Developing Resilient Autonomous Vehicles with Deep Reinforcement Learning

Edmund Summers

MSc Computer Science
The University of Bath
2022

Developing Resilient Autonomous Vehicles with Deep Reinforcement Learning

Submitted by: Edmund Summers

Abstract

Modern autonomous vehicles (AVs) can achieve point-to-point automation in well-known locations and under good weather conditions. Lifting these constraints is the most important obstacle between the current state of the art and full automation.

Reinforcement learning methods - particularly those that make use of deep neural networks - are a promising area of research for continuous control tasks like autonomous driving. This project investigates the use of the deep deterministic policy gradient family of deep reinforcement learning algorithms for learning driving policies that are resilient to unfamiliar road layouts or poor surface conditions, using a motorsport simulation as a testbed.

A reinforcement learning agent using twin delayed deep deterministic policy gradient was developed for the purpose of learning driving policies in the motorsport simulation. Instances of this agent trained on a variety of tracks in the simulation were then evaluated against a different selection of tracks, to test the degree to which their policies generalise to unseen locations.

The results presented demonstrate that given a training track with sufficient complexity, the agent can learn a driving policy that will generalise to confident driving on unseen tracks. This effect is particularly strong when the training track has a dirt surface: an agent trained on one dirt track was able to safely navigate almost all unseen tracks.

This outcome indicates that modern deep reinforcement learning algorithms have the potential to be used to bridge the gap between the current best level of vehicle automation and full autonomous driving, as part of a larger control system. Avenues of further research are suggested, including technical enhancements to the training process, investigating the ability of such an agent to cope with other road users, and directly examining the suitability of deep reinforcement learning for handling sensor degradation due to weather conditions.

Contents

List of Figures

List of Tables

Acknowledgements

I would like to express my gratitude to my supervisor, Dr Hongping Cai, for her guidance during this project. I would also like to thank my partner Amber for her patience and support.

Chapter 1

Introduction

1.1 Background

1.1.1 Autonomous Vehicles

Autonomous vehicles (AVs), sometimes known as ‘self-driving’, are those that can operate without direct human control. This technology has numerous and major applications, from replacing human-driven vehicles for private transport to autonomous heavy goods transport. As such, it is a topic of intense research in the academic and commercial worlds.

Designing AVs for use on public roads is a difficult problem, due to the complex sensory environment and the presence of other - possibly unpredictable - road users. The Society of Automotive Engineers (SAE) has defined a six-point scale for AV capability (SAE, 2021b). This scale begins with level zero - representing a vehicle with no automation and at most minimal driver aids - and ends with level five, representing a completely autonomous vehicle.

The best AVs today achieve level four on the SAE scale. The Waymo project - formerly the Google self driving car project - provides autonomous taxis confined to the city of Phoenix in Arizona, USA (Schwall et al., 2020). The remaining gap between vehicles like this and full automation - level five - concerns the ability to handle novel situations, such as driving in new locations, and the ability to operate in all weather conditions. Bad weather conditions challenge AV systems by degrading their sensory ability, and by increasing the risk of the vehicle experiencing a loss of grip, e.g. by sliding on ice (SAE, 2021b).

1.1.2 Deep Reinforcement Learning

In the field of artificial intelligence (AI), problems that can be modelled as a series of decisions under some set of constraints - that is, as a Markov decision process (MDP) - are difficult to solve using supervised learning techniques. This is because for any problem with a nontrivial state-action space, gathering sufficient labelled data to supervise every possible decision is at best impractical, and at worst impossible (Russell and Norvig, 2016, p.830).

This class of problem requires a different model of learning to solve, in which the agent can take action in the environment and receive a reward or punishment signal as a result. Reinforcement learning (RL) is a discipline of machine learning (ML) that models this process. This approach to learning is closely analogous to learning exhibited by natural intelligence, such as by infants learning basic motor skills (Sutton and Barto, 2018). RL techniques have widespread applications, since for most RL techniques the mechanism of learning is problem-agnostic. Despite this, research into the use of RL for AVs has only begun in earnest recently (Shalev-Shwartz, Shammah and Shashua, 2016).

Deep learning (DL) refers to a class of ML approaches that use deep - that is, multilayered - neural networks. Deep networks can achieve stronger representative power than shallow ones, and can form more complex representations from simpler subnetworks (Goodfellow, Bengio and Courville, 2016, p.5). A combination of DL with RL - deep reinforcement learning (DRL) (Mnih et al., 2015) - has been shown in recent years to be a strong general problem solver, most famously with DeepMind's AlphaGo program and its successor AlphaZero (Wang et al., 2016).

The most difficult remaining challenges in AV design relate to situations that are novel or unpredictable. It is proposed that experiential learning like DRL, which is more similar to the way human drivers learn than to systems designed from first principles to use object classification, trajectory prediction etc., proves superior in preparing agents for such situations. Understanding means for providing AVs with this sort of general-purpose, failure-tolerant control is an important obstacle to full automation.

1.1.3 Motorsport

Motorsport, or racing driving, offers an interesting opportunity for studying the suitability of different approaches for vehicle control alongside existing work on driving at lower speeds on public roads. The nature of racing driving is different, in that the car must be controlled closer to its performance limits. This introduces a strong sequence element to the driver's decisions: if, for example, it has not braked by the end of a long straight section, it will be too late to execute the upcoming turn without leaving the track.

If an AV encounters an unexpected patch of ice on the road, or runs over debris such as wet leaves, it may begin to slide, and need to control the vehicle in this more unstable regime. This effect also occurs in racing when the vehicle cannot maintain sufficient grip for a given corner at a given speed. A fast driver - autonomous or otherwise - must approach this limit as closely as possible to extract the best performance from the vehicle. This is also the case much of the time in races such as rally events, which include stages on unpaved roads.

Using motorsport also offers a clear reward function: sustained high speeds and short lap times should be rewarded, while time off-track or at low speed should be punished. Transfer of technology from motorsport to the public road is well-established: technologies like energy recovery systems and semi-automatic transmissions are used in many road cars but were largely developed in Formula One racing.

1.2 Research Objectives

This project aims to explore the use of deep reinforcement learning for autonomous vehicle control. Specifically, it aims to determine whether DRL is a useful technique for overcoming the particular challenges of (1) unfamiliar road layouts and (2) adverse road conditions, with motorsport used as a means of exercising these capabilities.

To answer this, the project aims to develop an agent capable of the following:

(i) Navigating an unfamiliar race track at speed

(ii) Maintaining a good level of performance on tracks with a dirt surface

Chapter 2

Technology Review

2.1 Simulators

Simulation offers a way of developing understanding without expensive or dangerous experiments. In this case, use of a racing simulation to develop such a model is essential, as using a physical track and car would be beyond the resources available for the project. Open-source racing simulators such as The Open Racing Car Simulator or TORCS (Wymann et al., 2014) are valuable tools for AV research.

TORCS is designed using a modular architecture to allow native programmatic control, which has made it popular in other research on AVs (Sallab et al., 2017; Wang, Jia and Weng, 2018). Other commercial racing simulation games such as Gran Turismo Sport (Fuchs et al., 2021) and World Rally Championship 6 (Jaritz et al., 2018) have been used in research, although these have required the development of custom APIs to control algorithmically.

TORCS also supports tracks with dirt surfaces, in addition to paved circuits. It would be possible to test the ability of the agent to generalise its training experience to low-grip situations directly by testing its performance on these tracks, or to include them in the training process to improve its ability to tolerate loss of grip on paved roads.

If successful, the agent's performance could be compared against lap times set by experienced human players, as this information is available for the built-in tracks in TORCS and would provide a convenient means of validation. DRL is known for producing superhuman performance in a wide variety of games (Mnih et al., 2015; Wang et al., 2016; Fuchs et al., 2021).

Due to TORCS' popularity, the open source community has created useful tools for working with it in a research context. An important one is Gym-TORCS (Yoshida, 2016; Dossa, 2018), which extends TORCS to provide a Python API for sensory feedback and player input, corresponding to OpenAI's Gym standard (OpenAI, 2016), which is a toolkit for developing reinforcement learning algorithms against an environment-agnostic interface.

Gym-TORCS is based on vTORCS, a build of TORCS that includes a server for programmatic control developed by Cuccu (2012), and SnakeOil, a Python client for connecting to the server (Edwards, 2013). Gym-TORCS is particularly useful because the leading tools for ML, such as TensorFlow (Abadi et al., 2015) and Keras (Chollet et al., 2015), are Python libraries.

Since it is desirable to minimise the time spent in the logistics of connecting the agent to the simulator, and TORCS has the additional benefit of supporting dirt tracks, TORCS will be used for this project, with a control system based on Dossa's expanded version of Gym-TORCS (Dossa, 2018).

Chapter 3

Literature Survey

3.1 Autonomous Vehicles

3.1.1 Early History

Research has been conducted on autonomous vehicles for around a hundred years, although designs prior to the 1980s required a specialised or contrived environment, e.g. magnetic cables buried in the road for the car to follow. The first work towards building independent AVs that could be suitable for public roads was published at Carnegie Mellon University (CMU) in the late 1980s.

Researchers at CMU demonstrated the guidance of the Autonomous Land Vehicle (ALV) - a van fitted with a video camera, LIDAR system and computer-controlled steering - around a 400-metre closed test track. This was first shown to be possible with a designed algorithm (Thorpe et al., 1987), and later with an early neural network (Pomerleau, 1989). While the former approach required less computational power, meaning the ALV could be driven at a higher speed with the same computer, the latter's network was able to learn a similar level of performance to the designed algorithm in half an hour of training, versus the several months required to develop the algorithm.

Despite some problems with the neural network approach, such as overfitting to the training set and degenerating into oscillation when faced with a fork in the road, machine learning approaches became the standard for further research into AV control. The performance and scope of AV control grew steadily through the 1990s and 2000s, along with the development of new neural network architectures and greater computational power to run them.

3.1.2 State of the Art

Modern cars include a range of partial automation technologies, ranging from driver aids like traction control, anti-lock braking systems and cruise control to more advanced features like automatic lane-keeping and overtaking, e.g. Tesla Motors' Autopilot system (Dikmen and Burns, 2016). Car manufacturers are now competing to develop models with higher levels of automation: cars modified to exhibit automation at level three on the SAE scale (SAE, 2021b) - see Table 3.1 - have been demonstrated in limited tests by BMW (Aeberhard et al., 2015) and Mercedes-Benz (Ziegler et al., 2014), able

Level	Description
0	Human driver is responsible for all control. Vehicle may offer warnings or momentary assistance.
1	Human driver remains responsible, but the vehicle may support one of the controls at a time.
2	Human driver remains responsible, but the vehicle may support all of the controls at one time.
3	The vehicle is responsible under limited conditions, but the human driver must be ready to take over.
4	The vehicle is autonomous and responsible, but only in a limited set of cases e.g. a particular location.
5	The vehicle is fully autonomous.

Table 3.1: SAE J3016 standard for vehicle automation (SAE, 2021a)

to navigate particular routes autonomously under human supervision. However, before Waymo launched its fleet, the most advanced commercially-available AVs achieved level 2 on the SAE scale (Mallozzi et al., 2019).

All current AVs capable of point-to-point autonomous navigation require extensive preparation in the area in which they operate. Before Waymo began operating its level-four autonomous taxis, the system was trained and tested over millions of kilometres of driving, and the area was mapped in great detail (Schwall et al., 2020). While this is a large undertaking, it should be noted that this is a potential advantage of AVs over human drivers, since the experiences of one vehicle in the fleet can be shared with the others, and the collective sum of experience is much greater than an individual human driver would accumulate over a lifetime.

Level five automation requires the ability to undertake any journey autonomously, without supervision and under any conditions (SAE, 2018). At the time of writing, no such AV exists.

3.1.3 AV Architecture

The general design of a modern AV is a pipeline broadly leading from sensory input to motor output. While different designs have been used successfully, this pipeline is usually characterised as being composed of three stages: perception, prediction and planning (Mallozzi et al., 2019). These tasks are particularly suited to different tools, so the design of each component can be quite different.

Perception

AVs use a variety of sensors, including technologies such as video cameras, LIDAR and GPS, to perceive their surroundings. The data provided by these sensors must be processed to provide a semantic representation of the environment. On public roads, and especially in urban areas, this is a difficult task, because there will be many objects nearby, including actors like other road users and pedestrians, but also obstacles such as parked vehicles and navigational cues such as the shape of the road and road signs.

Recently, deep learning approaches such as convolutional neural networks (CNNs) have become important tools for the perception step (Mallozzi et al., 2019), as they have been recognised as strong image classifiers (Krizhevsky, Sutskever and Hinton, 2012). State-of-the-art CNN architectures such as GoogLeNet v4 have achieved better-than-human image classification performance on the ImageNet test (Szegedy et al., 2017).

While classification of an image of a single object is quite well-solved, the problem of recognising many different objects in a visual field is more difficult - even more so when there are multiple senses that require combining (Feng et al., 2021). Approaches to this problem usually use a CNN to produce feature maps over an image, then make region proposals for specific parts of it. This is computationally expensive, but there are efficient approaches e.g. Faster R-CNN, which uses a pre-trained CNN such as GoogLeNet to generate a feature map, a convolutional region proposal network, and a classifier to give a prediction for each region (Ren et al., 2017). Optimising for performance is important for real-time applications like AVs, since it allows more frequent sampling, and hence earlier detection of dangerous situations (Feng et al., 2021).

