Race Line Optimisation

Using Deep Reinforcement Learning

AE4350 Bio-Inspired Intelligence and Learning

João Gonçalo Silva

Delft University of Technology

TU Delft

Race Line Optimisation

Using Deep Reinforcement Learning

by

João Gonçalo Silva

GitHub Repository:
`https://github.com/jgsilva747/RaceLineOptimisation_RL`

Instructor:	Dr. Erik-Jan van Kampen
Project Duration:	June, 2023 - August, 2023
Faculty:	Faculty of Aerospace Engineering, Delft

Cover: Two McLaren MC34s during a practice run at the USA Grand Prix 2019 by Dinesh Boaz (modified)

TU Delft

Contents

Nomenclature **ii**

1 Introduction . . . 1
1.1 Autonomous Car Racing . . . 1
1.2 Reinforcement Learning Fundamentals . . . 1
2 Simulation Environment . . . 2
2.1 Car Model . . . 3
2.2 Track Model . . . 4
3 Methodology . . . 4
3.1 State Formulation . . . 4
3.2 Hyperparameter Tuning . . . 5
3.3 Reward Signal Design . . . 7
4 Results . . . 9
5 Conclusion . . . 10

References **11**

A SAC **14**

B Integrator Tuning **16**

Nomenclature

Abbreviations

Abbreviation	Definition
ANN	Artificial Neural Network
DDPG	Deep Deterministic Policy Gradient
DRL	Deep Reinforcement Learning
GPU	Graphical Processing Unit
MPC	Model Predictive Control
MPPI	Model Predictive Path Integral Control
MSBE	Mean Squared Bellman Error
NN	Neural Network
ReLU	Rectified Linear Unit
RL	Reinforcement Learning
SAC	Soft Actor-Critic
TD3	Twin-Delayed Deep Deterministic Policy Gradient

Greek Letters

Letter	Definition
α	Temperature coefficient
γ	Discount factor of future rewards
$\boldsymbol{\epsilon}$	Input noise vector
θ_i	i-th entry of critic network parameter vector
λ	Learning rate
π	Policy
π^*	Optimal policy
π_ϕ	Policy approximation, parameterised by ϕ
$\pi(\boldsymbol{s})$	Action taken in state $\boldsymbol{s}$ under deterministic policy π
$\pi(\boldsymbol{u} \mid \boldsymbol{s})$	Probability of taking action $\boldsymbol{u}$ at state $\boldsymbol{s}$ under stochastic policy π
τ	Smoothing constant
ϕ	Policy network parameter

Latin Letters

Letter	Definition
$\boldsymbol{a}$	2D acceleration vector
$\mathcal{A}$	Action space
$\mathcal{B}$	Minibatch of transition samples

Letter	Definition
C_D	Drag coefficient
$\mathcal{D}$	Replay buffer of transition samples
$\boldsymbol{f}_\phi(\boldsymbol{\epsilon}, \boldsymbol{s})$	Policy network transformation
$\mathcal{H}$	Entropy
$\bar{\mathcal{H}}$	Target entropy
$J(X)$	Performance measure for variable X
$\mathcal{L}_{\theta_i}$	MSBE Loss of i-th Q-function approximation
m	Mass
mfr	Mass flow rate
N	Number of steps of an episode
$\mathcal{N}$	Gaussian distribution
$Q^\pi(\boldsymbol{s}, \boldsymbol{u})$	Q-function under policy π, starting with action $\boldsymbol{u}$ at state $\boldsymbol{s}$
$Q^\pi_{\theta_i}(\boldsymbol{s}, \boldsymbol{u})$	$Q^\pi(\boldsymbol{s}, \boldsymbol{u})$ approximation, parameterised by θ_i
r_t	Scalar reward obtained at time t
$r(\boldsymbol{s}, \boldsymbol{u})$	Reward obtained after taking action $\boldsymbol{u}$ at state $\boldsymbol{s}$
R	Curvature radius
R_t	Sum of future rewards, starting at time t
$\boldsymbol{s}_t$	State vector at time t
S_D	Drag Surface Area
$\mathcal{S}$	State space
t	Discrete time step
$\boldsymbol{u}$	Action vector
$\boldsymbol{v}$	2D velocity vector
$V^\pi(\boldsymbol{s})$	Value function under policy π, starting at state $\boldsymbol{s}$
wb	Vehicle's wheelbase
$\boldsymbol{x}$	2D position vector

Symbols

Symbol	Definition
$\leftarrow$	Assignment
$\doteq$	Equal relationship that is true by definition
$\mathbb{E}$	Expected value
$f : \mathcal{X} \to \mathcal{Y}$	Function f from set $\mathcal{X}$ to set $\mathcal{Y}$
$\mid \boldsymbol{x} \mid$	Norm of vector $\boldsymbol{x}$
$\mid \mathcal{X} \mid$	Number of elements in set $\mathcal{X}$
$X \sim x$	Variable X sampled from x

1. Introduction

The main goal of this project is to implement a Deep Reinforcement Learning algorithm in order to optimise the race line of a Formula 1 car in a previously-unseen circuit and understand the learning process. A secondary goal is to assess the robustness of the learning strategy by extending the controller to untrained circuits. This project is structured as follows: Subsection 1.1 discusses related work in the field of autonomous car racing. Then, the chosen method for this project is presented in Subsection 1.2, where the choice is justified using literature and performance expectations. Section 2 covers the car and circuit modelling before presenting the project's methodology in Section 3, where topics such as state formulation, effect of hyperparameters and reward design are discussed. The final result is then presented in Section 4, along with an analysis of the found solution.

1.1. Autonomous Car Racing

Autonomous car racing is a very challenging topic, as vehicles need to operate at the limit of their dynamics, navigating around curves in tight spaces at high speeds. This challenge has led to a growing research interest in this area. In this framework, the topic can be divided into: environment perception, trajectory planning, and vehicle control [1]. The focus of this project is on trajectory planning, which can be further split into global and local planning: both are concerned with completing a given task (*e.g.*, overtaking a vehicle or completing a lap) as quickly as possible, but while the former is usually related to generating the optimal overall path, also known as the race line, the latter works at a finer resolution and relates to tasks such as obstacle avoidance [2] or overtaking strategies [3].

There are different approaches for optimising the race line, *i.e.*, minimising the lap time: different optimisation methods such as Particle Swarm Optimisation [4], Genetic Algorithms [5], Convex Optimisation [6] or even formulating the task as an optimal (nonlinear) control problem [7] have proved to be successful in achieving this goal. However, more commonly used methods are Model Predictive Control (MPC) [8–11] and Model Predictive Path Integral Control (MPPI) [12, 13], which perform extremely well in real-life applications at high speeds due to the fact that they can accurately capture the complex dynamics of the vehicles.

A major drawback of MPC and MPPI is the fact that computation time is traded off for performance and dynamic fidelity. For instance, in [13], each trajectory is sampled in parallel, with each sample using up to 16 CUDA (Nvidia's parallel computing platform with access to Graphical Processing Units (GPUs), dramatically speeding up computing applications [14]) threads, which are used to make approximately 4.8 million queries to the full nonlinear dynamics model every second.

This high level of complexity and computational effort, which is not possible without the use of a modern GPU, led to the preference for an equally satisfactory but more computationally efficient approach: the use of Reinforcement Learning (RL). Deep Reinforcement Learning (DRL), which combines RL with Artificial Neural Networks (ANNs), enables the use of high-dimensional, continuous input spaces by using function approximators to learn the policy and value function [15, 16], which removes the need for heavy computational power, leading to its widespread and successful use in autonomous racing applications [2, 17–19].

1.2. Reinforcement Learning Fundamentals

RL is a bio-inspired subset of Machine Learning that seeks to mimic the way humans learn: by trial and error, based on the results of performed actions. In RL, an agent improves its decisions by interacting with an environment in order to ultimately find the optimal strategy, referred to as policy: at time t, the agent moves from state $\boldsymbol{s}_t$ to $\boldsymbol{s}_{t+1} \in \mathcal{S}$ by taking an action $\boldsymbol{u}_t \in \mathcal{A}$ given from its policy $\pi(\boldsymbol{s}) : \mathcal{S} \to \mathcal{A}$. The agent then receives feedback from the environment, called the reward $r_{t+1} \doteq r(\boldsymbol{s}_t, \boldsymbol{u}_t) \in \mathbb{R}$, at each time step [15, 16].

