in Figures 2.10, 2.11 and 2.12, thus simplifying the detection process. Once the vehicles were isolated, tracking their movement across the camera frame became significantly more straightforward. This method not only reduced the processing time but also maintained a high level of accuracy in vehicle identification and tracking.

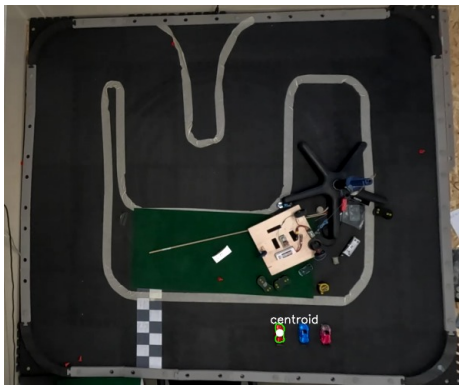

Figure 2.10: Red car detected.

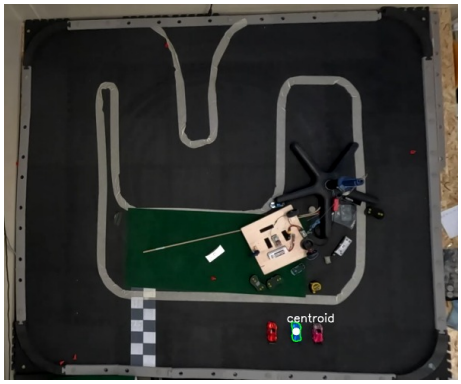

Figure 2.11: Blue car detected.

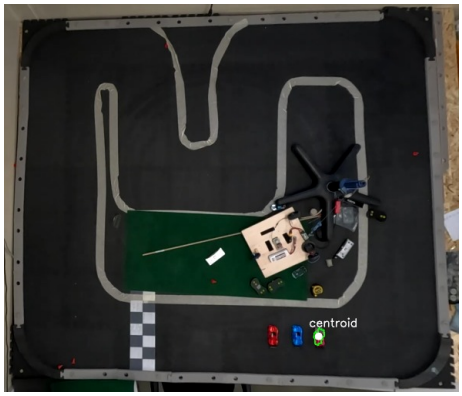

Figure 2.12: Pink car detected.

In order to accurately estimate the position, velocity, and orientation of vehicles within a dynamic environment, an Extended Kalman Filter (EKF) was formulated and applied. Traditional methods, such as the linear Kalman Filter, fall short of capturing the complexity of vehicle dynamics due to the inherent non-linear characteristics of these systems. These dynamics involve factors like changes in speed, direction, and orientation that do not follow linear patterns, especially under varying conditions.

The EKF emerges as a sophisticated solution to this challenge. It extends the capabilities of the standard Kalman Filter by incorporating a linear approximation approach to manage the non-linear aspects of vehicle dynamics. Through a mathematical process involving the Jacobian matrix, the EKF is able to predict iteratively and correct estimates of a vehicle's state, including its position, velocity, and orientation. This is a crucial advancement for technologies such as autonomous driving systems, aerial drones, and other applications requiring high precision in tracking and navigation.

2.4 Digital Twin

The work started with the creation of the physical model of the track and the vehicle using Blender, an advanced 3D modelling software. This initial step was crucial in achieving a high degree of accuracy in representing the components that exist in the real world, facilitating their seamless digital replication within the simulation environment. The result of this creation can be seen below in Figures 2.13 and 2.14.

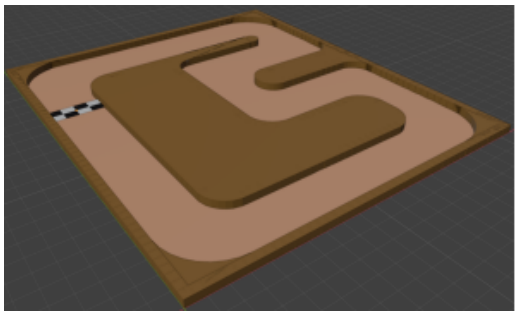

Figure 2.13: Finished physical track model.

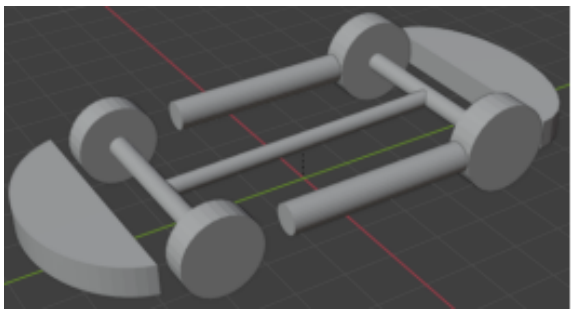

Figure 2.14: Finished physical car model.

Following the successful creation of these models, the next phase involved their integration into a sophisticated simulation framework. Unity was selected for this purpose, given its capability to simulate the dynamics and physics of the real-world system with remarkable accuracy. This choice was instrumental in developing the digital twin, providing a virtual counterpart to the physical system that could replicate its behaviour under various conditions.

Once the models were imported to Unity, the focus shifted to enabling physical interactions between the vehicle and its environment as it navigated the track. This required the implementation of rigid body dynamics and colliders, essential components that govern how objects interact physically within the simulation.

In order to control this vehicle within the simulation, a script was developed capable of translating user inputs, such as steering, acceleration, and braking, into the corresponding dynamics of the vehicle. This level of control was vital for conducting simulations that accurately reflected real-world driving conditions.

Two view cameras were strategically implemented to enhance the simulation experience. The first offered a top view of the track, providing a comprehensive overview of the car's progress, while the second camera, positioned behind the car, offered a more immersive experience. This dual-camera setup was useful to visualise the car's performance from different perspectives, enchanting the simulation experience. This two perspectives are shown in Figures 2.15 and 2.16.

Figure 2.15: Top view camera.

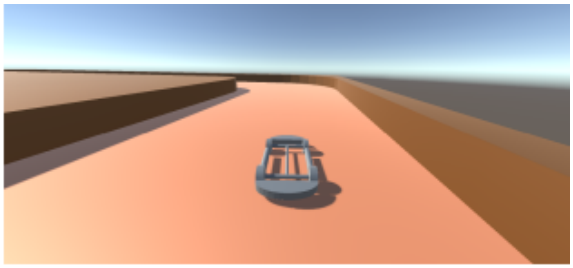

Figure 2.16: Race view camera.

In addition to the visual components, a UDP communication protocol was implemented to continuously transmit data towards the simulation to monitor the vehicle, allowing for real-time monitoring and analysis of the vehicle's state in the physical environment.

Chapter 3

Background on Autonomous Racing and Reinforcement Learning

3.1 Introduction

In this chapter, we begin by presenting the work that was developed in the field of autonomous racing. Following that, we will present the vehicle model and introduce the concept of imitation learning. Then, we will delve into the fundamental principles of reinforcement learning to provide a better understanding of the dissertation's work. On this second topic and firstly, we will give a brief overview of what reinforcement learning is. Secondly, we will explain how an RL system behaves and its constituents using a practical example. Finally, we will discuss the differences between Model-free and Model-based methods and elaborate on the method that was chosen for this work.

3.2 Control Methods in Autonomous Racing

Many different types of controls have been developed through the years. The survey "Autonomous Vehicles on the Edge" [8] provides a comprehensive overview of the latest advancements in autonomous racing, covering both software and hardware developments. Here, we present some of the papers referred to in that survey, summarising the work developed.

- Cai *et al.* [9] presented a novel approach to vision-based autonomous car racing through the application of deep imitative reinforcement learning. The study's objective is to leverage deep learning techniques to enable an autonomous vehicle to learn complex racing manoeuvres by imitating expert demonstrations. The author's objective is to equip the vehicle with the capacity to interpret and respond to its environment in real time through the utilisation of a vision-based system. This approach not only enhances the vehicle's decision-making capabilities but also improves its overall performance in racing scenarios. The integration of

deep imitative reinforcement learning demonstrates significant potential for advancing autonomous racing technologies, particularly in terms of handling dynamic and unpredictable environments.

- Perot *et al.* [10] investigated the potential of deep reinforcement learning for end-to-end driving in a realistic racing game. The authors highlight the importance of training an autonomous agent to control a vehicle directly from raw sensory inputs, eliminating the need for predefined rules or manual feature engineering. By utilising deep reinforcement learning, the agent is able to learn to make driving decisions through a process of trial and error, thereby optimising its performance over time. The study demonstrates the efficacy of this methodology in navigating the intricacies of a racing environment, exemplifying the potential of deep reinforcement learning to augment the autonomy and adaptability of racing game AI. This research contributes to the broader field of autonomous driving by demonstrating the feasibility of end-to-end learning systems in highly dynamic and competitive settings.

- Remonda *et al.* [11] introduced a framework termed "Formula RL," which employs deep reinforcement learning for autonomous racing, leveraging telemetry data. This paper outlines the implementation of a deep learning model trained on extensive telemetry data with the objective of enhancing the performance and decision-making capabilities of autonomous race cars. By analysing the telemetry data, the system is able to learn and subsequently optimise driving strategies, adapt to varying track conditions and improve overall race efficiency. The authors present evidence that deep reinforcement learning, when applied to telemetry data, markedly enhances the capacity of autonomous vehicles to perform effectively in competitive racing environments. This approach demonstrates the potential for synergy between advanced data analytics and reinforcement learning in advancing the capabilities of autonomous racing technology.

- Pan *et al.* [12] investigated the potential of agile autonomous driving through the application of end-to-end deep imitation learning. The objective of this research is to train an autonomous vehicle to perform high-speed manoeuvres by learning directly from expert demonstrations. The system employs deep neural networks to process raw sensory inputs, thereby enabling the generation of control commands that facilitate agile and responsive driving behaviour. The study demonstrates the efficacy of deep imitation learning in replicating sophisticated driving strategies and attaining superior performance in dynamic driving scenarios. By focusing on agility and adaptability, the authors demonstrate significant advancements in autonomous driving capabilities, particularly for applications requiring rapid decision-making and precision. This work makes a contribution to the field by demonstrating the potential of end-to-end learning approaches to enhance the agility and overall performance of autonomous vehicles.

- Chisari [13] investigated the use of simulation-to-reality (Sim2Real) techniques in the field of autonomous racing. The focus is on using simulated environments to train and improve autonomous racing algorithms before deploying them in the real world. The research team explores different methods to bridge the gap between simulation and reality, including domain adaptation and transfer learning. By leveraging simulated data and environments, the paper aims to enhance the efficiency and reliability of autonomous racing systems when transitioning from virtual to real-world scenarios. Ultimately, this contributes to the development of autonomous driving technology within the racing industry.

3.3 Model of the vehicle

There are various models that describe vehicle dynamics, ranging from simple representations to detailed simulations. At the most basic level, there's the point of mass model [14], which treats the vehicle as a single point of mass and allows for straightforward analysis of longitudinal and lateral movements by considering only external forces. Another basic model is the particle model with longitudinal dynamics [15], an extension of the point mass model that includes specific forces such as engine thrust and resistive forces, providing insights into acceleration and deceleration behaviours.

Moving up to more complex models, we have the single-track (bicycle) model [5], which represents the vehicle with two wheels and incorporates yaw dynamics, making it suitable for studying basic handling and stability. The double-track model [16] goes further by including all four wheels and accounting for lateral load transfer and tyre forces for more realistic handling dynamics. At the highest level of detail, the full car model [17] incorporates six degrees of freedom, including longitudinal, lateral, and vertical motions, as well as roll, pitch, and yaw, providing an in-depth simulation of overall vehicle dynamics.

For this dissertation work, the single-track (bicycle) model and the double-track model were two plausible choices that provide a good amount of fidelity without being too complex, which would have slowed down the system processing. Even though the double-track model offered a more realistic representation of the car, it was decided to favour the processing time. In conclusion, it was decided to use the single-track (bicycle) model.

3.3.1 Single-track (bicycle) model

In this thesis, we adopt the model for the steering dynamics [5] shown in figure 3.1 below. In the left image, we can see a vehicle from an overhead view. It has four wheels, where the wheelbase is b. The centre of mass is located a distance forward of the rear wheels. By assuming the motion of the front and rear pairs of wheels to be a single front wheel and a single rear wheel, we get an abstraction known as the bicycle model, shown on the right. The velocity at the centre of the mass has an angle α that represents the slip sideways relative to the length axis of the vehicle, and the steering angle is δ. The position of the vehicle is given by (x, y) and the orientation (heading) by θ.

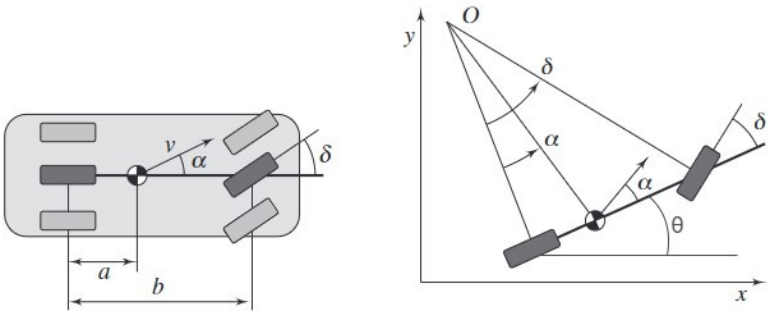

Figure 3.1: Model of the vehicle [5].

