Developing Effective Autonomous Driving for Sim Racing through Reinforcement Learning in Assetto Corsa

Jurre de Ruiter
j.deruiter-3@student.utwente.nl
University of Twente
Enschede, The Netherlands

ABSTRACT

This study contributes insights into the intersection of reinforcement learning (RL), sim(ulation) racing, and autonomous driving, specifically within Assetto Corsa (AC) as a sim racing environment. The difference in RL algorithms is explained with a reasoning for the suitability of the Soft-Actor-Critic (SAC) algorithm for an autonomous car racing agent in AC. Based on this, a system design is presented for using AC as an experimentation environment for training the model-free, off-policy SAC algorithm. Specific policy details, hyperparameters, and reward factors are discussed in the context of addressing the lap-completion problem. Here, the objective is to find a policy that achieves completion of a given track with a given car. Results are presented for five different reward functions, on which we conclude the fifth (using the heading and off-center errors) to be the most effective. Future steps for research are laid out with the goal of actually completing a full lap, and ultimately optimizing the minimum-time problem as well. Here the goal is not only to finish, but finish with minimal time.

KEYWORDS

sim racing, reinforcement learning, autonomous driving, assetto corsa, esports, racing performance, soft-actor-critic, data analysis

1 INTRODUCTION

Simulation racing, also known as 'sim racing', is the concept of virtual racing taking place on simulated software [21]. Here, the term "simulated" refers to the accuracy of the real-world like variables being mimicked by the cars and tracks in the software. Where traditional 'Arcade' racing video games focus on a more abstracted entertainment experience for the every-day gamer, sim racing aims to replicate the intricacies of real-world racing, providing highly realistic driving experiences. In recent years, the concept of sim racing has gotten significantly more popular through the rise of esports. Esports has seen significant growth [13], and countries are beginning to recognize professional gamers as athletes [16]. Additionally, the 2021 Olympics even featured an official esports event, including a sim racing tournament [4]. According to video game distribution platform Steam [5], the most popular realism focused sim racing games at the time of writing are Assetto Corsa (AC) [32], Assetto Corsa Competizione [31], Automobilista 2 [33], rFactor 2 [1], and iRacing [15].

The fact that sim racing environments simulate reality to a close extent, makes them candidate for a platform for research and experimentation with potential real-world impact. Simulated cars and real-world inspired tracks can generate vast quantities of data, offering a unique opportunity to blend the fields of autonomous driving and simulated racing. In autonomous driving, an agent takes full control over a vehicle, relying on environment analyses, decision-making processes, and precise control to navigate autonomously. Autonomous racing is a subfield of autonomous driving where the goal is to drive around a race track as fast as possible. Analyzing autonomous racing agents offers insights into novel strategies, providing the opportunity for racers to enhance their racing performance.

Central to the development of autonomous racing agents is the application of Artificial Intelligence (AI), in particular the field of Reinforcement Learning (hereafter referred to as 'RL'). RL considers an agent interacting with the environment, learning a policy, by trial and error, for sequential decision making problems in a wide range of fields in both natural and social sciences, and engineering [24]. By utilizing RL algorithms and training strategies, we can aim to develop a self-driving bot car capable of effectively navigating dynamic racing environments.

To ensure clarity of the problem at hand, we define the *autonomous racing problem* as the combination of two sub-problems:

- The *Lap-Completion Problem:* The aim to find a policy that achieves completion of a given track with a given car.
- The *Minimum-Time Problem:* The aim to find a policy that minimizes the total lap time for a given car and track.

While classical approaches for addressing autonomous racing problems have been researched extensively and have demonstrated impressive results, there is a noticeable gap in the literature regarding the utilization of AC as an environment for such studies. Despite being recognized as one of the leading racing simulators in its domain, known for its realism and support for extensive modifications, no prior research has explored the application of RL within the AC environment. Given the realism and widespread adoption of AC, establishing a framework for employing RL in this simulator is significant for professional racers who use AC as their training platform and for future research endeavors in the field.

TScIT 40, February 2, 2024, Enschede, The Netherlands

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

The primary challenge in this context is to design a framework capable of seamlessly collecting and processing data in real-time. There are several possible methods for data collection, and selecting the approach that offers the highest reliability and lowest latency is essential to a successful RL implementation. Additionally, the chosen RL algorithm must be not only effective but also tailored to suit the demands of sim racing environments, which tend to be relatively complex. Furthermore, it is imperative to make fitting observations to acquire the right data, so that a reward function can be developed for training the RL model to learn an effective policy.

This paper addresses these challenges by presenting a system design for using AC as an RL environment. It describes how the model-free, off-policy Soft-Actor-Critic (hereafter referred to as 'SAC') algorithm can be applied to achieve autonomous racing in AC. Furthermore, various reward functions are designed and tested using different observations as parameters. This is done with the aim of addressing the primary research question (RQ), which is the following: *"What factors and configuration enable a reinforcement learning algorithm to autonomously navigate a sim racing car to finish within Assetto Corsa?"*. This effectively means the focus of the research lies on addressing the lap-completion problem of the autonomous racing problem. In order to properly answer this research question, this study covers three sub-questions, which are the following:

RQ1: What are existing RL algorithms, and how do they differ in terms of use case?
RQ2: How can real-time data be effectively extracted from AC to enable the design of a system for autonomous driving?
RQ3: How can successful autonomous driving within sim racing be quantified into a reward function?

1.1 Contributions of this paper

By answering the research questions, this study aims to achieve the following contributions:

- Present a clear overview of the difference in algorithms within the RL domain, to inform future research of the avenues that can be taken to achieve autonomous racing.
- Explain the suitability and potential of the SAC algorithm in the domain of autonomous sim racing. The overarching goal here is to stimulate more research into the use of this algorithm, as it shows potential for further advancement of autonomous racing.
- Provide a technical system design for utilizing the racing simulator Assetto Corsa (AC) as an experimentation environment for training an RL model, effectively and efficiently managing real-time input and output. Given that no research has been done on RL in AC before, this system design aims to serve as a framework for future research using AC.
- Provide an overview of the observations, actions, completion signals, hyperparameters, and reward functions enabling effective utilization of SAC for developing an autonomous racing agent. The aim here is to inspire future research to build upon this study and address the minimum-time problem of the autonomous racing problem.