The original RL framework comprises discrete tabular methods, where each possible state has a corresponding policy entry. This is limiting for most real world control applications, since these are usually continuous and high-dimensional. To overcome run-time and storage capacity issues, RL can be combined with ANNs in a framework known as DRL, which enables handling high-dimensional input spaces [15, 16]. The use of DRL has been shown to supersede human performance in Atari games [20] with discrete action spaces, and was later extended to control applications with the Deep Deterministic Policy

Gradient (DDPG) algorithm [21], as it supports continuous action spaces. However, this algorithm is subject to severe hyperparameter sensitivity [22, 23].

Soft Actor-Critic (SAC) [24, 25] and Twin-Delayed Deep Deterministic policy gradient (TD3) [26] are state-of-the-art actor-critic algorithms that leverage DDPG to increase stability by using a pair of clipped Q-networks and a target Q network. In this project, the chosen algorithm is SAC because, as opposed to TD3, it uses an entropy-based stochastic policy, which has been shown to encourage exploration and increase sample efficiency [24, 27], and because the SAC implementation from `d3rlpy` incorporates *delayed policy update*, which is a feature from TD3 that was not originally present in the SAC algorithm.

SAC is an off-policy actor-critic DRL algorithm [24, 25] that applies the maximum entropy framework, which not only leads to better exploration [27] but also increases robustness against model or estimation errors [28]. The introduction of the entropy in the policy objective encourages the agent to maximise the reward while acting as randomly as possible. TD3 uses target policy smoothing because it trains a deterministic policy [26], but this is not needed in SAC since the noise from its stochastic policy is enough to achieve a similar effect.

Furthermore, SAC tries to mitigate the positive bias in the policy improvement [25] by using two soft Q-functions, modelled as two NNs parameterised by θ_1 and θ_2 and trained independently. In order to increase learning stability, a separate target value network $Q^{\pi_\phi}_{\text{target},\theta_i}$ that slowly tracks the main network is used. The target value network weights are updated less frequently than the main Q-network using an exponentially moving average with a smoothing constant τ. When updating the network parameters, SAC conservatively uses the minimum target Q-value from the two approximators.

The policy is modelled as a Gaussian distribution with mean and variance given by a NN parameterised by ϕ. The output is squashed by a tanh function in order to bound the actions in a finite range. The authors of SAC apply the NN transformation $\boldsymbol{u}_t = \boldsymbol{f}_\phi(\boldsymbol{\epsilon}_t, \boldsymbol{s}_t)$ to reparameterise the policy, which is a convenient trick that results in a lower variance estimator [25], where $\boldsymbol{\epsilon}$ is an input noise vector.

The original SAC algorithm [24] is very sensitive to the temperature hyperparameter α. In the new version [25], automatic entropy/temperature adjustment is implemented. While this is a simple modification, it was found that it largely reduces the need for hyperparameter tuning for each scenario. A more detailed description of SAC, including equations and pseudo-code, is provided in Appendix A.

Optimising a circuit's raceline will require the agent to find actions that are very close to the limits, such as driving as close to the wall as possible and at the maximum speed without loosing traction. This means that slight changes in these extreme manoeuvres will either lead to non-optimal trajectories or invalid trajectories. However, considering the characteristics of SAC and the success of previous implementations in the field of autonomous racing [17–19], it is expected that the use of SAC will be successful in achieving this project's goal, since this state-of-the-art algorithm is known to be efficient in high-dimensional and continuous input spaces with complex dynamics. Furthermore, the entropy-based stochastic policy ensures that the agent does not get stuck in local optima and will converge to a near-optimal result, provided that enough episodes are run, which means that the main challenge is expected to be linked to the reward signal design, since the agent can quite often find a non-trivial solution which maximises the reward but does not meet the designer's goal.

2. Simulation Environment

In this section, the implementation of the car and track models is described.

In this project, the raceline for a given circuit is optimised. The optimal raceline depends on the capabilities of the race car, since some cars are able to make sharper and faster turns due to different aerodynamics and tyre characteristics. The DRL `d3rlpy` library [29] leverages the `PyTorch` library [30], which in turn requires a `Gym` environment [31]. A commonly used environment used for RL research in the field of autonomous racing, compatible with `Gym`, is the `F1TENTH` environment [32], which already includes high-fidelity models of small race cars. However, in order to increase flexibility and to add to the didactic experience, a custom car model was created such that the RL agent can optimise the raceline for any type of race car, provided that it is modelled accurately.

2.1. Car Model

The car was modelled as a point mass, not accounting for drifting or spinning. This simplification was implemented because an ideal raceline does not include this type of motion, and it was found that penalising the situations where the car would drift/spin makes the agent drive within the car's limits, eliminating the need to model these dynamics. Therefore, the position $\boldsymbol{x}$, the velocity $\boldsymbol{v}$ and the mass m of the car are propagated as follows:

$$\begin{bmatrix} \dot{\boldsymbol{x}} & \dot{\boldsymbol{v}} & \dot{m} \end{bmatrix}^T = \begin{bmatrix} \boldsymbol{v} & \boldsymbol{a} & -mfr \cdot \max\{0, \text{throttle}\} \end{bmatrix}^T, \tag{1}$$

where mfr is the maximum mass flow rate of the vehicle, throttle is the throttle input, ranging from -1 to 1, and $\boldsymbol{a}$ is the car's acceleration vector. For car racing purposes, a 2 dimensional environment yields virtually the same results as a 3 dimensional environment while significantly reducing the overall complexity, except in specific circuits where great elevation changes are present. Therefore, the acceleration is described as a 2 dimensional vector with the following xy components:

$$\begin{bmatrix} a_x \\ a_y \end{bmatrix} = \begin{bmatrix} \frac{v_x}{|\boldsymbol{v}|} & -\frac{v_y}{|\boldsymbol{v}|} \\ \frac{v_y}{|\boldsymbol{v}|} & \frac{v_x}{|\boldsymbol{v}|} \end{bmatrix} \left(\begin{bmatrix} \cos(\text{steering}) \\ \sin(\text{steering}) \end{bmatrix} |\boldsymbol{a}| + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \frac{|\boldsymbol{v}|^2}{wb} \tan(\text{steering}) \right), \tag{2}$$

where wb is the wheelbase (distance between rear and front wheel axes) of the vehicle and steering is the input angle of the steering wheel. This decomposition of the acceleration vector in xy coordinates accounts for both the tangential and the normal components of the acceleration by making use of the relation between the curvature radius R, the wheelbase and the steering input: $R = wb/\tan(\text{steering})$.

The acceleration model is defined by the car for which the raceline is optimised. In this project, a Formula 1 car is considered. Figure 1 shows the typical Formula 1 acceleration profile accounting for tyre friction, aerodynamic forces and the engine's performance dependency on speed [33]. Formula 1 cars can accelerate from 0 to 100 km/h in 2.4 s, from 100 to 200 km/h in 1.8 s and from 200 to 300 km/h in 4.2 s [34], achieving top speeds of about 330 km/h, depending on the circuit [35], with a drag coefficient C_D ranging from 0.7 to 1 [36] and a drag surface area $S_D = 1.5$ m^2 [37]. The minimum car mass is 798 kg [34, 38], and for this project, an additional 100 kg of fuel are included.

However, modelling the engine power such that the car's acceleration profile matches that of Figure 1 and that the car has the performance described in the previous paragraph is a very complex task, particularly when accounting for the fact that these cars have eight forward gears. Such a detailed model would require an extensive study and could be a project in itself, which is outside the scope of this project, where the goal is to apply RL to solve a problem and thoroughly analyse the learning process and the results. It was then decided that the most important characteristic to be modelled is the acceleration times from 0 to 300 km/h at 100 km/h increments and the maximum speed, as this would approximate the simulated lap times to real life lap times. This led to the following empirical formulation of the maximum acceleration, accounting for both the engine and aerodynamic forces:

$$\begin{cases} m\,|\,\boldsymbol{a}\,|_{\max} = 4500 + 65.7\,|\,\boldsymbol{v}\,|^2 & \text{if} \quad |\,\boldsymbol{v}\,| < 15.278 \\ m\,|\,\boldsymbol{a}\,|_{\max} = 4500 + 65.7 \cdot 15.278^2 - 261\,(|\,\boldsymbol{v}\,| - 15.278) & \text{otherwise} \end{cases}, \tag{3}$$

which matches the described performance, as shown in Figure 2:

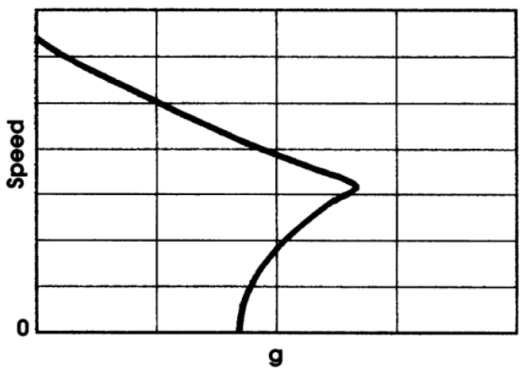

Figure 1: Speed vs Acceleration Illustration [33]