The vehicle model involves certain geometric parameters, namely, the steering angle δ, the velocity v, and distances a and b related to the turning centre O (see Figure 3.1). The parameters a and b are defined as $b = r_a \tan(\delta)$ and $a = r_a \tan(\alpha)$, where r_a is the distance between the centre of the turn and the rear axle.

The relation between the slip sideways angle (α) and the steering angle (δ) is given by:

$$\tan(\alpha) = \frac{a}{b} \cdot \tan(\delta) \tag{3.1}$$

This equation establishes a connection between how the front wheel angle and the steering angle are related during a turn.

The equation

$$\alpha(\delta) = \arctan\left(\frac{a \cdot \tan(\delta)}{b}\right) \tag{3.2}$$

provides a direct way to calculate the front wheel angle (α) based on the steering angle (δ).

Assuming no wheel slippage, the equations describing the motion of the vehicle in the x and y directions are given by:

$$\dot{x} = \frac{dx}{dt} = v \cdot \cos(\alpha + \theta) \tag{3.3}$$

$$\dot{y} = \frac{dy}{dt} = v \cdot \sin(\alpha + \theta) \tag{3.4}$$

These equations express how the position of the vehicle (x and y) changes over time as a function of the velocity (v) and the orientation angle (θ).

The angular velocity of the vehicle ($\dot{\theta}$) is given by:

$$\dot{\theta} = \frac{d\theta}{dt} = \frac{v}{r_a} = \frac{v}{b} \cdot \tan(\delta) \tag{3.5}$$

This equation expresses how the angular velocity of the vehicle ($\dot{\theta}$) is influenced by the velocity (v) and the steering angle (δ).

3.3.2 Simplified Model of the vehicle

Now, considering the simplification that the centre of the mass is in the rear axle as shown in Figure 3.2 below:

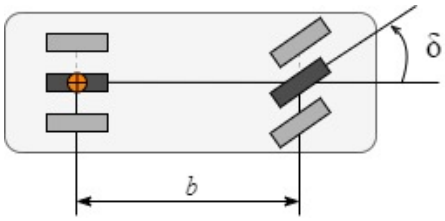

Figure 3.2: Model of the vehicle simplified.

and that the rear wheels do not slip sideways, then the angle α is equal to zero and the model simplifies to

$$\dot{x} = v \cdot \cos(\theta) \tag{3.6}$$

$$\dot{y} = v \cdot \sin(\theta) \tag{3.7}$$

$$\dot{\theta} = \frac{v}{b} \cdot \tan(\delta) \tag{3.8}$$

3.4 Imitation learning

Instead of using the previously proposed model, given by the equations 3.2-3.5, it is possible to adopt Imitation Learning, also known as Learning from Demonstrations. This is a machine learning technique where an agent learns a task by observing and imitating the actions of an expert, which can be a human or a piece of software itself. The aim of the model is to replicate the expert's behaviour in a given environment to achieve the desired outcome. Imitation learning is particularly useful in scenarios where it is difficult to specify a reward function or define the optimal policy directly.

Imitation Learning not only has the potential to replace complex models but can also serve as an alternative to traditional controllers, as shown in [18]. Traditional controllers often require extensive tuning and domain-specific knowledge to design control laws or rules that govern system behaviour. In contrast, imitation learning enables the system to learn these behaviours directly from demonstrations, reducing the need for manual design and tuning.

3.4.1 Application of Imitation leaning

Imitation learning has many practical applications in different domains, such as robotics [19], [20], autonomous vehicles [21], and game playing [22], [23].

It allows us to benefit from demonstrations to train models or controllers, which is particularly useful in situations where defining the best policy or reward function is difficult or impractical.

However, imitation learning can pose some challenges, such as dealing with suboptimal demonstrations and handling changes in the environment between the training and deployment phases. Researchers are actively working on developing techniques and improvements to overcome these challenges and improve the effectiveness of imitation learning algorithms.

3.4.2 Variations and extensions of Imitation learning

In addition to the usual Imitation learning process, there are variations and extensions of IL that incorporate additional methods and techniques. Some of these variations are the Inverse Reinforcement Learning (IRL), the Adversarial Imitation Learning (AIL), the Interactive Imitation Learning (IIL) and the Multi-modal Imitation Learning.

Inverse Reinforcement Learning involves the learning agent inferring the underlying reward or cost function from observed expert behaviour. The agent does not directly imitate actions, but instead seeks to understand the implicit preferences or objectives of the expert and optimise its own behaviour accordingly.

Adversarial Imitation Learning combines imitation learning with adversarial training. The learning agent competes with the discriminator, which differentiates between expert and agent-created trajectories. The agent aims to generate trajectories that are similar to those of the expert.

Interactive Imitation Learning is used in interactive scenarios, the learning agent may have the opportunity to query the expert for additional information or clarification during the learning process. This form of IL allows the agent to actively seek guidance from the expert.

In some cases, the learning agent may observe not only the expert's actions but also other modalities such as images, demonstrations, or language. Multi-modal Imitation Learning involves learning from a combination of these modalities to improve imitation accuracy.

3.5 Reinforcement learning

3.5.1 What is Reinforcement learning

Reinforcement learning refers to the process of learning how to make decisions, i.e., how to map situations to actions, in order to maximise a numerical reward signal. Unlike other forms of learning, the learner is not provided with any pre-defined set of actions to take. Instead, the learner must discover which actions lead to the most reward by trying them out. In the most interesting and challenging cases, the actions taken may have an impact not just on the immediate reward, but also on the subsequent rewards that follow. Trial-and-error search and delayed reward are two of the most important distinguishing features of reinforcement learning [24].

3.5.2 System of Reinforcement learning

In the typical configuration of a Reinforcement Learning (RL) system, it is found a network of interrelated components, with each element fulfilling a vital role in the intricate processes of learning and decision-making. Together, these components synergise to empower the system's capacity to

adjust and refine its behaviour within a given environment. Fundamental constituents of an RL system encompass:

- **Agent:** The agent is the entity responsible for interacting with the environment. It makes decisions by executing actions, ultimately impacting the state of the environment.
- **Environment:** The environment represents the external system with which the agent interacts. It can encompass physical spaces, simulations, or any system the agent aims to understand or control.
- **State (S):** The state encapsulates the current situation or configuration of the environment. It serves as a comprehensive representation containing all pertinent information necessary for the agent to make decisions. States can be discrete or continuous.
- **Action (A):** Actions are the choices or moves available to the agent within a given state. The collective set of possible actions constitutes the action space.
- **Policy (π):** The policy outlines the strategy or mapping from states to actions, dictating the agent's behaviour. It defines the probabilities associated with taking specific actions in a given state.
- **Reward (R):** At each time step, the agent receives a numerical reward or penalty from the environment based on the action taken and the resulting state. The agent's objective is to learn a policy that maximises the cumulative reward over time.
- **Value Function (V or Q):** The value function assesses the expected cumulative reward achievable from a specific state or state-action pair. The state value function (V) gauges the expected cumulative reward from a particular state, while the action value function (Q) estimates the expected cumulative reward of taking a specific action in a given state.
- **Exploration and Exploitation**: Balancing exploration and exploitation is crucial in reinforcement learning. Exploration involves trying out new actions to discover their effects, while exploitation entails selecting actions known to yield high rewards based on current knowledge.
- **Learning Algorithm:** The learning algorithm comprises a set of rules and procedures guiding the agent in updating its policy, value functions, or internal representations based on observed rewards and experiences. Common algorithms include Q-learning, Deep Q Network (DQN), Policy Gradient methods, among others.

In a Reinforcement Learning (RL) system, consider the Agent as the decision-making core, similar to the brain, which directs the next actions. When the Agent executes a specific action (denoted as A), it sets in motion a cascade of effects within the system. This action introduces new values into the system's inputs, thereby influencing the ongoing dynamics. These altered inputs interact with the environment, giving rise to a transformation in the system's state (S) and

an associated reward (R). The nature of this reward is contingent upon the overarching objective of the system, serving as an evaluative metric for the success or desirability of the undertaken action.

Subsequent to this interaction, the Agent undertakes a cognitive appraisal of the newly attained state and reward. Leveraging this assessment, the Agent strategically formulates plans for the subsequent action to be executed. This iterative cycle persists until the ultimate task or objective is satisfactorily achieved.

The intricate interplay between the Agent, actions, environment, states, and rewards is visually encapsulated in Figure 3.3, illustrating the cyclical and adaptive nature of the RL learning process.

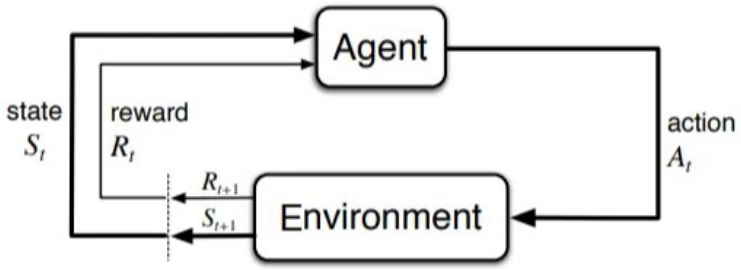

Figure 3.3: System of Reinforcement learning [6].

3.5.3 Practical example

An example to be more easily understood is the autonomous racing car. In this example, we consider an autonomous race car, that makes a manoeuvre (action). This manoeuvre will change the car's position on the track (environment). Then it will generate a new position and speed of the car (new state), and based on the consequences of the manoeuvre, the system will evaluate if the move was good or bad (reward). After that, the system will plan the next manoeuvre and repeat the process until the race is over.

Although this can seem simple, there is a lot of complexity behind a single manoeuvre. The manoeuvres are not independent of each other, which means that a manoeuvre that may seem optimal in the short term may not be the best for the long term. Therefore, the system must consider the potential consequences of each manoeuvre and make a strategic decision accordingly. This is one of the problems of the more challenging systems in reinforcement learning.

3.5.4 Fundamental equations

Now that we have established a foundational understanding of reinforcement learning, we delve deeper into its essential equations. We shift our focus from broad principles to the mathematical

bedrock and explore the intricate mechanisms that drive this learning system. Reinforcement learning relies on sophisticated algorithms that meticulously assign rewards and iteratively refine strategies over time. These equations, including reward functions, value functions, and policy optimisation, are the core of the learning process. They unlock the ability of agents to learn, adapt, and make intelligent decisions in dynamic environments.

Equation 3.9 captures the very essence of total return (G_t) in the reinforcement learning framework. It signifies the cumulative sum of future rewards from time $t+1$ until the end of an episode at time T, embodying the overarching goal of agents to maximize accumulated rewards.

$$\begin{aligned} G_t &= R_{t+1} + R_{t+2} + R_{t+3} + \ldots + R_T \\ G_t &= \sum_{k=0}^{T} R_{t+k+1} \end{aligned} \tag{3.9}$$

Moving forward, Equation 3.10 introduces a discount factor γ, reflecting the agent's inclination towards immediate rewards while acknowledging the significance of long-term consequences. Each subsequent reward is meticulously scaled by γ^k, acknowledging the diminishing impact of rewards over time.

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \ldots \\ G_t &= \sum_{k=0}^{T} \gamma^k R_{t+k+1} \end{aligned} \tag{3.10}$$

Equation 3.11 establishes a recursive relationship, portraying the total return at time t as the sum of the immediate reward (R_{t+1}) and the discounted total return at the next time step (γG_{t+1}). This recursive nature elegantly captures the evolving dynamics of the reinforcement learning process.

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \ldots \\ G_t &= R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \ldots) \\ G_t &= R_{t+1} + \gamma G_{t+1} \end{aligned} \tag{3.11}$$

Now, turning our attention to the value functions, Equation 3.12 introduces the state-value function $v_\pi(s)$. This function encapsulates the agent's anticipation of cumulative rewards, considering the infinite sum of discounted future rewards and encompassing all possible state and reward sequences.

$$v_\pi(s) = E_\pi[G_t | S_t = s] = E_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s \right] \tag{3.12}$$

Expanding upon this, Equation 3.13 extends the concept to action-value functions ($q_\pi(s,a)$). These functions represent the expected total return from state s when taking action a under policy π, capturing the intricate interplay between states, actions, and policy in shaping the expected cumulative rewards.

$$q_\pi(s,a) = E_\pi[G_t|S_t = s, A_t = a] = E_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s, A_t = a\right] \tag{3.13}$$

Now, we will explore the optimal scenarios. Equation 3.14 introduces the optimal state-value function $v_*(s)$, signifying the maximum expected total return from state s over all possible policies π. It serves as a benchmark for the best achievable performance from a given state.

$$v_*(s) = \max_\pi v_\pi(s) \tag{3.14}$$

Equation 3.15 defines the optimal action-value function $q_*(s,a)$, capturing the maximum expected total return from state s when taking action a over all possible policies π. It provides profound insight into the best achievable performance for a specific state-action pair.

$$q_*(s,a) = \max_\pi q_\pi(s,a) \tag{3.15}$$

Lastly, Equation 3.16 expresses the optimal action-value function in terms of the immediate reward (R_{t+1}) and the discounted value of the best next state ($v_*(S_{t+1})$). It illustrates that the best action-value function involves a delicate balance between the immediate reward and the expected value of the optimal subsequent state.

$$q(s,a) = E[R_{t+1} + \gamma v(S_{t+1})|S_t = s, A_t = a] \tag{3.16}$$

3.6 Model-free vs Model-based

Throughout the years, numerous algorithms have been developed to address the diverse problems in RL. These algorithms can be broadly classified into two groups:

- Model-Free
- Model-Based

The distinction between Model-free and Model-based reinforcement learning lies in their respective approaches to handling the environment. The Model-free methodology involves learning directly from interactions with the environment, bypassing the creation of an explicit model. Essentially, Model-free agents rely exclusively on trial-and-error learning. These methods focus on teaching either a policy, defining actions to take in a given state, or a value function, estimating the expected cumulative reward from a specific state.