1.2 Outline of this paper

Section 2 serves as an in-depth literature review, providing the theoretical background for the subsequent sections. Here, the theoretical and technical aspects of RL and the SAC algorithm are explained in detail. Furthermore, work related to the present study is mentioned, aiming to highlight the strengths and limitations of prior approaches, thereby reinforcing the contributions of this paper and identifying gaps that this research aims to address. Section 3 details the methodology employed in this research, focusing on the data collection, preparation, processing, and evaluation processes. This is supported by an overview of algorithmic input, output, hyperparameters, and the experiment setup. Section 4 presents the design of the system used for utilizing AC as an RL environment throughout the experiments. The subsequent section will present the results of the experiments outlined in the research methodology, supported by graphical overviews. Finally, conclusions will be drawn based on these results and suggestions will be made for future research based on a discussion of the conclusions.

2 LITERATURE REVIEW

In this section, background information will be provided upon which the subsequent sections of the paper will be built. In addition, existing work related to the relevant domains will be highlighted.

2.1 Assetto Corsa

Assetto Corsa (AC) [32], developed by Kunos Simulazioni, is a popular racing simulator known for its realism and support for extensive modifications. The game provides a diverse range of cars and tracks, offering a dynamic and immersive racing experience.

AC supports custom Python apps, which can be loaded into a session at runtime. To support this, the game provides an API (Application Programming Interface) [25] and shared memory reference [2]. This extensibility is a necessity for this research, as the ability to access simulator data in real-time allows for effective communication between the autonomous driving agent and the simulator environment.

There are several limitations to AC's API and modification systems. Most notably, the Python version (3.3.5) [9] supplied to external apps is dated, and the API's lap invalidation flag is not persistent. Understanding these limitations is crucial for the effective implementation and testing of a system for this research. In the subsequent sections, we will explore how these considerations shaped the methodology and implementation of the SAC algorithm for achieving autonomous driving in the AC simulation environment.

2.2 Reinforcement learning

In machine learning, various methods exist to train learning and decision-making. These methods can be divided into three groups: (I) supervised learning, (II) unsupervised learning, and (III) reinforcement learning approaches.

(I) Supervised learning involves a dataset with both data and corresponding ground truth labels, prompting agents to predict these labels. A notable sub-method within supervised learning is *imitation learning*. This approach involves training the agent to replicate the behaviour of a human based on a labeled dataset

of their actions. An example of existing research within the autonomous driving domain is that of Farag et al. [8], where recorded driver data was used as input for a Convolutional Neural Network to learn safe driving behavior and smooth steering maneuvering.

(II) Unsupervised learning still involves a dataset, but without the corresponding ground truth labels, requiring agents to discern patterns and group data based solely on its inherent structure.

(III) Reinforcement learning is the approach of this research. Here, the focus lies on learning what to do –how to map situations to actions– to maximize a numerical reward signal by trial and error. The learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them and generating its own data [24]. In autonomous driving, RL-based approaches can be used to teach an agent to navigate the racing environment, make decisions, and optimize its behavior over time without any prior knowledge.

The main components of RL are the *agent*, and the *environment* [20]. The environment is the world that the agent lives in and interacts with. At every step of interaction t, the agent sees a set of observations of the state of the world and a reward for how good the particular state is, and then decides on an action to take based on its policy $\pi_\theta(s)$. This agent-environment interaction loop is a Markov Decision Process (MDP), depicted in Figure 1. This loop occurs a finite number of times, after which the sequence of states and actions in the environment is called an episode, $\epsilon = (s_0, a_0, s_1, a_1, ...)$.

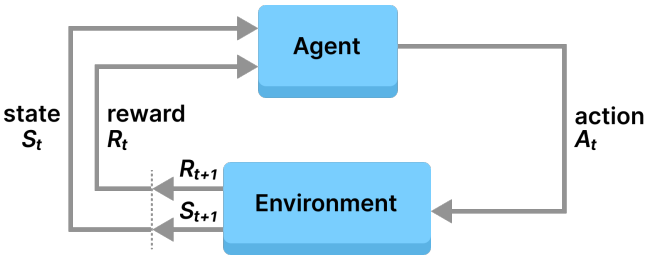


Figure 1: Agent-environment interaction loop

The goal of the agent is to maximize its gain G, which is the sum of future rewards r at every timestep t, discounted by γ:

$$G_t = \sum_{k=t}^{\infty} r_k \gamma^{k-t} \tag{1}$$

A discount factor of 1 implies the agent has no preference for short term or long term rewards. A discount value smaller than 1 implies a preference for immediate rewards. This preference becomes stronger as the discount factor gets closer to 0. At every timestep, the reward r_k is determined through a reward function R. Designing an effective reward function (reward shaping) is an essential part of solving the autonomous racing problem.

2.2.1 Types of RL algorithms. In RL, algorithms are categorized as either model-based or model-free, based on whether the agent has access to a model of the environment.

Model-based methods utilize a learnt model of the environment, allowing for potential gains in sample efficiency but introducing challenges in implementation and tuning. In the model-based domain, research has been conducted using classical approaches where the autonomous driving problem is broken down into trajectory planning and trajectory tracking. Model Predictive Control (MPC) [17][26] is a promising approach here, but has limitations such as possibly high computational complexity and the lack of flexibility in the cost function design.

