Reinforcement Learning Racecar Local Controller

Hefan Wang

4th Year Project Report
Computer Science
School of Informatics
University of Edinburgh

2023

Abstract

In this paper, we present a reinforcement learning-based approach for developing an autonomous race car controller capable of optimizing performance while safely navigating complex track configurations. Our methodology integrates a realistic vehicle dynamics simulation with a procedural race track generation algorithm, creating a rich learning environment that encourages the generalization of the trained agent. We employ the Proximal Policy Optimization (PPO) algorithm, renowned for its balance between sample efficiency, simplicity, and performance, as the foundation of our reinforcement learning framework. By carefully designing the observation space, reward function, and action space, we create an effective training environment for our agents. Our results demonstrate the effectiveness of our approach, with the trained agent exhibiting proficiency in autonomously controlling the simulated race car, optimizing performance, and adapting to various track configurations, displaying professional driver-like strategies and manoeuvres.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Hefan Wang*)

Acknowledgements

I sincerely appreciate my supervisor Dr Steve Tonneau for his kind support and guidance.

I want to thank all of my friends and family for their invaluable companionship.

I would also like to thank everyone on the Honours Project Day who voted for and awarded me and my project poster the best poster/presentation award. I went to some very fancy restaurants with the £100 prize, which served as a strong reward signal to reinforce my brain to be even more enthusiastic about the project.

Table of Contents

Chapter 1

Introduction

1.1 Motivations

The advancement of autonomous vehicle technology has gained significant attention in academia and industry. One of the primary challenges in this domain is the development of robust and efficient local controllers that can safely navigate vehicles through complex and dynamic environments. Developing a reinforcement learning (RL) based racecar local controller presents a unique opportunity to address this challenge. Compared to traditional control methods, it can lead to improved performance, adaptability, and generalizability.

The motivation behind building a reinforcement learning racecar local controller stems from the inherent limitations of classical control techniques[7]. While such methods have succeeded in well-defined scenarios, their reliance on fixed mathematical models can hinder their adaptability to novel or changing situations and limit their potential performance by human domain knowledge[27]. On the other hand, reinforcement learning has the potential to overcome these limitations by allowing the controller to learn from its interactions with the environment and adapt its actions accordingly[29]

Moreover, the highly dynamic nature of racing environments presents a compelling case for exploring reinforcement learning-based control. In these environments, vehicles need to maintain high speeds while navigating tight corners, avoiding collisions, and reacting to the actions of other competitors[14] The inherent ability of reinforcement learning algorithms to learn complex, non-linear control policies makes them well-suited for such tasks. Furthermore, deep reinforcement learning techniques can enable the controller to automatically extract relevant features from high-dimensional sensory input, thereby reducing the need for manual feature engineering[21]

In addition to the potential performance benefits, developing a reinforcement learning racecar local controller can provide valuable insights into the broader field of autonomous vehicle control. Lessons learned from this local controller's design, implementation, and evaluation can inform the development of more advanced control systems for a wide range of autonomous applications, from urban transportation to high-speed racing competitions[20] Ultimately, the pursuit of this research aims to

contribute to a deeper understanding of the capabilities and limitations of reinforcement learning in the context of vehicular control, thereby driving the field toward safer, more efficient, and more adaptable autonomous systems.

The majority of previous studies on reinforcement learning-based vehicle controllers rely on off-the-shelf racing simulations, which inherently constrain their trained models to the specific race tracks and vehicles used during the training process. This lack of customizability inhibits the generalizability of the developed models, restricting their applicability beyond the original training environment.[11][3][33]

Another important motivation comes from the opportunity to collaborate with the Edinburgh University Formula Student team. EUFS is one of the best teams in the United Kingdom's Formula Student autonomous car racing landscape and has become the champion of its autonomous racing event in 2022. Being a member of such a distinguished team provides a unique opportunity to contribute to cutting-edge automotive technologies' development and ongoing success. A significant motivation behind our research on reinforcement learning for autonomous race car controllers stems from the potential to deploy the developed algorithm on the team's real self-driving racecar. This collaboration opportunity would evaluate the practical applicability of our research and demonstrate our commitment to pushing the boundaries of autonomous vehicle technology.

1.2 Project Aims

This project aims to develop a comprehensive framework for designing, implementing, and evaluating a reinforcement learning-based racecar local controller. To achieve this goal, the project will focus on three primary objectives:

- Building a realistic racecar vehicle dynamics simulation
- Constructing a procedural race track generator
- Training a reinforcement learning agent that can safely and efficiently navigate the racecar around the track

The first objective, developing a realistic vehicle dynamics simulation, is crucial for creating an accurate and representative virtual environment in which the reinforcement learning agent can be trained and evaluated. This simulation should incorporate the essential physical properties of a racecar, such as its engine behaviours, suspension system, and tire characteristics, as well as the effects of various forces and torques on the vehicle's motion. By ensuring the fidelity of the simulation, the project aims to facilitate the transfer of the learned control policies to real-world racing scenarios, ultimately increasing the likelihood of successful deployment.

The second objective involves the creation of a procedural race track generator that can produce diverse and challenging race tracks for the reinforcement learning agent to navigate. By generating a wide variety of track layouts with varying degrees of complexity, the generator will allow for the assessment of the controller's robustness and adaptability under different conditions. The variety of track layouts is essential for

establishing the controller's capacity to handle novel and dynamic racing environments, which is a key requirement for real-world deployment. Furthermore, the procedural track generator will provide testing scenarios for benchmarking the performance of the proposed reinforcement learning controller against other control methods, enabling a rigorous and fair comparison.

Finally, the third objective centres around training a reinforcement learning agent to effectively control the racecar to achieve safety and high performance. This objective will involve selecting and implementing appropriate RL algorithms, environment observation and exploration strategies, and reward functions to drive the learning process. Utilizing the realistic vehicle dynamics simulation and the diverse tracks generated by the procedural race track generator, the reinforcement learning agent will be trained to make optimal real-time decisions based on its current state and the surrounding environment. Upon successful training, the agent should be able to navigate the racecar around the track in a safe and efficient manner while demonstrating the ability to adapt to varying track conditions and potential disturbances.

1.3 Report Structure

This report is structured into five main chapters, each dedicated to a specific aspect of our research on reinforcement learning for autonomous racecar local controllers.

In Chapter 2, "Background", we offer an extensive overview of our research's foundations, including the key concepts and techniques in reinforcement learning, racing, and autonomous vehicle local controllers. This chapter aims to provide readers with the necessary context and understanding of the relevant literature and state-of-the-art approaches in these domains, setting the stage for our proposed methodology and its implementation.

Chapter 3, "Environment and Task Setup", delves into the details of our research environment, including the implementation of the vehicle dynamics simulation and the procedural race track generator. We discuss the design and development of these essential components, ensuring that our training environment is realistic, diverse, and able to foster the learning of a generalized agent.

Chapter 4, "Methodology", outlines our reinforcement learning approach, specifically addressing the observation space, reward function, and action space. We explain the rationale behind the design choices made in this chapter and how they contribute to the effectiveness of our agent's training and its subsequent performance on the race track.

Chapter 5, "Experiment and Results", presents our experiments' results and our trained agent's performance. We detail the various stages of training, the impact of hyperparameter tuning, and the result of the agent's ability to control the simulated race car. This chapter highlights the success of our approach and its potential application in real-world autonomous vehicle control systems.

Chapter 2

Background

2.1 Reinforcement Learning

Reinforcement learning (RL) is a subfield of artificial intelligence and machine learning that focuses on training agents to make decisions by interacting with their environment. The central idea behind reinforcement learning is that an agent learns to map its current state to an optimal action by maximizing a cumulative reward signal, which gives feedback on the agent's performance[29]

In a typical reinforcement learning setup, an agent interacts with an environment through discrete time steps. At each time step, the agent perceives the current state of the environment, selects an action from its available action space, and executes the chosen action. In response, the environment transitions to a new state, giving the agent a scalar reward signal. The agent's goal is to learn a policy, which is a mapping from states to actions, that maximizes the expected cumulative reward over time[25]

Reinforcement learning algorithms employ a combination of value estimation and policy optimization techniques to maximize the cumulated reward. Value estimation methods estimate the expected cumulative reward obtained by taking a particular action in a given state. At the same time, policy optimization techniques update the agent's policy to select actions that lead to higher estimated values. A common RL value estimation approach uses temporal-difference (TD) learning algorithms, such as Q-learning[6] and SARSA[35], which update value estimates incrementally based on the difference between successive state-action value predictions. On the other hand, policy optimization methods, such as policy gradients[30] and actor-critic[17] algorithms, directly optimize the policy by computing and applying gradients of the objective function for the policy parameters.

In recent years, incorporating deep neural networks in reinforcement learning, known as deep reinforcement learning (DRL)[21], has led to significant advancements in the field. Deep neural networks can be used as function approximators to represent complex policies and value functions, enabling the RL agent to learn from high-dimensional sensory input and tackle problems with large state and action spaces. This development has resulted in breakthroughs in various applications, from playing video games[23]

and robotics[13] to natural language processing[31] and autonomous vehicles[1].

2.2 The Racing Problem

Time-trial-style car racing in closed circuits presents a challenging problem due to the complex interplay of various factors that contribute to the overall performance of the racecar. In this racing format, each vehicle competes individually against the clock, attempting to complete a set number of laps around a closed circuit in the shortest time possible. The focus of the competition shifts from direct driver-to-driver interaction to the optimization of individual lap times, placing a premium on vehicle control, strategy, and the efficient navigation of the track[11].

Several aspects contribute to the complexity of time-trial-style racing in closed circuits. First, the dynamic nature of the vehicle's motion and the intricate balance between various forces, such as aerodynamic downforce, tire grip, and engine torque, make predicting and controlling the car's behaviour challenging. Accurate modelling of these forces is essential for developing effective control strategies, yet the inherent nonlinearity and uncertainty in their interactions present significant obstacles.

Second, the track layout determines the optimal racing line and trajectory, influencing the vehicle's speed and cornering capabilities. The variability of track features, such as turns, elevation changes, and surface conditions, demands high adaptability from both the driver and the vehicle's control systems. Furthermore, varying constraints, such as track boundaries and dynamic obstacles like debris, add complexity to the problem[4].