In contrast, Model-based approaches aim to construct a detailed model of the environment. This explicit model becomes a valuable tool for planning and simulating potential future states

and outcomes. Here, the agent dedicates effort to understanding the dynamics of the environment, meaning it learns transition probabilities and rewards. This acquired knowledge becomes crucial for informed decision-making [24].

The selection between Model-free and Model-based methods depends on the specific problem at hand and the available resources. Each approach carries its own set of strengths and weaknesses, as outlined in the tables 3.1 and 3.2 below. These tables, based on these papers [24], [25], [26] and [27], provides a comprehensive overview of the advantages and disadvantages associated with each methodology, aiding researchers and practitioners in making informed choices based on the unique requirements of their reinforcement learning applications.

Table 3.1: Advantages and Disadvantages of the Model-Free algorithms.

Model-Free	
Advantages	**Disadvantages**
Simplicity and Flexibility: There is no need to create or maintain a specific model of the environment.	Exploration Challenges: In situations where exploration is difficult or risky, these algorithms may struggle to discover optimal or near-optimal policies.
Computational Efficiency: When the environment is intricate and their representation is difficult or requires a lot of resources.	Lack of Exploitation of Known Dynamics: In cases where the environment has patterns that can be used for efficient decision-making, model-free methods may miss potential benefits
Better Handling of Uncertainty: When the environment is uncertain or stochastic model-free can be more robust.	Limited Transferability: Model-free algorithms may struggle to transfer knowledge between tasks with differing dynamics, often requiring extensive retraining in new environments.
Suitability for Continuous Action Spaces: Deep neural networks are effective for handling continuous action spaces, making them well-suited for problems where actions lie in a continuous range.	Noisy Learning: Learning in a model-free manner from raw experiences can introduce variability and noise, potentially leading to suboptimal policies or slower convergence in certain scenarios.
Generalization: Function approximation methods, such as neural networks, display excellent generalization capabilities. They can adapt their learned value functions or policies to unseen states, resulting in more adaptive behavior in various situations.	Limited Long-Term Planning: Model-free algorithms often prioritize short-term rewards, potentially hindering performance in scenarios that require long-term planning or delayed rewards.
Empirical Success: Model-free algorithms have demonstrated success in real-world applications like playing complex games (e.g. AlphaGo, Atari games) and robotic control tasks by learning complex policies directly from raw sensory inputs.	Difficulty in Handling Partial Observability: In situations of partial observability, agents may struggle to make informed decisions without a complete understanding of the environment.

Table 3.2: Advantages and Disadvantages of the Model-Based algorithms.

Model-Free	
Advantages	**Disadvantages**
Simplicity and Flexibility: There is no need to create or maintain a specific model of the environment.	Exploration Challenges: In situations where exploration is difficult or risky, these algorithms may struggle to discover optimal or near-optimal policies.
Computational Efficiency: When the environment is intricate and their representation is difficult or requires a lot of resources.	Lack of Exploitation of Known Dynamics: In cases where the environment has patterns that can be used for efficient decision-making, model-free methods may miss potential benefits
Better Handling of Uncertainty: When the environment is uncertain or stochastic model-free can be more robust.	Limited Transferability: Model-free algorithms may struggle to transfer knowledge between tasks with differing dynamics, often requiring extensive retraining in new environments.
Suitability for Continuous Action Spaces: Deep neural networks are effective for handling continuous action spaces, making them well-suited for problems where actions lie in a continuous range.	Noisy Learning: Learning in a model-free manner from raw experiences can introduce variability and noise, potentially leading to suboptimal policies or slower convergence in certain scenarios.
Generalization: Function approximation methods, such as neural networks, display excellent generalization capabilities. They can adapt their learned value functions or policies to unseen states, resulting in more adaptive behavior in various situations.	Limited Long-Term Planning: Model-free algorithms often prioritize short-term rewards, potentially hindering performance in scenarios that require long-term planning or delayed rewards.
Empirical Success: Model-free algorithms have demonstrated success in real-world applications like playing complex games (e.g. AlphaGo, Atari games) and robotic control tasks by learning complex policies directly from raw sensory inputs.	Difficulty in Handling Partial Observability: In situations of partial observability, agents may struggle to make informed decisions without a complete understanding of the environment.

Moreover, delving into the realm of machine learning algorithms, we can intricately categorise the two overarching families into four distinctive sub-families, each encompassing specific methodologies tailored to optimise autonomous vehicle control. Within the model-free category, two prominent groups of methods take the spotlight: Policy Optimisation and Q-learning. On the contrasting side, the model-based category introduces two additional groups of methods, namely Learn the Model and Given the Model.

In the expansive landscape of Policy Optimisation, we encounter four sophisticated algorithms, each designed to refine and enhance the learning process. These algorithms include Policy Gradient, A2C/A3C (Actor-Critic), PPO (Proximal Policy Optimisation), and TRPO (Trust Region Policy Optimisation). Each of these methods employs distinct strategies to iteratively improve the policy governing the autonomous vehicle's actions, fostering adaptability and efficiency.

Meanwhile, the Q-learning category unfolds with its own set of four algorithms, each contributing to the augmentation of decision-making processes. These algorithms comprise DQN (Deep Q Network), C51, QR-DQN (Quantile Regression Deep Q Network), and HER (Hindsight Experience Replay). Through the utilisation of intricate reinforcement learning mechanisms, these algorithms aim to optimise the quality of decisions made by the autonomous vehicle, ultimately improving its overall racing performance.

Nestled between the Policy Optimisation and Q-learning categories, three additional algorithms emerge, each offering a unique approach to reinforcement learning. These include DDPG (Deep Deterministic Policy Gradient), TD3 (Twin Delayed Deep Deterministic), and SAC (Soft

Actor-Critic). These methods bridge the gap between policy-driven and value-driven learning, providing a nuanced and comprehensive framework for autonomous vehicle control.

Within the Learn the Model category, four algorithms stand out as pioneers in leveraging a model-based approach to enhance decision-making. These algorithms are Word Models, I2A (Imagination-Augmented Agents), MBMF (Magnitude Bounded Matrix Factorisation), and MBVE (Model-Based Value Expansion).

Finally, the Given the Model category is represented by a singular but formidable algorithm, AlphaZero.

By incorporating various forms of model-based learning, these algorithms seek to build an accurate representation of the environment, enabling the autonomous vehicle to make informed decisions based on acquired knowledge and predictions. This category presents a diverse array of methodologies, each with its strengths in adapting to different challenges posed by racing scenarios.

The figure 3.4 below represents these different methods organised in a taxonomy.

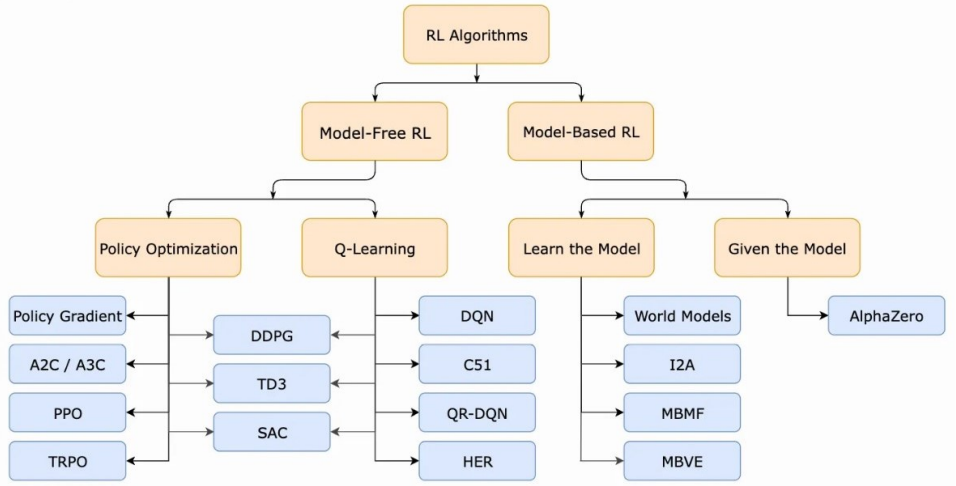

Figure 3.4: Taxonomy of RL algorithms [7].

3.7 Choosing the best method

Based on the methods discussed in section 3.6, there are four algorithms particularly well-suited to this task - Proximal Policy Optimisation (PPO), Trust Region Policy Optimisation (TRPO), Deep Deterministic Policy Gradient (DDPG), and Deep Q Network (DQN). These methods possess characteristics that are well adapted to the problem at hand.

To select the most suitable approach for this challenging and particular scenario, we will conduct an in-depth comparison of the individual strengths and considerations of each method.

3.7.1 Proximal Policy Optimisation (PPO)

Proximal Policy Optimisation (PPO) is a powerful reinforcement learning algorithm specifically designed to meet the challenges posed by environments with complex state and action spaces. At its core, PPO seeks to maximise the expected cumulative reward over time, encapsulated by the objective function $G(\theta)$. In this function, π represents the policy parameters, γ is the discount factor, and R_t is the reward at time t, as seen earlier in the equation 3.10.

PPO introduces an advanced replacement objective function, $L(\pi)$, to ensure stable and efficient learning. This surrogate function incorporates a clipping mechanism that prevents the policy from being updated in a drastic way that could disrupt the learning process. The clipped surrogate objective, with its careful balance between exploration and exploitation, is instrumental in maintaining stability during training.

The crux of the surrogate objective lies in the ratio $R_t(\pi)$, which compares the new policy probability with the old policy probability for a chosen action. The utility function A_t comes into play, providing insight into how each action performs relative to the average action. The surrogate objective is then calculated as a form of the clipped surrogate function, effectively constraining the size of the policy updates.

When implemented, PPO iteratively orchestrates a series of steps. Trajectories are collected by interacting with the environment. Benefits are computed to measure action performance, and the surrogate goal is determined. The optimisation step uses stochastic gradient ascent, a method for updating policy parameters, while respecting the trust region - a key aspect of PPO's stability.

Maintaining a trust region involves limiting the size of policy updates, a precaution against excessive deviations from the existing policy. This trust region, combined with the clipping mechanism, ensures that the learning process maintains a delicate balance and avoids abrupt changes that could impede progress.

The general workflow of PPO involves collecting samples, estimating benefits, computing the clipped surrogate objective, and iteratively updating the policy parameters. Hyperparameters, such as the clipping parameter , learning rate, and number of iterations, require careful tuning to optimise PPO's performance in different environments.

In summary, Proximal Policy Optimisation emerges as a robust and sophisticated reinforcement learning algorithm, providing a nuanced approach to policy optimisation that gracefully navigates the challenges posed by complex environments. Its clipped surrogate objective and attention to the trust region contribute significantly to its stability and effectiveness in diverse learning scenarios. More detailed information about these benefits can be found in [24] and [28].

3.7.2 Trust Region Policy Optimisation (TRPO)

Trust Region Policy Optimization (TRPO) is a reinforcement learning algorithm designed to refine an agent's policy in a way that ensures both stability and efficiency. Conceived by John Schulman

and his colleagues in 2015 [29], TRPO introduces the concept of a trust region to regulate the extent of policy updates, preventing radical departures from the existing policy while still promoting notable performance improvements.

In TRPO, the iterative process of policy improvement is fundamental. Starting with an initial policy, the algorithm continually refines it, seeking to maximise the expected cumulative reward, or in simpler terms, the expected return. The key lies in the trust region, a predefined area in the policy space that acts as a safety zone. Policy updates are then carefully confined within this trust region to avoid significant performance degradation.

The operational steps of TRPO start with sampling the current policy in the environment. Next, benefit estimates are calculated, which indicate the relative benefit of a particular policy in a given state compared to the average policy. A surrogate objective is then created that measures the improvement of the new policy over the old, based on the benefit estimates and action probability ratios. The policy update is determined by maximising this surrogate objective, subject to a constraint on the size of the update.

To ensure that the policy update remains within the trust region, TRPO uses a backtracking line search. This iterative process adjusts the step size until the policy update is deemed to comply with the trust region constraints. The final step involves applying the computed policy update to, well, update the policy.

The trust region constraint is a crucial aspect of the stability of the TRPO. Typically implemented through a Kullback-Leibler divergence constraint, it measures the difference between the new and old policies. By limiting this divergence to a small value, TRPO ensures that policy updates remain within acceptable limits.

While TRPO offers stability advantages over simple policy gradient methods, it is not without its challenges. The constrained optimisation problem it involves can be computationally intensive, and enforcing the trust region constraint accurately presents its own set of difficulties. Nevertheless, TRPO has paved the way for subsequent algorithms, such as Proximal Policy Optimisation (PPO), which attempt to address these challenges while retaining the stability benefits inherent in TRPO's approach.

3.7.3 Deep Deterministic Policy Gradient (DDPG)

Deep Deterministic Policy Gradient (DDPG) is a reinforcement learning algorithm designed to solve problems with continuous action spaces. It combines principles from deep learning and policy gradients, using an actor-critical architecture with several key components.

The actor network is responsible for determining the optimal policy by mapping states to continuous actions, while the critic network evaluates the chosen actions. DDPG introduces an experience replay buffer that stores and randomly samples past experiences to break temporal correlations and improve stability.