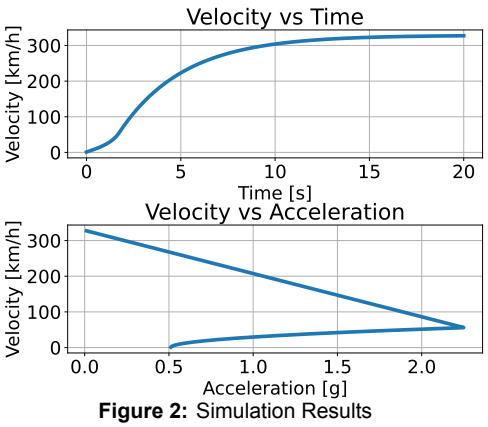

Figure 2: Simulation Results

Similarly, the maximum turning acceleration a_t and braking acceleration a_b were empirically formulated such that a_t is 2 g at 100 km/h, 3 g at 210 km/h and 6 g at 330 km/h [35], a_b is at most 5.5 g and the car is able to brake from 100 to 0 km/h in 1.5 s [39], and both acceleration profiles match the profile from Figure 3 [33]. The equations implemented to model the maximum accelerations as a function of the vehicle's speed are shown in Equation 4. Note that these are the maximum accelerations that the tyres can handle, and as such they are independent of the vehicle's mass m.

$$\begin{cases} a_t = 9 \times 10^{-4} \mid \boldsymbol{v} \mid^2 -4.45 \times 10^{-2} \mid \boldsymbol{v} \mid +2.5435 \\ a_b = 5 \times 10^{-4} \mid \boldsymbol{v} \mid^2 -6.7 \times 10^{-3} \mid \boldsymbol{v} \mid +1.5 \end{cases} \tag{4}$$

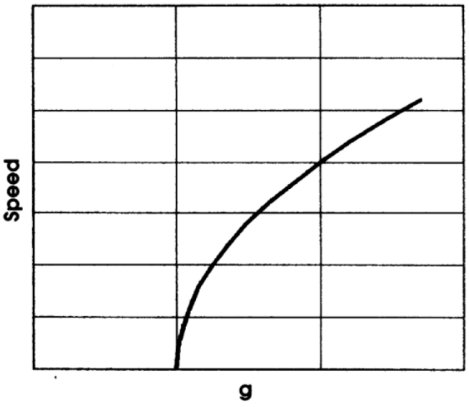

Figure 3: Cornering and Braking Acceleration Illustration [33]

2.2. Track Model

The track is modelled as a set of xy coordinates, based on the centre-line obtained from [40]. The coordinates were multiplied by a real world scaling factor $\texttt{circuit\_factor} = 10^5$ so that the modelled circuit had virtually the same dimensions as the real life track, so that the simulated lap times are comparable to real ones. The inner and outer limits of the track were obtained by adding a 5 m margin in both directions perpendicular to the centre-line, which was found to avoid overlapping lines in circuits with tight curves. Figure 4 shows the test curve that was used to test the algorithm, where the distance from the origin to the first corner is about 450 m and the total distance is about 840 m.

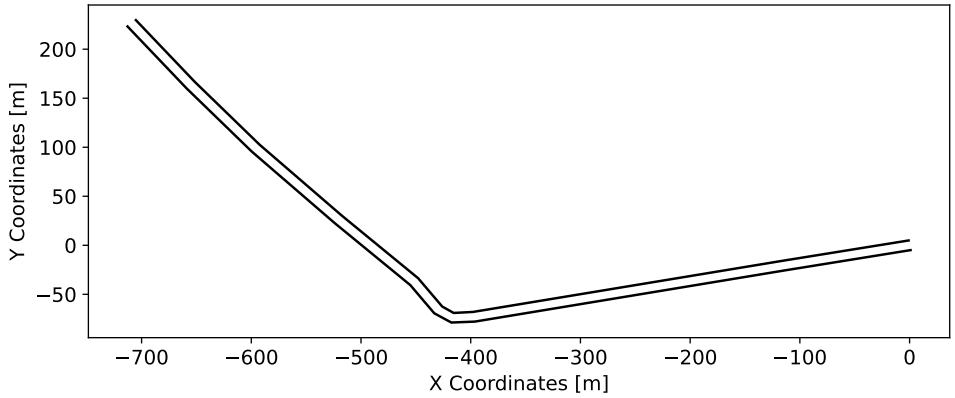

Figure 4: Test Curve (Turn 1 of Austria's Red Bull Ring). Direction: Right to Left

3. Methodology

In order to optimise a given circuit's raceline using DRL, the following methodology was adopted: (1) Define the state formulation to successfully complete one full run using SAC; (2) Analyse effect of hyperparameters and select best configuration (3) Iterate reward design to minimise lap times. This methodology is always applied to the test curve (Figure 4) due to the fact that a very large number of tests will be carried out, which would take significantly longer to run in a full circuit (a few days instead of a few hours). For reproducibility, every random generator's seed is always set to 0 and an Euler integrator is used with a step size of 0.15 s. A small integrator analysis is shown in Appendix B.

3.1. State Formulation

It was found that it is quite common in autonomous racing to include distance measurements in the agent's state, obtained using LiDAR measurements in multiple directions [2, 17, 19, 41]. Adding this to the state vector would allow the agent to associate a collision to the distance measurements, increasing the learning speed. Furthermore, this would allow the learnt policy to be transferred to other circuits since it would not be associated to specific xy coordinates, but instead to the distance to the track limits. The curvature of future track segments [17] should also be included in the state, since it allows the agent to associate the track curvature to breaking points and steering commands.

It seems that the overall logic of the state formulation should be to allow the agent to associate states to specific actions and rewards (such as learning to brake when the distance to a wall diminishes), meaning that defining the state as xy position coordinates is to lead to a poor performance. With this in mind, the state is formulated as:

Table 5: Updated State Formulation

State entry	Rationale
Velocity norm [m/s]	Allow the agent to learn to maximise speed and associate the speed to the curvature of a corner
Acceleration [m/s^2]	2 dimensional vector, allowing the agent to learn to maximise the forward acceleration and control the centripetal acceleration in curves
Velocity heading [rad]	Angle between velocity vector and centre-line, used as a way to let the agent know that it is moving forward [17]
Future curvatures [rad]	10 relative curvatures of future track segments, measured every 10 m, allowing the agent to prepare for a curve by breaking and turning at the right time
Distance to walls [m]	LiDAR measurements in the following directions: $\pm 90°, \pm 45°, \pm 30°, \pm 15°, 0°$. Allows the agent to avoid collisions and follow the wall-free path

This formulation results in a continuous 23 dimensional state. The expectation is that this will allow the learnt policy to be applied to any circuit, since the actions can directly be connected to the states which are not specific to any track, and no track-specific position coordinates are included.

3.2. Hyperparameter Tuning

With the new state formulation, employing the full car model with continuous actions and state spaces, and with a reward signal of one tenth of the total travelled distance (which encourages the agent to move forward), the agent is already capable of completing the full test curve with the default SAC implementation from [29]. Now, the effect of SAC's hyperparameters are analysed in order to understand how these influence the learning process. In the end, the best hyperparameters are picked so that the agent learns to complete the test curve in the smallest amount of episodes as possible.

Every parameter from [29] was analysed for 50,000 steps except the `observation_scaler`, which consists in scaling the state measurements, since the observations in this project are already simulated to resemble those of the real world. Therefore, 16 hyperparameters were analysed. The following figures show the effect of selected hyperparameters, where the red line represents the default values from [29]. In most of these figures, it can be noted that the reward usually hits a plateau of about 45 (which corresponds to reaching the first corner at 450 m) before jumping up to its maximum of about 84 (corresponding to the full test curve length of 840 m).