Lastly, the time-sensitive nature of time-trial racing necessitates exploring control strategies that prioritize safety and speed. This delicate balance requires the development of sophisticated algorithms that can account for the trade-offs between aggressive driving manoeuvres and the risk of losing control or causing damage to the vehicle. Finding the optimal racing strategy requires integrating advanced modelling techniques, adaptive control algorithms, and robust optimization methods.

2.3 Racecar Local Controller

In the domain of autonomous vehicular control, A local controller manages a vehicle's dynamics and actuation in real time, depending on the current conditions of the vehicle and its surrounding environments. The local controller's primary goal is to optimise several performance measures, including speed, acceleration, and cornering ability, to preserve the vehicle's control while maintaining the desired trajectory[32]. This is done by continuously monitoring the vehicle's sensor detections and states and applying control inputs, such as throttle, brake, and steering adjustments, to maintain the desired performance characteristics.

2.4 Related Works

2.4.1 Super-Human Performance in Gran Turismo Sport Using Deep Reinforcement Learning [11]

In this study, the authors demonstrate the capacity of a deep reinforcement learning agent to attain super-human lap times on specific race tracks and vehicles within the Gran Turismo Sport simulation environment. The researchers employ a state observation technique called edge distance measurements, which emulates the behaviour of lidar sensors through ray-casting, enabling the agent to extract proximate race track features. The agent's Soft Actor-Critic (SAC) reinforcement learning network receives input comprising measured distances from the car to race track edges, current speed, acceleration information, and a reward function proportional to the distance traversed by the car along the track's centre line. The inference network's output directly controls the car's steering, throttle, and brake commands. The trained agent outperforms human competitors in the given race conditions and exhibits awareness and application of common racing knowledge.

2.4.2 Outracing champion Gran Turismo drivers with deep reinforcement learning [33]

This research presents a deep reinforcement learning agent trained to compete with real-time players, in contrast to the previous study's static time-trial scenario. The proposed method concentrates on overtaking opponents and evading unlawful collisions with other race cars. The agent employs a simpler perception model that directly accesses the ground truth data of the race track a few seconds ahead of its current position without considering the car's facing angle or perceivable distance. Information about the agent's nearest opponents is also input into its Quantile Regression Soft Actor-Critic (QR-SAC) training network. A unique curriculum sequence of diverse training environments is utilised, from basic static single-agent time trials to multi-agent competitive scenarios starting at different track locations.

2.4.3 High-speed Autonomous Drifting with Deep Reinforcement Learning [3]

This paper explicitly addresses the drifting problem in car racing. Rather than minimising side-slipping like most human racers, the agent is rewarded for executing prolonged, controlled drifting. The training employs a reward allocation function proportional to the vehicle's lateral speed and wheel-slip duration. The agent is primarily trained using an imitation learning approach in the CARLA vehicle simulation environment, employing a SAC reinforcement learning network and a trajectory driven by a human drifting expert. The trained agent can maintain controlled drifting on a given set of tracks and vehicles. The performance of other reinforcement learning algorithms, including DDPG and DQN, is assessed and compared within the same training scenario.

2.5 Potential Improvements of Related Works

2.5.1 Improving Universality

Although the studies mentioned above yield remarkable results, most share a common limitation: their trained models apply only to the specific race track and vehicle on which they were trained. This constraint may stem from the off-the-shelf racing simulators used, which offer a preset range of race tracks and vehicles, lack variability, and thus risk overfitting. To address this issue, researchers could develop a customisable racing simulator, enabling them to randomise vehicle dynamics and track layouts for each training session. A customisable racing simulator could produce more robust, versatile, and useful agents combined with larger training networks.

2.5.2 Minimizing Prior Knowledge Guidance

While incorporating domain-specific prior knowledge into training can expedite convergence, it may also constrain the agent's potential performance. Instances of such limitations include rewarding agents for adhering to the track's centre line or following a human-generated trajectory. These methods counteract the sparse reward nature of racing problems. Future research should explore more natural and heuristic approaches. [10]

2.5.3 Optimizing Observation Encoding

By refining state observation encoding, agents may train and converge more rapidly. Ideal state observation encoding should be consistent, normalised, and highly pertinent to the agent's decision-making process. To incorporate ahead race track location and boundary information into the agent, the observation data should be filtered, preprocessed, and normalised to facilitate the agent's comprehension and interpretation. Potential data preprocessing methods may involve geometric transformations and optimisation techniques such as racing line calculation. Employing more sophisticated observation encoding could lead to more efficient learning processes and ultimately result in improved agent performance.

Chapter 3

Environment and Task Setup

3.1 Overview

The importance of developing a comprehensive environment and simulation setup is paramount in training and evaluating a reinforcement learning-based racecar local controller. By constructing a realistic vehicle dynamics simulation, we can accurately represent the complex physical interactions and the various subsystems that influence the racecar's behaviour. This level of realism is crucial for ensuring that the learned control policies are transferable to real-world racing scenarios, thereby increasing the likelihood of successful deployment and the possibility of discovering new racing strategies.

Additionally, the procedural race track generation method facilitates the creation of a wide array of diverse and challenging race tracks, which is essential for contributing to the controller's robustness, adaptability, and performance under various conditions. The ability to generate a multitude of track layouts and configurations enables the reinforcement learning agent to be exposed to a broader range of racing situations, ultimately enhancing its capacity to handle novel and dynamic racing environments. The combination of the vehicle dynamics simulation and the procedural race track generation provides a rigorous and controlled framework for training and benchmarking the performance of the proposed reinforcement learning controller.

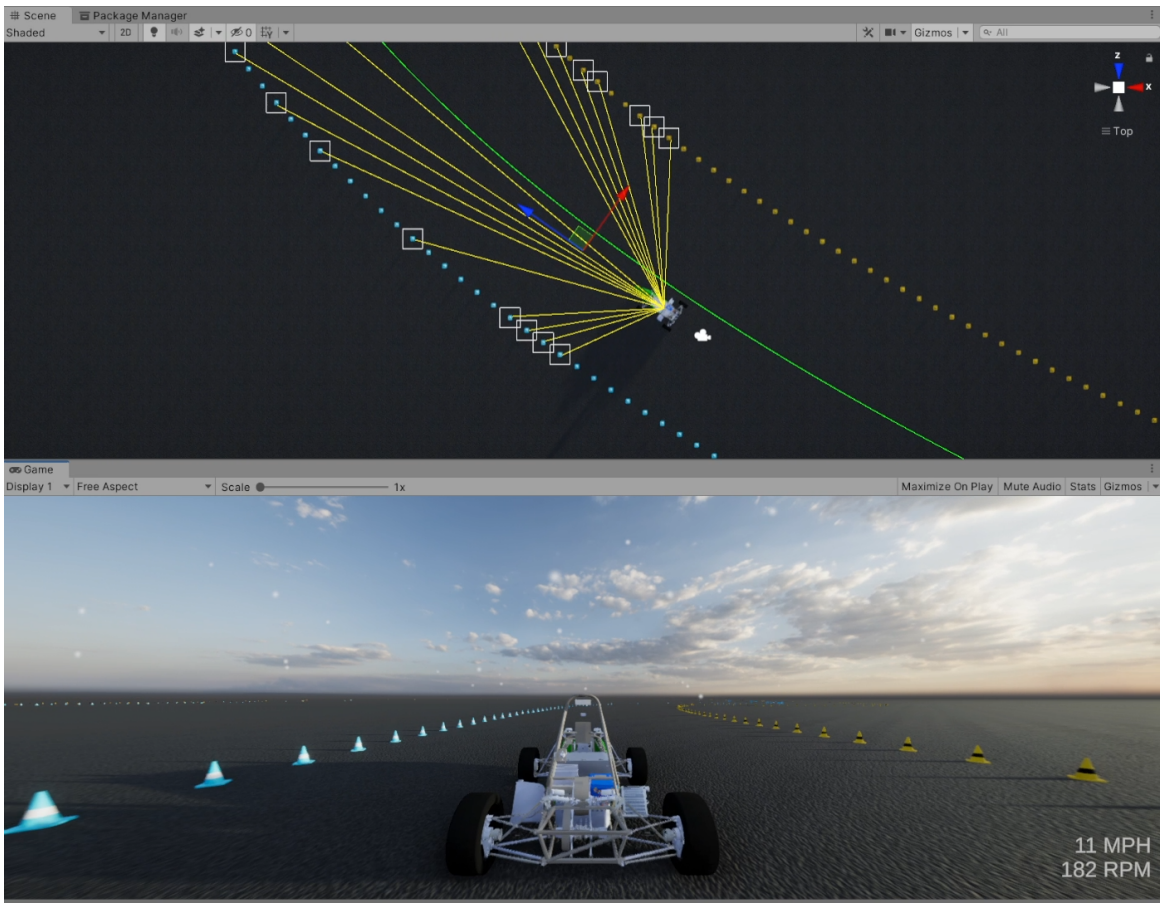

Figure 3.1: A Screen Shot of the Simulation Environment

3.2 Vehicle Dynamics Simulation

To realistically simulate the complex dynamics of a racecar, the modelling of various components, such as the car's body, engine, suspension system and wheels, plays a critical role in accurately representing the vehicle's behaviour. The simulation considers many vehicle parameters and inputs. Examples include wheelbase, rear track, turn radius, air friction coefficient, suspension stiffness, and engine torque curve. By accounting for these factors, the system effectively captures the nuanced interaction between different components, ensuring that the simulation accurately represents the physical reality of a vehicle's motion.[12]

3.2.1 Numerical Integration

Numerical integration plays a vital role in physics simulations, as it allows for approximating continuous functions and evaluating definite integrals using discrete data points [9]. In the case of simulating car physics, numerical integration is utilised to compute the motion and internal states of the racecar, considering various forces and torques acting on the vehicle and updating its position, velocity, orientation, suspension travel, and engine speeds over time.

The Euler method is one common approach to numerical integration employed in physics simulations. In the context of simulating car physics, the equations of motion can be represented as a system of first-order ordinary differential equations. They can be applied to update the vehicle's state over time. The basic formula for the Euler method is as follows:

$$x(t+\Delta t) \approx x(t) + \Delta t \cdot x'(t) \tag{3.1}$$

Where x(t) is the state variable at time t, x'(t) is the derivative of the state variable for time at time t, and delta t is the time step. This method is utilised in simulating various vehicle components, including the car's body movement and rotation, wheels rotation, suspension travel, and engine revolutions.