A notable feature is the use of target networks. Both actor and critic networks have two sets: current networks and target networks. Soft updates gradually align the target networks with the current ones, promoting stability during training.

The training process involves interacting with the environment, storing experience in the replay buffer, and updating the actor and critic networks. The critic is updated by minimising the time difference error, and the actor is updated by maximising the expected return. Exploration is encouraged by adding noise to the actor's output.

The algorithm's success hinges on tuning hyperparameters such as learning rates, discount factors, mini-batch sizes, and noise parameters. Despite its effectiveness, DDPG has challenges like overestimation bias, which can be mitigated using techniques like clipped double Q-learning.

In summary, DDPG is an off-policy actor-critic algorithm adept at handling continuous action spaces. By incorporating experience replay, target networks, and exploration noise, it achieves stability and efficiency in learning, making it a valuable tool in reinforcement learning. More detail information can be found in [30] and [24].

3.7.4 Deep Q Network (DQN)

The Deep Q Network (DQN) is an advanced reinforcement learning algorithm that combines Q-learning principles with deep neural networks. Essentially, DQN aims to learn a Q-function, which estimates the cumulative future rewards an agent can expect for taking a specific action in a given state.

The process begins with the agent interacting with an environment that is represented as a Markov Decision Process (MDP). The agent observes the current state and takes actions based on its policy. To enhance learning stability, DQN uses a technique called experience replay. Past experiences are stored in a replay buffer, breaking the temporal correlation between successive samples. This buffer includes tuples containing the current state, the action taken, the immediate reward, and the subsequent state.

The Q-function is approximated using a deep neural network called the Q-network. This network takes the current state as input and outputs Q-values for all possible actions. The parameters of this network are updated to minimise the temporal difference error, which is the difference between predicted and target Q-values.

To further stabilise learning, DQN introduces a target Q-network. This network has the same architecture as the Q-network but maintains frozen parameters for a specified number of steps. The target Q-values, derived from this network, are crucial in the Q-learning update.

Training involves sampling mini-batches from the replay buffer, and Q-network parameters are updated using gradient descent to minimise the mean squared TD error. Balancing exploration and exploitation is crucial, and DQN typically employs an epsilon-greedy strategy for action selection.

This iterative process continues until the agent's policy converges or a pre-defined stopping criterion is met. DQN has been successful in solving complex reinforcement learning problems such as video game playing and robotic control. Its fundamental nature has inspired subsequent advances in the field of deep reinforcement learning. More information about this method can be found in [31].

3.7.5 Methods comparison

In the specific context of optimising lap times for a 1:43 scale vehicle, considerations such as sample efficiency, real-time performance, and adaptability to continuous control are pivotal. While DQN is seen as a good candidate for this work due to its handling of complex state representations and versatility, which could prove advantageous in a scenario where data collection may be resource-intensive, its inability to handle continuous actions may be a limitation. Therefore, policy gradient methods like DDPG or PPO could be better suited, as they are designed for continuous control tasks and may offer more efficient exploration of the action space.

DDPG, in particular, could be a promising choice, as it combines the strengths of actor-critic architectures to learn both the policy and value function, potentially providing a balance between sample efficiency and control precision. Additionally, the use of Proximal Policy Optimisation (PPO) could be beneficial, as it has demonstrated robust performance across a wide range of continuous control tasks while being relatively simple to implement and tune compared to other policy gradient algorithms. The choice between these reinforcement learning methods ultimately depends on the specific requirements and constraints of the 1:43 scale vehicle optimisation task.

However, as we will talk in more detail in the next chapters, we will use Unity as our simulation environment, along with the ml-agents library, enabling us to implement reinforcement learning in this environment. Unfortunately, this library only provides us with some built-in methods. Thus, making us chose the PPO algorithm as it is the most well-suited reinforcement learning method available in the ml-agents toolkit for the 1:43 scale vehicle optimisation task.

Chapter 4

Path-Following Control in a Simplified Environment

In this chapter, we discuss the work that was carried out in the initial phase of the project when the final simulation environment was not yet ready. As a result, a simple simulation environment was developed using a Python library named Pygames [32]. In this environment, we implemented a PID, explored other types of reactive controllers, namely the carrot chase algorithm, and used imitation learning to replace the PID controller.

4.1 Simplified Model

In this setup, we used the simplified model mentioned in section 3.3.2 represented by the equations 3.6-3.8. For easier reference, these equations are repeated below:

$$\dot{x} = v \cdot \cos(\theta) \tag{4.1}$$

$$\dot{y} = v \cdot \sin(\theta) \tag{4.2}$$

$$\dot{\theta} = \frac{v}{b} \cdot \tan(\delta) \tag{4.3}$$

4.2 PID controller

The PID controller in this setup was utilised to steer the vehicle so that it stays in the middle of the track. To achieve this, two distinct strategies were developed and implemented. The initial strategy was designed to be more flexible, taking into consideration that the track might not be known beforehand. On the other hand, the second strategy assumes that the track is known, making it more restricted, aiming for precision in following the desired trajectory.

4.2.1 With an unknown environment

In our first approach, we addressed the scenario where the vehicle had no knowledge of its environment, which means that it did not have prior information about the path it was supposed to navigate. For this problem, our solution was to use the cross-track error (CTE), which is the car's lateral deviation from the desired trajectory. To determine that, the difference between the distance to both borders of the track was measured.

In order to determine the distance from the car to the track border regardless of the car's position, we utilised the projection of two lines from the car at specific angles, -70° and 70° relative to the car's current heading. It is important to note that these angles can vary. Using angles closer to perpendicular to the car's heading increases the accuracy as they result in points that are closer to the car's position, thus representing the real perpendicular points where the car is located. However, smaller angles mean the vehicle has less tolerance for being rotated from the track direction. On the other hand, larger angles provide more tolerance when the vehicle is rotated, but the represented points are further away from the car's current position, making the control a bit unstable.

Figure 4.1 below demonstrates how these angled lines extend from the vehicle towards the track borders. This visual aid makes it easier to comprehend how the distances to the track edge are obtained.

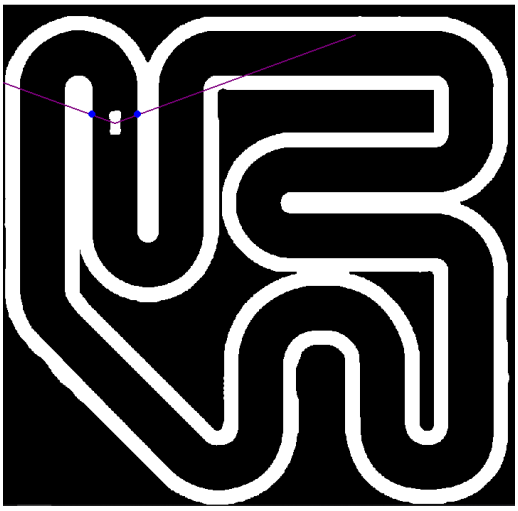

Figure 4.1: Representation of the car with the lines to get the interception with the borders of the track.

To compute the perpendicular distance, denoted as d_i^*, from a car to the closest wall, as shown in Figure 4.2, we used the orientation of the car represented by the angle θ, the inclination of the line in relation to the perpendicular of the car, which is given by the angle ϕ, and the straight-line distance from the centre of the car to the closest point on the wall, referenced as d_i. With these parameters, we calculated the shortest distance from the car to the wall using the equation 4.4 shown below. Then, the CTE was obtained by subtracting the two distances, d_i^*, of each side of that car.

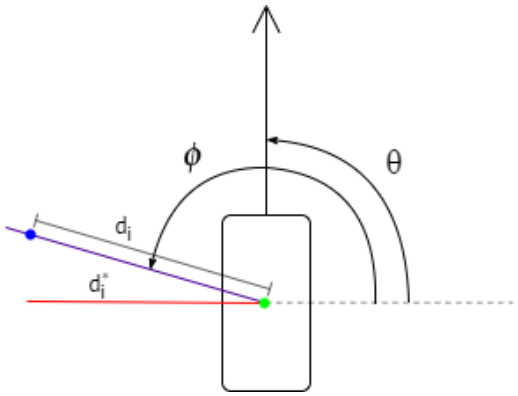

Figure 4.2: Calculation of the perpendicular distance to the car.

$$d_i^* = d_i \cdot \cos(\phi - \theta) \tag{4.4}$$

While this initial strategy was not a mandatory requirement for addressing the issue discussed in this dissertation, it was helpful in determining the desired path needed for the second approach. As said before, the focus of this dissertation was to control the vehicle in a scenario where the environment is known. By using this first approach, we were able to make the environment known and determine the path to follow, which is shown in Figure 4.3. This path was then used in the second approach as the desired trajectory for the car to follow.

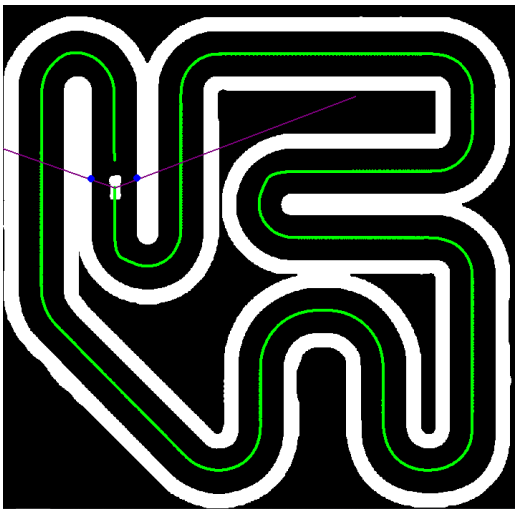

Figure 4.3: Representation of the trajectory mapped.

4.2.2 With a known environment

In this second approach, we assumed that the vehicle is well-acquainted with its environment and the path it must follow, and as in the first approach, we use the cross-track error (CTE) which is the distance between the centre of the car and the closest point on the desired trajectory.

In the implementation of this method, we employed a distance-based strategy wherein, in each cycle, we selected the point that lies ahead of the vehicle's current position, as well as a point that falls behind it on the path it intends to tread. Upon establishing these points, a line was drawn to connect them. With that line, we found the closest point to the car's centre, thus providing a tangible measure of the cross-track error. Figure 4.4 can be used for a more intuitive understanding of this process. Furthermore, this distance-based strategy was implemented in order to minimise the eventual imperfections that the trajectory could have, as this one was a result of the first approach, where the environment was unknown.

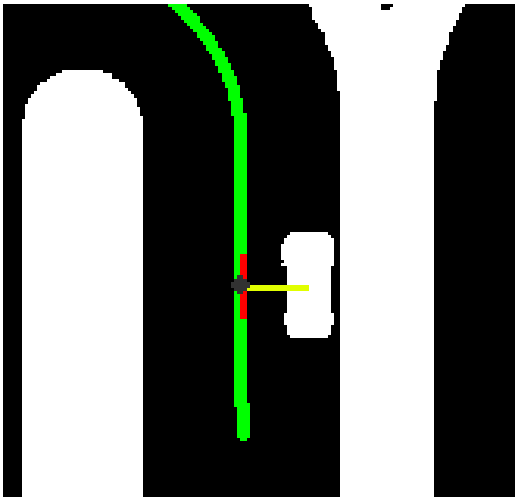

Figure 4.4: Representation of the closest point to the trajectory.

The length of the guiding line, shown in red in Figure 4.4, influences the car's trajectory and depends on the distance between the selected endpoints. When the distance between these points is relatively short, the car is able to adhere more faithfully to the desired trajectory, ensuring a movement pattern that closely matches the intended path. On the other hand, if the distance between points extends significantly, the car might begin to deviate from the precise path. In cases where the points are too far apart, there's a tendency for the car to take shortcuts, leading to a movement pattern that strays from the exact route, thereby compromising on precision. This phenomenon of cutting corners results in a discrepancy between the intended and actual paths taken by the car. These differences can be seen in Figures 4.5 and 4.7 for shorter distances and in Figures 4.6 and 4.8 for longer distances, respectively.

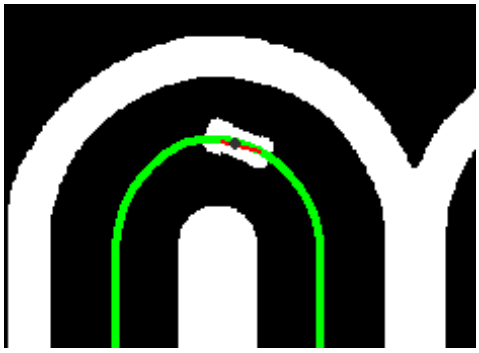

Figure 4.5: Guide line with 20 points of distance between the two endpoints.

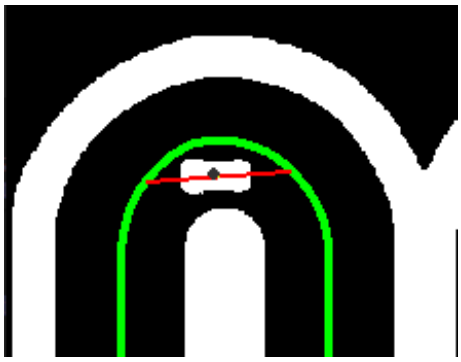

Figure 4.6: Guide line with 80 points of distance between the two endpoints.