Prediction

With sensory information in hand, an AV must make predictions about events in the future. Obvious examples of such predictions would include a pedestrian stepping into the road ahead or the movements of other vehicles, but the AV must also predict its own trajectory through the environment. The time-series nature of predicting future states from past observations lends itself to the use of recurrent neural networks (RNNs).

Researchers have demonstrated the use of long short-term memory networks (LSTMs) - a kind of RNN - for several AV-related tasks. These include predicting whether other vehicles on a motorway will change lane (Patel et al., 2019), predicting the movements of other vehicles at complex junctions (Jeong, Kim and Yi, 2020) and predicting whether pedestrians will step into the road (Lorenzo et al., 2020). Just as the perception step requires the control system to combine percepts from different senses into a semantic representation of the environment, so the prediction step requires the combination of separate predictions of future events into a probable series of future states.

Planning

Once the system has interpreted its sensory data and made predictions from it, it must make decisions about what actions to take in order to reach its goal state. This happens on several levels: the AV must plot a route to its destination, plan lane changes and turns ahead of time, and do all this by managing the vehicle's controls, i.e. steering, brakes, throttle etc. Some of these plans, like the route, are mostly decided in advance and have a longer time horizon, but lower-level decisions like control outputs are more subject to changing conditions in the environment.

Robotic control systems in general have historically used classical planning approaches like the Stanford Research Institute Problem Solver (STRIPS) (Fikes and Nilsson, 1971). STRIPS has proven successful in well-modelled problem domains, but the problem of AV control is both high-dimensional and stochastic, both in inputs and predictions of the behaviour of other actors. This has led to the use of RL in general and DRL in particular, since deep networks are able to approximate high-dimensional functions.

DRL appears promising for the lower-level planning tasks in the pipeline (Kiran et al., 2021), including navigating junctions where the ego vehicle's view is occluded (Isele et al., 2018), and lane-keeping and motorway merging (Aradi, 2020). It is this last stage in the pipeline, i.e. fine-grained control output, which is the focus of this project.

3.2 Reinforcement Learning

Reinforcement learning is a deep area of research, with a multitude of approaches and a large number of algorithms for different formal problems. This section will focus on a few key algorithms that highlight particular classes of approach, to give a background for section 3.3 on deep reinforcement learning.

3.2.1 Markov Decision Process

Reinforcement learning is a family of techniques for solving problems that can be characterised as Markov decision processes. Control of an AV can be characterised as an MDP because it exhibits the Markov property, i.e. that the history of actions taken until this point has no bearing on the optimal strategy in the current state (Sutton and Barto, 2018, p.49). It can be further characterised as a finite MDP, since the space of states, actions and rewards is finite.

An intuitive understanding of an MDP is as a formalism for sequential decision making, in which the agent trades off between immediate and delayed rewards (Sutton and Barto, 2018, p.47). The means by which this tradeoff is made is the policy (π), a function that specifies which action to choose in each possible state. An optimal policy (π^*) is one whose recommended action in each state maximises future utility, i.e. represents the best strategy.

Classically, an MDP is modelled as a four-tuple,

$$(s, a, P_a, R_a)$$

Where:

- s is the state space, the set of all possible states
- a is the action space, the set of all possible actions
- P_a is the probability distribution of action outcomes, i.e.

  $$P_a(s, s') = Pr(s_{t+1} = s' | s_t = s, a_t = a)$$

  is the probability that taking action a from state s at time t leads to state s_{t+1} at time $t + 1$
- R_a is the immediate expected reward distribution, i.e. $R_a(s, s')$ is the immediate expected reward received by transitioning from state s to state s'

A deterministic policy π is therefore defined as $\pi(s) = a$. In other words, π specifies that in state s, the agent should take action a. If the chosen approach models the policy as a probability distribution, i.e. the policy is stochastic, it can be expressed as:

$$\pi(a|s) = Pr(a_t = a|s_t = s)$$

3.2.2 Value-Based Methods

The value v of a state s can be represented as a state-value function under policy π as follows:

$$v_\pi(s) = \mathbb{E}_\pi[G_t|s_t = s]$$

Where:

- $\mathbb{E}_\pi$ is the expected value
- G_t is the estimated return

The value of taking a particular action a from state s can be more usefully defined using the action-value function (Sutton and Barto, 2018, p.58):

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t|s_t = s, a_t = a]$$

When estimating the estimated return G_t, we must take into account the future value of states, usually using a discount factor γ. Otherwise, the value function would only ever take into account the current state, and disregard long-term rewards. G_t can be expressed as:

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

Since the actions leading to these future rewards are defined by our action-value function, this infinite series intuitively tends towards the discount factor multiplied by the value of taking the most valuable action in the state that will be reached by taking action a in state s:

$$G_t = r_{t+1} + \gamma q_\pi(s_{t+1}, a_{t+1})$$

And, therefore:

$$q_\pi(s,a) = \mathbb{E}_\pi[r_{t+1} + \gamma q_\pi(s_{t+1}, a_{t+1}|s_t = s, a_t = a)]$$

This is known as the Bellman equation, and is used by the following algorithms to update their value functions. This process of estimating the value of a state based on estimates of the values of states accessible from it is known as bootstrapping (Sutton and Barto, 2018, p.264). Bootstrapping greatly improves data efficiency, at the cost of the potential for errors to compound in the function's estimates.

It follows that, given an accurate action-value function, the optimal policy can easily be derived by choosing the most valuable action in each state. Value-based approaches to RL seek to learn or approximate the action-value function, in order to be able to derive the optimal policy.

There are two well-known RL algorithms for deriving a learned action-value function Q - Sarsa (Rummery and Niranjan, 1994) and Q-learning (Watkins and Dayan, 1992). Sarsa is named for the five-tuple that describes the transition from one state to the next - (s, a, r', s', a') (Sutton and Barto, 2018, p.129). The general idea of these algorithms is, in state s, to take the action a chosen by the policy π, observe the reward signal R', and use this to update a table that maps each possible state-action pair to an estimated value. Both algorithms use this table approach, but they differ in the policy used for the search.

Sarsa

Sarsa is an on-policy algorithm, meaning that the Bellman equation update to the Q-table uses the estimated return from acting according to the existing policy:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r' + \gamma Q(s', a') - Q(s, a)]$$

Where:

- α is the step size, or learning rate
- r' is the reward signal from taking action a from state a
- γ is the discount factor, used to balance short and long-term rewards

Q-Learning

Q-learning is quite similar, with the main difference being that it is off-policy, meaning that when the Q-table is updated, it is updated with the estimated return from taking the greediest possible action, i.e. the one with the largest immediate reward:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma max_{a'} Q(s', a') - Q(s, a)]$$

Off-policy algorithms in general carry the benefit of better exploration: the learner is free to behave in a manner it does not think is advantageous, in order to expand its experience and avoid local minima.

3.2.3 Policy-Based Methods

While value-based methods for RL learn a policy indirectly - by learning the action-value function and forming a policy by following its recommendations - policy-based methods aim to estimate π^* directly (Kiran et al., 2021).

A particular advantage of this approach is that value-based methods require a discrete state-action space, and in large spaces are computationally demanding at best, and infeasible at worst. Real-world control problems like AV motion planning are not only high-dimensional and hence have very large state-action spaces, but include continuous dimensions e.g. steering angle or brake pressure. Policy-based methods can solve problems with continuous state-action spaces.

Some research on the use of value-based methods in this area has found success with discretising continuous parts of the state-action space, but finding the correct level of granularity is a challenge, and doing this for multiple elements results in the 'curse of dimensionality' as the number of possible states grows in a combinatorial manner (Kiran et al., 2021).

Policy Gradient

Policy gradient (PG) methods represent the policy as a parameterised probability distribution (Silver et al., 2014):

$$\pi_\theta(a|s) = Pr[a|s;\theta]$$

Where θ is the parameter vector. The agent decides on an action stochastically using this probability distribution and receives a reward signal from the environment in the form of a performance function $J(\theta)$ (Sutton and Barto, 2018, p.321). θ can then be updated via gradient ascent:

$$\theta_{t+1} = \theta_t + \alpha \Delta J(\theta_t)$$

Trust Region Policy Optimisation

Trust region policy optimisation (TRPO) is an on-policy technique for maximising the safe rate at which the policy can be learned. State-action spaces that have areas of high curvature, i.e. those where it is easy to make a serious mistake, are dangerous for algorithms whose learning rates have been tuned for flatter areas, as they are prone to making destructively large updates to the policy. The nature of policy gradient ascent can also make it difficult for a PG algorithm to recover from such a mistake (Schulman et al., 2015).

TRPO takes steps by searching for the best decision to take within a trusted region, an area whose size is governed by the curvature of the state-action space surface in that region. This is opposed to the classical gradient ascent approach of taking a step along the current gradient and assessing the result. Its inventors found that it was able to learn a variety of locomotive tasks in a physics simulator (Schulman et al., 2015).

Proximal Policy Optimisation

Proximal policy optimisation (PPO) is a refinement of TRPO proposed by the same lead author. Like TRPO, PPO alternates between experimenting with the environment and optimising a surrogate objective function through gradient ascent (Schulman et al., 2017), rather than optimising the policy parameters directly against the environment.

PPO performs multiple epochs of gradient ascent for each policy update and penalises large policy changes through a clipping parameter, which has proven simpler than TRPO while providing better performance (Schulman et al., 2017)(Kiran et al., 2021).

3.2.4 Actor-Critic Methods

Actor-critic methods combine value-based and policy-based methods, in that an 'actor' experiments against the environment according to a learned policy, and a 'critic' evaluates the decisions made by the actor using a learned value function (Sutton and Barto, 2018, p.321).

Deterministic Policy Gradient

Recall that policies for RL algorithms can either be stochastic or deterministic. Traditional policy gradient methods (see 3.2.3) learn a stochastic policy. A limitation of this is that the policy gradient must consider both the state and action spaces, while a deterministic policy need only consider the state space. In high-dimension spaces, this makes a purely stochastic approach much more expensive. The downfall of a purely deterministic approach is a limited degree of exploration (Silver et al., 2014).

Deterministic policy gradient (DPG) is an off-policy actor-critic approach in which the actor explores using a stochastic policy, but a deterministic target policy is learned. DPG has been shown to outperform traditional stochastic policy gradient methods by several orders of magnitude, and is especially useful in high-dimension spaces (Silver et al., 2014).

3.3 Deep Reinforcement Learning

Traditional reinforcement learning algorithms can be implemented using an explicit table of parameters representing the value function or policy, provided the state-action space is relatively small. For problems with large spaces and high dimensionality, this is impractical, and the function otherwise represented with a table can be approximated with a function approximator such as a neural network. Unfortunately, achieving good results with neural networks is not trivial, as the training process can prove unstable.

As noted, deep learning techniques are a subset of machine learning approaches that combine simpler subnetworks into deeper composite networks, so as to learn more complex representations (Goodfellow, Bengio and Courville, 2016, p.5). While this boundary is fairly clear, the boundary between reinforcement learning and deep reinforcement learning is less so: we might say that the distinction is less about algorithms and more about implementations, although some approaches e.g. deep Q-network (DQN) and deep deterministic policy gradient (DDPG) are usually described explicitly as using deep networks.

The ability of reinforcement learning algorithms that make use of deep learning techniques to learn complex representations in high-dimensional environments is particularly useful, as this allows agents to learn directly from unprocessed sensory input e.g. the pixels on-screen in a video game (Lillicrap et al., 2015). This is desirable for real-world applications, in which sensory input is naturally unprocessed.

3.3.1 Approaches

Deep Q-Network

Deep Q-network is a value-based off-policy approach that makes use of a deep CNN to approximate the action-value function (Mnih et al., 2015). It is off-policy in the same sense as classical Q-learning, i.e. the value of each action is evaluated using an ϵ-greedy policy, while actions are chosen according to the implicit current policy.

DQN, unlike TRPO and PPO, improves the stability of its network updates using a replay buffer, allowing each update to be derived from a batch of previous transitions - a technique now known as experience replay (Mnih et al., 2015). The intuitive explanation of experience replay is that it permits the agent to apply lessons from its past to current observations, rather than using experience to perform one update and then discarding it. Another stabilising mechanism introduced by DQN was the use of a target network, to which the Q-network's weights are gradually transferred during training.

Stabilising the network updates is necessary because the direct use of a neural network to approximate the Q-table has proven to be unstable in many environments (Lillicrap et al., 2015). The original work showed DQN to perform well on a variety of Atari 2600 games, but it is limited to solving problems with low-dimensional and discrete action spaces, and has not proven generally successful (Schulman et al., 2017).

Asynchronous Advantage Actor-Critic

While experience replay is a powerful tool for stabilising updates to deep networks, it is computationally expensive, and requires the exclusive use of off-policy algorithms (Mnih et al., 2016). Rather than storing a history of past experiences, it is possible to explore multiple instances of the training environment at once - using multiple agents - and use their collective observations to learn a solution. Using this parallelism as a stabilising mechanism permits the use of deep networks for a wider variety of fundamental RL algorithms (Mnih et al., 2016).