3.2.2 Ackermann Steering Geometry

The steering system plays a crucial role in simulating the vehicle's dynamics. The Ackermann steering geometry is employed in the simulation, which ensures that all wheels rotate around a common centre of curvature[22]. This geometry prevents tire scrubbing and provides a smooth steering experience. The steering input, which the user or an AI controller can provide, directly affects the calculation of the steering angles for each wheel.

To determine the steering angles, the system uses the Ackermann constants derived from the vehicle's wheelbase and rear track. The wheelbase is the distance between the front and rear wheel axles, while the rear track refers to the distance between the centres of the rear wheels. These parameters affect the vehicle's turning status and handling characteristics.[8]

$$\cot(\delta_L) - \cot(\delta_R) = \frac{RT}{WB} \tag{3.2}$$

$$\delta_{Ack} = \frac{\delta_{in}}{\gamma} \tag{3.3}$$

$$\delta_L = \tan^{-1}\left(\frac{WB\tan(\delta_{Ack})}{WB + 0.5RT\tan(\delta_{Ack})}\right) \tag{3.4}$$

$$\delta_R = \tan^{-1}\left(\frac{WB\tan(\delta_{Ack})}{WB - 0.5RT\tan(\delta_{Ack})}\right) \tag{3.5}$$

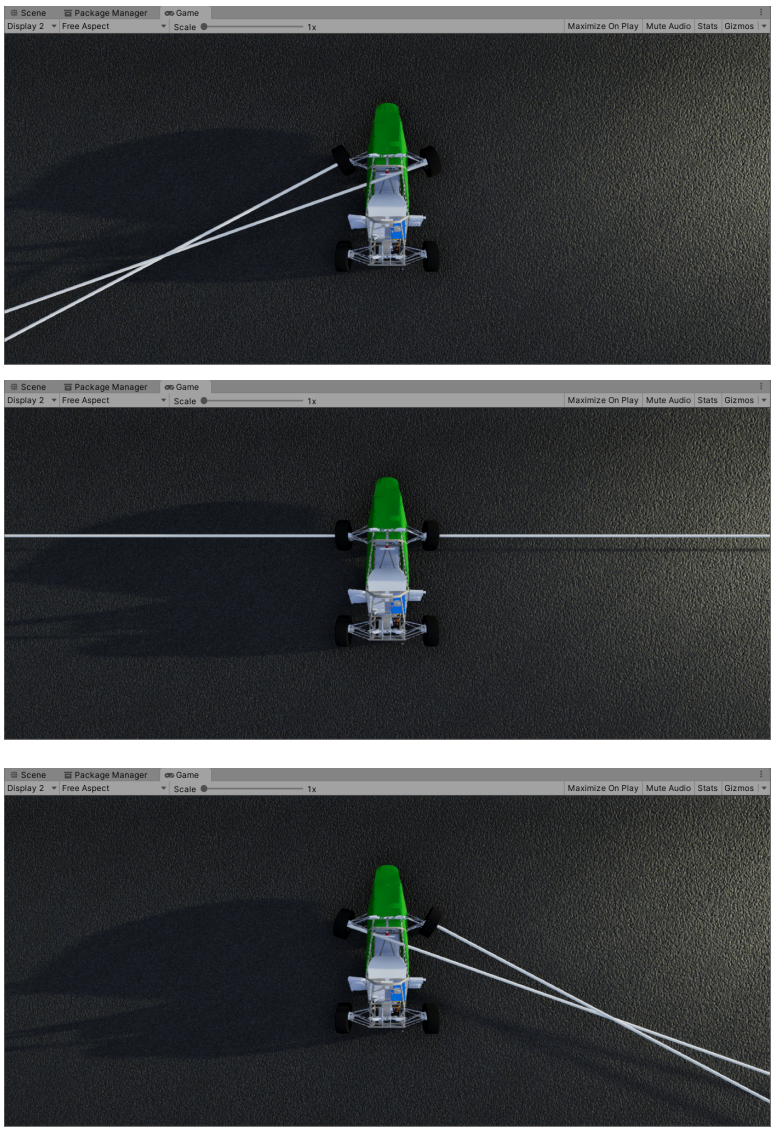

Figure 3.2: Visualisation of the Ackermann Steering Geometry

The Ackermann constants are then used in a trigonometric calculation to determine the steering angles for each wheel based on the steering input provided by the driver or an AI system. This method of calculating steering angles ensures that the simulation effectively models the correct steering behaviour for Racecars.

3.2.3 Engine and Transmission

The engine simulation is a vital component of the vehicle's dynamics simulation. It incorporates a torque curve, gear ratio, differential, and transmission efficiency to determine the drive torque applied to the drive wheels [34]. The torque curve is a function that represents the engine's torque output at different RPMs, and it is essential for simulating realistic engine behaviour.

The gear ratio and differential work together to transfer the engine's torque to the

drive wheels. The gear ratio determines how much the engine's torque is multiplied or reduced before being transmitted to the wheels. The differential helps distribute the torque evenly between the drive wheels, allowing them to rotate at different speeds when necessary. The transmission efficiency is a factor that accounts for power losses in the transmission system.

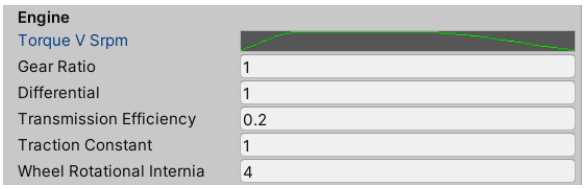

Figure 3.3: Engine Simulation Parameters

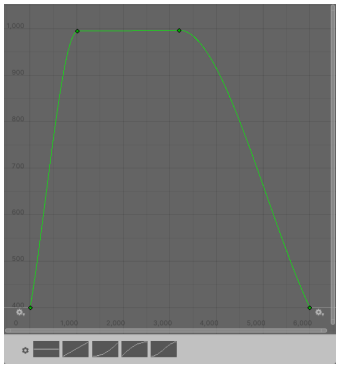

Figure 3.4: Engine Torque vs RPM Graph

The simulation also calculates the traction torque, rolling friction torque, and brake torque applied to the wheels. The traction torque is the force that propels the vehicle forward, while the rolling friction torque opposes the vehicle's motion and is caused by the tire's contact with the ground. Brake torque is the force generated by the braking system to slow down or stop the vehicle.

These torques influence the angular velocity and current RPM of the wheels. The simulation calculates the angular velocity of wheels based on the torques and the wheel's moment of inertia, and the current RPM is derived from the angular velocity. By considering these factors, the system accurately represents the engine's influence on the wheels and the vehicle's dynamics.

3.2.4 Suspension System

The vehicle's suspension is a crucial component that significantly impacts the vehicle's handling characteristics and overall performance. In the simulation, the suspension is modelled using a spring-damper system with adjustable parameters, including rest length, spring travel, spring stiffness, and damper stiffness. The rest length represents the uncompressed length of the spring, while the spring travel denotes the maximum displacement that the spring can undergo. The spring stiffness and damper stiffness are

coefficients that determine the forces generated by the spring and damper, respectively [16].

$$F_k = -kx \tag{3.6}$$

$$F_d = c\frac{dx}{dt} \tag{3.7}$$

The suspension forces, which consist of the spring force and damper force, are calculated based on the current length and velocity of the spring. The spring force is a function of the spring stiffness and the spring's displacement from its rest length, while the damper force is a function of the damper stiffness and the spring's velocity. These forces work together to provide a responsive and realistic suspension behaviour that reacts to different road surfaces and driving conditions. By adjusting these parameters, the system can simulate various suspension configurations and accurately represent their behaviour under various driving conditions.

The suspension forces are then applied to the vehicle's rigid body at the point of contact with the ground. This application of forces ensures that the vehicle's body responds appropriately to the forces generated by the suspension, simulating the interaction between the vehicle's body and the suspension system. As a result, the suspension system plays a crucial role in accurately simulating the vehicle's dynamics, providing a realistic representation of how the vehicle reacts to various road surfaces, inputs, and weight transferring and body rotations.

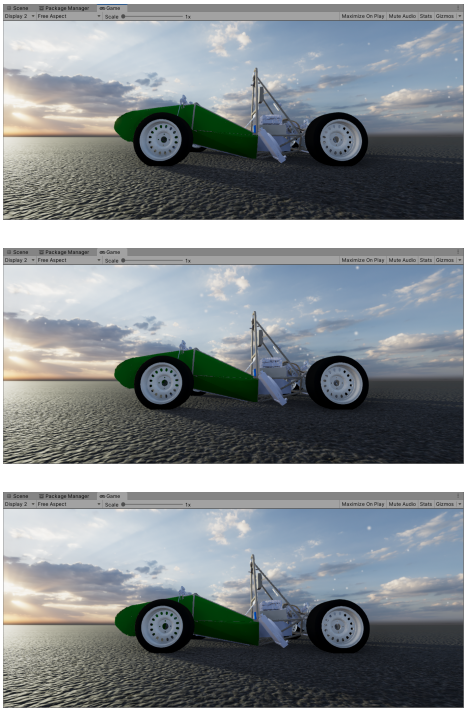

Figure 3.5: Visualisation of the suspension system's subtle influence on the vehicle body

3.2.5 Tire Friction Modelling

The lateral and longitudinal tire forces are modelled using a combination of slip angle, slip ratio, and friction coefficient, which are essential in determining the vehicle's traction and handling characteristics. These parameters capture the complex behaviour of the tire as it interacts with the road surface, which directly influences the vehicle's performance.

The slip angle is the angle between the tire's direction and its actual velocity, and it is used to determine the lateral forces acting on the tire. The slip angle is essential for simulating the tire's lateral grip and cornering capabilities.

The slip ratio represents the difference between the wheel's rotational and ground speeds. This ratio calculates the longitudinal forces acting on the tire, including acceleration and braking forces. The slip ratio is critical for simulating the tire's traction and braking performance.