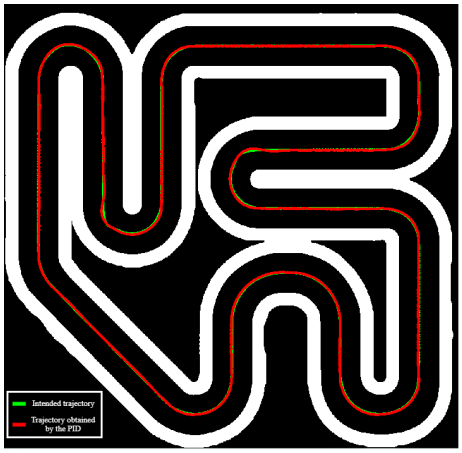

Figure 4.7: Result of the PID controller with a guide line with 20 points of distance between the two endpoints.

Figure 4.8: Result of the PID controller with a guide line with 80 points of distance between the two endpoints.

After testing the performance of the vehicle with guideline distances of 0, 5, 10, 20, 30, 40, 50, 60, 70, and 80 points between the two endpoints, we ultimately decided on a line length of 20 points. This particular length provided the optimal balance as it allowed us to follow the planned trajectory with greater accuracy while effectively smoothing out any occasional flaws in the path. This approach ensures both precision in tracking and an improved overall appearance by minimising the impact of any irregularities.

4.2.3 Tuning of the PID

The PID controller continuously monitors the cross-track error. Based on this error value, it calculates three separate components: the proportional part (P), which adjusts the output proportionally to the error; the integral part (I), which accounts for the sum of past errors, essentially attempting to eliminate residual errors by integrating the error value over time; and the derivative part (D), which predicts future errors by considering the rate at which the error is changing. The controller combines these three components—proportional, integral, and derivative—to produce an output

that aims to minimise the error and stabilise the system. In this particular case, the output, $u(t)$, is the steering angle. The PID is given by the equation 4.5 presented below:

$$u(t) = P \cdot e(t) + I \cdot \int_0^t e(t)\,dt + D \cdot \frac{de(t)}{dt} \tag{4.5}$$

Tuning a PID controller involves adjusting its three parameters: Proportional (P), Integral (I), and Derivative (D) gains, to get the desired response from a system. Proper tuning is crucial for achieving optimal performance, minimising errors, and ensuring system stability.

The process of tuning the parameters was manual, it started by setting the I and D gains to zero and increasing the P gain until the system exhibited sustained oscillations around the desired path. After that, the D gain was gradually increased to set a favourable damping response, and finally, the I gain was adjusted to minimise the steady-state error and improve the overall tracking performance of the system. This manual approach to tuning a PID controller was effective but also time-consuming.

The values obtained after adjusting these parameters were the following:

$$P = 0.5 \qquad I = 0.1 \qquad D = 0.05$$

4.3 Carrot chase algorithm

To further investigate the path-following methods, we implemented a strategy known as the "carrot chase" algorithm. This method focuses on pursuing a dynamically positioned target point ahead on the pathway. The "carrot chase" algorithm operates by propelling the car to move forward temporally, all the while aiming for a pre-selected point that acts as its current goal or "carrot".

During each iteration of this method, the car is assigned a new target point along the trajectory it must follow. It then adjusts its heading towards this target. This process is continuous, with the car recalculating its trajectory in each cycle to chase the subsequent "carrot."

As illustrated in Figure 4.9, the mechanism of action for the car involves following a blue point previously referred to as the "carrot". It is important to note, however, that the efficacy is influenced by the distance maintained between the vehicle and the carrot. If this distance is too great, the car will start to cut corners. This happens because the algorithm is designed to prioritise reaching the carrot over maintaining the ideal path.

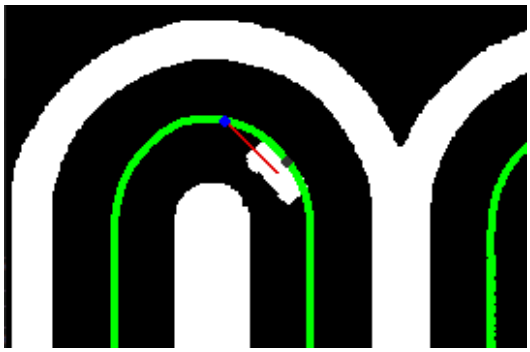

Figure 4.9: Representation of the carrot chase algorithm.

In order to determine the direction in which the car needs to turn, we calculated the angular difference between the vector that represents the target direction, passing through the "carrot" at an angle λ, and the vector representing the vehicle's current heading at an angle θ. In Figure 4.10, these two vectors are shown in red and black, respectively. This difference gives us the error angle Φ, which informs us whether we need to turn left or right to align with the desired path. We can calculate this using the following equation:

$$\Phi = \theta - \lambda \tag{4.6}$$

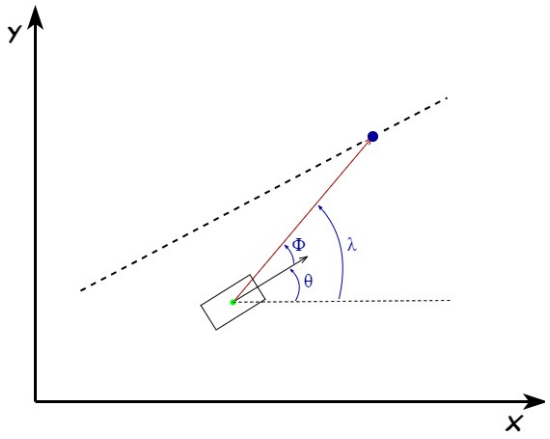

Figure 4.10: Representation of the angle Φ.

With angle Φ, we can determine the steering angle (δ) to orientate the vehicle to the "carrot" with the equation

$$\delta = \begin{cases} min(\Phi, \delta_{max}) & \text{if } \Phi \geq 0 \\ max(\Phi, -\delta_{max}) & \text{if } \Phi < 0 \end{cases} \tag{4.7}$$

where δ_{max} corresponds to the maximum steering angle of the vehicle.

Moreover, it is important to consider the speed of the vehicle when establishing the ideal distance between the car and the "carrot". The speed influences the required reaction time and distance needed to correct the trajectory. At higher velocities, the distance to the carrot needs to be greater than for lower velocities in order for the car to be able to turn in time. In this particular case, and after testing with numerous car-carrot lengths, we opted to choose a length of 40, which means that the carrot is always 40 points of the trajectory ahead of the car.

Figure 4.11 presents the resulting trajectory of using this algorithm; the carrot chase algorithm effectively guides the vehicle along the desired trajectory by continuously adjusting its heading to align with the dynamically positioned target point, or "carrot", resulting in a smooth and effective way to follow the intended path.

Figure 4.11: Result of the carrot chase algorithm.

4.4 Comparisons between the PID and the Carrot chase algorithm

The two path-following methods previously implemented, namely the PID controller and the Carrot chase algorithm, have both advantages and limitations, which will be discussed.

The PID controller is a widely used feedback loop mechanism in control systems. It operates by calculating the error between a desired setpoint and the current state of the system. Subsequently, the controller modifies the control inputs in accordance with three principal components: The proportional (P) component responds to the current error, the integral (I) component accounts for the accumulation of past errors, and the derivative (D) component predicts future errors based on their rate of change. This combination enables the PID controller to provide precise and stable control of the system. However, the process of tuning the PID parameters in order to achieve the desired performance can be time-consuming.

In contrast, the Carrot Chase algorithm is a relatively simple and intuitive path-following technique that, as previously stated, this algorithm employs a metaphorical representation of a rabbit pursuing a carrot positioned in its path. The "carrot" represents a target point that moves along

a desired path, and the vehicle continuously adjusts its direction to move towards this point. As the vehicle progresses, the carrot point advances, leading the car along the predetermined path. This method relies on simple geometric rules and feedback, making it relatively straightforward to implement and comprehend. However, its simplicity also means it can encounter difficulties with sharp turns or sudden changes in the trajectory, leading to less precise path-following in complex environments.

In terms of applications, the Carrot Chase Algorithm is frequently employed in scenarios where the environment is well-defined and the path is predetermined. This includes autonomous vehicles and mobile robots. The simplicity of implementation and the low computational demand of the Carrot Chase Algorithm make it an attractive choice for systems with limited processing power. On the other hand, the PID Controller is particularly well-suited to applications where maintaining a specific setpoint is of paramount importance, such as in industrial control systems, process control, and motor speed regulation. The capacity of this method to cope with disturbances and adapt to changes in system dynamics renders it a versatile tool for a variety of control problems.

Overall, the Carrot Chase Algorithm and PID Controller each have their unique strengths. The Carrot Chase Algorithm offers simplicity and efficiency, making it ideal for straightforward path-following tasks. Meanwhile, the PID Controller provides precise, adaptable control suitable for complex and dynamic systems, albeit with the need for careful tuning.

4.5 Imitation learning

After the implementation of the PID controller, our objective was to use imitation learning to replace it. Thus bridging the gap between classical controllers and those based on machine learning.

In the initial phase of this process, we began by establishing the state that would serve as the input for the neural network. To do this, we defined the parameters as the most recent N cross-track errors that the vehicle had encountered, with the cross-track error denoted as e. The state is represented by the equation 4.8 below.

$$X = [e_1 \quad e_2 \quad e_3 \quad \dots \quad e_N] \tag{4.8}$$

The architecture of our neural network features an input layer with a size of N. This is where the initial data enters the network, namely the N last CTEs. Following the input layer, the network is composed of 3 hidden layers. The first one comprises 1024, making it the largest layer of the neural network. The second and the third hidden layer contain 512 neurons each. In the end, we have the output layer that features a single neuron, representing the value that we would obtain from the PID controller. The detailed layout of this neural network is represented in Figure 4.12.

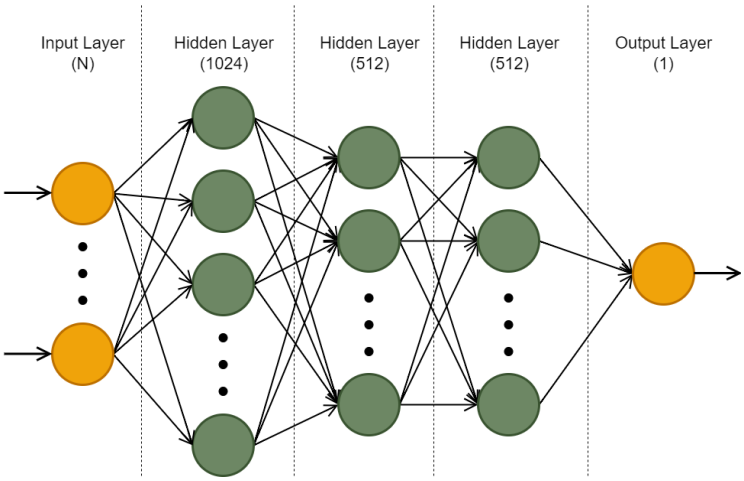

Figure 4.12: Diagram of the neural network.

With the objective of training the neural network, we used numerous examples demonstrating the behaviour of the PID controller in response to various cross-track errors. The collective data amounted to a total of 2 million points, which were used to train the neural network across 500 epochs.

The algorithm used is presented in Algorithm 1. First, in this algorithm, the number of interactions, N, is set to 2000000, representing the number of data points to generate and the initial CTE, denoted as *error*, is set to zero.

The main part of the algorithm is a loop that runs N times. In each iteration, the algorithm first checks if the absolute value of the *error* is less than $1e^{-4}$. If this is the case, it generates a random number, following a uniform distribution, belonging to the interval $[-10.0\ 10.0]$ and adds to the current car's position, making it the new target. This step ensures that the car's target position is varied, introducing new errors for the PID controller to correct.

Next, the algorithm updates the *error* value by calculating the difference between the car's current position and the target position. This updated error is then added to the *pid_input* list, which stores all the errors encountered during the simulation.

Finally, the algorithm updates the car's position based on the PID controller's correction, simulating the car's response to the correction. This new position will be used in the next iteration to calculate the new *error*.

This loop continues until two million data points have been generated. The resulting *pid_input* and *pid_output* lists form a solid training dataset for the neural network. By training on this dataset, the neural network can learn to predict and correct errors in a manner similar to the PID controller, potentially improving the car's control system.

Algorithm 1: Points for the neural network training.

Result: pid_input (vector of the PID input errors), pid_output (vector of the PID output corrections)

$N \leftarrow 2000000$;
$error \leftarrow 0$;
for $p \leftarrow 0$ **to** $N-1$ **do**
  if $|error| < 1e-4$ **then**
    $rand\_num \leftarrow$ random.uniform$(-10.0, 10.0)$;
    $pos\_x \leftarrow car.x + rand\_num$;
  end
  $error \leftarrow car.x - pos\_x$;
  Add the error to the vector of the PID input errors(pid_input);
  Calculate the PID's correction;
  Add the PID's output to the results vector(pid_output);
  Move the car depending on the PID's correction;
end

Our trials began by arbitrarily defining the state size with the value of 6, meaning we set it as the last 6 CTEs. However, the results obtained were not up to the mark. To address this issue, we decided to increase the state size to capture the last 12 cross-track errors. After we obtained the results, the improvements were visible, validating our hypothesis regarding the expansion of the state size to improve the performance.

Nevertheless, we experimented further by increasing the state size to 20, with the aspiration of achieving even better results. Despite this effort, the trajectory remained practically unchanged, suggesting that at a certain threshold, the performance wouldn't improve anymore.

In the end, we decided to revert to a state size of 12. We concluded that this configuration offered the optimal balance between complexity and performance, as can be seen below in Figures 4.13, 4.14 and 4.15.