Asynchronous advantage actor-critic (also called A3C) is a parallelised evolution of advantage actor-critic (A2C) that uses N-step returns, i.e. considers state-action outcomes over N steps, rather than just one. A2C is an actor-critic method in which the critic learns an advantage function - an approximation of how much better an action is than others in each state - rather than an actual value function. A3C has proven to outperform algorithms such as DQN and some policy gradient methods while using significantly fewer computational resources. However, its designers noted that for problems where interacting with the environment - i.e. gathering more data - is more expensive than updating the model, methods based on experience replay are still useful because they improve data efficiency (Mnih et al., 2016). This is true of environments like TORCS, where running the simulation itself is expensive.

Deep Deterministic Policy Gradient

Deep deterministic policy gradient is an off-policy actor-critic approach that builds on DPG and DQN (Lillicrap et al., 2015), in that it combines the use of experience replay and target networks to stabilise updates from DQN with DPG's ability to handle continuous action spaces. Its use of a deterministic policy gradient allows it to learn effectively using high-dimensional input such as raw image pixels. In the original work, the actor network was encouraged to explore the environment by introducing noise into its decisions, using the Ornstein-Uhlenbeck process (Uhlenbeck and Ornstein, 1930); this use of temporally-correlated noise to adjust the control policy was introduced earlier the same year by Wawrzynski (2015). However, more recent work has noted that this does not provide better results than uncorrelated Gaussian noise, which is simpler to implement (Fujimoto, Hoof and Meger, 2018; Barth-Maron et al., 2018).

DDPG was used to learn solutions to some of the same problems as DQN, generally taking around one-twentieth the number of training steps that DQN required to solve the problem to the same degree, as well as being able to solve several continuous motion control problems for which Q-learning and hence DQN are unsuitable (Lillicrap et al., 2015). DDPG also has the advantage of being relatively simple to implement. If the actor is encouraged to explore sufficiently, and updates are stabilised using layer normalization (Ba, Kiros and Hinton, 2016), DDPG has been shown to outperform several other DRL approaches, including PPO, which is more recent (Kidziński et al., 2018).

Distributed Distributional Deep Deterministic Policy Gradient

DeepMind, who first proposed DDPG (as well as A3C), have continued to develop its ideas. Barth-Maron et al. (2018) demonstrated an evolution of DDPG titled D4PG - distributed distributional deep deterministic policy gradient. In D4PG, parallel actors gather experience collectively and update the critic using a distributional Bellman update, as demonstrated by Bellemare, Dabney and Munos (2017). This greatly stabilises the critic updates, resulting in better guidance for the actor.

D4PG also incorporates N-step returns and prioritised experience replay. However, since the authors ran a variety of experiments with different combinations of their enhancements, they were able to determine that the latter provided only a marginal improvement in simple tasks, and actually destabilised updates in complex ones. D4PG was shown to provide state-of-the-art performance on several difficult continuous control tasks in a stable manner (Barth-Maron et al., 2018), although the use of wall time to compare its training performance against DDPG is somewhat misleading, as significantly more computational resources are required to gather experience in parallel simulations.

Twin Delayed Deep Deterministic Policy Gradient

Methods that incorporate a value function, such as value-based methods like Q-learning as well as actor-critic methods, use bootstrapping to update the estimated value of state-action pairs based on existing estimates. Bootstrapping increases sample efficiency at the cost of being prone to a problem known as value overestimation.

Value overestimation is a compounding of small errors in the estimates used for bootstrapping, which occurs because the Bellman equation update errs on the side of a high estimate. This leads to low-value states eventually being estimated as high-value, and therefore incorrect policy updates to the actor (Fujimoto, Hoof and Meger, 2018). This phenomenon is one of the explanations for the divergent behaviour often seen when training function approximators without stabilising mechanisms.

Fujimoto, Hoof and Meger (2018) propose a variant of DDPG called twin delayed deep deterministic policy gradient (TD3) that aims to reduce the effects of value overestimation.

The ‘twin’ in TD3 refers to the use of two critic networks, from which the least critical is used to update the actor - a technique taken from a method called double Q-learning (van Hasselt, 2010). ‘Delayed’ is in this case refers to the actor being updated less frequently than the critic, with the idea that this gives fewer opportunities for noisy estimates to compound into large errors.

Chapter 4

Agent Design

4.1 Modelling Vehicle Control

Fundamentally, to learn a policy for controlling a vehicle, an agent must be aware of the shape of the environment and the ego vehicle's dynamics, and be able to make control inputs. The particular senses and control mechanisms available in this case are those exposed by the Gym-TORCS interface developed by Dossa (2018).

4.1.1 State Space

Table 4.1 lists the sensory inputs that were chosen from those availabe. While the visual output of the simulation is one of the available senses, this was not included, as this prevents the simulation from being run headlessly, and greatly increases the state space. The rangefinding sense τ (see Table 4.1) conveys the relevant information about the shape of the track with many fewer degrees of freedom, which is advantageous for training against. Since the available senses describe the environment as well as the ego vehicle's state, the input from these senses at any particular time describes the full state of the simulation.

4.1.2 Action Space

Given this state-space, we must also define the actions available for the agent to take. The simulation offers the option to control the steering, throttle, brake and gearbox; for this project, the transmission was set to automatic, meaning the agent need only control the steering, throttle and brake - see Table 4.2.

4.2 Algorithm Design

The problem as stated has a high number of state-space dimensions and two continuous action-space dimensions. This precludes the use of fully value-based approaches, instead requiring a policy-based or hybrid method. Recall that traditional policy-based methods, i.e. ones that use a stochastic policy, are greatly outperformed by actor-critic methods that can make use of a deterministic policy (Silver et al., 2014). This would suggest the use of either DDPG or A3C.

Name	Symbol	Range	Description
`angle`	θ	$[-\pi, +\pi]$	Angle between ego vehicle longitudinal axis and track axis
`speedX`	v_x	$[-\infty, +\infty]$	Velocity along longitudinal axis (forward/backward) in kilometres per hour
`speedY`	v_y	$[-\infty, +\infty]$	Velocity along transverse axis (sideslip) in kilometres per hour
`speedZ`	v_z	$[-\infty, +\infty]$	Velocity in vertical direction (climb/descent) in kilometres per hour
`track`	τ	$19 * [-1, 200]$	Distance between the ego vehicle and track edge in metres, in 19 directions. Values are capped at 200, and all values will be -1 if the vehicle is off-track
`trackPos`	p	$[-1, +1]$	Transverse displacement between track axis and ego vehicle, where 0 means the vehicle is in the centre, 1 means it is at the edge, and -1 means it is off-track
`wheelSpinVel`	ω	$4 * [0, +\infty]$	Rotation rate of each wheel, in radians per second

Table 4.1: Agent sensory inputs

Name	Symbol	Range	Description
`steering`	σ	$[-1, +1]$	The steering angle, from -1 (full left lock) to $+1$ (full right lock), with 0 being neutral
`throttle`	γ	$[-1, +1]$	The throttle/braking input, from -1 (full braking force) to $+1$ (full throttle)

Table 4.2: Agent action space

DDPG was selected as the basis for the agent's design, as experience replay improves sample efficiency in costly environments such as TORCS, and enhancements such as the distributional critic update from D4PG and twin critic stabilisation from TD3 offer state-of-the-art performance.

4.2.1 DDPG in Depth

This section will discuss the mechanics of DDPG as implemented in the agent in this study, with a formal summary given in Figure 4.1. As noted in Section 3.3.1, DDPG is an off-policy actor-critic deep reinforcement learning algorithm for continuous action spaces.

Four function approximators are used in DDPG: a pair of working networks representing the value function, or critic ($Q(s, a|\theta^Q)$), and the policy function, or actor ($\mu(s|\theta^\mu)$), and their "target" counterparts, Q' and μ'. The target networks are used to achieve more stable updates and were introduced by DQN (Mnih et al., 2015). It is this duality of networks that makes DDPG off-policy. These networks are initialised with uniform random weights and biases, but with each working-target pair having the same values i.e. $\theta^{Q'} \leftarrow \theta^Q$ and $\theta^{\mu'} \leftarrow \theta^\mu$.

The other technique drawn from DQN is the replay buffer used for experience replay (R). This is usually implemented as a list of tuples, where each tuple contains a state transition i.e. (s_t, a_t, r_t, s_{t+1}). The general idea of the algorithm is to, each time step, sample a new experience from the environment, and train the actor and critic using a sample of the replay buffer's contents.

At each time step and in state s_t, the actor μ selects an action a_t according to its policy, i.e. $a_t = \mu(s_t|\theta^\mu)$. a_t is then taken in the environment, resulting in the next state s_{t+1} and a reward signal r_t. This state transition is then stored in R.

In the original DDPG study, the selection of a_t was augmented with temporally-correlated noise produced by an Ornstein-Uhlenbeck process (Lillicrap et al., 2015), to encourage exploration. As noted in Section 3.3.1, later work by Fujimoto, Hoof and Meger (2018) and Barth-Maron et al. (2018) has shown that this temporal correlation is unnecessary, and uncorrelated Gaussian noise is sufficient. For this project, Gaussian noise was used.

While the use of noise - correlated or otherwise - to encourage exploration is well-established in reinforcement learning, this is usually applied directly to the decisions made by the agent. Recent research by Plappert et al. (2017) has demonstrated superior exploration behaviour by applying noise to an agent's network parameters, rather than its outputs; this is now known as parameter space noise. Parameter space noise can be applied trivially to off-policy algorithms such as DDPG by perturbing the policy while experience is gathered, and learning as normal from sampled experiences. To start with, parameter space noise was applied to θ^μ.

With the new state transition added to R, the next step is to randomly sample N tuples (s_i, a_i, r_i, s_{i+1}) from R. The loss term L of the working critic can then be derived using:

$$L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$$

Where y_i is given by:

$$y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$$

This term y_i represents the discounted long-term value of the target actor's (μ') prospective decisions in each s_i in the sampled experiences, as estimated by the target critic Q'. The target networks are used because they represent the agent's most stable estimates of the optimal policy and Q-function. The critic's parameters θ^Q are then updated using L via mean-squared error.

The critic can then produce the policy gradient, with which to update θ^μ via gradient ascent:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

Finally, the two target networks are updated 'softly', i.e. their parameters $\theta^{Q'}$ and $\theta^{\mu'}$ are moved towards θ^Q and θ^μ by a ratio of $\frac{1}{\tau}$. The next time step begins with selecting a_{t+1}, and the process continues until the specified number of training steps T has elapsed.

4.2.2 Design Parameters

As in the original DDPG paper, the actor and critic network were constructed with two hidden layers of 400 and 300 units respectively, using rectified linear activation (Lillicrap et al., 2015). The Adam optimizer developed by Kingma and Ba (2014) was used, with initial learning rates of 10^{-4} for the actor (α) and 10^{-3} for the critic (β) respectively, as in the original implementation.

Unlike the original implementation, however, the state and action inputs to the critic were concatenated in the input layer, rather than having the action input skip the first hidden layer. This is in line with the approach used by the TD3 paper (Fujimoto, Hoof and Meger, 2018). The actor's output layer consisted of two units using *tanh* activation, owing to the $[-1, +1]$ range of both control outputs (see Table 4.2).

The list of remaining training hyperparameters is given in Table 4.3, with the values used at the start of the design iteration process. These are included in-text here rather than in an appendix because they will be the subject of further discussion.

4.3 Architecture

The software developed for this project (henceforth "the system") follows a simple client-server architecture. The system developed interfaces with the vTORCS (Cuccu, 2012) simulation via two steps. First, the system consumes Gym-TORCS (Dossa, 2018) as a local Python library. Gym-TORCS uses the SnakeOil Python client developed by Edwards (2013) to communicate over a designated local network port with vTORCS itself. Figure 4.2 shows the relationship between these entities.