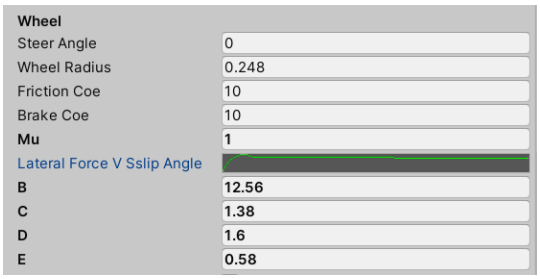

Figure 3.6: Tire Friction Modelling Parameters

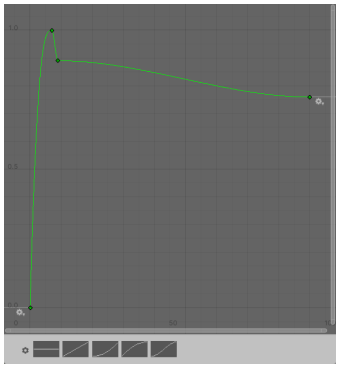

Figure 3.7: Tire Lateral Force VS Slip Angle Graph

The friction coefficient is another vital parameter that influences tire forces. It represents the tire's grip on the road surface and affects lateral and longitudinal forces. The friction coefficient can change based on factors such as tire compound, road surface, and temperature.

$$F_x = f(\kappa, F_z) = F_z \cdot D \cdot \sin(C \cdot \arctan B\kappa - E[B\kappa - \arctan(B\kappa)]) \tag{3.8}$$

The horizontal and lateral tire forces are calculated using these parameters combined

in the Pacejka Magic Formula tire model that describes the relationship between these parameters and the forces generated by the tire [24]. The calculated tire forces are then applied to the relative wheel positions on the vehicle's rigid body, ensuring that the tire's performance characteristics directly influence the vehicle's motion and response to inputs. By accurately representing this relationship, the simulation can effectively capture the tire's behaviour under various driving conditions and road surfaces, such as dry, wet, or icy pavement and different suspension configurations.

3.2.6 Aerodynamics Approximation

The air resistance force is another critical factor in vehicle dynamics simulation, as it significantly impacts the vehicle's acceleration and top speed. The simulation calculates the air resistance force based on the air friction coefficient, frontal area, air density, and the square of the vehicle's velocity [15]. The air friction coefficient is a dimensionless parameter representing the resistance of the vehicle's shape to the airflow. At the same time, the frontal area is the cross-sectional area of the vehicle that faces the direction of travel.

$$F_{air} = \frac{1}{2}\rho C_D A v^2 \tag{3.9}$$

The air density parameter is a pre-computed variable affected by the ambient temperature, pressure, and altitude, and it directly affects the magnitude of the air resistance force. By accounting for these parameters, the system can accurately calculate the air resistance force experienced by the vehicle as it travels at various speeds and through different environments. This force is then applied to the vehicle's rigid body in the opposite direction of its velocity, ensuring that the simulation effectively represents the influence of air resistance on the vehicle's performance.

3.3 Procedural Race Track Generation

3.3.1 Race Tracks

Race tracks are specialized venues designed to host various racing competitions, including automobiles, motorcycles, and even horse racing. These tracks are meticulously engineered to meet the requirements of the racing events they host, with consideration given to aspects such as track surface, layout, safety features, and spectator amenities. The primary objective of a race track is to provide a challenging and exciting environment for competitors and spectators while ensuring the safety of all participants.

Race tracks are generally characterized by a closed-circuit design, forming a continuous loop without any intersecting paths. This allows for a clearly defined start and finish line, consistent lap timing and performance measurement. The length, width, and overall layout of race tracks can vary significantly depending on the type of racing event and the desired difficulty level [19].

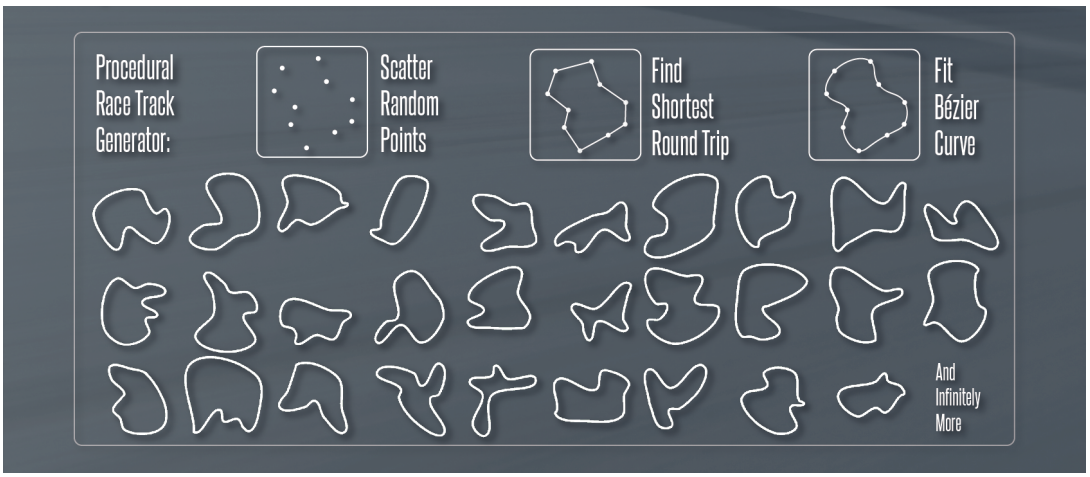

Figure 3.8: Procedural race track generation system overview

A key aspect of race track design is incorporating various track features that present unique challenges to drivers or riders. These may include a combination of straights, curves, and corners. Straights are the track sections where vehicles can reach their top speeds due to the lack of curvature. These stretches are crucial for overtaking opportunities and strategically planning one's approach to upcoming corners or curves. The length and number of straights on a track can significantly impact the nature of the competition and the vehicles' performance characteristics.

Curves and corners, on the other hand, require precise handling and tactical manoeuvres from the drivers or riders. These track features are defined by their geometry, which includes factors such as radius, angle, and direction. The radius of a curve refers to the distance between its centre and the outer edge of the track. A larger radius results in a gentler curve, allowing for higher speeds, while a smaller radius demands more significant adjustments to a vehicle's speed and trajectory. The angle of a corner represents the change in direction it causes, with a greater angle requiring a more pronounced steering input and braking. Corners can also be classified based on their direction, as either left-hand or right-hand turns.

The transitions between straights, curves, and corners form a race track's overall layout and flow. Designers must carefully balance the demands of high-speed straights with the technical challenges posed by curves and corners and consider the potential overtaking opportunities and strategic implications of each feature. The track's geometry can significantly influence a race's dynamics, with tighter, more technical circuits favouring vehicles with superior handling and agility. In contrast, more open and fast tracks benefit those with higher top speeds and aerodynamic efficiency.

3.3.2 Pseudo-Random Number Generator

Random number generators (RNGs) are algorithms or devices designed to produce a sequence of numbers that lack any discernible pattern or predictability. These numbers exhibit statistical randomness, meaning that each number in the sequence is independent of its predecessors and has an equal probability of appearing. RNGs are vital in

numerous fields, including cryptography, computer simulations, and procedural content generation [2].

Two primary categories of random number generators are true random number generators (TRNGs) and pseudo-random number generators (PRNGs). TRNGs derive their randomness from physical processes, such as electronic noise, radioactive decay, or atmospheric phenomena. These generators capitalise on the inherent unpredictability of such processes to produce truly random numbers. However, TRNGs can be slower, more expensive, and may require specialised hardware, which can limit their practical applications.

In contrast, PRNGs are algorithmic approaches that employ deterministic processes to generate seemingly random sequences of numbers. PRNGs utilise an initial value, the seed, to determine the starting point for the number sequence. Given the same seed, PRNGs produce the same sequence of numbers, allowing for reproducibility in simulations or other applications. Although PRNGs may not offer true randomness, their output can approximate the properties of random sequences, provided that the algorithm is well-designed and properly initialised.

Procedural generation is a technique commonly used in computer graphics, video games, and simulations, where content is created algorithmically rather than manually. By utilising random number generators, procedural generation can generate diverse and seemingly unique results. In this context, PRNGs are particularly advantageous, as their deterministic nature allows developers to reproduce specific configurations or tweak their parameters to fine-tune the generated content.

3.3.3 Random Point Scattering

The first essential step for the procedural race track generation is to utilise pseudo-random number generators (PRNGs) to scatter random points on a two-dimensional plane to generate a set of coordinates (x, y) that represent points in a Cartesian coordinate system with two separate PRNGs, one for each coordinate. First, the PRNG responsible for the x-coordinate is initialised with a seed value and then used to produce a sequence of pseudo-random numbers within the desired range of x values. Simultaneously, a second PRNG, initialised with a different seed, generates a sequence of pseudo-random numbers within the desired range of y values. Combining the x and y values produced by these PRNGs creates a set of two-dimensional points with a seemingly random distribution.

3.3.4 The Travelling Salesman Problem

The Traveling Salesman Problem (TSP) is a computational challenge that has captured the attention of researchers and practitioners from various fields, such as operations research, computer science, and applied mathematics. The TSP's main objective is to find the shortest route that visits a set of given points and returns to the starting point. This problem has broad applicability in logistics, manufacturing, and network design.

The TSP is formally defined as follows: Given a set of points (or nodes) and a distance

matrix representing the pairwise distances between these points, the objective is to find a Hamiltonian cycle—a closed loop that visits each point exactly once—such that the total distance is minimised. There are two primary categories of TSP - symmetric and asymmetric - depending on whether the distances between pairs of points are identical in both directions or not [18].

The TSP is a challenging problem due to its computational complexity, which is classified as NP-hard. The number of permutations grows factorially, doing an exhaustive search for an optimal solution infeasible for moderately-sized instances. This means that an efficient algorithm capable of optimally solving all instances of the problem is unlikely to be developed without significant advancements in computing.

Researchers have developed heuristic and metaheuristic algorithms to address these computational challenges to find near-optimal solutions within a reasonable time frame. Heuristic methods, such as the nearest neighbour and minimum spanning tree, exploit problem-specific knowledge to construct a solution. Metaheuristic techniques, such as simulated annealing, genetic algorithms, and ant colony optimisation, explore the solution space through stochastic processes. While these methods do not guarantee an optimal solution, they often yield high-quality results relatively quickly, making them well-suited for practical applications.

In procedural race track generation, solving the TSP for a set of randomly scattered points on a 2D plane can help determine the optimal layout for the track. By minimising the total distance required to traverse the entire set of points, the track's design can be optimised and made into a closed loop without any intersecting paths, forming a closed circuit.

3.3.5 Genetic Algorithm

In our work, we utilised the genetic algorithm (GA), a population-based metaheuristic, to tackle the Traveling Salesman Problem (TSP) as a critical component of our procedural race track generation process. Genetic algorithms draw inspiration from natural selection and evolution processes, offering an adaptable and efficient method for searching large and complex solution spaces [18]. In this context, the GA proved effective in determining near-optimal layouts for the race track by optimising the order in which the randomly scattered points on a 2D plane were connected.