Model-free methods are favored for their ease of implementation and tuning. The biggest advantages in the context of sim racing and AC is that they do not require perfect knowledge about the vehicle and its environment. Given that it is more widely useful to develop a generalized and adaptive learning system, this research focuses on the more popular model-free space. Within model-free learning, another critical branching point is the question of what to learn. The two main approaches are:

- **Policy-based Learning:** This approach involves learning an approximator $V_\phi(s)$ for optimizing the parameters θ of the policy $\pi_\theta(a|s)$. Policy optimization is often *on-policy*, as it only uses data collected while acting according to the most recent policy. The approximator function is denoted by:

$$V^\pi(s) = \underset{\tau\sim\pi}{E}[R(\tau)\,|s_0 = s\,] \tag{2}$$

- **Value-based Learning:** In this= approach, an approximator $Q_\theta(s, a)$ is learnt for the optimal action-value function $Q^*(s, a)$. This optimization is almost always *off-policy*, as each update can use data collected at any point during training. The corresponding policy is obtained via the connection between Q^* and π^* through $a(s) = argmax Q_\theta(s, a)$. The optimal action-value function provides the expected return when starting in state s, taking an arbitrary action a, and then forever after acting according to the optimal policy:

$$Q^*(s, a) = \max_\pi \underset{\tau\sim\pi}{E}[R(\tau)\,|s_0 = s, a_0 = a\,] \tag{3}$$

2.2.2 Existing applications. The application of RL algorithms has already proven to be highly effective in achieving AI advancements in virtual game domains, such as Atari [19], Go [27][28], StarCraft [37], and Dota [3].

Within the autonomous racing space, several papers have been published about applying computational intelligence techniques to achieve autonomous driving. To create an autonomous driving agent, there are three main tasks involved, namely Recognition, Prediction, and Planning. In one of the earliest of these papers by Pyeatt and Howe [22], adaptable agents were designed for highly dynamic environments, testing tuning, decomposition, and coordination of the low level behaviours. Togelius and Lucas [34][35][36] pioneered further research by conducting several studies where neural networks were evolved to drive autonomously, and simulated rangefinder sensors were used as primary inputs.

Research within the context of overtaking in autonomous racing has been conducted by Y. Song et al. using RL in Gran Turismo Sport [30]. Additionally, investigations by P. Wurman et al. [39] and F. Fuchs et al. [10] explore the application of deep RL to optimize track completion speed. These studies utilize a SAC algorithm implementation. Numerous studies [23][14][18][38] have been conducted on the utilization of Deep Deterministic Policy Gradients (DDPG) with

TORCS (The Open Racing Car Simulator) as the designated environment. These studies offer valuable insights into observation spaces and effective reward shaping strategies. An important observation is that a predominant focus on the DDPG algorithm is evident in these studies, leaving the exploration of the SAC algorithm within this context relatively unexplored. This is another gap this study aims to address.

2.2.3 Soft-Actor-Critic. The algorithm used in this research is Soft-Actor-Critic (SAC), introduced by Haarnoja et al. in 2018 [12]. Actor-critic algorithms are model-free and combine policy-based and value-based methods for more effective learning. The actor represents the policy, defining the mapping from states to actions. It is responsible for deciding which actions to take in a given state. The critic evaluates the actions chosen by the actor by estimating the expected cumulative reward, representing the value function from value-based learning. The actor subsequently uses feedback from the critic to improve its policy, and the critic is updated based on the observed rewards and the actor's chosen actions.

The objective function of SAC is to maximize the expected sum of rewards, including an entropy term:

$$\mathcal{J}(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{\infty}\gamma^t\left(r(s_t,a_t) + \alpha\mathcal{H}(\pi_\theta(\cdot|s_t))\right)\right]$$

Here, π_θ represents the policy, τ is a trajectory, γ is the discount factor, $r(s_t, a_t)$ is the immediate reward, and $\mathcal{H}(\pi_\theta(\cdot|s_t))$ is the entropy of the policy.

3 METHODOLOGY

In this section, the methodology adopted for this research is outlined. First, the data collection and preparation process is explained. Second, an in-depth breakdown of the policy network details the data processing through an overview of inputs, outputs, and configuration of the SAC algorithm. Additionally, the reasoning behind the use of the SAC algorithm for this research is explained. Third, reward shaping is covered, presenting various reward functions that determine how the effective policy is learnt. Last, the experiment setup used to acquire results is presented. This section aims to lay the groundwork for the presentation of results.

3.1 Data collection

The data collected for the experiments consists of quantitative data produced by the AC environment. This data collection is achieved through communication with AC's internal API. The data is consequently prepared into a standardized dictionary format in Python, after which it is ready to be processed as observation data. Technical details of the data collection are provided in Section 4: System Design.

3.2 Policy network

In the SAC algorithm, the actor learns a policy π_θ through its policy network. This network takes the collected data observations from the environment as input and applies data processing to produce corresponding actions, instructing the virtual controller on how to interact with the environment.

The SAC algorithm was chosen as method for the data processing for a number of reasons:

- **Continuous Action Spaces:** SAC is suitable for problems with continuous action spaces, making it a fitting choice for the autonomous racing problem where the action space involves a range of steering angles, throttle, and brake values.
- **Entropy Regularization:** The inclusion of entropy regularization encourages exploration in the learning process. This is particularly beneficial when the agent needs to navigate complex and diverse environments, such as that of AC
- **Sample Efficiency:** SAC tends to be sample-efficient compared to other RL algorithms. The exploration from the entropy regularization and (self-tuning) temperature parameter encourages the agent to explore different actions and states, leading to more efficient learning. Furthermore, having two Q-functions helps in stabilizing the learning process and mitigating overestimation biases, which can lead to improved sample efficiency. This is beneficial as sim racing games often involve a large state and action space, and collecting samples can be computationally expensive.
- **Robustness to Hyperparameters:** SAC is relatively robust to hyperparameter choices, making it more forgiving during the tuning phase.

3.2.1 Observation Space. The observation space is described in Table 1. We denote the observation space vector as $o_t = [\rho_t, v_t, \eta_t, i_t, c_t, \rho_{t-1}, \phi_t, \theta_t, d_{c_t}]$. These observations are used as input to the neural networks. ρ_t represents a value between $[0, 1]$, corresponding to how far the car has come along the track spline, effectively indicating its progress to passing the finish line. This value does not decrease if the car moves back in the wrong direction, but instead remains at the furthest point reached in the lap. v_t represents the speed of the car at a given time in kilometers per hour. The car's coordinates in the world at a given time are represented by the vector $\eta_t = [\eta_{x_t}, \eta_{y_t}, \eta_{z_t}]$. There is a binary flag i_t, which is set to 1 when a lap should be considered invalid. As the native AC API flag for lap invalidation malfunctions, i_t is set to 1 once more than 2 tires get off the track. The lap count c_t is provided to correctly determine whether a lap has been completed. In addition to the track progress factor ρ_t, the progress value of the preceding episode timestep is represented as ρ_{t-1}, allowing for computation of the progress increment between episodes $\Delta_\rho = \rho_t - \rho_{t-1}$. The velocity of the car is represented as $\phi_t = [\phi_{x_t}, \phi_{y_t}, \phi_{z_t}]$. Finally, the heading error θ_t is defined as the angle between the car and track's center axis in radians, while the off-center error d_{c_t} is defined as the distance between the car and the track's center axis. These errors are visualized in Figure 2.