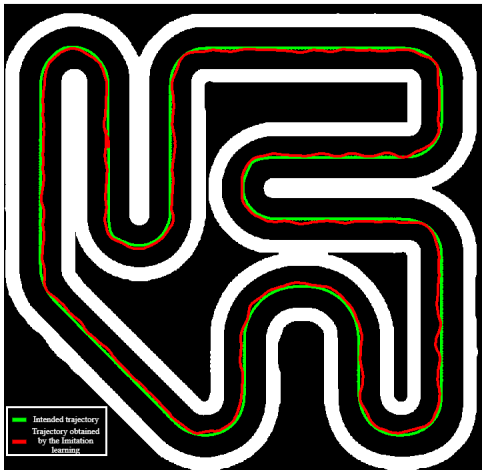

Figure 4.13: Path using Imitation Learning with a state size of 6.

Figure 4.14: Path using Imitation Learning with a state size of 12.

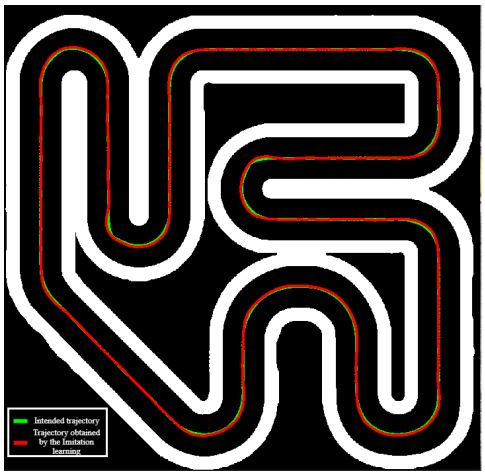

Figure 4.15: Path using Imitation Learning with a state size of 20.

After this training, the neural network was able to understand and mimic the behaviour of the PID controller. As a result of this successful imitation, the neural network demonstrated its capability to replace the traditional PID controller effectively. This can be seen in Figure 4.16 below. In this figure, the intended navigation path is represented with a green colour, while the path controlled by the neural network through imitation learning is represented in red.

Figure 4.16: Path using Imitation Learning.

4.6 Comparisons between the PID and the Imitation learning

The PID controller and imitation learning represent two distinct approaches to controlling systems, each with its own methodology, advantages, and limitations.

As described above in section 4.4, the PID controller is a traditional control method using feedback to correct errors between a desired setpoint and the current state of the system.

Imitation learning, on the other hand, is a type of machine learning where an agent learns to perform a task by observing and mimicking the actions of an expert. Instead of explicitly programming the controller, the agent learns a policy by training on data that consists of state-action pairs from expert demonstrations. In this specific case, the objective was to mimic the behaviour of the PID.

Imitation learning shines in applications where the task is too complex for manual programming or traditional control methods. It is particularly effective in autonomous driving. The primary advantage of imitation learning is its ability to leverage expert demonstrations to learn sophisticated behaviours, making it highly adaptable and capable of improving as more data is gathered.

Nevertheless, imitation learning comes with its own set of challenges. The performance of an imitation learning model heavily relies on the quality and quantity of the expert demonstrations. Poor or insufficient demonstrations can lead to suboptimal or even unsafe behaviour.

In conclusion, PID controllers provide a conventional and accurate approach to control systems with well-understood dynamics. In contrast, imitation learning offers a flexible, data-driven method that can perform well in complex and poorly understood environments by utilizing expert demonstrations. The selection between these methods relies on the particular application needs, such as the complexity of the task, availability of expert data, and the requirement for adaptability.

Chapter 5

Planning and Control with Higher Fidelity Models

This chapter will examine the work that was developed in the final simulation environment, Unity. In this environment, an updated version of the PID was implemented, with the dynamics of the car being implemented in a more realistic manner than in the previous environment. Subsequently, an implementation of the algorithm of reinforcement learning PPO was developed and trained, enabling the car to drive around the track.

5.1 Model used

The model used for the work presented in this chapter was the one mentioned in section 3.3.1, and it is represented by the equations 3.2-3.5. For easier reference, these equations are repeated below:

$$\alpha(\delta) = \arctan\left(\frac{a \cdot \tan(\delta)}{b}\right) \tag{5.1}$$

$$\dot{x} = v \cdot \cos(\alpha + \theta) \tag{5.2}$$

$$\dot{y} = v \cdot \sin(\alpha + \theta) \tag{5.3}$$

$$\dot{\theta} = \frac{v}{r_a} = \frac{v}{b} \cdot \tan(\delta) \tag{5.4}$$

5.2 PID controller

In this particular implementation of the PID controller, we make the assumption that the layout of the track is already known. To accomplish this, we have strategically placed checkpoints along the track, shown in Figure 5.1, dividing it into 8 segments such as straights and curves. This division allows us to define a trajectory along the racetrack, as illustrated in Figure 5.2.

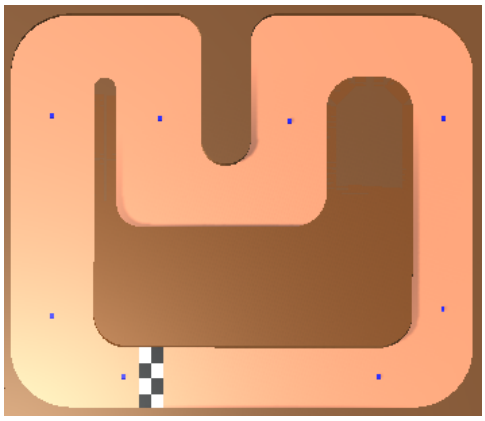

Figure 5.1: Track marked with the checkpoints.

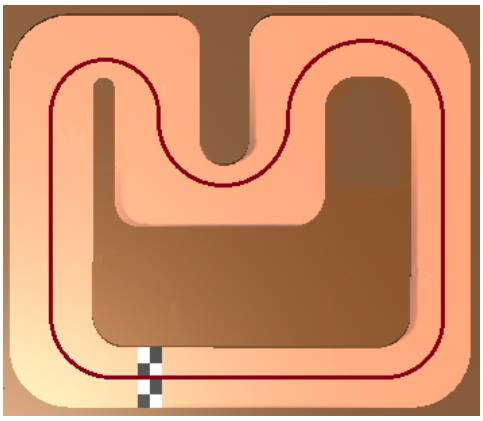

Figure 5.2: Track marked with the trajectory.

This trajectory allows us to identify the point along the path that is closest to the current position of the car. This process involves determining the segment where the vehicle is located and then finding the nearest point. Two methods have been implemented to find this point: one for straight segments and another for curved segments.

Closest point in a straight segment

In a straight segment, the process begins by defining the vectors AB and AC. Points A and B represent the extremes of the segment, and C is the car's position. The equations for the vectors are denoted as 5.5 and 5.6.

$$AB = B - A \tag{5.5}$$

$$AC = C - A \tag{5.6}$$

Subsequently, we utilised vectors AC and AB to calculate the projection factor (t) of AC onto AB, as demonstrated in equation 5.7.

$$t = \frac{AC \cdot AB}{AB \cdot AB} \tag{5.7}$$

Finally, the determination of the closest point (P) was performed using the projection factor (t), the point A, and the vector AB, as demonstrated in equation 5.8.

$$P = A + t \cdot AB \tag{5.8}$$

Figure 5.3 serves as a visual aid to facilitate the comprehension of this process.

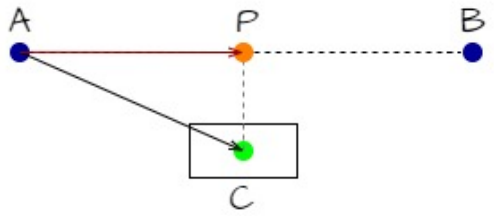

Figure 5.3: Representation of the method to find the closest point of the trajectory in a straight.

Closest point to a curved segment

In a curve segment, the process begins by defining the vector OC, where O is the centre of the curve formed by the extremes A and B, and C is the car's position. The equation 5.9 for this vector is shown below.

$$OC = C - O \tag{5.9}$$

Next, we normalised this vector to get its orientation, as can be seen in equation 5.10.

$$OC_{normalised} = \frac{OC}{||OC||} \tag{5.10}$$

Finally, to obtain the closest point (P), we scale the normalised OC vector with the radius (r). This is shown in equation 5.11

$$P = O + OC_{normalised} \cdot r \tag{5.11}$$

Figure 5.4 serves as a visual aid to facilitate the comprehension of this process.

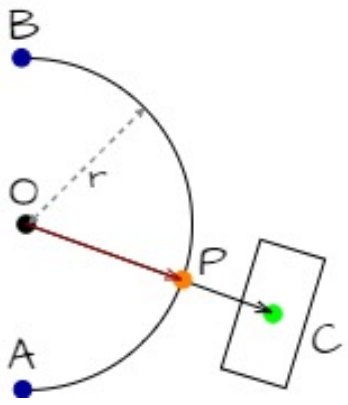

Figure 5.4: Representation of the method to find the closest point of the trajectory in a curve.

With the nearest point of the trajectory, we proceed to calculate the cross-track error. This error is defined as the perpendicular distance between the car's position and the closest point on the trajectory. In a simpler way, it measures how far the car is to the specified path.

Having the cross-track error, we use it as an input of the PID controller. With the PID being a feedback control loop mechanism, it uses this input to adjust the steering angle of the car in order to minimise this error, resulting in the vehicle driving back towards the desired trajectory. In this process, we applied the same equation for the PID controller (equation 4.5) that was used in section 4.2.3.

5.2.1 Results

Overall, the vehicle was able to effectively follow the desired trajectory, as shown in Figure 5.5, demonstrating the PID controller's effectiveness. However, there were occasions when the car experienced some delays in recovery, namely at the end of each curve.

The delays in the recovery are due to the need for adjustments in the PID parameters. One way to reduce the recovery time is by modifying these parameters. However, this comes with a trade-off because decreasing the recovery time would result in a more oscillatory path. Therefore, we concluded that a slower recovery time would be preferable to a more oscillatory trajectory.

The tuning of the three gains of the PID controller—proportional, integral, and derivative—was performed manually, using the same process as in section 4.2.3. In the end, we obtained the following values for the parameters:

$$P = 1.7 \qquad I = 0.1 \qquad D = 0.5$$

The recovery time was also influenced by the vehicle's maximum steering angle (δ), which was limited to 13.5 degrees. A greater steering angle would allow the vehicle to make sharper turns and quick corrections, improving its agility and overall performance, but it would not necessarily reflect the real physical constraints of the hardware.

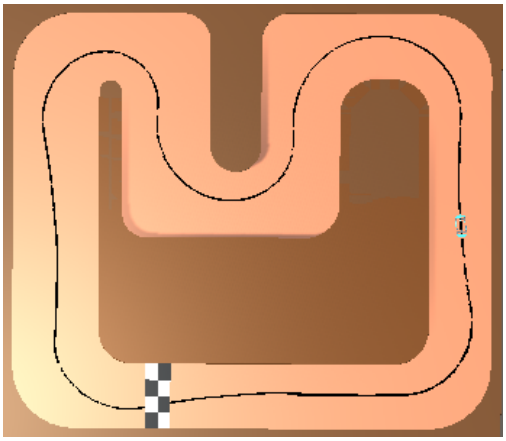

Figure 5.5: Path using the updated PID in the new environment.

5.3 Reinforcement Learning

To implement reinforcement learning in Unity, we used the Unity Machine Learning Agents Toolkit (ML-Agents) [33]. This toolkit is an open-source project that allows games and simulations to be used as environments for training intelligent agents using deep reinforcement learning and imitation learning. The main component of the ML-Agents framework used in this project was the Proximal Policy Optimisation (PPO) algorithm for reinforcement learning.

5.3.1 State Space and Control Inputs

For the implementation of this algorithm, we needed to define the state space and control inputs. The state space consists of the relevant variables that describe the environment that the vehicle is inserted into and the vehicle's internal state. This state will determine the actions that the agent can take in order to maximise the reward and reach the desired goal. Furthermore, these actions will have consequences on the subsequent states of the system, which the agent must learn to predict and exploit in order to optimise its behaviour over time. The control inputs are the parameters that the vehicle can manipulate in order to influence the state of the system and drive towards the desired goal. The system's state and control inputs are presented below in equations 5.12 and 5.13, respectively, with a complementary explanation of what each parameter represents:

$$X = [s \quad e \quad \mu \quad v_x \quad v_z \quad w \quad f \quad \delta] \tag{5.12}$$

$$Y = [f^* \quad \delta^*] \tag{5.13}$$

where

- s - is the progress made into the track, given by the following expression $s = s_t - s_0$
- e - is the current cross-track error (CTE)
- μ - is the difference in angle between the car and the reference trajectory direction as illustrated in Figure 5.6
- v_x - is the velocity in the x-axes
- v_z - is the velocity in the z-axes
- w - is the angular velocity
- f - is the last input of the acceleration control
- δ - is the last input of the steering angle
- f^* - is the control input of the acceleration control
- δ^* - is the control input of the steering angle

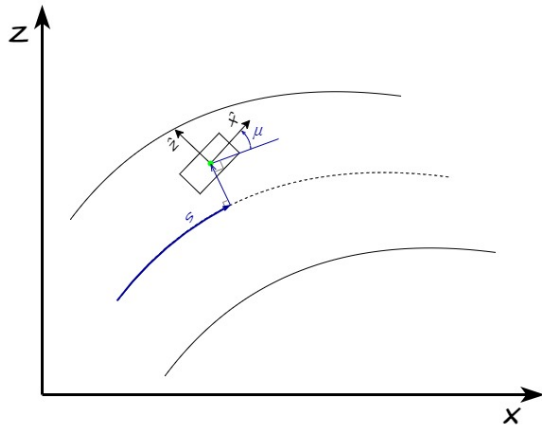

Figure 5.6: Difference in angle between the car and the trajectory.