Gym-TORCS allows its users to create and destroy simulation instances, observe the environment and perform actions via the Gym interface (OpenAI, 2016) in a simple, imperative manner. This setup allows simulator and session configuration to be controlled

```
Deep Deterministic Policy Gradient
  Let $\tau$ be some value such that $\tau \ll 1$
  Let $\rho$ be some value such that $\rho \gg 1$
  Let $\gamma$ be some value such that $0 < \gamma < 1$
  Let $T$ be some value such that $T \gg 1$
  Let $N$ be some value such that $1 \ll N \ll \rho$
  Let $\sigma$ be some value such that $\sigma > 0$

  Initialise critic network $Q(s, a|\theta^Q)$
  Initialise actor network $\mu(s|\theta^\mu)$, with Gaussian parameter noise scaled by $\sigma$
  Initialise target networks $Q'$ and $\mu'$ with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
  Initialise replay buffer $R$ with size $\rho$
  Receive initial environment state $s_1$

  **for** $t = 1, T$ **do**                                    ▷ For non-episodic environments
      Execute action $a_t = \mu(s_t|\theta^\mu)$
      Observe new state $s_{t+1}$ and reward $r_t$
      Store the experience $(s_t, a_t, r_t, s_{t+1})$ in $R$

      Randomly sample $N$ tuples $(s_i, a_i, r_i, s_{i+1})$ from $R$
      Derive loss term from target networks $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
      Update critic by minimising the loss $L = \frac{1}{N}\sum_i(y_i - Q(s_i, a_i|\theta^Q))^2$
      Update actor policy using sampled policy gradient:

          $$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|s_i$$

      Update the target networks with:

          $$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$

          $$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  **end for**
```

Figure 4.1: DDPG Algorithm as implemented, adapted from Lillicrap et al. (2015)

Symbol	Value	Description
α	10^{-4}	Actor learning rate
β	10^{-3}	Critic learning rate
τ	10^{-3}	Soft update rate for target networks
ρ	10^{6}	Number of experiences to store in the replay buffer
γ	0.99	Discount factor for future rewards
T	10^{6}	Number of training steps to take
N	100	Number of experiences to sample for each training update
σ	0.3	Gaussian noise scale factor

Table 4.3: Agent hyperparameters, with initial values

with code: one of its particular benefits is that by disabling rendering the simulation to the screen, the simulation can be greatly accelerated. Gym-TORCS permits a time-step approach to client-server interaction, rather than having to synchronise actions with observations as a human player would, by pausing the simulation in the time between reporting the state and being instructed of the next action.

4.4 Reward Function Design

Intuitively, the agent's goal is to drive around the track in the minimum possible time. However, the obvious approach of having fast lap times provide a reward is too naive. Problems with large state-action spaces in which the agent must achieve a distant goal to receive a reward, i.e. those whose environments offer sparse rewards, are difficult to solve because the agent must happen upon a good policy by chance before receiving any reward at all (Sutton and Barto, 2018, pp.469-470).

A solution to the sparse reward problem is reward shaping, which refers to the use of intermediate rewards to guide exploration towards the actual desired behaviour (Skinner, 1938). This could take the form of providing smaller rewards for achieving certain 'checkpoints', or providing proxy rewards that encourage behaviour in the correct direction.

It is also important to design a reward function that will discourage the agent from settling on local maxima. This is of particular concern in our case, since exploration is a difficult problem when learning in continuous action spaces (Lillicrap et al., 2015). The design of the agent as described in Section 4.2.1 incorporates noise to encourage exploration. However, since the critic's evaluation and the updates to both the actor and critic are informed by the reward signal, this is not sufficient to overcome a reward function that does not incentivise exploration.

An intuitive example of a local maximum could arise if our reward function simply rewarded high speed (v_x). Imagine the ego vehicle starting the simulation on a straight section of track and accelerating in a straight line. To learn to brake for the corner at the end of the straight section would require it to choose to reduce its reward, and the agent may not be explorative enough to try this.

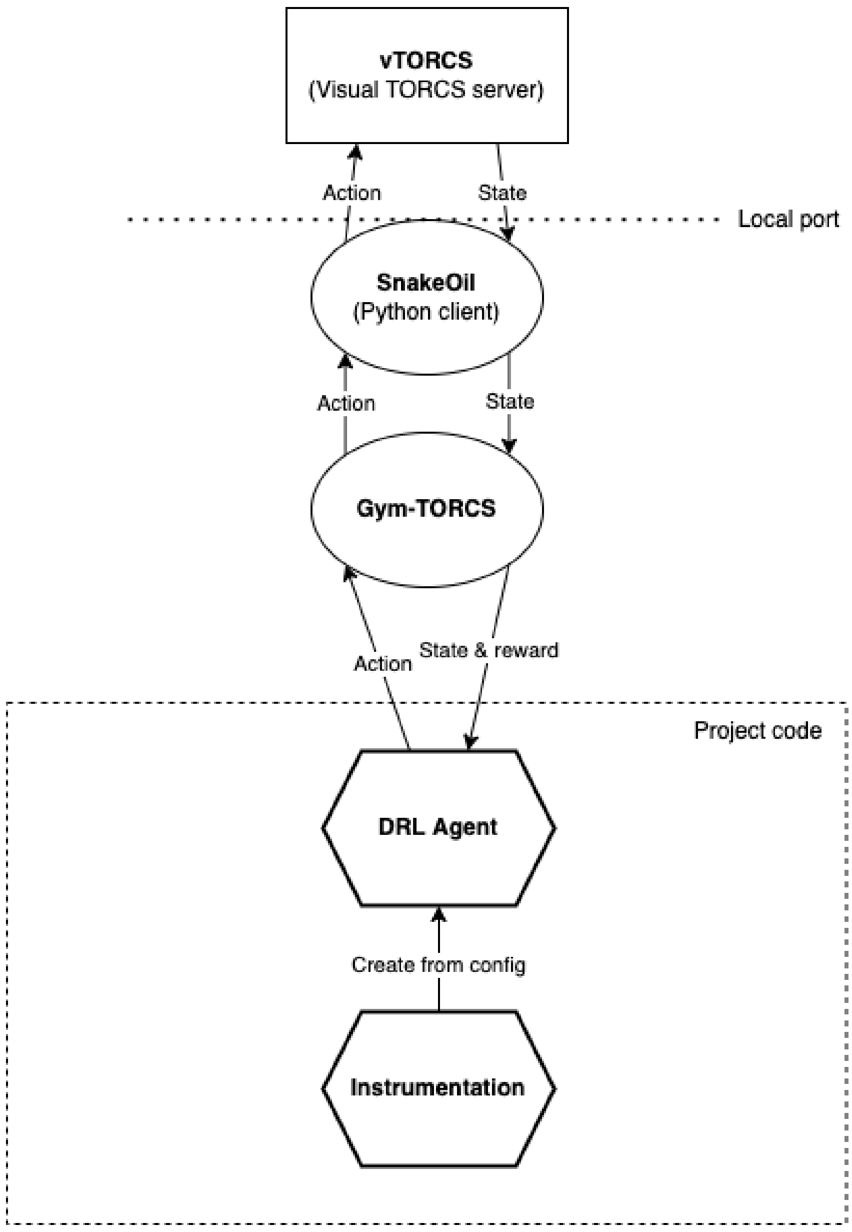

Figure 4.2: System architecture

Relatedly, the reward function's shape must be relevant to the desired behaviour. If we wanted to discourage this sort of short-term behaviour and instead incentivised total distance covered, the agent may not learn to drive particularly quickly, as this would increase the risk of going off-track and being punished.

If a vehicle can perform a lap of a track in a short time, this requires maintaining a high average velocity along the track. Since we have both the ego vehicle's forward velocity (v_x) and the angle between its longitudinal axis and the axis of the track (θ), we can easily derive its resultant velocity along the track axis:

$$r = v_\theta = v_x cos\theta$$

Rewarding v_θ allows us to provide a continuous reward signal, alleviating the sparse reward problem, while also incentivising the desired behaviour. It is also a simple means of encouraging the agent to follow the track curvature, while allowing it to use more of the available space. When a racing vehicle navigates a corner at speed, better performance can be achieved by using the full width of the track, as this reduces the effective curvature of the turn, increasing the minimum velocity reached. The initial agent design uses the above reward function as a starting point.

In addition to rewarding v_θ, the agent would receive a reward of -1 and the episode would end if the agent moved backwards along the track or crossed the boundaries at the track edge (i.e. sensor $p = -1$).

Chapter 5

Training Methodology

Developing and training the agent was an iterative process, with the precise design and methodologies used evolving over time. This chapter will describe the general plan of training, while Chapter 6 will explore the changes that were made to the agent's design over the course of the project.

5.1 Hardware

The experiments described in this dissertation were performed on a computer running Ubuntu Linux with an AMD Ryzen 7 3700X CPU and an Nvidia GeForce GTX 970 GPU. The agent was developed using the Keras machine learning framework (Chollet et al., 2015) with TensorFlow as the backing tensor processing library (Abadi et al., 2015). Both of these libraries offer versions supporting discrete GPU acceleration via Nvidia's CUDA API (Nvidia, 2021), which allowed between a two-fold and five-fold reduction in training time.

An unfortunate drawback of using GPU acceleration is that training outcomes become nondeterministic, or at least more so. This is despite setting constant random seeds for the libraries used by the system. Although there seems to be some element of nondeterminism inherent within the TORCS simulation, training on CPU only resulted in similar learning outcomes between identical trials.

The advantage offered by offloading some work to the GPU is that it can be performed in a highly parallel manner: this however means that the pseudorandom numbers produced by the system's random seeds are consumed in a nondeterministic order, rendering the ordering they provide meaningless. Nondeterminism is a known problem for GPU-accelerated machine learning research, and the major tensor processing libraries are being updated to optionally provide determinism guarantees. However, it is easily argued that a model that relies on the precise ordering of pseudorandom numbers to achieve a particular performance is not robust.

Once an agent has been trained, reproducible results are obtainable in testing by running on CPU only, which is acceptable.

5.2 Normalisation

The improvements to the original Gym-TORCS wrapper made by Dossa (2018) normalise the sensory inputs listed in Table 4.1, i.e. scale them to the range $[-1, +1]$ or $[0, 1]$ rather than $[-\pi, +\pi]$ or $[-\infty, +\infty]$. Inputs such as τ are already constrained to a known range, making this trivial. For senses with unlimited values, such as v_x, a sensible maximum value is used e.g. 300 $km \cdot h^{-1}$.

This is an important adjustment: without it, signals that have naturally larger numbers such as v_x would drown out inputs like p when propagating through both networks. This would cause the model to fit to these inputs in proportion to their larger size, which is undesirable.

5.3 Instrumentation

One component of the system developed for this study was a means of defining experiments in advance, so as to run them and save the results automatically. This included specifications for agent hyperparameters, training parameters, and logistical details like disabling rendering and track selection. There was also a means of sharing replay buffers between experiments, although a means for saving buffers to disk for use later was not developed.

This allowed different hypotheses to be tested without waiting for the training to run: some sessions could take up to several hours. This was particularly useful because training outcomes proved sensitive to the hyperparameters used - this is a known limitation of DDPG and of DRL in general (Barth-Maron et al., 2018). An advantage of creating a system for writing model weights to disk and reading them back was that training experience could be re-used between trials where relevant, as well as being able to observe the agent's behaviour with rendering enabled, to get a subjective sense of the quality of its policy.

5.4 Strategy

5.4.1 Format

Experiments take place over several simulation episodes, where each episode begins with the vehicle at the track's start line and ends either after a preset time or distance limit, or when the agent misbehaves by leaving the track or moving backwards along it.

After a configured number of training episodes in which the full DDPG algorithm is run, a validation episode is run on the same track in order to evaluate learning. This uses the target policy μ' without any noise, does not add experiences to the replay buffer, and does not update any network weights. The sum of the reward signals received at every time step during the validation episode is then stored for later reporting.

This cycle of training and validation continues until a configured number of total training steps T (see Table 4.3) is reached. An experiment with 10^6 training steps took between three and ten hours to complete, depending on other parameters, e.g. a larger batch size N requires more computational work at each step. However, the relationship between

N and total runtime was nonlinear, as the batch network update can take advantage of GPU parallelism.

5.4.2 Track Selection

TORCS offers a wide variety of tracks with different surfaces and layouts. These range in complexity from simple ovals of the kind used for NASCAR and other stock car racing series to replicas of technical circuits like Brands Hatch, Laguna Seca, and Suzuka.

Intuitively, since the agent can only learn from experiences it is able to access, training it on a track without sufficient complexity would limit the generality of policy it could learn. If, for example, the track used was an infinitely long straight road, the agent would converge on a policy of driving quickly in a straight line, and never learn how to brake or navigate corners.

In fact, the learning outcomes observable from training are quite sensitive to the track selection. The default track configured by Gym-TORCS - `g-track-1` - has a mix of straight sections and high- and low-speed corners, but its particular arrangement means the first time the agent needs to brake is after a long high-speed section, for a tight hairpin. In such a situation, the car is travelling quickly, meaning there is less time available to explore the option of braking before the car fails to make the turn and leaves the road, ending the episode.

This manifests as the agent optimising well for navigating this first part of the track, but only very rarely learning to brake in time to take the first hairpin successfully, even given hundreds of thousands of training steps per session. Being 'stuck' in a particular set of states like this makes the agent less likely to learn to escape them, as the target networks are slowly trained towards that local minimum. Tracks such as `g-track-2`, which feature corners requiring braking earlier on in the track and at lower speeds, allow a more general policy to be learned much more quickly.

However, tracks such as `spring` are so complex that an agent trained on it fails to learn any reasonable policy. A track or tracks must be selected that are technical and varied enough to encourage learning a general policy, but not so difficult that not even the training track can be learned.

5.4.3 Validation

While the agent's performance was recorded during validation episodes within the training session, this data was mainly used to assess the learning curves displayed by different configurations.

Once an agent had been trained, its policy was evaluated by testing it on tracks that it did not see during training. Five road tracks and two dirt tracks were selected for this; their layouts are shown in Figure 5.1. Trained agents were evaluated over a ten-lap time trial on each of these circuits. Success was measured by the ability to complete ten laps, with fastest lap times also recorded.