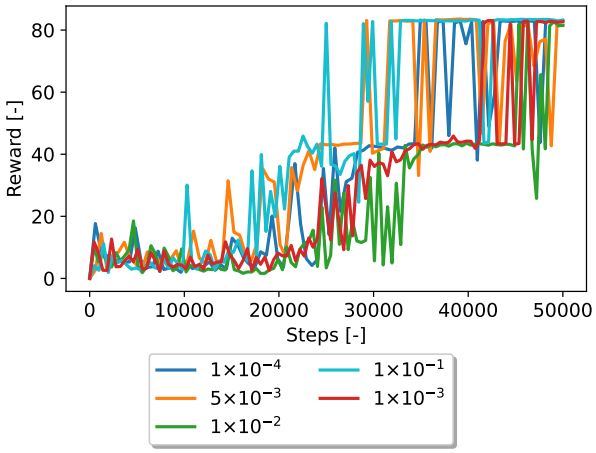

Figure 5: Effect of τ on Reward

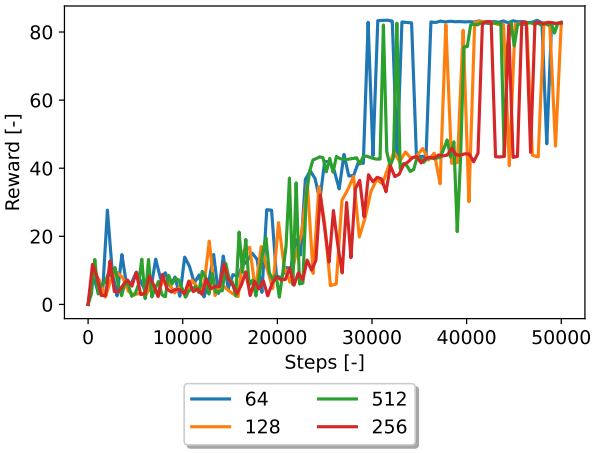

Figure 6: Effect of minibatch size on Reward

Figure 5 shows that the effect of τ is not linear, *i.e.*, the performance increases with values lower and greater than the default value. However, the values of τ that increase the speed of the learning process

the most are 5×10^{-3} and 1×10^{-1}, but the latter leads to a smoother reward evolution, and is therefore preferred. Figure 6 shows the effect of the minibatch size, which reveals that a smaller batch size of 64 leads to the best performance. Although this result was initially unexpected, it was later found that it actually matches the theory and smaller batch sizes do perform better, since large batch sizes can lead to very sharp losses [42].

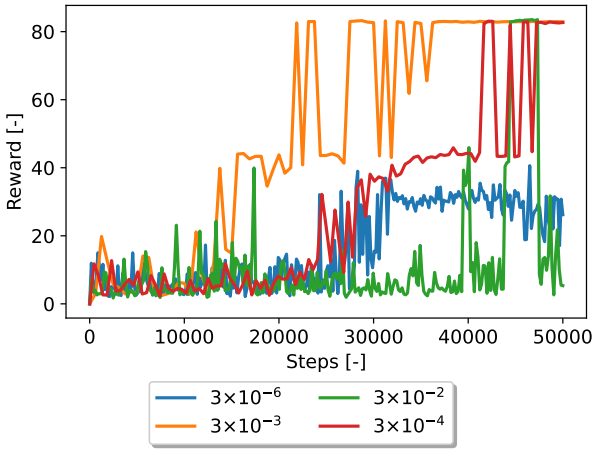

Figure 7: Effect of Critic's Learning Rate on Reward

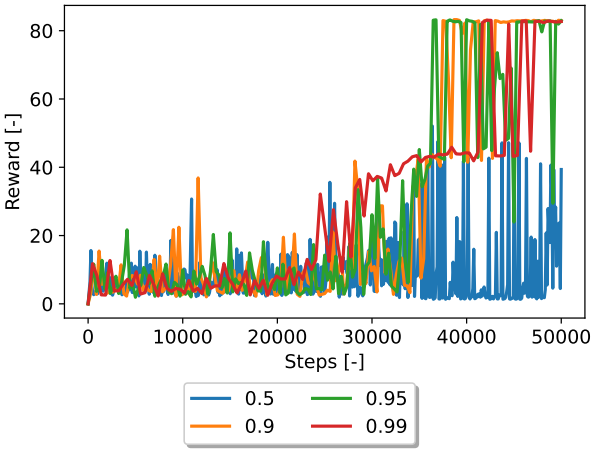

Figure 8: Effect of γ on Reward

Figure 7 shows the effect of the learning rate of the critic network on the reward. It is found that scaling the default value by a factor of 10 (orange) significantly improves the agent's performance. However, a further increase by a factor of 10 (green) is already excessive and leads to worse results than the default value. This means that a too high learning rate leads to instability, likely because associations between the actions and rewards are made to quickly which could lead to wrong connections. Mathematically, it could be that the loss cannot be properly minimised because of too large updates (due to the large learning rate which multiplies the error gradients), leading to oscillations or even divergences. A too low learning rate (blue) also leads to poor performances, since the update rate can get so small that the loss converges to some value far from 0 and no longer decreases.

Figure 8 shows that a very small value of γ (blue) leads to a bad performance, since less importance is given to future rewards. Note that in order to get a higher reward after the first curve, the agent needs to brake and turn right, which can trigger the centripetal acceleration penalty, and the agent ends up preferring not to be penalised in the short run to increase rewards in the long run. Values of 0.9 and 0.95 achieve the maximum reward in less episodes than with 0.99, but the latter appears to have a smoother/more stable reward evolution from 0 to the maximum. Furthermore, it is expected that the short term reward will very often have to be sacrificed in a real circuit (in each curve) in order to maximise the future rewards, and as such, the larger value of 0.99 is preferred.

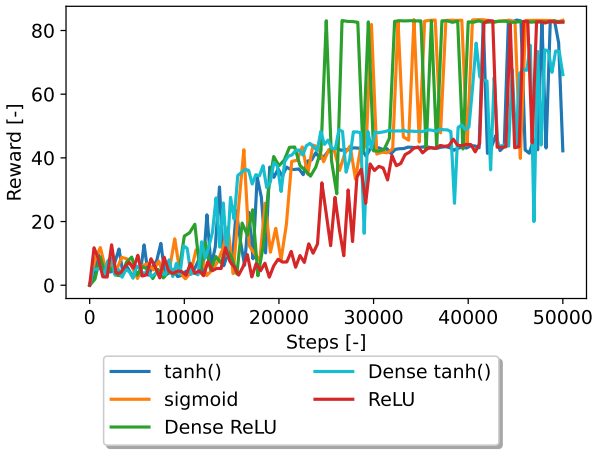

Figure 9: Effect of NN Architecture on Reward

Figure 10: Effect of Reward Scale on Reward

Figure 9 shows how the architecture of the NNs affects the reward. For this analysis, all NNs were changed concurrently. From theory, it was expected that the sigmoid activation performed better than the Rectified Linear Unit (ReLU) activation [43, 44], which can be verified in the figure. Furthermore, the dense architecture was also expected to perform better [45], which was also verified. Although the dense architecture (green) performs better than the default architecture with sigmoid activation (orange), it was found that the dense architecture leads to very large computational times, and as such the default architecture with the sigmoid activation is preferred. Figure 10 shows that the scale of the reward has a tremendous effect on the reward evolution. In this case, scaling the reward by a factor of 20 proves to be extremely beneficial.

After analysing the 16 hyperparameters and understanding their behaviour, these were updated as shown in Table 6, and the respective reward evolution is shown in Figure 11. Hyperparameters not present in the table have the default value from [29].

Table 6: Updated Hyperparameters

Hyper parameters	Reward Scale	τ	NN activation (critic, actor & temperature)	Random steps Update start step Steps per epoch	critic learning rate	actor learning rate	batch size
Value	x 1 $\rightarrow$ x 20	1×10^{-4} $\rightarrow 0.1$	ReLU $\rightarrow$ sigmoid	1000 $\rightarrow$ 500	3×10^{-4} $\rightarrow 3 \times 10^{-3}$	3×10^{-4} $\rightarrow 3 \times 10^{-3}$	256 $\rightarrow$ 64

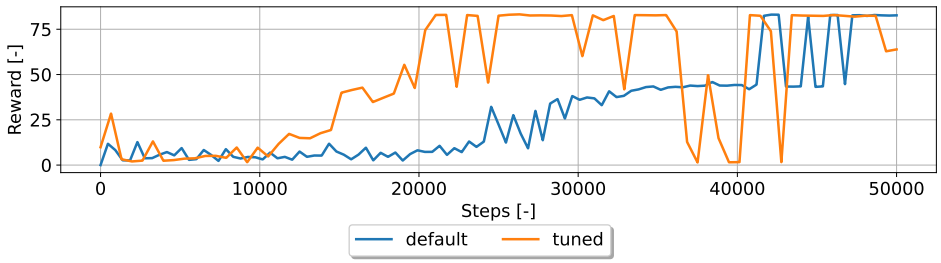

Figure 11: Reward Evolution with Final Hyperparameter Choice

Figure 11 shows how the tuned hyperparameters lead to the agent reaching the end of the test curve faster than with the default values, indicating that the hyperparameter tuning process was successful. Additionally, among the tested hyperparameters that were tested but not mentioned in this report, it is worth pointing out that it was found that the Adam optimiser had the best performance, as expected from theory [46], and that it was found that using 4 critics instead of 2 improved the performance, but only 2 are used because it was found that the computations with 4 critics take longer.