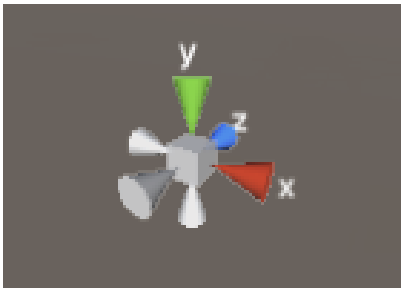

Figure 5.7: Representation of the referential used in Unity.

Additionally, it was taken into account that some of these parameters have physical limitations. Therefore, additional constraints were imposed to ensure the feasibility and safety of the vehicle's actions. In this case, the restrictions were:

$$
\begin{aligned}
v_x &\in [0 \quad V_{xmax}] \\
v_z &\in [0 \quad V_{zmax}] \\
f^* &\in \{-1, \quad 0, \quad 1\} \\
\delta^* &\in [-\delta_{max} \quad \delta_{max}]
\end{aligned}
$$

where

- $V_{max} = V_{xmax} = V_{zmax} = 91 cm/s$
- $\delta_{max} = 13.5°$

5.3.2 Reward function

The reward function used in the training process was designed to encourage the vehicle to prioritise increasing the distance it could travel without incidents. This was achieved by rewarding the vehicle for every progress on the track, based on the distance covered, thus motivating continuous forward progression. Simultaneously, to ensure that this drive for distance did not compromise safety or adherence to the driving track, penalties were introduced within the reward structure for any form of misconduct, such as colliding with the boundaries of the track or engaging in reverse driving, which could signify confusion or incorrect navigation. Notably, collisions with other vehicles were not penalized during this training, as we assumed the car was alone on the track.

The implemented reward function is presented in equation 5.14, where Δs is the progress made in the track between two consecutive cycles, $\Delta s = s_t - s_{t-1}$.

$$R = \begin{cases} -5, & \text{if hit a wall} \\ -1, & \text{if } \Delta s \text{ is negative} \\ \Delta s, & \text{if } \Delta s \text{ is positive/zero} \end{cases} \tag{5.14}$$

This balance between encouraging exploration and covering distance while discouraging potentially dangerous behaviours creates a feedback loop that is crucial for the vehicle's learning algorithm. It enables the vehicle to understand and internalise not only the mechanics of motion but also the strategic importance of safe and rule-compliant navigation. By continuously interacting with the environment and learning from the feedback provided by the reward function, the vehicle's reinforcement learning algorithm can steadily improve its decision-making processes. This leads to better driving efficiency and safety, aligning the vehicle's performance with the main objectives established at the beginning of the training process.

5.3.3 Training process

With these parameters, we proceeded to train the vehicle in the simulator environment. For that, we utilised a powerful server equipped with an AMD Ryzen 7 5800X processor and 32GB of RAM. The Ryzen 7 5800X is a high-performance, 8-core, 16-thread CPU that provides excellent processing power for the demanding computational tasks required in the training process. The 32GB of RAM ensures that the server has ample memory to handle the large datasets and complex algorithms involved in the training simulations.

The choice of this hardware configuration was based on two factors: the availability and the need to efficiently process the vast amount of sensor data, environmental information, and vehicle dynamics that are essential for training the autonomous driving system. The Ryzen 7 5800X's powerful cores and generous RAM capacity allowed the server to run the training simulations with minimal latency and maximum responsiveness, enabling the vehicle to learn and adapt to the virtual driving scenarios in a timely and effective manner.

By utilising this robust hardware setup, we were able to accelerate the training process, allowing the vehicle to rapidly acquire the necessary skills and knowledge to navigate the simulated environment with precision and safety. The high-performance computing resources ensured that the training could be conducted at a faster pace, ultimately leading to a more efficient and reliable autonomous driving system.

For each training episode, we spawned the car in a random starting position, P_{car}, near a randomly selected checkpoint, with the checkpoint position denoted as P_{cp}. Additionally, the vehicle's initial orientation, in relation to the track, and velocity were also randomised, represented by the angle μ and velocity v, respectively. These initial conditions were restricted to specific value ranges, as can be seen below, to ensure a diverse set of starting scenarios for the training, ultimately enhancing the robustness and adaptability of the autonomous driving system.

$$\begin{aligned}
\mu &\in [-30^\circ \quad 30^\circ] \\
v &\in [0 \quad V_{max}] \\
P_{carX} &\in [P_{cpX} - 10 \quad P_{cpX} + 10] \\
P_{carZ} &\in [P_{cpZ} - 10 \quad P_{cpZ} + 10]
\end{aligned}$$

Our system was trained for a total of 44.27 million episodes, with a total duration of 30 and a half hours, and each episode was set to have a duration of 360 ticks. This temporal framework was defined so that the vehicle is limited to executing only one or two turns at a time, depending on its initial placement on the track. This constraint serves an educational purpose. By limiting the episode's length, the learning curve for the car is significantly steepened, facilitating an accelerated grasp of its own dynamics. This essential understanding is vital, as it underpins the car's ability to adapt effectively to the numerous challenges presented by each corner of the track.

Additionally, if the duration of the episodes extends excessively, the vehicle's ability to accurately comprehend and assimilate its operational dynamics could be compromised. This excessive length might not only hinder the vehicle's learning process but also present substantial challenges in mastering its intended functions efficiently. As a result, the vehicle may struggle to develop a reliable understanding of its environment and how to navigate it effectively, leading to potential difficulties in achieving its goals.

5.3.4 Behaviour parameters

For training the agent using the library Ml-Agents in Unity, it was necessary to specify a set of behaviour parameters. These parameters were crucial as they set the foundational guidelines for the training process. They determine several key aspects of the training, including the batch size and buffer size. The batch size refers to the amount of training data the agent processes before updating its learning, while the buffer size indicates the amount of data stored from past experiences to learn from.

Additionally, these parameters also dictate the rate at which the agent learns the policies, which is essential for ensuring the agent can adapt its actions to achieve optimal performance. The structure and size of the neural network are also defined within these parameters. This is important because the neural network's architecture directly influences the agent's ability to process and learn from complex data patterns.

Moreover, the duration of the training and the number of epochs on each episode are specified within these parameters. The length of the training is significant for fully developing the agent's capabilities, ensuring it has sufficient time to learn and adapt. Epochs per episode refer to the number of complete passes through the training dataset, allowing the agent multiple opportunities to learn and refine its strategies.

To accomplish this, we have defined a yaml file as shown in 5.1 with the behaviour parameters to train the agent.

```
behaviors:
  Car:
    trainer_type: ppo
    hyperparameters:
      batch_size: 256
      buffer_size: 10240
      learning_rate: 3.0e-4
      beta: 5.0e-4
      epsilon: 0.2
      lambd: 0.99
      num_epoch: 3
      learning_rate_schedule: linear
    network_settings:
      normalize: false
      hidden_units: 128
      num_layers: 2
    reward_signals:
      extrinsic:
        gamma: 0.99
        strength: 1.0
    max_steps: 60000000
    keep_checkpoints: 20
    checkpoint_interval: 500000
    time_horizon: 64
    summary_freq: 10000
```

Code 5.1: Code of the behaviour parameters to train the agent.

5.3.5 Results

After completing the training, the agent was able to navigate the environment effectively, consistently, and reliably complete the proposed task. These results will be presented and discussed in more detail.

At first, we will analyse the graphs showing the cumulative reward and the length of each episode over the course of the training, which are shown in Figures 5.8 and 5.9, respectively. The cumulative reward graph demonstrates a steady increase over successive training episodes, indicating that the agent was able to learn and improve its performance through the reinforcement learning process. This is also proven by the length of each episode, as it also steadily increases over time, suggesting that the agent was able to navigate the environment more efficiently and for a longer time.

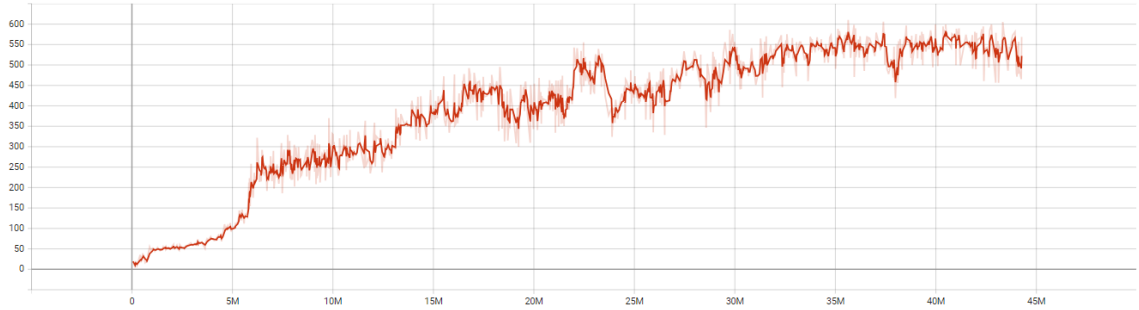

Figure 5.8: Environment_Cumulative Reward.

Figure 5.9: Environment_Episode Length.

Next, we will analyse the losses throughout the training process, namely the policy loss and the value loss, which are presented in Figures 5.10 and 5.11, respectively. The policy loss graph shows a high loss that decreases rapidly at the beginning, which is expected as the agent explores and learns about the environment. After the initial learning phase, the policy loss stabilises and remains relatively low. This suggests that the agent is learning a stable policy. However, there can be occasional spikes in the policy loss, indicating periods where the policy may be exploring new strategies or experiencing instability, like what happened around the episode 35 million.

The value loss graph presents a decrease over time, suggesting that the agent is improving its value predictions, although it does not stabilise as much as the policy loss. Additionally, there are more pronounced fluctuations in the value loss compared to the policy loss. This can occur because value estimation is generally more complex and subject to more variability.

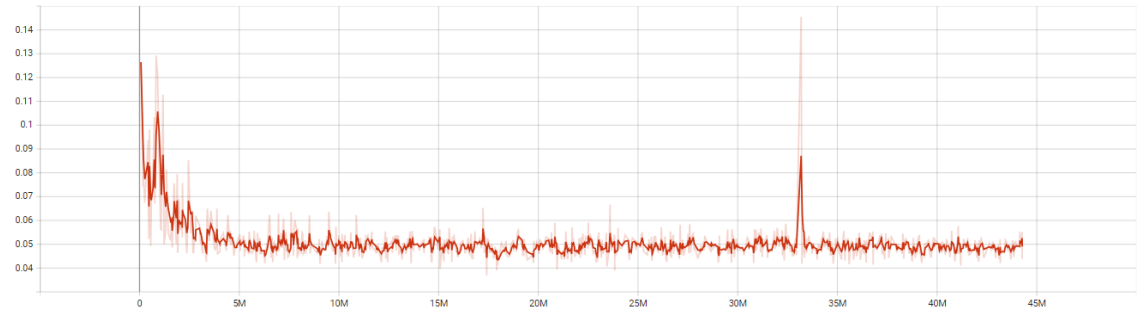

Figure 5.10: Losses_Policy Loss.

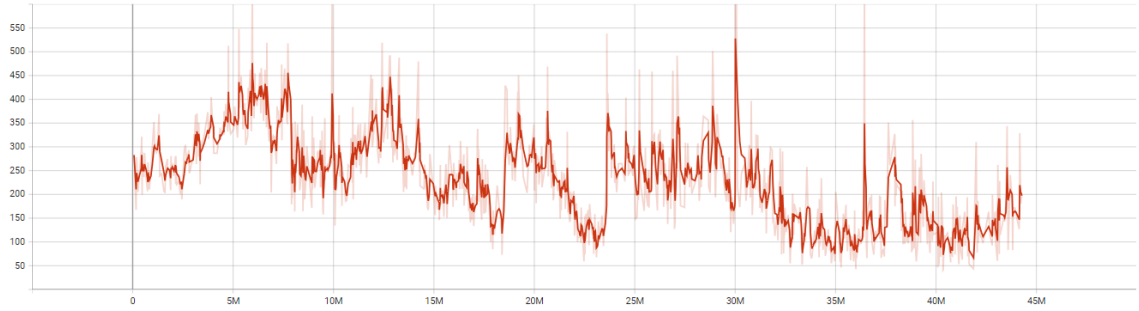

Figure 5.11: Losses_Value Loss.

Furthermore, we will examine the evolution of the policy parameters over the course of the training episodes. The graph of the Beta, shown in Figure 5.12, corresponds to the strength of the entropy regularisation, which makes the policy "more random". For all of the train's duration, the Beta decreases linearly, reducing the influence of the randomness of the policy. The graph represented in Figure 5.13 shows the exploration rate, which decreases over time. This means that the system explores more in the initial part of the training process and gradually shifts to exploiting the learned policy. The Learning Rate graph, shown in Figure 5.14, corresponds to the strength of each gradient descent update step. Initially, the learning rate is high, helping the vehicle to make significant progress, while at the end, this learning rate is lower in order to fine-tune the model parameters.