The road tracks were selected for their emphasis on different aspects of vehicle handling. `wheel-1` and `eroad` have a mix of low-speed and high-speed corners to exercise general handling. `forza` - which is a replica of the Monza grand prix circuit from the 1970s,

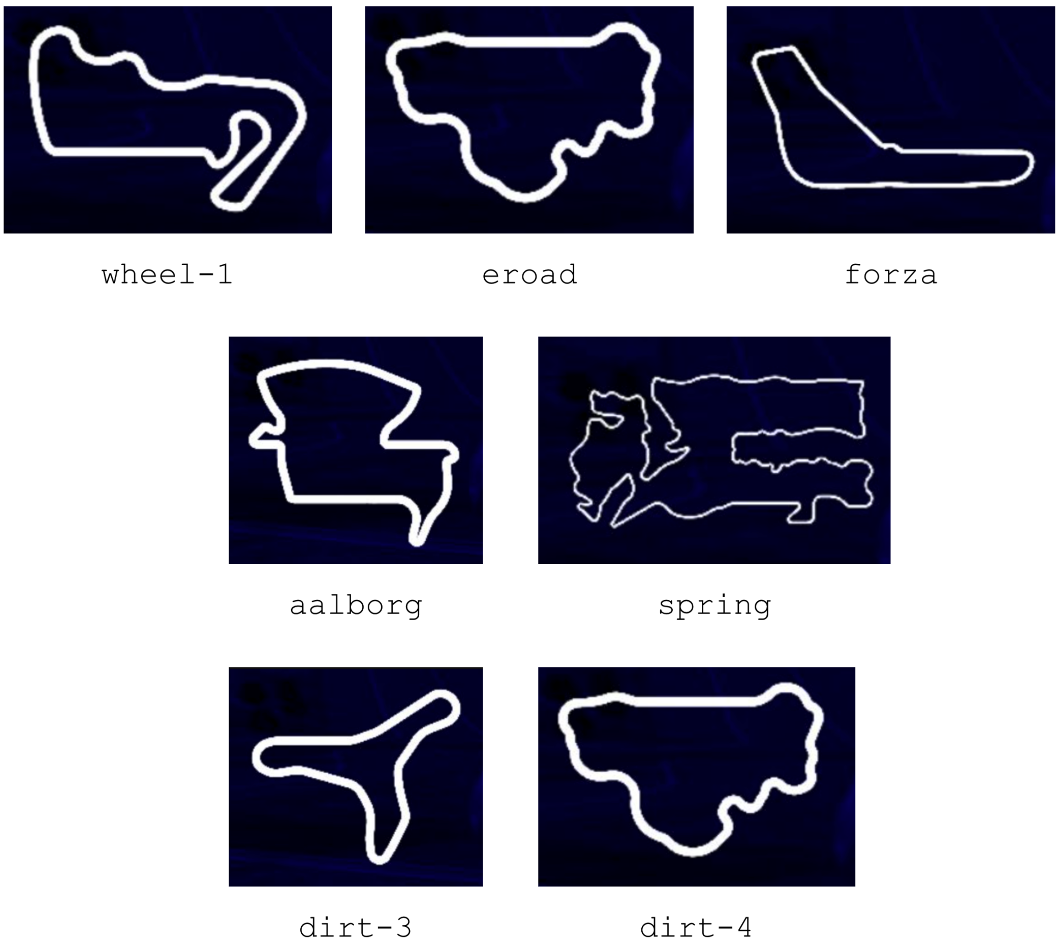

Figure 5.1: Validation tracks (Wymann et al., 2014)

before chicanes were added to slow drivers down - has long straights with a few high-speed corners, requiring stability and smoothness of input at high speed. Similarly, `aalborg` is a tight and technical circuit with many sharp corners, requiring heavy braking. `spring` is a long and complex circuit with many difficult parts.

Two fairly simple dirt circuits were selected: `dirt-3`, which is a replica of a former Australian rallycross circuit called Catalina Park (see Figure 5.3), and `dirt-4`, which is a dirt-surface version of `eroad`.

In addition to recording lap times and total reward figures, it was possible to watch the agent under test. As driving is an easily-understood activity for a human observer, inspecting the agent's behaviour or style of driving offered subjective insights into the quality of learning. This is made easer by the simulation's control input overlay - observe the bottom right corner of Figures 5.2 and 5.3. For example, excessive oscillation around the track median indicated a noisy learning process, and failure to attempt to brake for a sharp corner suggested an insufficient degree of exploration. These observations were useful in tuning the hyperparameters used, as well as determining the need for the refinements discussed in Chapter 6.

It was also observed that the characteristics of the track or tracks used for training influenced the driving style of the agent. Agents trained on `alpine-1`, which is a narrow track with many sharp turns, tended to drive more slowly and with more caution on other tracks. Intuitively, these agents had learned that in their experience, the risk of colliding with the track edge outweighed the benefit of going more quickly. Conversely, an agent trained on a faster and more open track like `wheel-2` - a replica of the Suzuka grand prix circuit (Figure 5.2) - tended to be more aggressive on the validation tracks, achieving better lap times.

Observing the trained agents' driving in a subjective sense was also useful for debugging the simulation. During training it became apparent that for some tracks, the episode would terminate without an obvious cause, e.g. the car leaving the track or going backwards. TORCS supports simulation of damage from collisions with objects in the environment or other vehicles, and this is exposed as one of the sensory inputs by the Gym-TORCS wrapper.

Gym-TORCS' default behaviour is to treat any damage signal as grounds to end the episode. It transpired that on some circuits this can happen at random moments for no apparent reason, while the car is driving normally. This negatively impacted the learning process, as some random subset of otherwise valid states were marked as terminal, and the agent learned to avoid them, introducing additional noise. The fork of Gym-TORCS used for the project was modified to remove the damage signal check.

Figure 5.2: Trained agent driving on `wheel-2` (Suzuka Circuit)

Figure 5.3: Trained agent driving on `dirt-3` (Catalina Park)

Chapter 6

Agent Refinement

6.1 Training Format

At first, it was expected that allowing an episode to last a long time would be beneficial: if an episode contains a large number of steps, this is because the agent is performing well enough to navigate the track without driving off it.

However, allowing arbitrarily long validation episodes is problematic. Once the agent has learnt a good enough policy to complete multiple laps, a small misstep leading to moving off the track in an early lap greatly reduces the reward from that episode. Conversely, there will by chance be occasional outstanding performances with total reward an order of magnitude larger. This leads to a long-tailed distribution of validation performance, which appears spiky when graphed, making it more difficult to reason about the outcomes of different experiments in relation to one another.

Further, this long-tailed distribution can give some states excessively large values in the critic's Q-function: this is because they were encountered before completing a large number of laps, rather than because they are intrinsically more valuable. This effect can learn to the agent overfitting to the set of states it has observed, limiting the generality of its policy.

The Gym-TORCS wrapper allows the introduction of a lap limit, which was then used. This is useful for comparing trained agents on the same track, as it offers an intuitive test case. However, limiting the number of laps for training episodes is even more problematic. If the agent is limited to a small number of laps e.g. three, this will produce a more attractive learning curve, but once it is capable of finishing all three laps, the reward function fails to encourage learning.

The reward function (see Section 4.4) rewards in proportion to velocity along the track axis. However, it is applied at a constant rate with respect to simulation time. If the vehicle is travelling faster, a larger distance will be covered between training step intervals. A fixed number of laps represents a fixed distance to cover, rather than number of steps.

Comparing two hypothetical agents that can complete three laps in 100 and 200 seconds respectively, the former is clearly superior, but each will receive the same reward signal. Since the former agent covers distance twice as quickly, it is rewarded twice as richly over half the number of steps, cancelling out the speed advantage. Therefore, it is essential to limit episode length to a sensible value during training sessions using a fixed number of steps, rather than a fixed number of laps.

6.2 Reward Function

The original reward function in Section (4.4) rewards longitudinal velocity only. This means the maximum reward per step r_{max} is equal to the vehicle's maximum speed, and the minimum is zero. While this is a correct reward function in the sense that transverse velocity and position are common in efficient racing lines, i.e. paths of minimum curvature (Cardamone et al., 2010), in practice it was found that discouraging transverse velocity promotes better learning:

$$r = v_x(cos\theta - sin\theta)$$

With this function, r_{max} is unchanged, but the minimum (moving perpendicular to track axis) is $-r_{max}$. The equilibrium point of zero reward is found at 45°. Intuitively, this provides more encouragement for the agent to follow the track curvature, and the transverse angle involved in an efficient racing line is seldom greater than 45°.

Some time was spent investigating penalising transverse position as well, i.e. encouraging the agent to move away from the edges of the track:

$$r = v_x(cos\theta - sin\theta - kp)$$

Where k is some scalar e.g. $\frac{1}{2}$, and p is the degree of displacement between the track axis and edge (see Table 4.1). However, this tended to introduce an oscillation around the track median, and did not provide any learning advantages. Without the kp term, the agent could be observed after training to find efficient paths through corners, using the full width of the track.

6.3 Algorithm Design

While the classic DDPG implementation described in Section 4.2.1 was found to be able to learn good driving policies some of the time, it suffered from unpredictable training outcomes and noisy learning processes. Despite the use of experience replay with a large buffer and large buffer sample size, the algorithm was prone to destructive updates and divergent behaviour.

Recall that improvements to DDPG introduced by TD3 (Fujimoto, Hoof and Meger, 2018) and D4PG (Barth-Maron et al., 2018) stabilise and improve learning outcomes. While D4PG offers state-of-the-art performance, the use of many concurrent actors requires high-performance computing hardware that was not available for the present study. For example, the designers of D4PG reported their results after training the model for 12 hours with 32 parallel actors (Barth-Maron et al., 2018); to perform the same workload would likely take at least several days on the consumer hardware available, which would be impractical.

6.3.1 TD3 in Depth

As twin delayed deep deterministic policy gradient (TD3, see Section 3.3.1) presents improvements to the stability of DDPG with only minor changes and a marginal increase in required computing resources, modifications were made to the agent to implement it. This section lists the changes introduced by TD3, with reference to the formal summary in Figure 6.1. The new hyperparameters introduced in Figure 6.1 are described in Table 6.1.

Twin Critics

As noted, the 'twin' in TD3 refers to the use of two critics, each with their own target network. The compounding effect of value overestimation is mitigated by using the more conservative estimate of the two when computing the loss term using the Bellman operator, i.e. rather than the following, in DDPG:

$$y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$$

We would instead take:

$$y_i = r_i + \gamma \min_{i=1,2} Q_i'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q_i'})$$

As in the original TD3 paper by Fujimoto, Hoof and Meger (2018), the critics are then updated using mean-squared error, given the loss L:

$$L = argmin_{\theta_i} \frac{1}{N} \sum (y - Q_i(s,a))^2$$

An unfortunate drawback of having this additional critic network is that the algorithm is less performant during training: since there is more work to do, the same hardware can run fewer training steps per second versus DDPG.

Delayed Policy Update

TD3 incorporates a delayed policy update in that the actor's policy and all target networks are only updated every d time steps. In the original paper, as in our agent, $d = 2$, i.e. the update happens every other step. The critics are still updated every time step.

Target Policy Smoothing

The last change from DDPG is the use of target policy smoothing, a regularisation strategy based on the learning update from SARSA (Rummery and Niranjan, 1994). This adds clipped Gaussian noise to actions chosen by the target policy when deriving the critic target value:

$$\tilde{a} = \mu'(s') + \epsilon, \quad \epsilon \sim clip(\mathcal{N}(0, \tilde{\sigma}), -c, c)$$

This $\tilde{a}$ is then substituted for the $\mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q_i'}$ term when deriving y_i.

Twin Delayed Deep Deterministic Policy Gradient

```
Let $\tau$ be some value such that $\tau \ll 1$
Let $\rho$ be some value such that $\rho \gg 1$
Let $\gamma$ be some value such that $0 < \gamma < 1$
Let $T$ be some value such that $T \gg 1$
Let $N$ be some value such that $1 \ll N \ll \rho$
Let $\sigma$ be some value such that $\sigma > 0$
Let $d$ be some value such that $d > 1$
Let $c$ be some value such that $c > 0$

Initialise critic networks $Q_1(s, a|\theta^{Q_1})$ and $Q_2(s, a|\theta^{Q_2})$
Initialise actor network $\mu(s|\theta^{\mu})$
Initialise target networks $Q'_1$, $Q'_2$ and $\mu'$ with weights $\theta^{Q'_1} \leftarrow \theta^{Q}_1$, $\theta^{Q'_2} \leftarrow \theta^{Q}_2$, $\theta^{\mu'} \leftarrow \theta^{\mu}$
Initialise replay buffer $R$ with size $\rho$
Receive initial environment state $s_1$

for $t = 1, T$ do                                    ▷ For non-episodic environments
    Execute action $a_t = \mu(s_t|\theta^{\mu})$ with noise $\epsilon \sim \mathcal{N}(0, \sigma)$
    Observe new state $s_{t+1}$ and reward $r_t$
    Store the experience $(s_t, a_t, r_t, s_{t+1})$ in $R$

    Randomly sample $N$ tuples $(s_i, a_i, r_i, s_{i+1})$ from $R$
    $\tilde{a} \leftarrow \mu'(s') + \epsilon$, where $\epsilon \sim$ clip $(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
    Derive loss term $y \leftarrow r + \gamma \min_{i=1,2} Q'_i(s', \tilde{a})$
    Update critics by minimising loss $L = \arg\min_{\theta_i} \frac{1}{N} \sum (y - Q_i(s, a))^2$

    if $t$ mod $d = 0$ then
        Update actor policy using sampled policy gradient:
            $$\nabla_{\theta^{\mu}} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^{Q})|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^{\mu}} \mu(s|\theta^{\mu})|s_i$$
        Update the target networks with:
            $$\theta^{Q'_i} \leftarrow \tau\theta^{Q} + (1 - \tau)\theta^{Q'_i}$$
            $$\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1 - \tau)\theta^{\mu'}$$
    end if
end for
```

Figure 6.1: TD3 Algorithm, adapted from Fujimoto, Hoof and Meger (2018)

Symbol	Value	Description
d	2	Number of critic updates for each actor and target update
c	0.5	Maximum noise magnitude

Table 6.1: TD3-specific hyperparameters

6.3.2 Other Changes

In accordance with the TD3 paper, and after observing no difference in practice, parameter space noise was replaced with conventional Gaussian action noise. However, unlike the TD3 paper, the target policy smoothing approach of clipping the applied noise to $[-c, c]$ was also used for the exploration noise. This was found to provide a small benefit to training stability, and reduced undesirable oscillation in steering outputs.