3.3. Reward Signal Design

Employing the full car model with continuous action and state spaces and the distance reward, the agent is capable of completing the full test curve in 40.16 s (very slow when compared with the results from Table 7) without crashing into a wall using the implementation of SAC from [29] with tuned hyperparamters. In order to limit the centripetal acceleration, a linear penalty with a factor of 1 is applied when the agent exceeds the limit, computed according to Equation 4.

With the goal of completing a lap as quickly as possible, common reward signals are: minimum steering (penalty that changes linearly with the absolute value of $\mathrm{steering}$) [2, 47], maximum velocity (reward that changes linearly with the total speed) [47], constant steering (penalty that changes linearly with the absolute difference between consecutive steering inputs) [47], distance with exponential time discount (reward proportional to travelled distance that exponentially decreases with time) [17] and collision penalties (either a constant scalar or proportional to the kinetic energy) [2, 17, 19, 41, 47]. The lap time itself is not a good reward because it is too sparse and the agent cannot associate the reward to any specific action [17].

Besides the presented reward signals, implemented as described in their respective papers, a few more signals were designed based on intuition: *maximum acceleration*, where the agent is encouraged to apply the maximum throttle as much as possible ($r = k$ if $\mathrm{throttle} > 0.99$, else $-dt$, where k is a positive scalar still to be defined); *forward velocity*, where the agent is encouraged to drive as fast as

possible in the direction of the centre-line (reward that changes linearly with the velocity in the centre-line direction); and *straight line*, where the agent is rewarded for not using the steering wheel as to avoid zigzagging, also known as weaving ($r = k$ if $\mid \text{steering} \mid < 0.01$ rad , else $-dt$).

As expected, the reward signal design was the most intricate and time consuming phase of this project and required many iterations. The design process was structured as follows: firstly, the presented reward signals were ran individually and concurrently for 100,000 steps, employing weighting factors to normalise the rewards to the same order of magnitude as the distance reward, in order to identify the most promising signals and find instances of unexpected results; then, the combinations with the fastest trajectories were analysed in greater detail by changing their weighting factors, in order to further improve the learnt strategy; finally, a sensitivity analysis on the random generator's seed is carried out.

In the first and second design phases, over 70 reward signal combinations were ran for 100,000 steps, always including a collision penalty of -1000, with each combination taking about 2 h to run on a Ryzen 7 5800H @ 3.2GHz. Since a lot of these combinations were found to be frivolous, only the relevant resulting times are shown in Table 7.

Table 7: Initial Reward Signal Tests

Reward Signal	Lap Time [s]
Forward Velocity	12.59
Max Velocity & Minimum Steering	12.87
Max Velocity & Constant Steering	13.77
Constant Steering & Max Acc.	14.21
Forward Velocity & Max Acc.	12.61
Forward Velocity	12.23
Max Acc. & Straight Line	11.26
Min Curvature & Max Acc.	10.90
Distance w/ exponential time discount	11.67

The trajectories are clearly faster than with the distance reward, as anticipated. However, the best trajectories show some weaving, and the agent sometimes decelerates on the straights, which could be a sign that the agent realised that "existing" for more time leads to a higher reward than finishing the lap quickly, as shown in Figure 12.

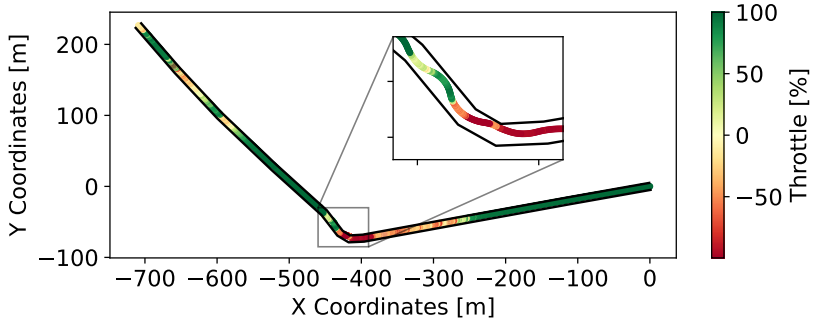

Figure 12: Weaving and Unnecessary Deceleration (Direction: Right to Left)

In order to avoid this, the rewards were modified as to penalise weaving ever more strongly for selected reward signals. This resulted in an improvement of the lap times, as shown in Table 8. However, it was found that too severe a penalty leads to the agent learning that it is better to crash and end the episode than to be penalised for turning, which shows how delicate the reward design process can be.

Table 8: Lap Times after Weight Tuning

Reward Signal	Lap Time Before Tuning [s]	Lap Time After Tuning [s]
Constant Steering & Max Acc.	14.21	13.71
Max Acc. & Straight Line	11.26	10.89
Distance w/ exponential discount	11.67	10.27
Min Curvature & Max Acc.	10.90	10.90

At last, the reward signal can be chosen: the distance with exponential discount, as described in [17]. A final sensitivity analysis is now performed in order to assess if the same results are obtained with different random generator's seeds:

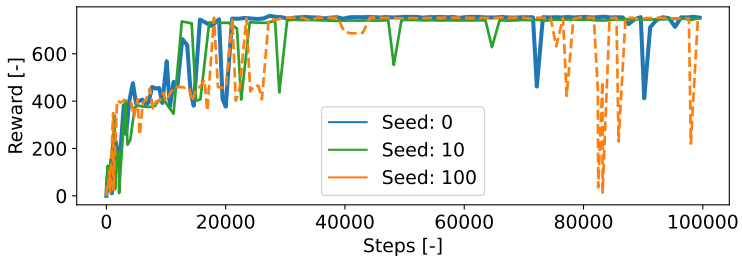

Figure 13: Reward Evolution for Different Random Generator Seeds

Figure 13 shows how training the agent using seed 10 or seed 100 leads to the same maximum reward as the seed in which the reward signal and the hyperparameters were tuned (seed 0), showing that the algorithm performs well regardless of the seed, as expected from SAC due to its entropy-based reward.

4. Results

The final result is analysed on the circuit of Silverstone, where the agent was trained for 750,000 steps. Figure 14 shows the resulting lap times as a function of the maximum rewards achieved. The best lap time in 750,000 steps of training was 84.06 s, which is 6 seconds faster than the fastest lap time set by Max Verstappen in 2023: 90.275 s [48]. This likely occurs due to mismatches between reality and the simulation model. Figure 15 shows the resulting race line at Silverstone.

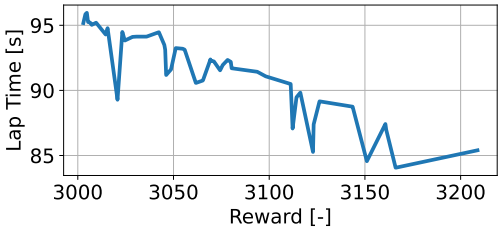

Figure 14: Evolution of Lap Time with Reward

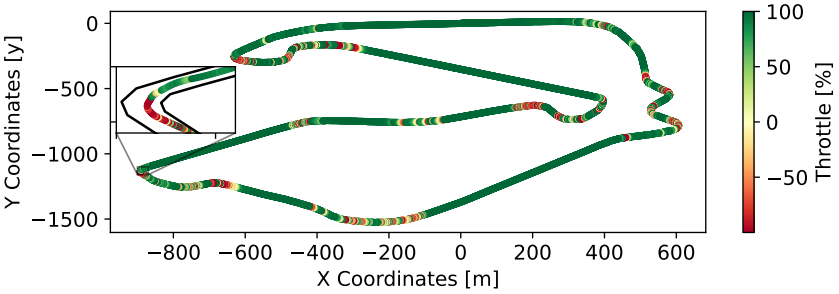

Figure 15: Resulting Race Line at Silverstone (Direction: Clockwise)

Comparing Figure 15 to Figure 12, big improvements can be seen: namely, the agent does not brake randomly in the straights (the throttle is now at 100%) and there is no unnecessary weaving. It appears that the agent learned to correctly brake before the curves and turn smoothly.