3.2.2 Action Space. The action space is described in Table 2. We denote the action space vector as $a_t = [\omega_t, \delta_t]$. Through the use of a virtual controller, AC can receive signals for throttle, brake, and steering. Therefore, these are the three independent continuous actions in the action space. Because the throttle and brake are rarely engaged at the same time, and for the sake of simplicity, the agent has one combined continuous action dimension for throttle and brake $\omega_t = [-1, 1]$. Here, $\omega_t = -1$ denotes full braking, $\omega_t = 1$ denotes full throttle, and $\omega_t = 0$ denotes neither throttle nor braking.

Table 1: The observation space ($o_t \in \mathbb{R}^{11} \times \{0,1\}^2$)

Symbol	Description	Space
ρ_t	Track progress factor	$\mathbb{R}$
v_t	Car speed (km/h)	$\mathbb{R}$
η_t	World location coordinates	$\mathbb{R}^3$
i_t	Binary flag for lap invalidation	$\{0,1\}$
c_t	Lap count	$\{0,1\}$
ρ_{t-1}	Previous track progress	$\mathbb{R}$
ϕ_t	Car velocity vector	$\mathbb{R}^3$
θ_t	Heading error	$\mathbb{R}$
d_{c_t}	Off-Center error	$\mathbb{R}$

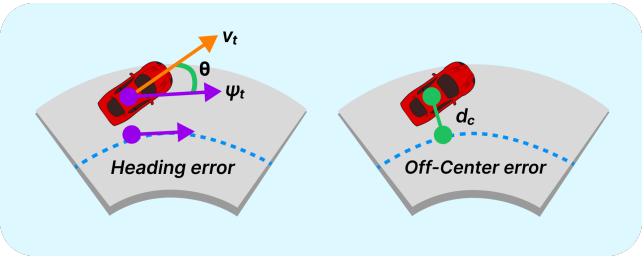


Figure 2: Heading error θ and Off-Center error d_c

The steering angle is denoted by $\delta_t = [-1, 1]$. A value of $\delta_t = -1$ refers to a full left turn and a value of $\delta_t = 1$ refers to a full right turn. The effective steering angle in degrees depends on the steering sensitivity settings and car being used. The Ferrari 458 GT2, for instance, has a range of $[-270, 270]$ degrees when using the settings described in the experiment setup. It is important to note that, for the sake of simplicity, gear switching is handled automatically.

Table 2: The action space ($a_t \in \mathbb{R}^2$)

Symbol	Description	Space
ω_t	Throttle-brake signal	$\mathbb{R}$
δ_t	Steering angle factor	$\mathbb{R}$

3.2.3 Hyperparameters. The effectiveness of the SAC algorithm is influenced by several hyperparameters that govern its learning dynamics. Considering common values (mostly default values proposed by the original paper) and the context of autonomous driving, the following hyperparameters were selected:

- **Learning Rate (lr):** 10^{-3}. This is the rate at which the model adapts its policy based on the feedback received.
- **Discount Factor (γ):** *0.99.* This parameter determines the importance of future rewards in the learning process. A higher value emphasizes long-term rewards, while a lower value focuses more on immediate rewards.
- **Entropy Coefficient (α_{entropy}):** *0.02.* This coefficient regulates the importance of the entropy term in the objective function, balancing between exploration and exploitation.
- **Polyak Coefficient (ρ):** *0.995.* The interpolation factor in polyak averaging for target networks. This is close to 1 to slow down the rate at which the target networks are updated, creating a more stable and less volatile estimate of the value function.
- **Batch Size:** *100.* The number of experiences sampled from the replay buffer in each iteration.
- **Replay Buffer Size:** 10^6. The capacity of the replay buffer, which stores past experiences for learning.
- **Initial Random Steps:** 10^4. The number of steps where uniform-random action selection is used to help exploration, before running the real policy.
- **Update After:** 10^4. The number of environment interactions to collect before starting gradient descent updates.
- **Update Every:** *50.* The number of environment interactions that should elapse between gradient descent updates.

These hyperparameters were fine-tuned through iterative experimentation to optimize the training process and results.

3.3 Reward Shaping

In formulating the reward function, several variables come into play. The primary objective of the reward function is to guide the learning agent along the correct trajectory towards the finish line, ensuring it adheres to the track boundaries, with the goal of solving the lap-completion problem. To explore various strategies, five distinct reward functions were designed and iteratively refined throughout the research. The subsequent reward functions were explored:

(1) **Speed:** A reward is given for the normalized speed:

$$R_1 = v_t / v_{max} \tag{4}$$

(2) **Speed, Progress:** A reward is given for the normalized speed and the track progress factor:

$$R_2 = v_t / v_{max} + \rho_t \tag{5}$$

(3) **ΔProgress:** A reward is assigned based on the difference between the current observed track progress factor and the track progress factor observed in the previous step:

$$R_3 = \Delta\rho_t = \rho_t - \rho_{t-1} \tag{6}$$

(4) **Speed, ΔProgress:**

$$R_4 = v_t / v_{max} + \Delta\rho_t \tag{7}$$

(5) **Speed, Track Errors:** This reward function places emphasis on track-related information rather than progression. Drawing inspiration from the dissertation by B. Evans [7], it penalizes based on two track-related errors: the heading error θ and the off-center error d_c, demonstrated in Figure 2. Four functions were evaluated using these errors, which also take inspiration from existing studies [38][11]:

$$R_{5_1} = V_t(cos(\theta) - \alpha sin(\theta) - \beta|d_c|) \tag{8}$$

$$R_{5_2} = V_t + (cos(\theta) - \alpha sin(\theta) - \beta|d_c|) \tag{9}$$

$$R_{5_3} = V_t + V_t(cos(\theta) - \alpha sin(\theta) - \beta|d_c|) \tag{10}$$

$$R_{5_4} = \frac{v_t}{v_{max}} cos(\theta) - d_c \tag{11}$$

All reward functions were tested with –and without– fixed penalties for the car going off-track or coming to a halt. Furthermore, an additional bonus was included in the final reward if the car successfully completed a lap. The signal indicating the end of an episode was determined through either:

- **Termination:** Occurred when a lap was finished, the progress goal was reached (allowing for checkpoints), or the lap was invalidated due to the car going off-track.
- **Truncation:** Applied if the episode exceeded a specified number of steps before reaching the end goal of completing a lap.

It is crucial to highlight that multiple experiments were undertaken, systematically examining the consequences of both including and excluding the reward and termination signals mentioned above.

3.4 Experiment Setup

For the experiments, an environment setup was carefully configured to ensure consistency and reproducibility. The following settings are employed in the experiment setup:

- **Mode:** Practice Mode is used and "Silverstone 1967" is selected as the track as it has a relatively simple layout. The starting point is set to "Starting Line" to ensure the car starts right in front of the starting/finish line.
- **Vehicle:** The "Ferrari 458 GT2" is used as the designated vehicle. The default tire options are employed to maintain a standard baseline for the experiments.
- **Opponents:** To focus solely on the autonomous driving agent's performance, the number of AI opponents is set to 0.
- **Conditions:** To simplify the problem space, the conditions are set to ideal. Penalties are enabled to enable off-track detection.
- **Framerate:** The framerate is intentionally limited to 30 FPS to limit the amount of steps the agent has to take per second.
- **Controls:** Gamepad is selected as the control input device. Additionally, specific settings are adjusted for the gamepad to ensure responsiveness and accuracy. These settings include setting speed sensitivity to 0, steering speed to 100%, steering gamma to 100%, and steering filter to 0%.
- **Automatic Gear Switching:** Automatic gear switching is enabled to simplify the control scheme and allow the autonomous agent to focus on steering and acceleration without manual gear management.

Each experiment used a computing environment with an NVIDIA GeForce RTX 3070. Moreover, the NVIDIA CUDA toolkit was installed on each machine to allow for effective training on the GPU. To initiate an experiment, the system is executed, and subsequently the training process is started by clicking the "Start Training" button within AC. This streamlined approach ensures a controlled and standardized environment for evaluating the autonomous driving agent's performance.

4 SYSTEM DESIGN

A core objective of this research is to establish a robust technical framework for utilizing AC as an experimentation environment for training an RL model. Based on the methodology needs, this section presents a system design to achieve the aforementioned goal with focus on reproducibility and shaping a solid path for future research. Three different methods were considered for the data collection, of which the third method was ultimately chosen for the final implementation. The first and second method were merely prototyped and consequently discarded for their lower reliability and/or higher latency in comparison to the third method.

(I) In-app: The initial approach involves running all code within an AC application (app). The app reads data from the API and uses this as input to the SAC model. All output handling is handled within AC as well. This approach was discarded due to compatibility issues, as AC apps run on Python 3.3.5, which is insufficient for modern machine learning libraries like PyTorch.

(II) Screengrab The second method incorporates an AC app and a separate standalone app running on the same machine. Here, the AC app displays real-time API observations on a window in the AC game environment. The standalone app continuously grabs screenshots of the screen, which it analyzes through an Optical Character Recognition (OCR) engine like Google's Tesseract [29]. The observation data from the OCR is used as input to the RL model, running on the standalone application. This method was deemed suboptimal, as several tests exhibited an average latency of 400-600ms. This high of a latency is unsuitable for a real-time RL algorithm where we want to execute, analyse, and train multiple steps per second.

(III) Socket: The final method maintains the structure of the second method: an AC app and a separate standalone app, both running on the same machine. However, instead of retrieving data through OCR, data is communicated via a local socket. The AC app listens for data observation requests from the socket, reads API data, and sends it back over the socket. The standalone application uses this observation data to train the SAC model. After several tests, this method was deemed to be most suitable due to its low latency (0-1ms) and reliability of local socket communication.

Upon selecting the socket method, the system illustrated in Figure 3 was developed. This system comprises two separate components running on the same machine: an Assetto Corsa Python application and a standalone Python application, connected through local socket communication and input devices. The implementation details and technical architecture of both components are explained in the following subsections. The code for the full system is publicly available on GitHub [6].

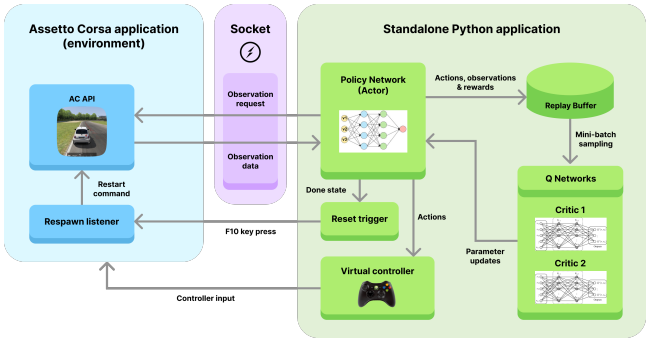


Figure 3: System overview

4.1 Assetto Corsa Application

The AC application performs two critical functions: communicating observation data from the API and facilitating the reset of the car to its initial state. For the first task, the app continuously listens for a "next_state" message in bytes. Upon reception of this message, it retrieves observation data in the observation space from its API, which it will return over the established local socket connection. For the second task, a dedicated thread continuously listens for a keypress event, specifically from the "F10" key. Upon detection of this keypress, the application triggers an internal command within AC to initiate a reset to the start session screen. Additionally, it issues a command to start the session. AC will move the car based on controller inputs from the virtual controller, managed by the standalone application running on the same machine.