Also, while the TD3 authors allowed their agents to behave in a purely exploratory manner for the first 10^3 time steps in episodic environments (Fujimoto, Hoof and Meger, 2018), this was not implemented in the present project, as an episode can terminate in only a few steps.

Finally, while the TD3 authors used the same hidden layers of 400 and 300 neurons in their implementation as in the DDPG paper, it was observed that reducing the sizes of these layers to 256 neurons each improved runtime with no loss in representative power. A full diagram of the architectures used for the actor and critics can be found in Appendix A.

6.3.3 Hyperparameter Tuning

The TD3 paper demonstrated its best results with $\tau = 5 \times 10^{-3}$ and a learning rate of 10^{-3} for both actor and critic (Fujimoto, Hoof and Meger, 2018). However, the DDPG learning rates of 10^{-4} for the actor, 10^{-3} for the critic, and $\tau = 10^{-3}$ showed better learning in the TORCS environment.

The replay buffer was sized to contain all experiences gathered during the training process, as in the TD3 paper, and the number of samples to take at each time step (N) was increased from 100 to 256, as this was found to improve stability.

A full list of all hyperparameters used is available in Appendix B (see Table B.1).

Chapter 7

Results

7.1 Training Performance

As noted in Section 5.4.2, TORCS offers a wide variety of tracks. The selection of the track used for training has a large impact on the outcome, and while it was impractical to exhaustively trial all the supported tracks, a subset of tracks was chosen for evaluation. These were selected on the basis that they were considered challenging enough for the agent to learn a general policy, but not so challenging as to make learning impossible.

Trials of 10^6 training steps were conducted on each of these tracks, with the resulting trained agent validated against the tracks listed in Section 5.4.3 in ten-lap time trials. Of these, the tracks that facilitated the best learning were `wheel-2`, or Suzuka, and `dirt-6`, a technical dirt course. Their layouts are shown in Figure 7.1.

The agents' target policy was evaluated against the training track every 10 episodes. Their performance - expressed as the total reward signal received during each validation episode - can be seen in Figures 7.2 and 7.3.

In these figures, the horizontal axis - labelled "Trial" - counts the number of validation episodes. Since more successful policies are able to drive for longer without leaving the track, the total number of training steps that passed during the ten episodes between each of these validation trials was not only variable, but increased with performance. This

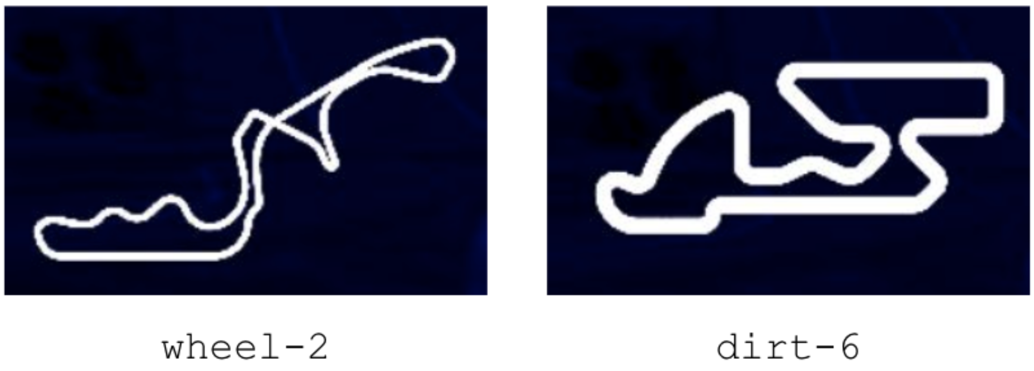

Figure 7.1: Training tracks (Wymann et al., 2014)

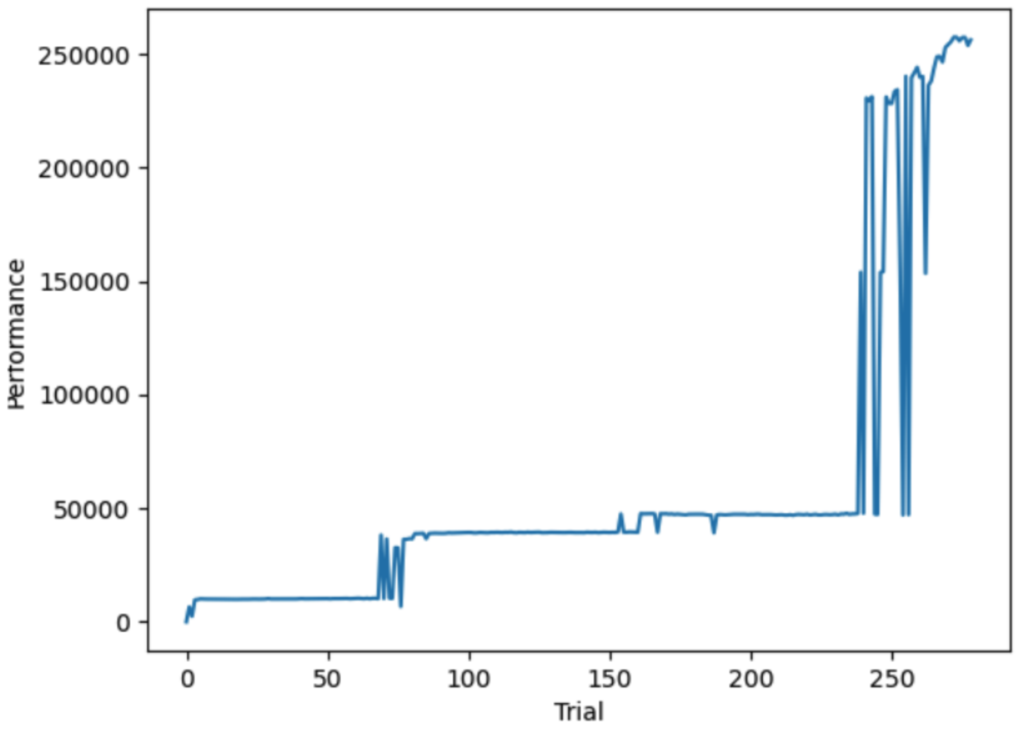

Figure 7.2: Training performance curve for agent trained on `wheel-2`

means that the horizontal axis does not divide up the training time evenly. This is also why the number of total trials differs betwen Figures 7.2 and 7.3 - the constant is the number of training steps, i.e. 10^6.

Since the agent was rewarded approximately in proportion to longitudinal velocity with respect to the track axis, and the two tracks used have different shapes, the total reward achieved is not comparable between tracks, and is presented only to give scale to the curve within each graph.

7.2 Validation Performance

In Tables 7.1 and 7.2, the performance of each agent can be seen as measured on the seven validation tracks. A trial was considered ‘successful’ if the agent was able to complete all ten laps. This trial length was chosen on the grounds of providing a high level of confidence that the driving policy was sustainable: from testing, it was found that some agents that could complete five laps could not complete ten, but an agent that could complete ten was never observed to fail to complete 20.

Comparing the two tables, it is clear that the agent trained on `dirt-6` learned a far more general policy, even on non-dirt tracks, i.e. those of a different character to its training experience. It was able to consistently navigate all but one of the unseen tracks, including `aalborg` and `spring`, which were among the most difficult for other agents to manage. Since `spring` is so long, ten laps at around 800 seconds per lap is over two hours of driving in simulation, which this agent completed without making a serious mistake.

Of the 16 agents trained, none managed to complete more than one lap of `dirt-3`, despite

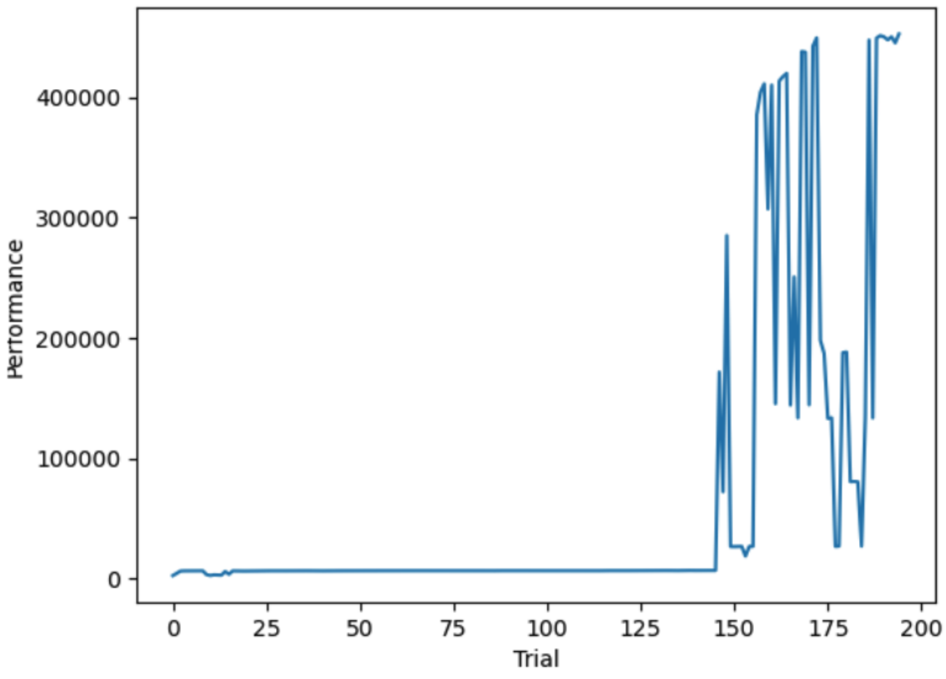

Figure 7.3: Training performance curve for agent trained on `dirt-6`

Track	**Laps**	**Fastest Lap (s)**	**Successful**
`wheel-1`	10	107.02	Yes
`eroad`	10	85.68	Yes
`forza`	10	157.71	Yes
`aalborg`	0	-	No
`spring`	0	-	No
`dirt-3`	0	-	No
`dirt-4`	0	-	No

Table 7.1: Validation performance of agent trained on `wheel-2`

Track	**Laps**	**Fastest Lap (s)**	**Successful**
`wheel-1`	10	140.78	Yes
`eroad`	10	113.96	Yes
`forza`	10	184.92	Yes
`aalborg`	10	111.52	Yes
`spring`	10	807.82	Yes
`dirt-3`	0	-	No
`dirt-4`	10	113.91	Yes

Table 7.2: Validation performance of agent trained on `dirt-6`

its apparently simple layout. This may have been due to its vertical shape: it includes more elevation changes and cambered corners than other tracks.

Despite the greater generality of the `dirt-6` agent, the agent trained on `wheel-2` is nonetheless included as a finalist due to its much higher speed on road courses. While speed is not the primary goal of AV research, in this case it serves as a proxy measure of the agent's 'confidence' on the road. It is worth noting that this agent was 15-25% faster than the one trained on `dirt-6` on all the tracks it could navigate.

In fact, it may be the case that the agent trained on `dirt-6` was more successful at other tracks precisely because it was slower, i.e. by driving further from the car's limits it gave itself more time to react to corners. While this strategy is not desirable for a racing driver, it may well be desirable for the public road.

Chapter 8

Discussion

8.1 Summary

This project has presented an investigation into the use of deep reinforcement learning for the control of autonomous vehicles in unfamiliar locations and poor road conditions. It was demonstrated that the deep deterministic policy gradient family of algorithms - specifically, twin delayed deep deterministic policy gradient - is able to learn driving policies that generalise to handling unseen road layouts safely, to the extent of being able to stay within the track limits.