Implementing the GA for the TSP involved several key steps, beginning with creating an initial population. Each population member represented a candidate solution, encoded as a permutation of the points. We employed various initialisation strategies to generate a diverse population, including random permutations and the insertion of some heuristically-constructed solutions.

Following the initialisation, the GA proceeded with iterative selection, crossover, and mutation cycles. During the selection process, we adopted a fitness-based approach, in which individuals with lower total distances and, thus, higher fitness were more likely to be selected for reproduction. This allowed us to maintain a balance between exploration and exploitation, preserving reasonable solutions while continuing to search for better alternatives.

Crossover, also known as recombination, played a crucial role in generating offspring by combining the genetic information of two parent solutions. We used order-based crossover operators designed explicitly for the TSP to ensure the generation of valid offspring permutations. These operators maintained each parent's relative order of points while creating new potential solutions.

Mutation operators were also employed to introduce random alterations to the offspring solutions, adding diversity to the population and mitigating premature convergence. We implemented various mutation operators, such as swap, insertion, and inversion, each of which modified the order of the points in the offspring solutions in distinct ways.

The GA iterations continued until a predefined stopping criterion was reached, such as a maximum number of generations or a target solution quality. Upon termination, the best solution found by the algorithm was used as the layout for the race track. Our implementation of the GA exhibited a strong capability to solve the TSP in the context of procedural race track generation, producing high-quality race track layouts within reasonable time frames.

3.3.6 Bézier Curves

Bezier curves, named after the French mathematician Pierre Bezier, have found wide-ranging applications in various domains, such as computer graphics and design, due to their mathematical elegance and ability to generate smooth curves. These parametric curves are defined by a set of control points, which in turn determine the shape and properties of the curve. Bezier curves can be constructed of various orders, with quadratic and cubic Bezier curves being the most common. While higher-order curves offer greater control and flexibility in shaping the curve, they also introduce increased complexity in calculations [5].

One of the key properties of Bezier curves is their containment within the convex hull formed by their control points. This ensures that the generated curves do not deviate significantly from the desired path. Bezier curves also possess a local control property, which entails modifying a control point only influences the curve in its vicinity. The combination of these properties and the ability to create continuous and differentiable curves make Bezier curves particularly well-suited for generating smooth and visually appealing paths.

In the context of race track generation, Bezier curves are an advantageous choice due to their smoothness, local control, and ability to generate continuous paths. A seamless and engaging race track layout can be created using Bezier curves to interpolate between the points.

After obtaining the near-optimal layout for the race track using the genetic algorithm to solve the TSP, we employed Bezier curves to interpolate between the scattered points and generate smooth, realistic race track segments. This process was critical to our procedural race track generation pipeline, ensuring that the generated tracks offered a continuous and realistic racing experience. We first created a sequence of control points based on the TSP solution to achieve this, accounting for the geometric properties required to produce a smooth race track.

For each pair of adjacent points in the layout, additional control points were strategically placed to shape the curve and maintain desired geometric properties, such as tangent directions at the start and end of each curve segment. The placement of these additional control points was governed by a set of heuristics and constraints that aimed to preserve the overall structure of the layout while producing a smooth race track. By considering factors such as curvature, the distance between points, and the desired racing line, we ensured that the generated curves were visually appealing and conducive to an engaging racing experience.

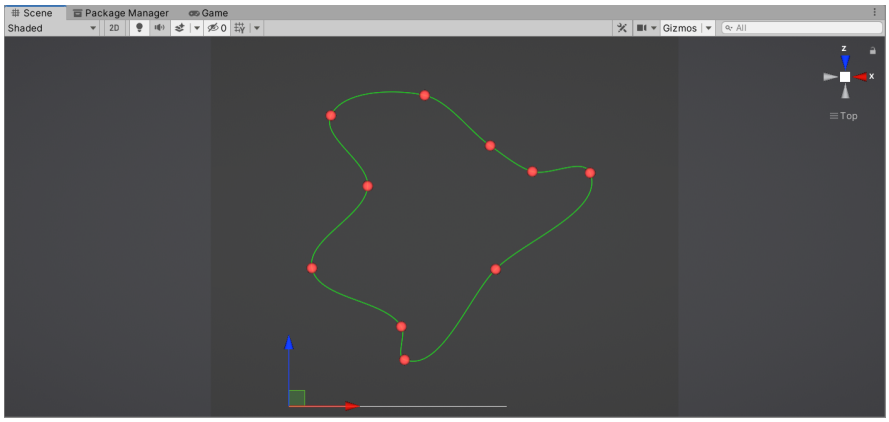

Figure 3.9: Visualisation of Bezier curves race track fitting

Cubic Bezier curves offered sufficient flexibility to generate the necessary curvature while maintaining computational efficiency. Utilising cubic Bezier curves, defined by four control points per segment, we created the continuous path between the points in the TSP solution. By connecting the Bezier curve segments and ensuring continuity in the first and second derivatives at the joining points, we produced a seamless race track with realistic turns and transitions.

In the final stage of the race track generation process, we connected the Bezier curve segments to form a smooth and continuous path. This ensures that the first and second derivatives are continuous at the joining points of the curve segments, leading to seamless transitions between different track sections. This continuity was crucial for providing a realistic racing experience, allowing racers to maintain their momentum while negotiating the track's turns and twists.

3.3.7 Race Track Elements Placement

3.3.7.1 Traffic Cones

In order to provide a clear and visually coherent delineation of the race track boundaries, we used the normal and tangent information extracted from the generated race track Bezier curve to place a series of traffic cones alongside the track strategically. This step enhanced the visual appeal of the track and served as an essential guide for racers, helping them navigate the course and understand the boundaries of the racing surface.

Utilising the mathematical properties of the Bezier curve, we calculated the tangent vector at various points along the curve. The tangent vector is the curve's first derivative and represents the curve's direction at each sampled point. This information was essential for determining the orientation of the traffic cones, ensuring they were placed consistently with the direction of the race track. By normalising the tangent vector, we obtained a unit vector that provided a consistent reference for cone placement.

Subsequently, we calculated the normal vector by taking the orthogonal vector to the tangent vector. This normal vector represented the direction perpendicular to the curve at each sampled point and was used to offset the cone positions from the main path of the track. By selecting an appropriate distance from the main path, we ensured that the cones were placed at a consistent offset from the race track, effectively marking the boundaries of the racing surface.

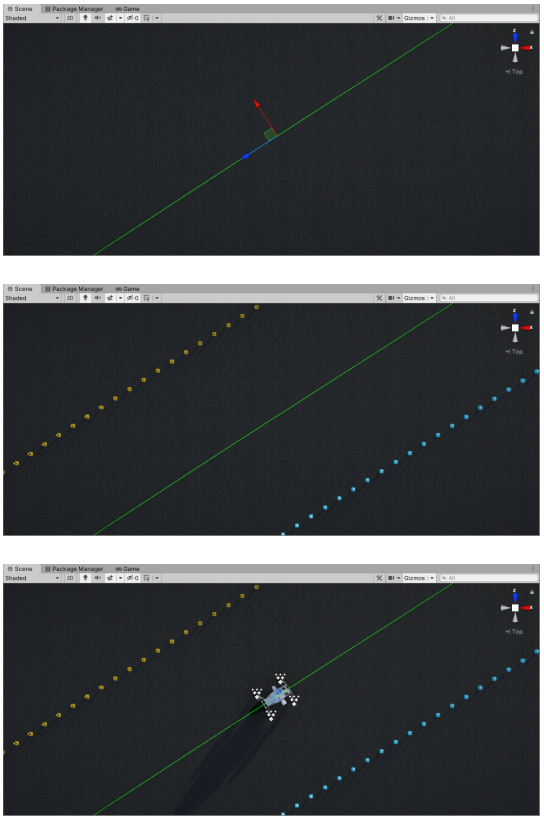

Figure 3.10: Visualisation of the race track elements placement

To evenly distribute the traffic cones along the race track, we employed a sampling strategy based on the arc length parameterisation of the Bezier curve. This approach allowed us to maintain a consistent spacing between the cones, providing a visually pleasing and easily interpretable boundary for the racers. By varying the density of the cones based on the curvature and complexity of the track sections, we ensured that the cones provided clear guidance for the racers, especially in more challenging areas of the course, like tight corners.

3.3.7.2 Racecar spawning

Once the race track has been procedurally generated using Bezier curves and traffic cones placed to mark the track boundaries, the next step is to spawn the race car onto the track, ensuring it is correctly oriented and positioned. To achieve this, we selected a suitable starting point along the Bezier curve and calculated the tangent vector at that point. This tangent vector indicates the direction of the track, enabling us to orient the race car either facing forwards or backwards, depending on the desired initial racing direction. By aligning the race car's forward vector with the tangent vector, we ensured that the car was correctly oriented along the track direction, thus allowing for a seamless commencement of the racing simulation.

In addition to spawning the race car on the generated track, it is crucial to initialise the vehicle dynamics simulation parameters for a realistic and immersive racing experience. To do so, we set the initial values for various parameters such as wheelbase, rear track, turn radius, air friction coefficient, frontal area, air density, suspension properties, and engine behaviour. By carefully calibrating these parameters based on the specific attributes of the race car being simulated, we could ensure high fidelity in the vehicle's motion and response to various inputs, such as steering, throttle, and braking. This level of detail in the vehicle dynamics simulation significantly contributed to the overall realism of the racing simulation.

Chapter 4

Methodology

4.1 The Reinforcement Learning Problem

Our study aimed to investigate the application of reinforcement learning (RL) techniques for driving the racecar vehicle dynamics model we developed on the procedurally generated race tracks. To frame this as an RL problem, we first needed to define the essential components of the learning task, including the observation, reward function, and action space. Our goal was to design an autonomous driving agent capable of learning to navigate the tracks efficiently and safely, using a suite of custom-designed observation methods for environment perception, reward function for behaviour encouragement and action mapping for decision actuation and execution.

4.2 Observations

4.2.1 LiDAR and Ray-casting

Light Detection and Ranging (LiDAR) is a widely utilised remote sensing technique in autonomous driving. LiDAR enables accurate assessment of distances between objects in the surrounding environment by releasing laser pulses, measuring the time it takes for the light to reflect back to the sensor, and then estimating the distance to nearby objects. LiDAR offers useful information about the area around the vehicle in the context of autonomous driving, enabling very accurate and reliable obstacle identification and tracking [26].