4.2 Standalone Application

The standalone application is responsible for the RL aspect of the system. It establishes a connection to the local socket, and commences the training loop with the specified hyperparameters. A high-level pseudocode of the training loop is given in algorithm 1. Following the authors of the SAC paper [12], the SAC agent is configured with a squashed Gaussian Multilayer Perceptron (MLP) for the actor network and two MLP Q-networks for the critics. The SAC Network Architecture, depicted in Figure 4, showcases these networks, each comprising two hidden layers of 256 neurons. ReLU activations are applied to the hidden layers and Gaussian distributions on actions, with TanH activation on the output layer. In addition to the networks, the standalone application implements a "First In, First Out" (FIFO) replay buffer. This buffer stores actions, observations, and rewards from the policy network (actor) and facilitates mini-batch sampling by the Q-networks (critics) during the learning process. The actions generated by the policy network are transmitted to a virtual Xbox 360 controller, which in turn maps throttle to the right trigger, brake to the left trigger, and steering to the left joystick. Upon the completion of an episode, the policy network triggers a reset by simulating a virtual F10 keypress, effectively signaling the Assetto Corsa application to reset for the next episode.

Algorithm 1: SAC Training Loop

```
for each episode do
    observation = reset_environment();
    while not step_done do
        if total steps < initial random action steps then
            action ← random_action();
        else
            action ← sample_action_space();
        observation_, reward, step_done =
          step_environment(action);
        store_in_replaybuffer(observation, action, reward,
          observation_, done);
        observation ← observation_;
        if time to learn then
            batch ← sample_replaybuffer();
            update_networks(batch);
```

5 RESULTS

In this section, the results from training the SAC agent with the various reward functions will be illustrated. A full overview of the average distance, highest distance, average speed, and standard deviation of distance reached for every reward function can be found in Table 3. Moreover, the results for the distance reached (in percentage) are depicted in Figure 5, and the results for the average speed (in km/h) are depicted in Figure 6. The convergence curves for the reward values are depicted in Figure 7.

As for reward function R_5, a multitude of experiments was conducted, which led to the following observations:

- R_{5_1}**:** The agent learns to just stand still, as this minimizes the negative reward by lowering the speed factor.
- R_{5_2}**:** The agent now understands that actually making speed compensates the negative penalties. It also tries to stay on the center of the road.
- R_{5_3}**:** The agent no longer learns to stand still, but it goes off-road to end the run quickly and minimize penalties. After this experiment, the end signal for going off-road was omitted.
- R_{5_4}**:** The agent, again, learns to just stand still, as this minimizes the negative reward by lowering the speed factor.

The reward function R_{5_2} was chosen for the depiction of the final results in Figure 5 and Figure 6, as its results were the most valuable.

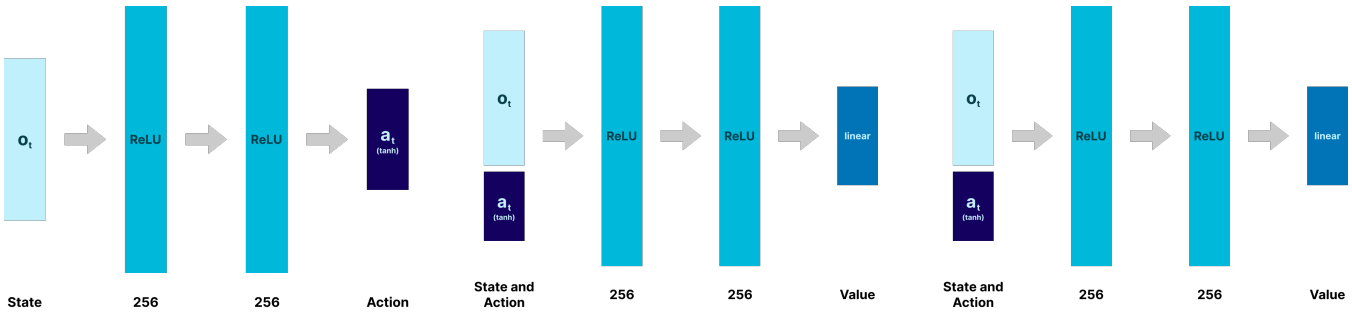
When looking at the results, we find that the car was not able to finish a lap on the track; it has not even managed to pass the 10% progress mark. Nevertheless, the depicted results are still valuable. It is clear that considering speed for the reward function (R_1) gives the highest average speed, as expected. It, however, also came with a relatively high average distance and the highest total distance. This can be explained by the fact that the track starts out straight, so the agent just gets quite far because of the track's design, not because the agent understand the track particularly well.

Table 3: Results per reward function R

R	Avg Dist (%)	Dist High (%)	Avg v (km/h)	Std. Dist
R_1	2.578	9.841	77.825	1.449
R_2	1.163	3.961	22.894	0.463
R_3	2.132	7.422	74.410	1.386
R_4	2.309	7.367	19.288	1.290
R_{5_2}	2.862	7.682	17.158	2.343

6 CONCLUSION

A thorough research has been conducted on the RL space with an explanation of why the SAC algorithm is suitable for autonomous sim racing. In addition to this, a technical framework has successfully been developed for utilizing AC as a real-time RL environment. Now that we have results, a conclusion can be made on the most suitable reward function.