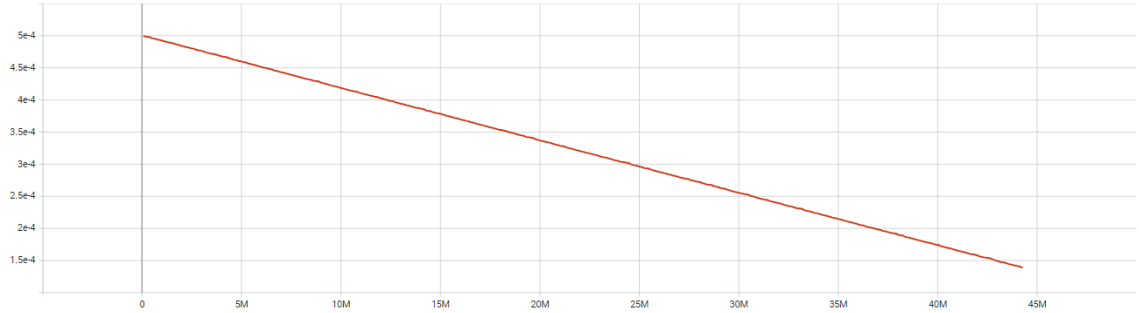

Figure 5.12: Policy_Beta.

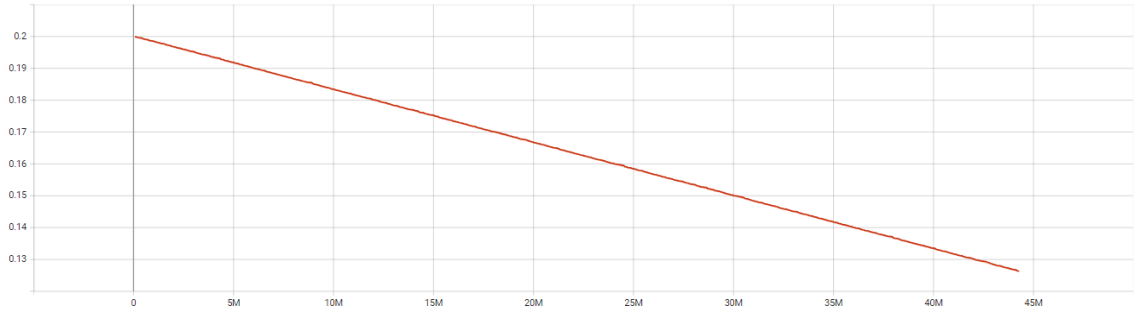

Figure 5.13: Policy_Epsilon.

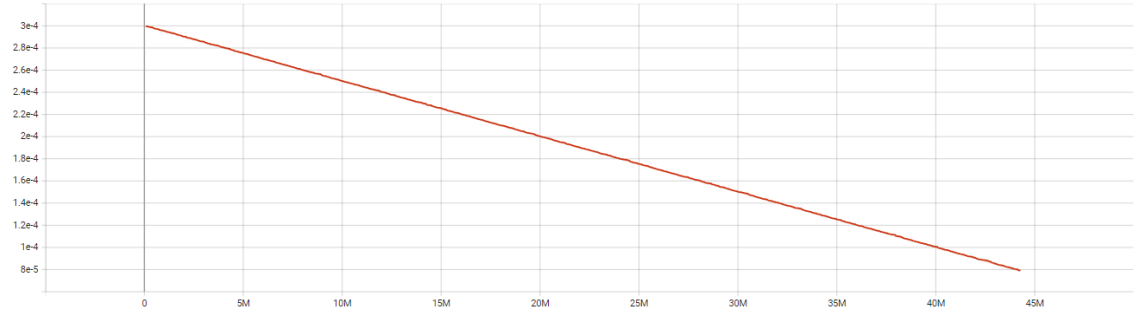

Figure 5.14: Policy_Learning Rate.

The policy entropy graph, presented in Figure 5.15 measures the randomness of the agent's actions. This starts relatively low and increases rapidly, peaking around the 5 million mark, representing an initial increase in its exploration. After reaching the peak, the entropy stabilises and then shows a slight decline. This indicates that the agent is gradually shifting from exploration to exploitation, settling into a more stable policy where it has learned effective actions.

Figure 5.15: Policy_Entropy.

In the policy extrinsic value estimation graph, shown in Figure 5.16, it is visible the agent's estimation of the cumulative reward it expects to receive from the environment. In the beginning, there is a rapid increase in the value estimate, which indicates that the agent quickly learns to identify and pursue rewarding strategies. The value estimate continues to increase with fluctuations. These fluctuations indicate ongoing adjustments and fine-tuning of the agent's strategy as it learns from the environment. Towards the end, the value estimate appears to stabilise, suggesting that the agent's understanding of the environment's rewards has become more consistent and accurate.

Figure 5.16: Policy_Extrinsic Value Estimate.

Chapter 6

Conclusions and Future Work

Firstly, in this chapter, we will summarise the topics that were addressed in this thesis. Then we will provide an account of the work that would be interesting to develop in order to expand this project.

6.1 Conclusion

In conclusion, this dissertation has demonstrated the effective implementation and comparison of various control strategies for autonomous vehicle navigation.

Initially, a traditional PID (proportional-integral-derivative) controller was deployed, which demonstrated its reliability in managing the car's movements. This established control methodology provided a robust basis for subsequent investigations. Building upon this, a neural network trained through imitation learning successfully replicated the PID controller's functionality, thereby demonstrating the potential of machine learning techniques in control applications. The neural network was able to learn the underlying patterns and decision-making processes of the PID controller, thereby demonstrating the power of data-driven approaches in replicating complex control behaviours.

A further advance in the complexity and autonomy of the system was made with the implementation of the reinforcement learning algorithm Proximal Policy Optimisation (PPO). The PPO algorithm enabled the vehicle to navigate the track autonomously, adapting to various conditions and optimising its performance over time. This approach permitted the vehicle to learn and refine its control policies through a process of trial and error, thereby achieving more robust and adaptive navigation capabilities.

The results of this research demonstrate the efficacy of integrating neural networks and reinforcement learning for controlling a vehicle to drive on a track, thereby paving the way for future advancements in autonomous vehicle technology. These findings contribute to the broader field of artificial intelligence and control systems, offering valuable insights and methodologies that can be applied to a range of autonomous applications, including robotics, unmanned aerial vehicles, and intelligent transportation systems.

On a final note, this work serves as a foundation for future investigations, where researchers can build upon these established techniques, improving the overall system.

6.2 Future work

Building upon this project holds immense potential for enhancing the dynamic nature and the challenge it presents. It would be captivating to start by implementing the PPO reinforcement learning method physically. Then, enchanting it to make the cars capable of dodging obstacles. Initially, these obstacles could be static, creating a foundational challenge for navigational algorithms. As the system evolves, introducing obstacles with movement, albeit at slow speeds, would add a fascinating layer of complexity. This change forces the cars to perform overtakes by predicting and reacting to moving targets, mimicking real-life driving scenarios more closely.

Progressing further, an intriguing escalation of this challenge would be to have the cars race side by side, engaging in a dynamic competition where they must overtake each other. The critical aspect here would be enabling the cars to execute these manoeuvres without any collisions. This requires sophisticated perception and decision-making algorithms that can handle high-speed dynamics and intricate spatial awareness.

Broadening the scope of the project, an additional proposal worth exploring is to implement the driving challenges using different reinforcement learning methods. By employing various algorithms, it's possible to conduct an in-depth comparison and analysis of their performance under identical conditions. This comparative study would be instrumental in understanding the strengths and weaknesses of each method, providing insights into which algorithms are most effective for specific types of navigational and obstacle avoidance tasks.

Another promising direction to take the project would be to enhance the system by transitioning to a vehicle of a 1:10 scale. This upgrade would not only provide a more realistic driving experience but also introduce the possibility of mounting a camera on the vehicle. The presence of a camera opens up new avenues for training the vehicle using reinforcement learning, with a significant emphasis on visual perception. This adjustment would elevate the challenge, as the vehicle would now need to interpret visual inputs to navigate its environment, closely mirroring how humans drive. This approach could revolutionise the training process, potentially leading to more agile and autonomously intelligent vehicles capable of complex navigation and decision-making based on real-time visual data.

References

[1] CNN. Roborace: The world's first driverless electric racing car. *CNN*, 2017. Accessed: 2024-07-27. URL: https://edition.cnn.com/2017/03/01/motorsport/roborace-robot-car-ai-autonomous-driverless-electric/index.html.

[2] John Larkin. Indy autonomous challenge unveils next gen autonomous vehicle platform iac av-24. *AI Online*, Jan 2024. URL: https://www.ai-online.com/2024/01/indy-autonomous-challenge-unveils-next-gen.

[3] Alex Liniger. Autonomous rc car racing, 2015. Accessed: January 8, 2024. URL: https://control.ee.ethz.ch/research/team-projects/autonomous-rc-car-racing.html.

[4] WEMOS. S2 mini Documentation, 2024. Accessed: 2024-06-14. URL: https://www.wemos.cc/en/latest/s2/s2_mini.html.

[5] K. J. Åström and R. M. Murray. *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton University Press, Princeton, NJ, USA, 2nd edition, 2019.

[6] blackburn. Introduction to reinforcement learning: Markov decision process, 2019. URL: https://towardsdatascience.com/introduction-to-reinforcement-learning-markov-decision-process.

[7] C. R. Wolfe. Policy gradients: The foundation of reinforcement learning, 2023. Accessed: December 11, 2023. URL: https://cameronrwolfe.substack.com/p/policy-gradients-the-foundation-of.

[8] Betz, Johannes, Zheng, Hongrui, Liniger, Alexander, Rosolia, Ugo, Karle, Phillip, Behl, Madhur, Krovi, Venkat, and Mangharam, Rahul. Autonomous vehicles on the edge: A survey on autonomous vehicle racing. 3:458–488. doi:10.1109/ojits.2022.3181510.

[9] Peide Cai, Hengli Wang, Huaiyang Huang, Yuxuan Liu, and Ming Liu. Vision-based autonomous car racing using deep imitative reinforcement learning. *IEEE Robotics and Automation Letters*, 6(4):7262–7269, 2021.

[10] Etienne Perot, Maximilian Jaritz, Marin Toromanoff, and Raoul De Charette. End-to-end driving in a realistic racing game with deep reinforcement learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 3–4, 2017.

[11] Adrian Remonda, Sarah Krebs, Eduardo Veas, Granit Luzhnica, and Roman Kern. Formula rl: Deep reinforcement learning for autonomous racing using telemetry data. *arXiv preprint arXiv:2104.11106*, 2021.

[12] Yunpeng Pan, Ching-An Cheng, Kamil Saigol, Keuntaek Lee, Xinyan Yan, Evangelos Theodorou, and Byron Boots. Agile autonomous driving using end-to-end deep imitation learning. *arXiv preprint arXiv:1709.07174*, 2017.

[13] Eugenio Chisari. *Learning from Simulation, Racing in Reality: Sim2Real Methods for Autonomous Racing*. PhD thesis, March 2020.

[14] Rajesh Rajamani. *Vehicle dynamics and control*. Springer Science & Business Media, 2011.

[15] Thomas Gillespie. *Fundamentals of vehicle dynamics*. SAE international, 2021.

[16] Jo Yung Wong. *Theory of ground vehicles*. John Wiley & Sons, 2022.

[17] Dean C Karnopp, Donald L Margolis, and Ronald C Rosenberg. *System dynamics: modeling, simulation, and control of mechatronic systems*. John Wiley & Sons, 2012.

[18] MathWorks. Imitate mpc controller for lane keeping assist. https://www.mathworks.com/help/reinforcement-learning/ug/imitate-mpc-controller-for-lane-keeping-assist.html, 2023. Accessed: 2024-07-30.

[19] Alain Droniou, Serena Ivaldi, and Olivier Sigaud. Learning a repertoire of actions with deep neural networks. In *Proceedings of the 2014 Joint IEEE International Conferences on Development and Learning and Epigenetic Robotics (ICDL-Epirob'14)*, pages 229–234. IEEE, 2014.

[20] Leonel Rozo, Pablo Jimenez, and Carme Torras. A robot learning from demonstration framework to perform force-based manipulation tasks. *Intelligent Service Robotics*, 6(1):33–51, 2013.

[21] Adam Coates, Pieter Abbeel, and Andrew Y. Ng. Learning for control from multiple demonstrations. In *Proceedings of the 25th International Conference on Machine Learning*, pages 144–151. ACM, 2008.

[22] Juan Ortega, Noor Shaker, Julian Togelius, and Georgios N. Yannakakis. Imitating human playing styles in super mario bros. *Entertainment Computing*, 4(2):93–104, 2013.

[23] Stephane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. *arXiv preprint arXiv:1011.0686*, 2010.

[24] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, USA, 2nd edition, 2018.

[25] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[26] Kurt Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*, 2018.

[27] Tomorrow Bio. Comparison between model-free vs model-based reinforcement learning. Website, 2023. URL: https://www.tomorrow.bio/post/comparison-between-model-free-vs-model-based.

[28] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[29] John Schulman, Sergey Levine, Pieter Abbeel, Michael I. Jordan, and Philipp Moritz. Trust region policy optimization. *arXiv preprint arXiv:1502.05477*, 2015. URL: https://arxiv.org/abs/1502.05477.

[30] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2016.

[31] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, and et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[32] Pygame Community. Pygame News. https://www.pygame.org/news. Accessed: 2024-07-25.

[33] Unity Technologies. Unity Machine Learning Agents (ML-Agents). https://github.com/Unity-Technologies/ml-agents. Accessed: 2024-07-25.