As mentioned in the beginning of this project, a secondary goal was to extend the learnt strategy to an untrained circuit. Therefore, the agent with the policy from Silverstone is used in Zandvoort:

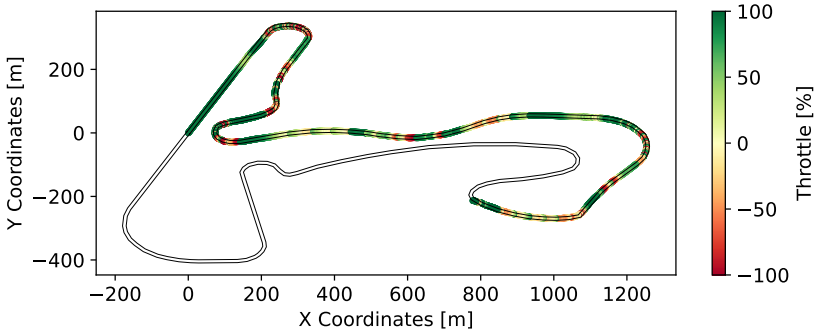

Figure 16: Resulting Race Line at Zandvoort

Figure 16 shows that the policy from Silverstone only allows the agent to complete only half of the Zandvoort circuit, crashing in 42.61 s, even though the agent learnt to drive straight on the straights, brake before curves and turn smoothly. This can be interpreted as a partial success, since the agent can drive for some time before colliding with the wall. However, in a real life scenario, this would not be acceptable. A potential solution could be to either increase the learning steps or to train the agent on a very complex circuit, where it could be prepared for all kinds of curves.

For a deeper insight into the learnt strategy, an algorithm known as DeepSHAP is used, which consists in a framework used to explain the outputs from DNNs [49], using the implementation from [50].

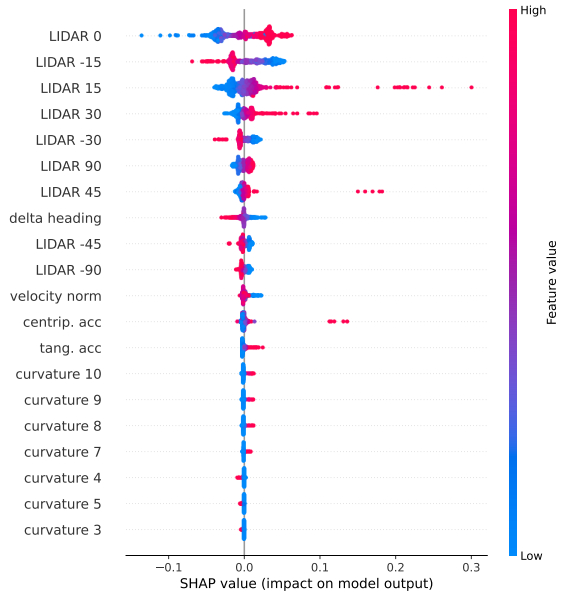

Figure 17: Throttle Outputs

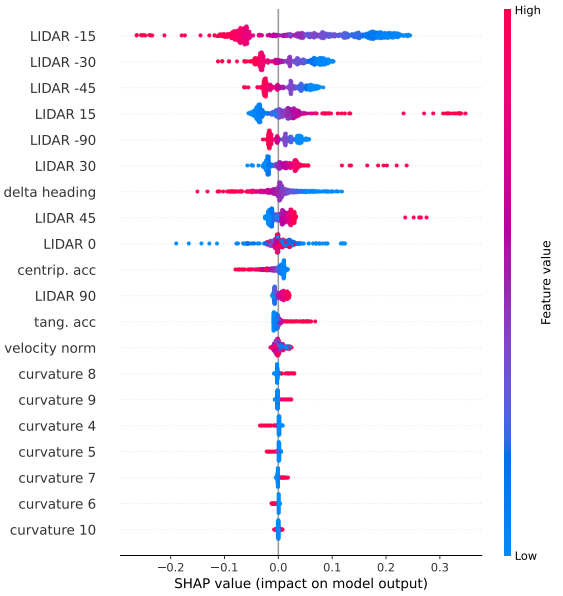

Figure 18: Wheel Outputs

Figures 17 and 18 show how the states relate to the throttle and wheel outputs, respectively. As expected, the agent learnt to accelerate when it is far from walls: for example, Figure 17 shows that, for LIDAR 0 (the LiDAR measurement at 0°), large values lead to positive throttle outputs (accelerating), whereas small values (indicating that a wall is close) lead to negative outputs (braking). Additionally, Figure 18 shows how the agent learnt to turn right when LIDAR -15 (the LiDAR measurement that points 15° to the left) has a small value, indicating that there is a wall to the left, and it learnt to turn right when LIDAR 15 (LiDAR measurement that points 15° to the right) has a small value (wall on the right). It was expected that the future curvatures would have a stronger effect on the output, but this likely could be fixed by training the agent for more steps. These results, combined with some verification and validation work, could eventually be used to develop a simple PID controller.

5. Conclusion

The main goal of this project was to implement a DRL algorithm that would optimise the race line of a Formula 1 car in an unseen circuit and without any knowledge of the controls of the vehicle. By developing a car and a track model, adequately formulating a state, understanding the effects of hyperparameters (and tuning them) and tuning the reward signal, this goal was successfully achieved. Additionally, the learnt strategy was understood by analysing the resulting race line at Silverstone and by applying the DeepSHAP algorithm. The secondary goal of being able to extend a learnt policy to an unseen circuit was partly achieved, as the agent was able to complete half a lap at Zandvoort. A potential way to fix this could be to train the agent on a more complex circuit first.

This was the first time I worked with Machine Learning of any kind, which proved to be a tough challenge. My main takeaways from the project are: (1) I should report my own work to myself, in order to facilitate tracking changes in code and in data files, and (2) the reward design is very intricate as the agent often finds unforeseen ways to maximise the reward, such as finding the optimal tradeoff between driving fast and receiving the reward for as long as possible (instead of simply driving as fast as possible).

References

[1] Johannes Betz et al. 'Autonomous vehicles on the edge: A survey on autonomous vehicle racing'. In: *IEEE Open Journal of Intelligent Transportation Systems* 3 (2022), pp. 458–488.

[2] Benjamin Evans, Herman A Engelbrecht and Hendrik W Jordaan. 'From navigation to racing: Reward signal design for autonomous racing'. In: *arxiv. abs/2103.10098*. 2021. URL: https://arxiv.org/pdf/2103.10098v1.pdf (visited on 02/08/2023).

[3] Jayanth Bhargav et al. 'Deriving spatial policies for overtaking maneuvers with autonomous vehicles'. In: *2022 14th International Conference on COMmunication Systems & NETworkS (COMSNETS)*. IEEE. 2022, pp. 859–864.

[4] M. Bevilacqua, A. Tsourdos and A. Starr. 'Particle swarm for path planning in a racing circuit simulation'. In: *2017 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*. 2017, pp. 1–6. DOI: 10.1109/I2MTC.2017.7969735.

[5] Luigi Cardamone et al. 'Searching for the Optimal Racing Line Using Genetic Algorithms'. In: 2010, pp. 388–394. DOI: 10.1109/ITW.2010.5593330.

[6] Nitin R. Kapania, John Subosits and J. Christian Gerdes. 'A Sequential Two-Step Algorithm for Fast Generation of Vehicle Racing Trajectories'. In: *Journal of Dynamic Systems, Measurement, and Control* 138.9 (2016). DOI: 10.1115/1.4033311. URL: https://doi.org/10.1115%2F1.4033311.

[7] Thomas Herrmann et al. 'Minimum Race-Time Planning-Strategy for an Autonomous Electric Racecar'. In: *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2020. DOI: 10.1109/itsc45102.2020.9294681. URL: https://doi.org/10.1109%2Fitsc45102.2020.9294681.

[8] Alexander Liniger, Alexander Domahidi and Manfred Morari. 'Optimization-based autonomous racing of 1: 43 scale RC cars'. In: *Optimal Control Applications and Methods* 36.5 (2015), pp. 628–647.

[9] Tommaso Novi et al. 'Real-time control for at-limit handling driving on a predefined path'. In: *Vehicle system dynamics* (2019).

[10] Juraj Kabzan et al. 'Learning-Based Model Predictive Control for Autonomous Racing'. In: *IEEE Robotics and Automation Letters* 4.4 (2019), pp. 3363–3370. DOI: 10.1109/LRA.2019.2926677.

[11] Mahmoud F. Elmorshedy et al. 'Recent Achievements in Model Predictive Control Techniques for Industrial Motor: A Comprehensive State-of-the-Art'. In: *IEEE Access* 9 (2021), pp. 58170–58191. DOI: 10.1109/ACCESS.2021.3073020.