(a) SAC π_ϕ Policy Network (Actor) (b) SAC Q_{θ_1} Network (Critic 1) (c) SAC Q_{θ_2} Network (Critic 2)

Figure 4: SAC Network Architecture

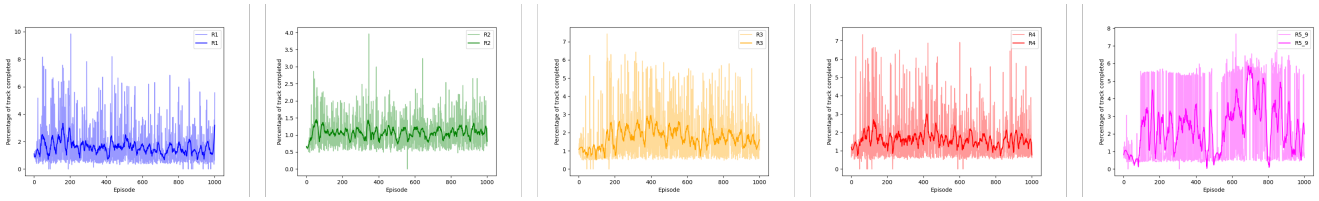


Figure 5: Distance reached (%) over episodes

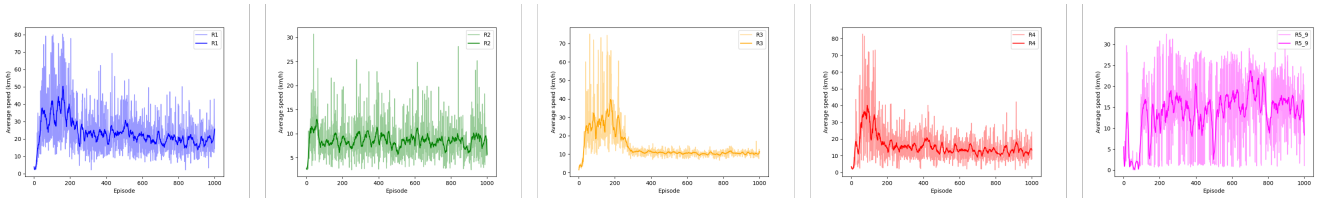


Figure 6: Average speed (km/h) over episodes

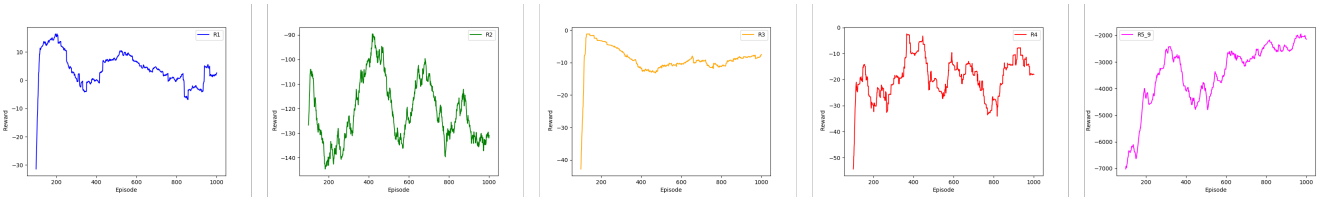


Figure 7: Convergence curves for reward over episodes

Looking at speed and progress (R_2) gave the worst results, so this should not be used for future research. Ultimately, R_5 seems to show the most promise with a growing convergence curve for reward. This should be used for future research, as more training time could bring better results as the reward curve continues to grow.

6.1 Future Directions

The principal contribution of this research is a solid framework upon which future research can build. There are three key avenues that should be explored in future research. The first avenue for future research should be the perfection of the reward function for solving the lap-completion problem, as none of the reward functions presented were able to facilitate that yet. The next avenue considers adding the lap time to the observation space, which can then be analyzed to minimize the minimum-time problem. This would make for a full solution of the autonomous racing problem. The last suggested avenue for future research is generalization. Only one singular track was used in this research, making the agent vulnerable to overfitting and solely learning the specifics of the particular track. By running experiments with a multitude of tracks, this can be mitigated. Furthermore, generalizations may be made over the sim racing domain as a whole, expanding the research outside of the AC environment.

REFERENCES

[1] Studio 397. 2013. *rFactor 2.* https://www.studio-397.com/rfactor2/ Accessed: January 14, 2024.

[2] assettocorsamods. 2014. *Assetto Corsa Shared Memory Reference.* https://assettocorsamods.net/threads/doc-shared-memory-reference.58/ Accessed: January 14, 2024.

[3] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Christopher Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub W. Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. 2019. Dota 2 with Large Scale Deep Reinforcement Learning. *ArXiv* abs/1912.06680 (2019). https://api.semanticscholar.org/CorpusID:209376771

[4] International Olympic Committee. 2021. *IOC makes landmark move into virtual sports by announcing first-ever Olympic Virtual Series.* https://olympics.com/ioc/news/international-olympic-committee-makes-landmark-move-into-virtual-sports-by-announcing-first-ever-olympic-virtual-series Accessed: November 28, 2023.

[5] Valve Corporation. 2003. *Steam.* https://store.steampowered.com/ Accessed: January 14, 2024.

[6] Jurre de Ruiter. 2024. ACRL: Assetto Corsa Reinforcement Learning. https://github.com/Jurredr/ACRL. Accessed: January 21, 2024.

[7] Benjamin David Evans. 2023. *Accelerating Deep Reinforcement Learning for Autonomous Racing.* Ph. D. Dissertation. Stellenbosch University. https://scholar.sun.ac.za/bitstream/handle/10019.1/127272/evans_deep_2023.pdf?sequence=1

[8] Wael Farag and Zakaria Saleh. 2018. Behavior Cloning for Autonomous Driving using Convolutional Neural Networks. In *2018 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT).* 1–7. https://doi.org/10.1109/3ICT.2018.8855753

[9] Python Software Foundation. 2014. *Python 3.3.5.* https://www.python.org/downloads/release/python-335/ Accessed: January 14, 2024.

[10] Florian Fuchs, Yunlong Song, Elia Kaufmann, Davide Scaramuzza, and Peter Dürr. 2020. Super-Human Performance in Gran Turismo Sport Using Deep Reinforcement Learning. *IEEE Robotics and Automation Letters* 6 (2020), 4257–4264. https://api.semanticscholar.org/CorpusID:221151057

[11] Kıvanç Güçkıran and Bülent Bolat. 2019. Autonomous Car Racing in Simulation Environment Using Deep Reinforcement Learning. In *2019 Innovations in Intelligent Systems and Applications Conference (ASYU).* 1–6. https://doi.org/10.1109/ASYU48272.2019.8946332

[12] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. arXiv:1801.01290 [cs.LG]

[13] Johanna Hamilton. 2019. The Rise of Esports. *ITNOW* 61 (09 2019), 28–29. https://doi.org/10.1093/itnow/bwz068

[14] Zhiqing Huang, Ji Zhang, Rui Tian, and Yanxin Zhang. 2019. End-to-End Autonomous Driving Decision Based on Deep Reinforcement Learning. In *2019 5th International Conference on Control, Automation and Robotics (ICCAR).* 658–662. https://doi.org/10.1109/ICCAR.2019.8813431

[15] iRacing.com Motorsport Simulations. 2008. *iRacing.* https://www.iracing.com/ Accessed: January 14, 2024.