An interesting parallel can be drawn between LiDAR in autonomous driving and ray-casting techniques in computer graphics. Ray casting is a rendering technique that involves tracing rays from a viewpoint into the scene and computing the intersections between the rays and the objects in the environment. This approach allows for determining the visible surfaces in a 3D scene and is widely used in various computer graphics applications, such as video games and robotic simulations.

In our implementation, we employed a method inspired by LiDAR and ray-casting principles to detect the cones surrounding the race car in a 2D planar environment.

The method involved emitting rays from the vehicle in multiple directions, extending outwards in a planar fashion, and computing the intersections between the rays and the cones' positions. This process enabled us to determine the distances between the vehicle and the cones, which were then incorporated as part of the observation provided to the reinforcement learning agent. This approach not only maintained a reasonable level of computational efficiency in the simulation but also provided a realistic representation and simulation of the vehicle's onboard sensors that could potentially be applied in real-life scenarios.

4.2.2 Observation Space Processing

Utilising raw LiDAR detection data as the observation space in reinforcement learning applications can be problematic for several reasons. Firstly, the raw data is often not normalised, which means that the values representing the distances between the vehicle and the cones can vary significantly. This can pose challenges for reinforcement learning algorithms, as they rely on consistent and normalised input values to learn and make decisions effectively. Secondly, relying solely on raw LiDAR detection data can lead to inconsistencies in the observation space, where certain cones might be missed by the detection algorithm, especially in challenging track conditions such as tight corners and hairpins.

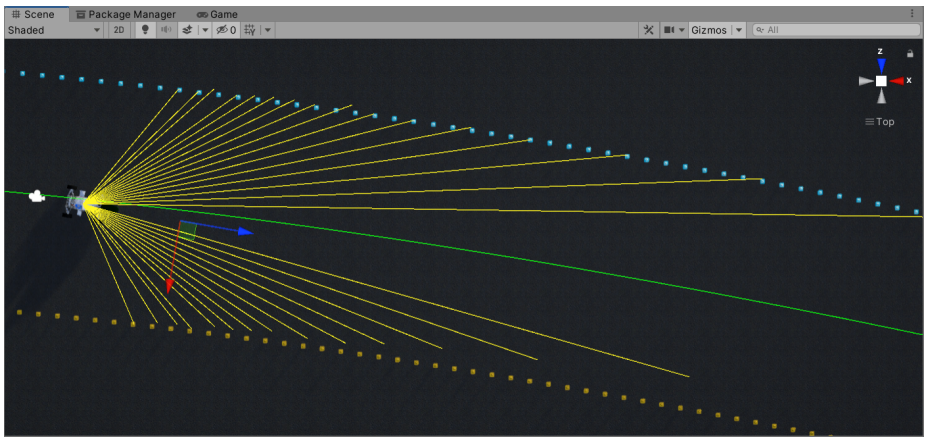

Figure 4.1: Visualisation of processed LiDAR observation input

In our implementation, we addressed the issues of normalisation and consistency by dividing the vehicle's front-facing directions into 40 evenly distributed sectors and recording only the distance of the closest cone in each sector. By adopting this approach, we achieved a normalised observation space, as the distances within each of the 40 sectors were consistent and comparable. Moreover, this method ensured that cones positioned between LiDAR rays would not be missed by the detection algorithm, allowing the agent to accurately perceive every cone in its environment, even in challenging track sections such as tight corners and hairpins.

4.2.3 Additional Observations

Incorporating the vehicle's velocity, acceleration, and orientation information into the reinforcement learning agent's observation is vital for several reasons.

First, the vehicle's velocity is a crucial observation component, as it directly affects the agent's ability to perceive the vehicle's dynamics and respond to different track conditions. A higher velocity, for instance, can require greater control and finer adjustments to steering and throttle inputs, while lower speeds might necessitate a different approach. By including the velocity in the observation, the agent can learn to adapt its control policy according to the vehicle's current speed.

Second, acceleration is essential to the vehicle's state observation, as it gives the agent insight into how the vehicle responds to its inputs. For example, a sudden increase in acceleration may indicate that the agent is applying too much throttle, resulting in wheel spin or loss of traction. A sudden decrease in acceleration may suggest that the agent is applying the brakes or that the vehicle is losing speed due to external factors, such as track conditions or air resistance. By incorporating acceleration data into the observation space, the agent can learn the result of its control inputs, optimising its performance on the track.

Finally, the vehicle's orientation is also important to the agent's decision-making process, which provides crucial information about the vehicle's alignment relative to the track. Understanding the vehicle's orientation allows the agent to predict how the vehicle will respond to steering inputs and anticipate upcoming changes in the track layout. This information can help the agent to learn when to adjust its steering, throttle, and braking inputs to navigate the track optimally.

4.3 Reward shaping

4.3.1 Collision Penalty

Implementing a penalty or negative reward upon collision with traffic cones serves a critical purpose in the reinforcement learning process for our autonomous vehicle. By assigning a negative reward of -10 for these events, the agent is encouraged to learn behaviours that avoid colliding with the cones, ultimately leading to more efficient and safe driving on the generated race tracks.

This design choice is rooted in the desire to train an agent that can navigate the race track without hitting the traffic cones as they mark the track's boundaries. By penalising collisions with cones, the reinforcement learning process is guided towards an optimal policy that maintains the vehicle within the bounds of the track, ultimately resulting in smoother and more controlled driving.

In our implementation, once the agent's vehicle collides with a traffic cone, we apply a significant negative reward, signalling to the agent that the incurred action was undesirable. This feedback helps the agent adjust its policy to avoid similar consequences in the future. Additionally, upon collision, we reset the training episode and generated an entirely new race track. This approach ensures that the agent encounters a diverse range

of track layouts, encouraging it to learn generalised strategies for handling various track configurations rather than memorising specific track patterns.

4.3.2 Speed Encouragement

Incorporating a positive reward for achieving faster speeds in the reinforcement learning process incentivises the agent to develop a policy that navigates the race track safely, increases efficiency, and decreases lap times. By providing the agent with a reward signal directly proportional to the race car's current speed at each time step with a value of 0.01 * current speed, we encourage the development of a driving strategy that balances safety with optimal performance.

This design choice stems from the goal of training an autonomous vehicle that can navigate different race tracks at higher speeds while avoiding collisions. The positive reward for increased speed is implemented in a way that complements the negative reward given for colliding with traffic cones. This combination helps the agent to strike a balance between avoiding collisions and driving at a faster pace.

In our implementation, we apply the speed reward only once the race car reaches a small initial speed threshold, greater than 1 MPH. This ensures that the agent is not incentivised to remain stationary or drive at extremely low speeds to avoid collisions, which would result in an inefficient driving strategy. By applying the speed reward past this initial threshold, we promote the learning of an optimal policy that maintains higher speeds while navigating the track safely and efficiently.

The importance of incorporating the speed reward into the reinforcement learning process is evident in developing an agent capable of navigating the race tracks at higher speeds without compromising safety. By rewarding the agent for achieving faster speeds while concurrently penalising collisions, we create an environment conducive to the learning of a robust policy that can handle a diverse range of racing conditions with proficiency and agility. This design choice aligns with the ultimate goal of producing an autonomous vehicle that can perform effectively in a racing scenario, where the optimisation of speed and efficiency is of paramount importance.

4.4 Action Space

4.4.1 Steering Output

Designing an appropriate action space is crucial for effectively training a reinforcement learning agent, as it determines the available actions the agent can take to interact with its environment. In our case, we map the action output of the agent, which is a continuous floating-point number ranging from -1 to 1, to the left-to-right steering input of the racecar vehicle dynamics model.

To accomplish this mapping, we interpret the agent's action output as the steering command for the racecar's front wheels. A value of -1 corresponds to the maximum left steering angle, 1 corresponds to the maximum right steering angle, and 0 indicates neutral steering with the wheels pointing straight ahead. The continuous range of values

between -1 and 1 allows for proportional control, allowing the agent to produce a broad spectrum of steering commands for refined racecar control.

This approach to action space mapping provides several benefits. Firstly, it ensures the agent can learn to produce a wide range of steering commands to navigate various track configurations and conditions. Secondly, the continuous nature of the action space enables smoother and more precise control of the racecar, allowing the agent to make subtle adjustments as needed to maintain optimal racing lines and avoid collisions.

4.4.2 Throttle & Brake Output

Another crucial aspect of designing the action space for our reinforcement learning agent involves defining how the agent controls the acceleration and deceleration of the racecar. To address this, we map a second action output of the agent, which is also a continuous floating-point number ranging from -1 to 1, to the throttle and braking inputs of the racecar vehicle dynamics model. This mapping allows the agent to modulate its speed, enabling it to achieve optimal lap times and respond appropriately to different track conditions.

We interpret the agent's action output as a command for both the throttle and brake systems of the racecar. A value of -1 corresponds to full braking force, a value of 1 corresponds to full throttle, and a value of 0 represents no input to either system, causing the car to coast. The continuous range of values between -1 and 1 allows for proportional control over the throttle and brake, providing the agent with the capability to fine-tune the racecar's speed and adapt to the dynamic racing conditions.

This dual-input action space mapping confers several advantages. First, it allows the agent to learn to produce a variety of throttle and braking commands that cater to different racing scenarios, such as accelerating out of corners or decelerating before entering a sharp turn. Second, the continuous nature of the action space ensures smooth and precise control over the racecar's speed, preventing sudden or jerky movements that could destabilize the vehicle. Implementing this mapping mechanism for throttle and brake inputs empowers the agent with the necessary control mechanisms to optimize its performance on race tracks.

4.5 Proximal Policy Optimisation [28]

Our study employed Proximal Policy Optimization (PPO), a reinforcement learning algorithm proposed by Schulman et al [28]. PPO is a policy gradient method that alternates between sampling data through interaction with the environment and optimising a surrogate objective function using stochastic gradient ascent. Unlike standard policy gradient methods, which perform one gradient update per data sample, PPO utilises a novel objective function that enables multiple epochs of minibatch updates. This approach shares some of the benefits of Trust Region Policy Optimization (TRPO), such as data efficiency and reliable performance. However, it is simpler to implement, more general, and exhibits better sample complexity.