[12] Grady Williams et al. 'Aggressive driving with model predictive path integral control'. In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. 2016, pp. 1433–1440. DOI: 10.1109/ICRA.2016.7487277.

[13] Grady Williams et al. *Information Theoretic Model Predictive Control: Theory and Applications to Autonomous Driving*. 2017. arXiv: 1707.02342 [cs.RO].

[14] Nvidia. *CUDA*. URL: https://developer.nvidia.com/cuda-zone (visited on 26/07/2023).

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

[16] Muddasar Naeem, Syed Tahir Hussain Rizvi and Antonio Coronato. 'A Gentle Introduction to Reinforcement Learning and its Application in Different Fields'. In: *IEEE Access* 8 (2020), pp. 209320–209344. DOI: 10.1109/ACCESS.2020.3038605.

[17] Florian Fuchs et al. 'Super-Human Performance in Gran Turismo Sport Using Deep Reinforcement Learning'. In: *IEEE Robotics and Automation Letters* 6.3 (2021), pp. 4257–4264. DOI: 10.1109/lra.2021.3064284. URL: https://doi.org/10.1109%2Flra.2021.3064284.

[18] Peide Cai et al. 'High-Speed Autonomous Drifting With Deep Reinforcement Learning'. In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 1247–1254. DOI: 10.1109/lra.2020.2967299. URL: https://doi.org/10.1109%2Flra.2020.2967299.

[19] Edoardo Ghignone, Nicolas Baumann and Michele Magno. 'TC-Driver: A Trajectory Conditioned Reinforcement Learning Approach to Zero-Shot Autonomous Racing'. In: *Field Robotics* 3.1 (2023), pp. 637–651. DOI: 10.55417/fr.2023020.

[20] V. Mnih et al. 'Human-level control through deep reinforcement learning'. In: *Nature* 518 (2015), pp. 529–533. DOI: 10.1038/nature14236.

[21] Timothy P Lillicrap et al. 'Continuous control with deep reinforcement learning'. In: *arXiv preprint arXiv:1509.02971* (2015).

[22] Peter Henderson et al. 'Deep reinforcement learning that matters'. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 32. 1. 2018.

[23] Yan Duan et al. 'Benchmarking deep reinforcement learning for continuous control'. In: *International conference on machine learning*. PMLR. 2016, pp. 1329–1338.

[24] Tuomas Haarnoja et al. 'Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor'. In: *International conference on machine learning*. PMLR. 2018, pp. 1861–1870.

[25] Tuomas Haarnoja et al. 'Soft actor-critic algorithms and applications'. In: *arXiv preprint arXiv:1812.05905* (2018).

[26] Scott Fujimoto, Herke Hoof and David Meger. 'Addressing function approximation error in actor-critic methods'. In: *International conference on machine learning*. PMLR. 2018, pp. 1587–1596.

[27] Tuomas Haarnoja et al. 'Reinforcement learning with deep energy-based policies'. In: *International conference on machine learning*. PMLR. 2017, pp. 1352–1361.

[28] Brian D Ziebart. *Modeling purposeful adaptive behavior with the principle of maximum causal entropy*. Carnegie Mellon University, 2010.

[29] Takuma Seno and Michita Imai. 'd3rlpy: An Offline Deep Reinforcement Learning Library'. In: *Journal of Machine Learning Research* 23.315 (2022), pp. 1–20. URL: http://jmlr.org/papers/v23/22-0017.html.

[30] Adam Paszke et al. 'PyTorch: An Imperative Style, High-Performance Deep Learning Library'. In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[31] Greg Brockman et al. 'Openai gym'. In: *arXiv preprint arXiv:1606.01540* (2016). URL: https://gymnasium.farama.org/ (visited on 17/06/2023).

[32] Matthew O'Kelly et al. 'textscF1TENTH: An Open-source Evaluation Environment for Continuous Control and Reinforcement Learning'. In: *NeurIPS 2019 Competition and Demonstration Track*. PMLR. 2020, pp. 77–89.

[33] Peter Wright and Tony Matthews. *Formula 1 technology*. SAE International, 2001.

[34] Flow Racers. *How Fast Do F1 Cars Accelerate?* URL: https://flowracers.com/blog/how-fast-f1-cars-accelerate/ (visited on 18/06/2023).

[35] Technical Formula 1 Dictionary. *Turning Acceleration*. URL: https://www.formula1-dictionary.net/turning_acceleration.html (visited on 18/06/2023).

[36] Technical Formula 1 Dictionary. *Aerodynamics of F1*. URL: https://www.formula1-dictionary.net/aerodynamics_of_f1.html (visited on 18/06/2023).

[37] D.J.N. Limebeer, G. Perantoni and A.V. Rao. 'Optimal control of Formula One car energy recovery systems'. In: *International Journal of Control* 87.10 (2014), pp. 2065–2080. DOI: 10.1080/00207179.2014.900705.

[38] Motorsport.com. *How much does an F1 car weigh?* URL: https://www.motorsport.com/f1/news/how-much-does-an-f1-car-weigh-in-2023/10437685/ (visited on 19/06/2023).

[39] Professional's HQ. *Average F1 Car Speed 2023*. URL: https://professionalshq.com/average-f1-car-speed-we-pulled-the-real-data-2023/ (visited on 19/06/2023).

[40] Tomislav Bacinger. *Formula 1 Circuits*. https://github.com/bacinger/f1-circuits/tree/master. June 2023.

[41] Axel Brunnbauer et al. *Latent Imagination Facilitates Zero-Shot Transfer in Autonomous Racing*. 2022. arXiv: 2103.04909 [cs.LG].

[42] Nitish Shirish Keskar et al. *On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima*. 2017. arXiv: 1609.04836 [cs.LG].

[43] Prajit Ramachandran, Barret Zoph and Quoc V Le. 'Searching for activation functions'. In: *arXiv preprint arXiv:1710.05941* (2017).

[44] Bing Xu et al. 'Empirical evaluation of rectified activations in convolutional network'. In: *arXiv preprint arXiv:1505.00853* (2015).

[45] Samarth Sinha et al. 'D2rl: Deep dense architectures in reinforcement learning'. In: *arXiv preprint arXiv:2010.09163* (2020).

[46] Sebastian Ruder. 'An overview of gradient descent optimization algorithms'. In: *arXiv preprint arXiv:1609.04747* (2016).

[47] Ruiqi Zhang et al. 'Residual Policy Learning Facilitates Efficient Model-Free Autonomous Racing'. In: *IEEE Robotics and Automation Letters* 7.4 (2022), pp. 11625–11632. DOI: 10.1109/LRA.2022.3192770.

[48] Formula 1. *Lap Times at Silverstone 2023*. URL: https://www.formula1.com/en/results.html/2023/races/1214/great-britain/fastest-laps.html (visited on 12/08/2023).

[49] Hugh Chen, Scott Lundberg and Su-In Lee. *Explaining Models by Propagating Shapley Values of Local Components*. 2019. arXiv: 1911.11888 [cs.LG].

[50] SHAP documentation. *Deep Explainer*. URL: https://shap-lrjball.readthedocs.io/en/latest/examples.html#deep-explainer (visited on 05/08/2023).

[51] Joshua Achiam. *Spinning Up in Deep Reinforcement Learning*. 2018. URL: https://spinningup.openai.com/en/latest/algorithms/sac.html (visited on 02/08/2023).

[52] Tuomas Haarnoja et al. *Learning to Walk via Deep Reinforcement Learning*. 2019. arXiv: 1812.11103 [cs.LG].

Appendix A

SAC

SAC is an off-policy actor-critic DRL algorithm [24, 25] that applies the maximum entropy framework, which not only leads to better exploration [27] but also increases robustness against model or estimation errors [28]. TD3 uses target policy smoothing because it trains a deterministic policy [26], but this is not needed in SAC since the noise from its stochastic policy is enough to achieve a similar effect. The policy objective $J(\pi)$ including the entropy term $\mathcal{H}$ is given by:

$$J(\pi) = \sum_{t=0}^{N} \mathop{\mathbb{E}}_{\boldsymbol{a}_t \sim \pi} \left[\gamma^t \left(r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \alpha \mathcal{H} \left(\pi(\cdot \mid \boldsymbol{s}_t) \right) \right) \right], \tag{A.1}$$

$$\text{with } \mathcal{H} \left(\pi(\cdot \mid \boldsymbol{s}_t) \right) = \mathop{\mathbb{E}}_{\boldsymbol{a}_t \sim \pi} \left[-\log \pi(\boldsymbol{a} \mid \boldsymbol{s}_t) \right], \tag{A.2}$$

where α is the temperature parameter that controls the relative importance of the entropy term, controlling the stochasticity of the optimal policy. This formulation of the objective encourages the agent to maximise the reward while acting as randomly as possible.