In contrast to the second stated research aim (see Section 1.2), it was found that training such an agent on low-grip dirt surfaces improves the generality of policy learnt for driving on a normal road, rather than *vice versa*. This is at the cost of reduced 'confidence', as measured by average lap time. A possible intuitive explanation for this is that the agent did not encounter enough low-grip situations while training on a road surface to prepare it for the dynamics of driving on dirt, whereas training exclusively on a low-grip dirt surface prepared it well for the few low-grip situations it encountered when driving on the road.

8.2 Limitations

8.2.1 Training Experience

One important limitation of the experimental format used was that for each of the agents whose performance was reported in Chapter 7, the training experience only encompassed a single track. While this means achieving any level of generality for other tracks is a good result, it also limits the maximum generality achievable. Since TORCS supports the addition of custom tracks, and there are track generators available, it would be possible to generate and use a very long track containing many different kinds of topology, but a better alternative would be to train on multiple tracks.

Using multiple tracks was attempted during the project, but this proved non-trivial. The nature of the algorithm's policy and value updates caused lessons learned on previous tracks to be destroyed. For example, one agent was trained for 10^6 steps on `dirt-6` followed by another 10^6 steps on `wheel-2`. The models' weights were saved after the first training period for comparison with the final product, and the replay buffer was sized to include all 2×10^6 steps.

Unfortunately, the version of the agent that had trained on both tracks was inferior in almost every respect to the one that had half the total experience, and had only seen `dirt-6`. This instability in the learning process is a troubling limitation for this kind of serial training. However, this is not necessarily an intrinsic limitation of this style of deep learning: there may be implementation problems that remain undiscovered. It is also possible that training for too long on any track causes a divergence and degeneration in the learning process, i.e. it may be possible to learn productively on tracks in series if the total number of training steps is kept smaller.

8.2.2 Oscillation

Observing the agents presented in Chapter 7 while driving on unfamiliar roads, a behaviour of oscillating steering outputs can clearly be seen on straight sections. When cornering, this behaviour is generally absent, with the steering output being quite smooth.

This tendency limits the maximum speed achievable in straight sections, and hence discourages the agent to accelerate: steering back and forth at high speed will upset the car's balance and lead to a spin, ending the episode. Agents tended to learn to avoid this, but not to avoid the root cause, i.e. the oscillation.

It seems reasonable that this might be caused by the reward function chosen: by rewarding longitudinal velocity and penalising transverse velocity, we cause the reward signal to reduce if the agent steers away from the track axis. On a high-speed straight section, anything more significant than a slight correction will lead to an over-correction, setting up the oscillation.

However, after experimenting with a reward function that only rewarded forward vehicle speed - i.e. $r = v_x$ - the oscillation was still observed. This behaviour may instead be due to exploration noise being gradually pushed into the target policy by the training process.

8.2.3 Hyperparameter Tuning

It is well-known that reinforcement learning outcomes are unpredictable and can be difficult to reproduce. This is especially true for algorithms that fall into the "Deadly Triad" described by Sutton and Barto (2018, p.264): those that use function approximation, which includes all DRL algorithms by definition, those that use bootstrapping, and those that are off-policy, i.e. where training is performed on transitions not produced by the target policy. DDPG and TD3 fall into all three of these categories. It is important to note that Sutton and Barto do not recommend against algorithms that make use of these techniques, as they each have their own advantages.

This unpredictability means that certain aspects of the training and refinement process, especially experimenting with different hyperparameter values e.g. learning rates and sample batch sizes, can be quite challenging. When outcomes are sensitive to many factors, and are not precisely repeatable even with no change in preconditions due to GPU acceleration, it can be impossible to know whether a change is truly beneficial. There may be different arrangements of hyperparameters that can deliver better and more stable learning outcomes that were not explored during this project.

One partial solution to this problem would be to guarantee determinism at least for the hyperparameter optimisation process, either by using a future tensor processing library that offers determinism guarantees or by sacrificing GPU acceleration altogether. This would eliminate the problem of repeatable training outcomes for a given set of preconditions, but not the deadly triad problem.

8.3 Future Work

8.3.1 Experience Gathering

In Section 8.2, it was noted that the format used was unable to produce good results from training on multiple tracks, due to later updates destroying earlier learning. One possible solution for this could be to use a parallel algorithm like A3C or D4PG, and have each of the parallel agents gather experience on a different track simultaneously. This would give a more equal weighting to experience gathered from each, in that the network updates are not coupled to a particular ordering of tracks.

If the hardware resources to perform such an experiment were not available, it might be possible to emulate this to a degree with the current system by running each training episode on a rotation of tracks. This would require either restarting the TORCS environment every episode or using the parallelism feature of Gym-TORCS to maintain parallel instances, either of which should be feasible.

8.3.2 Multiple Agents

A natural extension of the research presented in this project would be to investigate the ability of DRL agents to cope with other road users. TORCS supports other AI drivers, i.e. those that use its own built-in AI system, and the agent's reward function could be adjusted to incentivise it to keep a safe distance from other vehicles. The dynamics between multiple DRL agents could also be explored with tools like MADRaS (Santara et al., 2021), which provides a 'multiplayer' version of Gym-TORCS for just this purpose.

8.3.3 Sensor Degradation

Although poor weather conditions can degrade AV control performance by reducing the grip available from the road surface, this is not the only mechanism. Substances such as rain, dust, or sand in the air can degrade sensor performance, introducing noise into the system's inputs or causing some sensors to become useless. In order to further answer the question of whether DRL is a useful tool for overcoming the challenge of inclement weather, it might be possible to simulate such a situation by adding noise to sensory inputs before passing them to the agent. Another option would be to enable graphics rendering, learn directly from the pixels on-screen as in Lillicrap et al. (2015), and include rain or dust in the simulation. This would require the use of a simulator that supported such weather.

8.3.4 Practical Applications

Prior work on the use of DRL for AV control has focussed on specific lower-level planning tasks, e.g. lane-keeping and motorway merging (see Section 3.1.3). Given a number of

DRL-based agents for specific motor tasks required for driving automation, an AV control system could delegate to each agent in turn according to its understanding of the current situation, e.g. giving control over to the lane-changing subsystem when the higher-level plan requires that activity.

The project presented in this dissertation has demonstrated that DRL can also be a useful tool for general-purpose low-level navigation: projects such as Waymo (Schwall et al., 2020) require more detailed maps of a location than would be necessary for a human driver, but these maps are not available in all locations. Using a DRL-based subsystem for road handling could help bridge the gap in areas with only consumer-grade maps.

Similarly, this project has demonstrated the potential for a DRL-based subsystem to maintain control of the vehicle when low-grip surfaces like ice or dirt on the road is encountered. The project as described does not provide an agent with fine enough control for e.g. lane-keeping on a public road, but the principle appears sound.

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y. and Zheng, X., 2015. TensorFlow: Large-scale machine learning on heterogeneous systems [Online]. Software available from tensorflow.org. Available from: `https://www.tensorflow.org/`.

Aeberhard, M., Rauch, S., Bahram, M., Tanzmeister, G., Thomas, J., Pilat, Y., Homm, F., Huber, W. and Kaempchen, N., 2015. Experience, results and lessons learned from automated driving on germany's highways. *IEEE intelligent transportation systems magazine* [Online], 7(1), pp.42–57. Available from: `https://doi.org/10.1109/MITS.2014.2360306` [Accessed 26 February 2021].

Aradi, S., 2020. Survey of deep reinforcement learning for motion planning of autonomous vehicles. *Ieee transactions on intelligent transportation systems* [Online], pp.1–20. Available from: `https://doi.org/10.1109/TITS.2020.3024655`.

Ba, J.L., Kiros, J.R. and Hinton, G.E., 2016. Layer normalization. *arxiv preprint arxiv:1607.06450*.

Barth-Maron, G., Hoffman, M.W., Budden, D., Dabney, W., Horgan, D., Tb, D., Muldal, A., Heess, N. and Lillicrap, T., 2018. Distributed distributional deterministic policy gradients. *arxiv preprint arxiv:1804.08617*.

Bellemare, M.G., Dabney, W. and Munos, R., 2017. A distributional perspective on reinforcement learning. *International conference on machine learning*. PMLR, pp.449–458.

Cardamone, L., Loiacono, D., Lanzi, P.L. and Bardelli, A.P., 2010. Searching for the optimal racing line using genetic algorithms. *Proceedings of the 2010 ieee conference on computational intelligence and games*. IEEE, pp.388–394.

Chollet, F. et al., 2015. Keras. `https://keras.io`.

Cuccu, G., 2012. Visual torcs [Online]. Available from: `https://github.com/giuse/vtorcs` [Accessed 18 Feb 2022].

Dikmen, M. and Burns, C.M., 2016. Autonomous driving in the real world: Experiences with tesla autopilot and summon. *Proceedings of the 8th international conference on automotive user interfaces and interactive vehicular applications*. pp.225–228.

Dossa, R.F.J., 2018. Gymtorcs: An openai gym-style wrapper for the torcs racing car simulator [Online]. Available from: `https://dosssman.github.io/projects/gym-torcs/` [Accessed 27 October 2021].

Edwards, C.X., 2013. Snakeoil [Online]. Available from: `https://xed.ch/project/snakeoil/index.html` [Accessed 18 Feb 2022].

Feng, D., Haase-Schütz, C., Rosenbaum, L., Hertlein, H., Gläser, C., Timm, F., Wiesbeck, W. and Dietmayer, K., 2021. Deep multi-modal object detection and semantic segmentation for autonomous driving: Datasets, methods, and challenges. *Ieee transactions on intelligent transportation systems* [Online], 22(3), pp.1341–1360. Available from: `https://doi.org/10.1109/TITS.2020.2972974`.

Fikes, R.E. and Nilsson, N.J., 1971. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4), pp.189–208.

Fuchs, F., Song, Y., Kaufmann, E., Scaramuzza, D. and Duerr, P., 2021. Super-human performance in gran turismo sport using deep reinforcement learning. *Ieee robotics and automation letters*.

Fujimoto, S., Hoof, H. and Meger, D., 2018. Addressing function approximation error in actor-critic methods. *International conference on machine learning*. PMLR, pp.1587–1596.

Goodfellow, I., Bengio, Y. and Courville, A., 2016. *Deep learning*. MIT Press. `http://www.deeplearningbook.org`.

Hasselt, H. van, 2010. Double q-learning. *Advances in neural information processing systems*, 23.

Isele, D., Rahimi, R., Cosgun, A., Subramanian, K. and Fujimura, K., 2018. Navigating occluded intersections with autonomous vehicles using deep reinforcement learning. *2018 ieee international conference on robotics and automation (icra)*. IEEE, pp.2034–2039.

Jaritz, M., De Charette, R., Toromanoff, M., Perot, E. and Nashashibi, F., 2018. End-to-end race driving with deep reinforcement learning. *2018 ieee international conference on robotics and automation (icra)*. IEEE, pp.2070–2075.

Jeong, Y., Kim, S. and Yi, K., 2020. Surround vehicle motion prediction using lstm-rnn for motion planning of autonomous vehicles at multi-lane turn intersections. *Ieee open journal of intelligent transportation systems* [Online], 1, pp.2–14. Available from: `https://doi.org/10.1109/OJITS.2020.2965969`.

Kidziński, Ł., Mohanty, S.P., Ong, C.F., Huang, Z., Zhou, S., Pechenko, A., Stelmaszczyk, A., Jarosik, P., Pavlov, M., Kolesnikov, S. et al., 2018. Learning to run challenge solutions: Adapting reinforcement learning methods for neuromusculoskeletal environments. *The nips'17 competition: Building intelligent systems*. Springer, pp.121–153.

Kingma, D.P. and Ba, J., 2014. Adam: A method for stochastic optimization. *arxiv preprint arxiv:1412.6980*.

Kiran, B.R., Sobh, I., Talpaert, V., Mannion, P., Al Sallab, A.A., Yogamani, S. and Pérez, P., 2021. Deep reinforcement learning for autonomous driving: A survey. *Ieee transactions on intelligent transportation systems*.

Krizhevsky, A., Sutskever, I. and Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, pp.1097–1105.

Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D. and Wierstra, D., 2015. Continuous control with deep reinforcement learning. *arxiv preprint arxiv:1509.02971*.

Lorenzo, J., Parra, I., Wirth, F., Stiller, C., Llorca, D.F. and Sotelo, M.A., 2020. Rnn-based pedestrian crossing prediction using activity and pose-related features. *2020 ieee intelligent vehicles symposium (iv)* [Online]. pp.1801–1806. Available from: `https://doi.org/10.1109/IV47402.2020.9304652`.

Mallozzi, P., Pelliccione, P., Knauss, A., Berger, C. and Mohammadiha, N., 2019. Autonomous vehicles: State of the art, future trends, and challenges. *Automotive systems and software engineering*. Cham: Springer International Publishing, pp.347–367.

Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D. and Kavukcuoglu, K., 2016. Asynchronous methods for deep reinforcement learning. *International conference on machine learning*. PMLR, pp.1928–1937.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G. et al., 2015. Human-level control through deep reinforcement learning. *nature*, 518(7540), pp.529–533.

Nvidia, 2021. *CUDA GPUs | nvidia developer* [Online]. Nvidia. Available from: `https://developer.nvidia.com/cuda-gpus` [Accessed 25 April 2021].