The PPO algorithm introduces a novel objective with clipped probability ratios, which forms a pessimistic estimate (i.e., lower bound) of the policy's performance. To optimise policies, PPO alternates between sampling data from the policy and performing several epochs of optimisation on the sampled data. Empirical results suggest that the surrogate objective with clipped probability ratios performs the best compared to other surrogate objective versions. PPO has demonstrated superior performance on benchmark tasks, such as continuous control tasks and Atari games, compared to other online policy gradient methods.

PPO strikes a favourable balance between sample complexity, simplicity, and wall time, making it an appealing choice for our reinforcement learning problem. Its scalability allows it to handle large models and parallel implementations while maintaining data efficiency and robustness across various problems without extensive hyperparameter tuning. Furthermore, PPO's compatibility with first-order optimisation makes it suitable when more complicated optimisation techniques, such as TRPO, are not feasible.

Chapter 5

Experimentation and Results

5.1 Training

This chapter presents the training method to teach the reinforcement learning agent to drive the racecar on various race tracks. We aim to develop an agent capable of navigating a diverse range of race track environments while avoiding overfitting to specific track layouts. The foundation of our method involved generating a new race track for each training episode, initialising the racecar in the appropriate starting position, and allowing the agent to control the vehicle while receiving reward signals based on its performance.

We employed the procedural track generator previously discussed to generate new race tracks for each training episode. This generator provided a continuous supply of novel track layouts for the agent to navigate, thereby ensuring that the agent faced an array of driving scenarios throughout the training process. In turn, this approach fostered the development of a more versatile and adaptable agent by preventing it from memorising specific patterns or sequences.

Generating a new race track for each training episode served to combat overfitting and ensure the development of a more generalised agent capable of driving on various race tracks. By constantly exposing the agent to diverse driving scenarios, our methodology prevented the reinforcement learning algorithm from over-optimising for specific track layouts or patterns, resulting in a more versatile and adaptable driving policy.

Before each training episode commenced, the racecar was initialised and placed in the appropriate starting position on the newly generated track. Once every training episode begins, the agent starts to issue control commands to the vehicle simulation and receives reward signals based on performance metrics measured from speed and collisions with traffic cones.

During the training process, if an unwanted collision occurred, a negative reward signal for collision penalty was issued, and the training episode was terminated and reset. This design decision aimed to provide immediate feedback to the agent when an undesirable event transpired, encouraging the development of driving strategies that prioritise safe and efficient navigation. By concluding an episode immediately after a collision event,

the reinforcement learning algorithm could adjust the policy to avoid similar scenarios in the future, fostering the development of more effective driving strategies.

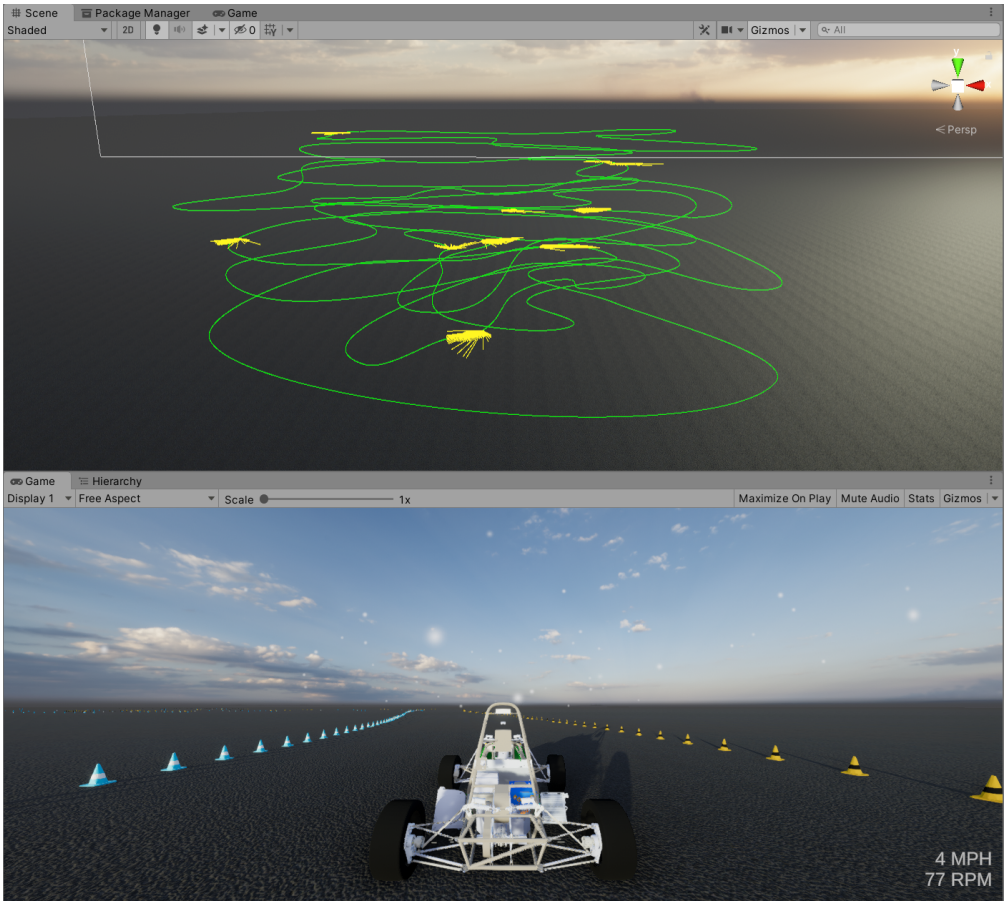

Figure 5.1: 10 Instances of the training environment running simultaneously

To expedite training and capitalise on modern hardware's parallelism, we ran ten instances of the training environments simultaneously. This approach enabled the agent to learn from multiple episodes concurrently, drastically reducing the time required for training. Additionally, this methodology provided the agent with a broader and more diverse range of driving experiences, allowing it to gain insights into various track configurations and challenges more quickly.

Multiple training environments also minimised any potential biases introduced by individual track layouts, ensuring that the agent could develop a more robust and generalised understanding of driving dynamics. This approach allowed the agent to learn to adapt to a range of new track configurations simultaneously.

The maximum episode length for each episode was set to 3000 simulation steps. This constraint ensured that the agent's training focused on shorter, more manageable track sections rather than attempting to learn from exceedingly long episodes. This design choice allowed the agent to build its driving capabilities incrementally, mastering smaller segments before tackling more complex or lengthy track configurations.

5.2 Hyper-parameter Tuning

In this chapter, we discuss the process of tuning hyperparameters for the proximal policy optimisation (PPO) algorithm employed in our reinforcement learning agent's training. By assessing the agent's performance with different hyperparameter combinations, we aimed to discover the configuration that resulted in the most effective and efficient training. To identify the optimal combination of hyperparameters that yielded the highest cumulative reward, we programmed the training algorithm to systematically sweep through a set of predefined values for batch size, buffer size, learning rate, epsilon, lambda, hidden units number, and hidden layers number.

```yaml
behaviors:
  Drive:
    trainer_type: ppo
    hyperparameters:
      batch_size: 1024
      buffer_size: 10240
      learning_rate: 3.0e-4
      beta: 5.0e-4
      epsilon: 0.2
      lambd: 0.95
      num_epoch: 3
      learning_rate_schedule: linear
      beta_schedule: linear
      epsilon_schedule: linear
    network_settings:
      normalize: false
      hidden_units: 128
      num_layers: 2
    reward_signals:
      extrinsic:
        gamma: 0.99
        strength: 1.0
    max_steps: 50000000
    time_horizon: 64
    summary_freq: 5000
```

Figure 5.2: The set of training hyper parameters for the final agent

The batch size hyperparameter refers to the number of samples used in a single update step of the gradient optimisation process. A larger batch size generally reduces the variance in the update steps, potentially improving the stability of the training process. However, larger batch sizes may also increase the computational requirements and slow down the overall training process.

Buffer size determines the number of transitions (state, action, reward, and next state

tuples) stored in the replay buffer for the reinforcement learning agent to learn. A larger buffer size can enhance the diversity of samples available for training, improving the agent's ability to learn from a broader range of experiences. Conversely, a smaller buffer size may limit the agent's learning capacity due to a reduced variety of experiences, but it can also reduce memory requirements and improve computational efficiency.

The learning rate hyperparameter controls the magnitude of the weight updates during the optimisation process. A larger learning rate can speed up convergence to an optimal policy but may also lead to instability or oscillation in the training process. On the other hand, a smaller learning rate can result in more stable training but at the expense of potentially slower convergence.

Epsilon, a parameter specific to the PPO algorithm, represents the clipping threshold used to bound the ratio of old and new policy probabilities. This hyperparameter constrains the magnitude of the policy updates to prevent excessively large deviations from the current policy. Balancing the value of epsilon is crucial, as a smaller value provides a tighter constraint and greater stability, while a larger value allows for more flexibility in policy updates, potentially enabling faster convergence.

Lambda is a parameter in the generalised advantage estimation (GAE) used for computing the advantage function in PPO. Adjusting lambda influences the balance between the bias and variance of the advantage estimates. Higher lambda values reduce the bias but may increase the variance, while lower values result in lower variance but increased bias.

The number of hidden units and layers in the neural network architecture can significantly impact the model's capacity and the complexity of the function approximations. Increasing the number of hidden units or layers can improve the expressive power of the network, enabling it to learn more complex policies. However, larger networks can also increase the risk of overfitting and require more computational resources.

5.3 Results & Evaluation

In the results section, we present the comprehensive outcomes of our experimentation after training the reinforcement learning algorithm for 28,270,000 steps. As the training progressed, we observed that the cumulative reward exhibited a converging trend, ultimately stabilising at a steady level at around 26. This observation implies that the agent had achieved a level of proficiency in controlling the simulated race car within the simulation environment, demonstrating a degree of mastery in adapting to various race track configurations and maximising the car's performance whenever possible.

The agent's driving behaviour exhibited a high level of sophistication, firstly in its ability to maintain a straight trajectory on predominantly straight sections of the race track. The agent maximised acceleration and speed by keeping the vehicle aligned in a straight line while travelling through straight sections of the track, taking full advantage of the car's performance capabilities. This behaviour showcases the agent's understanding of optimising vehicle speed for straightaways.