[16] Daniel Kane and Brandon Spradley. 2017. Recognizing ESports as a Sport. *The Sport Journal* 19 (05 2017).

[17] Alexander Liniger, Alexander Domahidi, and Manfred Morari. 2014. Optimization-based autonomous racing of 1:43 scale RC cars. *Optimal Control Applications and Methods* 36, 5 (July 2014), 628–647. https://doi.org/10.1002/oca.2123

[18] Kai Liu, Qin Wan, and Yanjie Li. 2018. A Deep Reinforcement Learning Algorithm with Expert Demonstrations and Supervised Loss and its application in Autonomous Driving. In *2018 37th Chinese Control Conference (CCC).* 2944–2949. https://doi.org/10.23919/ChiCC.2018.8482790

[19] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing Atari with Deep Reinforcement Learning. https://doi.org/10.48550/arXiv.1312.5602 arXiv:1312.5602 [cs.LG]

[20] OpenAI. 2018. *Spinning Up in Deep Learning.* https://spinningup.openai.com/en/latest/spinningup/rl_intro.html Accessed: January 21, 2024.

[21] D. Perel. 2021. *What is Sim Racing and How Do I Get Started?* https://coachdaveacademy.com/tutorials/what-is-sim-racing/ Accessed: November 28, 2023.

[22] Larry D. Pyeatt and Adele E. Howe. 1998. Learning to Race: Experiments with a Simulated Race Car. In *The Florida AI Research Society.* https://api.semanticscholar.org/CorpusID:15433604

[23] Adrian Remonda, Sarah Krebs, Eduardo Veas, Granit Luzhnica, and Roman Kern. 2022. Formula RL: Deep Reinforcement Learning for Autonomous Racing using Telemetry Data. arXiv:2104.11106 [cs.AI]

[24] Andrew G. Barto Richard S. Sutton. 2018. *Reinforcement Learning: An Introduction (second edition).* The MIT Press.

[25] Giovanni Romagnoli. 2017. *Assetto Corsa API documentation.* https://www.assettocorsa.net/forum/index.php?threads/python-doc-update-25-05-2017.517/ Accessed: January 14, 2024.

[26] Ugo Rosolia and Francesco Borrelli. 2019. Learning How to Autonomously Race a Car: a Predictive Control Approach. arXiv:1901.08184 [cs.SY]

[27] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, L. Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy P. Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529 (2016), 484–489. https://api.semanticscholar.org/CorpusID:515925

[28] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy P. Lillicrap, Fan Hui, L. Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. 2017. Mastering the game of Go without human knowledge. *Nature* 550 (2017), 354–359. https://api.semanticscholar.org/CorpusID:205261034

[29] R. Smith. 2007. An Overview of the Tesseract OCR Engine. In *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, Vol. 2. 629–633. https://doi.org/10.1109/ICDAR.2007.4376991

[30] Yunlong Song, HaoChih Lin, Elia Kaufmann, Peter A. Duerr, and Davide Scaramuzza. 2021. Autonomous Overtaking in Gran Turismo Sport Using Curriculum Reinforcement Learning. *2021 IEEE International Conference on Robotics and Automation (ICRA)* (2021), 9403–9409. https://api.semanticscholar.org/CorpusID:232404855

[31] KUNOS Simulazioni Srl. 2018. *Assetto Corsa Competizione.* https://assettocorsa.gg/assetto-corsa-competizione/ Accessed: January 14, 2024.

[32] KUNOS Simulazioni Srl. 2019. *Assetto Corsa.* https://assettocorsa.gg/ Accessed: November 28, 2023.

[33] Reiza Studios. 2020. *Automobilista 2.* https://www.game-automobilista2.com/ Accessed: January 14, 2024.

[34] Julian Togelius and Simon M. M. Lucas. 2005. Evolving controllers for simulated car racing. *2005 IEEE Congress on Evolutionary Computation* 2 (2005), 1906–1913 Vol. 2. https://api.semanticscholar.org/CorpusID:1073693

[35] Julian Togelius and Simon M. M. Lucas. 2006. Arms Races and Car Races. In *Parallel Problem Solving from Nature.* https://api.semanticscholar.org/CorpusID:14233196

[36] Julian Togelius and Simon M. M. Lucas. 2006. Evolving robust and specialized car racing skills. *2006 IEEE International Conference on Evolutionary Computation* (2006), 1187–1194. https://api.semanticscholar.org/CorpusID:7659584

[37] Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, L. Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander Sasha Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom Le Paine, Caglar Gulcehre, Ziyun Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy P. Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. 2019. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature* 575 (2019), 350 – 354. https://api.semanticscholar.org/CorpusID:204972004

[38] Sen Wang, Daoyuan Jia, and Xinshuo Weng. 2019. Deep Reinforcement Learning for Autonomous Driving. arXiv:1811.11329 [cs.CV]

[39] Peter R. Wurman, Samuel Barrett, Kenta Kawamoto, James MacGlashan, Kaushik Subramanian, Thomas J. Walsh, Roberto Capobianco, Alisa Devlic, Franziska Eckert, Florian Fuchs, Leilani Gilpin, Piyush Khandelwal, Varun Kompella, HaoChih Lin, Patrick MacAlpine, Declan Oller, Takuma Seno, Craig Sherstan, Michael D. Thomure, Houmehr Aghabozorgi, Leon Barrett, Rory Douglas, Dion Whitehead, Peter Dürr, Peter Stone, Michael Spranger, and Hiroaki Kitano. 2022. Outracing champion Gran Turismo drivers with deep reinforcement learning. *Nature* 602 (2022), 223 – 228. https://api.semanticscholar.org/CorpusID:246701687