OpenAI, 2016. Gym [Online]. Available from: `https://github.com/openai/gym` [Accessed 18 Feb 2022].

Patel, S., Griffin, B., Kusano, K. and Corso, J.J., 2019. Predicting future lane changes of other highway vehicles using rnn-based deep models. `1801.04340`.

Plappert, M., Houthooft, R., Dhariwal, P., Sidor, S., Chen, R.Y., Chen, X., Asfour, T., Abbeel, P. and Andrychowicz, M., 2017. Parameter space noise for exploration. *arxiv preprint arxiv:1706.01905*.

Pomerleau, D.A., 1989. *Alvinn: An autonomous land vehicle in a neural network*.

Ren, S., He, K., Girshick, R. and Sun, J., 2017. Faster r-cnn: Towards real-time object detection with region proposal networks. *Ieee transactions on pattern analysis and machine intelligence* [Online], 39(6), pp.1137–1149. Available from: `https://doi.org/10.1109/TPAMI.2016.2577031`.

Rummery, G.A. and Niranjan, M., 1994. *On-line q-learning using connectionist systems*, vol. 37. Citeseer.

Russell, S. and Norvig, P., 2016. *Artificial intelligence: A modern approach*. 3rd ed. Harlow: Pearson Education Limited.

SAE, 2018. Sae international releases updated visual chart for its "levels of driving automation" standard for self-driving vehicles [Online]. Available from: `https://www.sae.org/news/press-room/2018/12/sae-international-releases-updated-visual-chart-for-its-\T1\textquotedblleftlevels-of-driving-`

automation\T1\textquotedblright-standard-for-self-driving-vehicles [Accessed 27 October 2021].

SAE, 2021a. Sae levels of driving automation™ refined for clarity and international audience [Online]. Available from: https://www.sae.org/blog/sae-j3016-update [Accessed 15 January 2022].

SAE, 2021b. *Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles* [Online]. Available from: https://www.sae.org/standards/content/j3016_202104/ [Accessed 1 Oct 2021].

Sallab, A.E., Abdou, M., Perot, E. and Yogamani, S., 2017. Deep reinforcement learning framework for autonomous driving. *Electronic imaging*, 2017(19), pp.70–76.

Santara, A., Rudra, S., Buridi, S.A., Kaushik, M., Naik, A., Kaul, B. and Ravindran, B., 2021. Madras: Multi agent driving simulator. *Journal of artificial intelligence research*, 70, pp.1517–1555.

Schulman, J., Levine, S., Abbeel, P., Jordan, M. and Moritz, P., 2015. Trust region policy optimization. *International conference on machine learning*. PMLR, pp.1889–1897.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A. and Klimov, O., 2017. Proximal policy optimization algorithms. *arxiv preprint arxiv:1707.06347*.

Schwall, M., Daniel, T., Victor, T., Favarò, F. and Hohnhold, H., 2020. *Waymo public road safety performance data* [Online]. Waymo. Available from: https://storage.googleapis.com/sdc-prod/v1/safety-report/Waymo-Public-Road-Safety-Performance-Data.pdf [Accessed 9 April 2021].

Shalev-Shwartz, S., Shammah, S. and Shashua, A., 2016. Safe, multi-agent, reinforcement learning for autonomous driving. *arxiv preprint arxiv:1610.03295*.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D. and Riedmiller, M., 2014. Deterministic policy gradient algorithms. *International conference on machine learning*. PMLR, pp.387–395.

Skinner, B., 1938. *The behavior of organisms: an experimental analysis*. Appleton-Century.

Sutton, R.S. and Barto, A.G., 2018. *Reinforcement learning: An introduction*. 2nd ed. Cambridge, MA: MIT press.

Szegedy, C., Ioffe, S., Vanhoucke, V. and Alemi, A.A., 2017. Inception-v4, inception-resnet and the impact of residual connections on learning. *Thirty-first aaai conference on artificial intelligence* [Online]. Available from: https://www.aaai.org/ocs/index.php/AAAI/AAAI17/paper/viewPaper/14806.

Thorpe, C., Hebert, M., Kanade, T. and Shafer, S., 1987. Vision and navigation for the Carnegie-Mellon NAVLAB. *Annual review of computer science*, 2(1), pp.521–556.

Uhlenbeck, G.E. and Ornstein, L.S., 1930. On the theory of the brownian motion. *Physical review*, 36(5), p.823.

Wang, F.Y., Zhang, J.J., Zheng, X., Wang, X., Yuan, Y., Dai, X., Zhang, J. and Yang, L., 2016. Where does alphago go: From church-turing thesis to alphago thesis and beyond. *Ieee/caa journal of automatica sinica*, 3(2), pp.113–120.

Wang, S., Jia, D. and Weng, X., 2018. Deep reinforcement learning for autonomous driving. *arxiv preprint arxiv:1811.11329.*

Watkins, C.J. and Dayan, P., 1992. Q-learning. *Machine learning*, 8(3), pp.279–292.

Wawrzynski, P., 2015. Control policy with autocorrelated noise in reinforcement learning for robotics. *International journal of machine learning and computing*, 5(2), p.91.

Wymann, B., Espié, E., Guionneau, C., Dimitrakakis, C., Coulom, R. and Sumner, A., 2014. TORCS, The Open Racing Car Simulator [Online]. Available from: `http://torcs.sourceforge.net/index.php` [Accessed 9 April 2021].

Yoshida, N., 2016. Gym-torcs [Online]. Available from: `https://github.com/ugo-nama-kun/gym_torcs` [Accessed 25 April 2021].

Ziegler, J., Bender, P., Schreiber, M., Lategahn, H., Strauss, T., Stiller, C., Dang, T., Franke, U., Appenrodt, N., Keller, C.G., Kaus, E., Herrtwich, R.G., Rabe, C., Pfeiffer, D., Lindner, F., Stein, F., Erbs, F., Enzweiler, M., Knöppel, C., Hipp, J., Haueis, M., Trepte, M., Brenk, C., Tamke, A., Ghanaat, M., Braun, M., Joos, A., Fritz, H., Mock, H., Hein, M. and Zeeb, E., 2014. Making bertha drive—an autonomous journey on a historic route. *IEEE intelligent transportation systems magazine* [Online], 6(2), pp.8–20. Available from: `https://doi.org/10.1109/MITS.2014.2306552` [Accessed 26 February 2021].

Appendix A

Agent Network Architectures

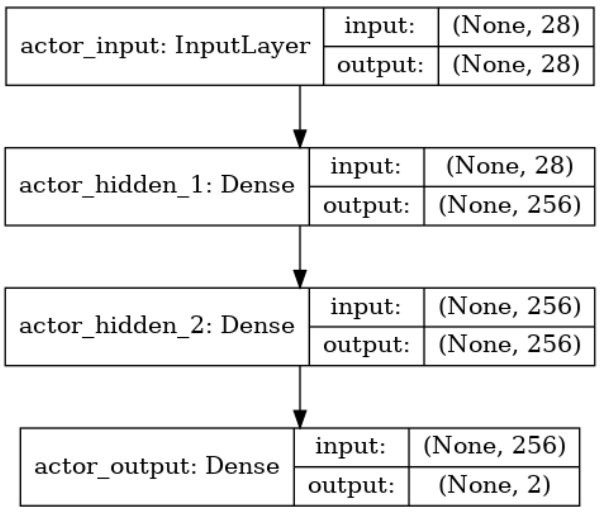

Figure A.1: Actor network architecture, used for working and target networks

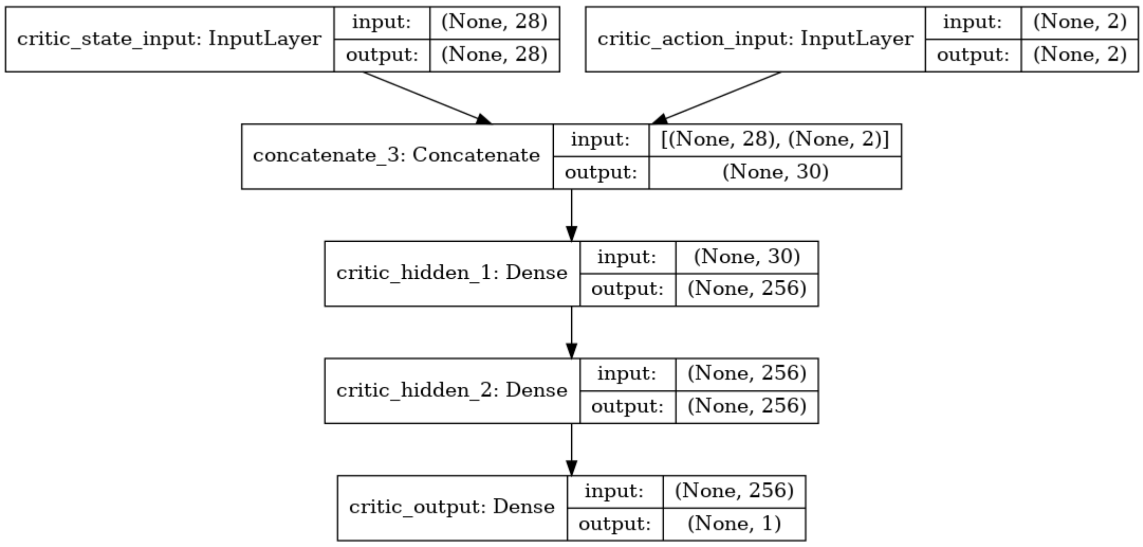

Figure A.2: Critic network architecture, used for working and target networks

Appendix B

Agent Hyperparameters

Symbol	**Value**	**Description**
α	10^{-4}	Actor learning rate
β	10^{-3}	Critic learning rate
τ	10^{-3}	Soft update rate for target networks
ρ	∞	Number of experiences to store in the replay buffer
γ	0.99	Discount factor for future rewards
N	256	Number of experiences to sample for each training update
σ	0.2	Gaussian noise scale factor
d	2	Number of critic updates for each actor and target update
c	0.5	Maximum noise magnitude

Table B.1: Final agent hyperparameters

Appendix C

University of Bath Ethics Checklist

This form must be attached to the dissertation as an appendix.

Department of Computer Science

12-Point Ethics Checklist for UG and MSc Projects

Student	Edmund Summers
Academic Year or Project Title	Developing Resilient Autonomous Vehicles with Deep Reinforcement Learning
Supervisor	Dr Hongping Cai

Does your project involve people for the collection of data other than you and your supervisor(s)? **NO**

If the answer to the previous question is YES, you need to answer the following questions, otherwise you can ignore them.

This document describes the 12 issues that need to be considered carefully before students or staff involve other people ('participants' or 'volunteers') for the collection of information as part of their project or research. Replace the text beneath each question with a statement of how you address the issue in your project.

1. *Will you prepare a Participant Information Sheet for volunteers?* YES / NO

   This means telling someone enough in advance so that they can understand what is involved and why – it is what makes informed consent informed.

2. *Will the participants be informed that they could withdraw at any time?* YES / NO

   All participants have the right to withdraw at any time during the investigation, and to withdraw their data up to the point at which it is anonymised. They should be told this in the briefing script.

3. *Will there be any intentional deception of the participants?* YES / NO

   Withholding information or misleading participants is unacceptable if participants are likely to object or show unease when debriefed.

4. *Will participants be de-briefed?* YES / NO

   The investigator must provide the participants with sufficient information in the debriefing to enable them to understand the nature of the investigation. This phase might wait until after the study is completed where this is necessary to protect the integrity of the study.

5. *Will participants voluntarily give informed consent?* YES / NO
Participants MUST consent before taking part in the study, informed by the briefing sheet. Participants should give their consent explicitly and in a form that is persistent –e.g. signing a form or sending an email. Signed consent forms should be kept by the supervisor after the study is complete. If your data collection is entirely anonymous and does not include collection of personal data you do not need to collect a signature. Instead, you should include a checkbox, which must be checked by the participant to indicate that informed consent has been given.

6. *Will the participants be exposed to any risks greater than those encountered in their normal work life (e.g., through the use of non-standard equipment)?* YES / NO
Investigators have a responsibility to protect participants from physical and mental harm during the investigation. The risk of harm must be no greater than in ordinary life.

7. *Will you be offering any incentive to the participants?* YES / NO
The payment of participants must not be used to induce them to risk harm beyond that which they risk without payment in their normal lifestyle.

8. *Will you be in a position of authority or influence over any of your participants?* YES / NO
A position of authority or influence over any participant must not be allowed to pressurise participants to take part in, or remain in, any experiment.

9. *Will any of your participants be under the age of 16?* YES / NO
Parental consent is required for participants under the age of 16.

10. *Will any of your participants have an impairment that will limit Their understanding or communication?* YES / NO
Additional consent is required for participants with impairments.

11. *Will the participants be informed of your contact details?* YES / NO
All participants must be able to contact the investigator after the investigation. They should be given the details of the Supervisor as part of the debriefing.

12.	*Will you have a data management plan for all recorded data?* Personal data is anything which could be used to identify a person, or which can be related to an identifiable person. All personal data (hard copy and/or soft copy) should be anonymized (with the exception of consent forms) and stored securely on university servers (not the cloud).	YES / NO