When approaching corners, the agent displayed further evidence of skilful vehicle

control as it applied the brakes in a measured manner to decrease speed. This braking behaviour showcased the agent's understanding of the need to adjust the vehicle's speed in response to changing track conditions, as well as its ability to steer dynamically based on the track's layout. Furthermore, the agent demonstrated a keen awareness of varying corner tightness levels, as evidenced by more aggressive braking and earlier brake points when encountering sharper turns. These nuanced driving behaviours suggest that the agent had developed a high degree of control over the simulated race car, ensuring optimal performance while navigating diverse track features.

Remarkably, the agent's driving pattern bore a striking resemblance to the racing techniques employed by professional race car drivers. The agent consistently followed a vague racing line, a path along the track that optimises the distance travelled and enables the vehicle to maintain the highest possible speed through corners. Adhering to the racing line demonstrates the agent's ability to strategically and efficiently traverse the race track.

In addition to adhering to the racing line, the agent displayed an aptitude for minimising its turning radius while maintaining speed, strategically approaching corner apexes while preserving a safe distance from the traffic cones lining the track's edge. This behaviour highlights the agent's capacity to balance performance optimisation and safety, showcasing the efficacy of the training methodology employed in our study.

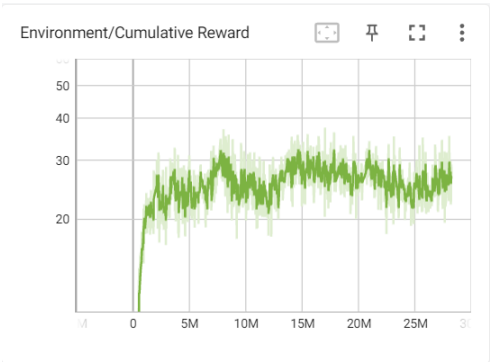

Figure 5.3: Graph of Cumulative reward over training steps

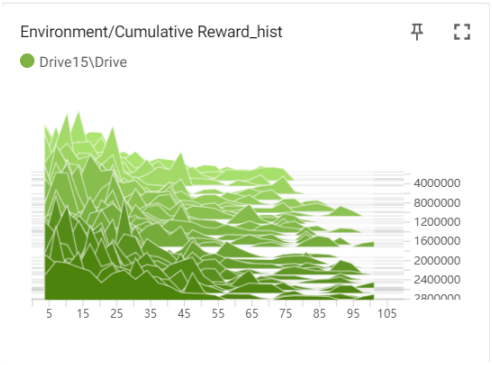

Figure 5.4: Cumulative reward histogram over training steps

Chapter 6

Conclusions

6.1 Conclusions

This research project was motivated by the goal of developing an advanced reinforcement learning agent capable of autonomously controlling a race car while optimising its performance and safely navigating complex track configurations. Our approach involved the implementation of a realistic vehicle dynamics simulation, developing a procedural race track generation algorithm, and using state-of-the-art reinforcement learning techniques to train an autonomous driving agent.

The vehicle dynamics simulation served as the foundation of our project, providing an accurate representation of the race car's behaviour under various driving conditions. The procedural race track generation algorithm further augmented our experimental setup by allowing us to create diverse race tracks, thereby ensuring the generalisation of our reinforcement learning agent. This unique simulation and track generation combination was crucial for providing a rich and challenging environment in which our agents could learn and adapt.

Our reinforcement learning framework was built around the Proximal Policy Optimization (PPO) algorithm, chosen for its excellent balance between sample efficiency, simplicity, and performance. By carefully designing the observation space, reward function, and action space, we created a learning environment conducive to training a robust and versatile driving agent. Additionally, we conducted an extensive hyperparameter tuning process, which contributed to the success of our PPO training algorithm.

Our experiments' results demonstrated our approach's effectiveness, with the trained reinforcement learning agent displaying remarkable proficiency in autonomously controlling the simulated race car. Our agent successfully optimised the race car's performance while navigating a wide variety of race track configurations and adapting its driving behaviour to the changing track conditions, displaying self-driving autonomy similar to professional racecar drivers.

6.2 Future Works

We intend to explore the deployment of our reinforcement learning algorithm onto our real Edinburgh University Formula Student autonomous race car. The evaluation of our approach in a real-world scenario will serve as a crucial step in validating the effectiveness and robustness of our trained agent. Moreover, this would open up new opportunities for further optimisation and refinement of our algorithm and the potential application of our findings to broader autonomous driving contexts.

Bibliography

[1] Szilárd Aradi. Survey of deep reinforcement learning for motion planning of autonomous vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 23(2):740–759, 2020.

[2] Lenore Blum, Manuel Blum, and Mike Shub. A simple unpredictable pseudo-random number generator. *SIAM Journal on computing*, 15(2):364–383, 1986.

[3] Peide Cai, Xiaodong Mei, Lei Tai, Yuxiang Sun, and Ming Liu. High-speed autonomous drifting with deep reinforcement learning. *IEEE Robotics and Automation Letters*, 5(2):1247–1254, 2020.

[4] Luigi Cardamone, Daniele Loiacono, Pier Luca Lanzi, and Alessandro Pietro Bardelli. Searching for the optimal racing line using genetic algorithms. In *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*, pages 388–394. IEEE, 2010.

[5] Ji-wung Choi, Renwick Curry, and Gabriel Elkaim. Path planning based on bézier curve for autonomous ground vehicles. In *Advances in Electrical and Electronics Engineering-IAENG Special Edition of the World Congress on Engineering and Computer Science 2008*, pages 158–166. IEEE, 2008.

[6] Jesse Clifton and Eric Laber. Q-learning: Theory and applications. *Annual Review of Statistics and Its Application*, 7:279–301, 2020.

[7] R Craig Coulter. Implementation of the pure pursuit path tracking algorithm. Technical report, Carnegie-Mellon UNIV Pittsburgh PA Robotics INST, 1992.

[8] David Crolla. *Encyclopedia of automotive engineering*. John Wiley & Sons, 2015.

[9] Philip J Davis and Philip Rabinowitz. *Methods of numerical integration*. Courier Corporation, 2007.

[10] Peter Finnman and Max Winberg. Deep reinforcement learning compared with q-table learning applied to backgammon, 2016.

[11] Florian Fuchs, Yunlong Song, Elia Kaufmann, Davide Scaramuzza, and Peter Dürr. Super-human performance in gran turismo sport using deep reinforcement learning. *IEEE Robotics and Automation Letters*, 6(3):4257–4264, 2021.

[12] Giancarlo Genta. *Motor vehicle dynamics: modeling and simulation*, volume 43. World Scientific, 1997.

[13] Julian Ibarz, Jie Tan, Chelsea Finn, Mrinal Kalakrishnan, Peter Pastor, and Sergey Levine. How to train your robot with deep reinforcement learning: lessons we have learned. *The International Journal of Robotics Research*, 40(4-5):698–721, 2021.

[14] Radoslav Ivanov, Taylor J Carpenter, James Weimer, Rajeev Alur, George J Pappas, and Insup Lee. Case study: verifying the safety of an autonomous racing car with a neural network controller. In *Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control*, pages 1–7, 2020.

[15] Joseph Katz. Race car aerodynamics. *Robert Bentley*, 1995.

[16] C Kim and PI Ro. Reduced-order modelling and parameter estimation for a quarter-car suspension system. *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, 214(8):851–864, 2000.

[17] Vijay Konda and John Tsitsiklis. Actor-critic algorithms. *Advances in neural information processing systems*, 12, 1999.

[18] Pedro Larranaga, Cindy M. H. Kuijpers, Roberto H. Murga, Inaki Inza, and Sejla Dizdarevic. Genetic algorithms for the travelling salesman problem: A review of representations and operators. *Artificial intelligence review*, 13:129–170, 1999.

[19] L Leonard, A Lim, TJS Chesser, SA Norton, and JP Nolan. Does changing the configuration of a motor racing circuit make it safer? *British journal of sports medicine*, 39(3):159–161, 2005.

[20] Nan Li, Dave W Oyler, Mengxuan Zhang, Yildiray Yildiz, Ilya Kolmanovsky, and Anouck R Girard. Game theoretic modeling of driver and vehicle interactions for verification and validation of autonomous vehicle control systems. *IEEE Transactions on control systems technology*, 26(5):1782–1797, 2017.

[21] Yuxi Li. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*, 2017.

[22] Wm C Mitchell, Allan Staniforth, and Ian Scott. Analysis of ackermann steering geometry. Technical report, SAE Technical Paper, 2006.

[23] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[24] Hans B Pacejka and Egbert Bakker. The magic formula tyre model. *Vehicle system dynamics*, 21(S1):1–18, 1992.

[25] Martin L Puterman. Markov decision processes. *Handbooks in operations research and management science*, 2:331–434, 1990.

[26] Thinal Raj, Fazida Hanim Hashim, Aqilah Baseri Huddin, Mohd Faisal Ibrahim, and Aini Hussain. A survey on lidar scanning mechanisms. *Electronics*, 9(5):741, 2020.

[27] Moveh Samuel, Mohamed Hussein, and Maziah Binti Mohamad. A review of some pure-pursuit based path tracking techniques for control of autonomous vehicle. *International Journal of Computer Applications*, 135(1):35–38, 2016.

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

[29] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[30] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

[31] Victor Uc-Cetina, Nicolas Navarro-Guerrero, Anabel Martin-Gonzalez, Cornelius Weber, and Stefan Wermter. Survey on reinforcement learning for language processing. *Artificial Intelligence Review*, pages 1–33, 2022.

[32] Sandor M Veres, Levente Molnar, Nick K Lincoln, and Colin P Morice. Autonomous vehicle control systems—a review of decision making. *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, 225(2):155–195, 2011.

[33] Peter R Wurman, Samuel Barrett, Kenta Kawamoto, James MacGlashan, Kaushik Subramanian, Thomas J Walsh, Roberto Capobianco, Alisa Devlic, Franziska Eckert, Florian Fuchs, et al. Outracing champion gran turismo drivers with deep reinforcement learning. *Nature*, 602(7896):223–228, 2022.

[34] Bernhard Wymann, Eric Espié, Christophe Guionneau, Christos Dimitrakakis, Rémi Coulom, and Andrew Sumner. Torcs, the open racing car simulator. *Software available at http://torcs. sourceforge. net*, 4(6):2, 2000.

[35] Dongbin Zhao, Haitao Wang, Kun Shao, and Yuanheng Zhu. Deep reinforcement learning with experience replay based on sarsa. In *2016 IEEE symposium series on computational intelligence (SSCI)*, pages 1–6. IEEE, 2016.