In order to mitigate the positive bias in the policy improvement [25], SAC makes use of two soft Q-functions, modelled as two NNs parameterised by θ_1 and θ_2 and trained independently. The Q-function is obtained by modifying the Bellman equation to include the entropy's log-likelihood term from Equation A.2 and the temperature parameter α:

$$Q_{\theta_i}^{\pi_\phi} (\boldsymbol{s}_t, \boldsymbol{a}_t) = \mathop{\mathbb{E}}_{\boldsymbol{a}_{t+1} \sim \pi_\phi} \left[r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma \left(Q_{\theta_i}^{\pi_\phi} (\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1}) - \alpha \log \pi_\phi (\boldsymbol{a}_{t+1} \mid \boldsymbol{s}_{t+1}) \right) \right]. \tag{A.3}$$

In order to increase learning stability, a separate target value network $Q_{\text{target},\theta_i}^{\pi_\phi}$ that slowly tracks the main network is used. The target value network weights are updated less frequently than the main Q-network with a smoothing constant τ:

$$\theta_{i,\text{target}} = \tau \theta_{i,\text{target}} + (1 - \tau) \theta_i. \tag{A.4}$$

The critic's goal is to minimise the loss $\mathcal{L}_{\theta_i}$ computed with the mean squared Bellman error. Since there are two target networks, SAC conservatively uses the minimum target Q-value from the two approximators. The tuple $(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1})$ is sampled from a minibatch $\mathcal{B}$ from the replay buffer, since SAC is an off-policy algorithm which makes use of past experiences. This is how SAC increases sampling efficiency compared to the previously mentioned on-policy algorithms. The action $\boldsymbol{a}_{t+1}$, on the other hand, is obtained directly from the current policy:

$$\mathcal{L}_{\theta_i} = \mathop{\mathbb{E}}_{\substack{\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1} \sim \mathcal{B} \\ \boldsymbol{a}_{t+1} \sim \pi_\phi}} \left[r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma \left(\min_{j=1,2} Q_{\theta_j}^{\pi_\phi} (\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1}) - \alpha \log \pi_\phi (\boldsymbol{a}_{t+1} \mid \boldsymbol{s}_{t+1}) \right) - Q_{\theta_i}^{\pi_\phi} (\boldsymbol{s}_t, \boldsymbol{a}_t) \right]^2. \tag{A.5}$$

The policy is modelled as a Gaussian distribution with mean and variance given by a NN parameterised by ϕ. The output is squashed by a tanh function in order to bound the actions in a finite range. The

authors of SAC apply the NN transformation $\boldsymbol{a}_t = \boldsymbol{f}_\phi(\boldsymbol{\epsilon}_t, \boldsymbol{s}_t)$ to reparameterise the policy, which is a convenient trick that results in a lower variance estimator [25], where $\boldsymbol{\epsilon}$ is an input noise vector. The actor network's objective is expressed as:

$$J(\phi) = \underset{\substack{\boldsymbol{s}_t \sim \mathcal{B} \\ \boldsymbol{a}_t \sim \pi_\phi \\ \boldsymbol{\epsilon}_t \sim \mathcal{N}}}{\mathbb{E}} \left[\min_{i=1,2} Q_{\theta_i}^{\pi_\phi}\left(\boldsymbol{s}_t, \boldsymbol{f}_\phi(\boldsymbol{\epsilon}_t, \boldsymbol{s}_t)\right) - \alpha \log \pi_\phi\left(\boldsymbol{f}_\phi(\boldsymbol{\epsilon}_t, \boldsymbol{s}_t) \mid \boldsymbol{s}_t\right) \right]. \tag{A.6}$$

The original SAC algorithm [24] is very sensitive to the temperature hyperparameter α. In the new version [25], automatic entropy/temperature adjustment is implemented. While this is a simple modification, it was found that it largely reduces the need for hyperparameter tuning for each scenario. This automatic adjustment makes use of the optimal policy π^* and its objective is given by:

$$J(\alpha) = \underset{\substack{\boldsymbol{s}_t \sim \mathcal{B} \\ \boldsymbol{a}_t \sim \pi_t^*}}{\mathbb{E}} \left[-\alpha \log \pi_t^*(\boldsymbol{a}_t \mid \boldsymbol{s}_t) - \alpha \bar{\mathcal{H}} \right], \tag{A.7}$$

where $\bar{\mathcal{H}}$ is the target entropy, optimally defined as $-\mid \mathcal{A} \mid$ [25].

The SAC algorithm is presented in pseudo-code in Algorithm 1:

Algorithm 1 SAC Algorithm, Adapted from [25, 51, 52]

Input: Learning Rates λ_θ, λ_ϕ, λ_α, Smoothing Parameter τ
  Initialise policy parameter π, double Q-function parameters θ_i, $i \in \{1, 2\}$
  Initialise empty replay buffer: $\mathcal{D} \leftarrow \varnothing$
  for each iteration **do**
    for each environment time step t **do**
      Sample action: $\boldsymbol{a}_t \sim \pi_\phi(\cdot \mid \boldsymbol{s}_t)$
      Sample state transition and reward from the environment $\boldsymbol{s}_{t+1}$, $r(\boldsymbol{s}_t, \boldsymbol{a}_t)$
      Store transition sample in memory buffer: $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1})\}$
      Sample minibatch $\mathcal{B}$ from $\mathcal{D}$
      if $\boldsymbol{s}_{t+1}$ is terminal **then**
        Reset environment
      end if
    end for
    for each gradient step **do**
      Update Q-function parameters:

$$\theta_i \leftarrow \theta_i - \lambda_\theta \nabla_{\theta_i} \frac{1}{\mid \mathcal{B} \mid} \sum_{\{(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1})\} \in \mathcal{B}} \mathcal{L}_{\theta_i} \quad , \quad i \in \{1, 2\}$$

      Update policy parameter:

$$\phi \leftarrow \phi + \lambda_\phi \nabla_\phi \frac{1}{\mid \mathcal{B} \mid} \sum_{\boldsymbol{s}_t \in \mathcal{B}} J(\phi)$$

      Update temperature hyperparameter: $\alpha \leftarrow \alpha - \lambda_\alpha \nabla_\alpha J(\alpha)$
    end for
    Update target network: $\theta_{\text{target},\, i} \leftarrow \tau \theta_{\text{target},\, i} + (1 - \tau)\theta_i \quad , \quad i \in \{1, 2\}$
  end for

Appendix B

Integrator Tuning

With the goal of minimising the wall-clock simulation time, an integrator analysis is carried out in order to trade-off integration errors for a reduced run-time. Figure B.1 shows how the integration error changes with the fixed step size for the Euler and the Runge-Kutta 4 (RK4) integrators:

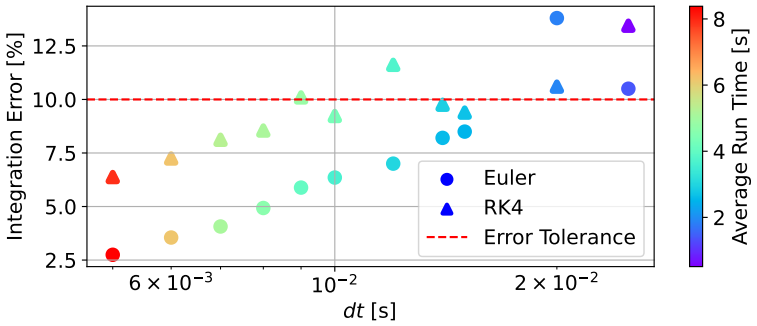

Figure B.1: Integration Errors

Since actions can change very quickly and unexpectedly, it was considered that variable step integrators would not produce optimal trajectories, as they could lead to skipping the breaking or turning points. Additionally, with the goal of comparing the fastest integrators, only the Euler and RK4 integrators were tested: Euler was chosen due to its simplicity, and RK4 was chosen because it is the highest-order explicit RK method that requires the same number of steps as the order of accuracy. Figure B.1 was obtained by comparing the maximum velocity in a curve with different step sizes to the maximum velocity for an extremely small step size, assumed to have a negligible integration error. By arbitrarily establishing a maximum error of 10% (about 30 km/h), the fastest integrator that can be chosen (the right-most point below the red line) is the Euler integrator with a fixed step size $dt = 0.015$ s, where each test curve episode takes about 2 s, which is slightly faster than the RK4 with the same step size.