School of Computing
FACULTY OF ENGINEERING AND
PHYSICAL SCIENCES

Final Report

Reinforcement Learning for Self Driving Cars

Muhammad Ahmad

Submitted in accordance with the requirements for the degree of
MEng, BSc Computer Science with Artificial Intelligence

2022/23

COMP3931 Individual Project

The candidate confirms that the following have been submitted.

Items	Format	Recipient(s) and Date
Final Report	PDF file	Uploaded to Minerva (02/05/23)
Link to online code repository	URL	Sent to supervisor and assessor (02/05/23)
User manuals	Jupyter Notebooks	Sent to supervisor and assessor (02/05/23)

The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

(Signature of Student) Muhammad Ahmad

Summary

Machine Learning and Deep Learning have been in development for a few decades. However, in the last decade there has been significant progress, such as theoretical and practical progress in efficiently training Deep Neural Networks (neural networks with more units and more layers) and advances in availability of data and hardware (such as more powerful Graphics Processing Units). This has lead to amazing use cases of Deep Learning in many different problem domains.

Reinforcement Learning can utilise Deep Learning, however, its method of learning is different from Deep Learning. Self-Driving cars are now becoming a reality and the project aims to see how successful Reinforcement Learning along with Deep Learning is for tasks in Self-Driving Cars.

In the name of Allah, the Gracious, the Merciful.

وَأَن لَّيْسَ لِلْإِنسَٰنِ إِلَّا مَا سَعَىٰ

And that man will have nothing but what he strives for. [Holy Quran, Surah An-Najm (The Star) 53:40]

أَلَيْسَ ٱللَّهُ بِكَافٍ عَبْدَهُۥ ۖ

Is not Allah sufficient for His servant? [Holy Quran, Surah Az-Zumar 39:37]

Acknowledgements

I would like to thank my supervisor Prof. Abdulrahman Altahhan for his consistent support and guidance throughout this project. I would also like to thank Prof. Julian Brooks for providing crucial suggestions for my report structure. My parents were very helpful throughout this year, providing support in whatever way they could.

Contents

List of Figures

List of Tables

Nomenclature

γ	Discount Factor
$A(s, a)$	Advantage Function that takes a state and an action as input: the output estimates how much of an advantage taking this action has over other actions in this state
$Q(s, a)$	Function that maps a state and an action to its Q-value
$V(s)$	Value Function that maps a state to its estimated value
DDQN	Double Deep Q-network
DNN	Deep Neural Network
DQN	Deep Q-network
MDP	Markov Decision Process
PER	Prioritised Experience Replay
PRNG	Pseudo Random Number Generator
ReLU	Rectified Linear Unit

Chapter 1

Introduction and Background Research

1.1 Introduction

Machine Learning has been in development for a very long time. However, in the last decade there has been significant progress, such as theoretical and practical progress in efficiently training large Deep Neural Networks (DNN) and advances in availability of data and hardware (such as more powerful Graphics Processing Units). This has lead to amazing applications Deep Learning in many different real world problem domains [1].
Reinforcement Learning is a promising sub-field of Machine Learning. Its training paradigm is similar to Supervised Deep Learning, however, there are fundamental differences in the training process. In Supervised Deep Learning, the aim is to predict something of interest using an input based on **ground truth** data collected before hand. In Reinforcement Learning, a ground truth dataset is not required; instead the agent observes and interacts with the environment, receiving feedback from the environment. The agent has to learn behaviours based on this sparse feedback. This makes Reinforcement Learning suitable for sequential decision making problems, where collecting ground truth data is not possible due to physical or theoretical limitations etc. However, Reinforcement Learning can benefit greatly from using Deep Learning techniques: DNNs are very useful for **automatic feature extraction**, which makes the learning task easier.
Self-Driving cars are now becoming a reality: this project explores how suitable Reinforcement Learning with Deep Learning is on Autonomous Driving tasks.

Figure 1.1: A real world example of an early self driving car (Creative Commons)

1.2 Background Research

1.2.1 Reinforcement Learning Overview

Reinforcement Learning (RL) has been successfully applied to various problem domains, where Sequential Decisions need to be made.
During training, the agent receives an observation and takes an action based on this observation. There are two components of interest here: the agent and the environment. The agent for our task represents the Reinforcement Learning agent whose objective is to learn how to drive the car within its current lane. The agent goes through three stages [2], which are described below:

1. After receiving the observation for the current time-step, the agent will take an action i.e. it will **interact** with the environment.

2. The agent will then receive a reward from the environment (feedback) based on the action it took previously. Using this, the agent can **evaluate** the action it took.

3. Based on this, the agent will **improve** its understanding of the environment and crucially improve its understanding of the relationship between the observations, actions and potential rewards it will get.

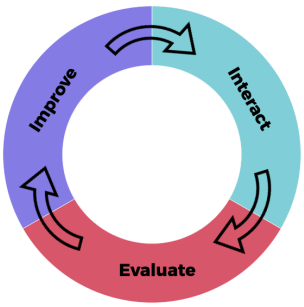

Figure 1.2: Different stages of the agent

Figure 1.2 shows that the three stages above form a loop. [1] The environment can be modelled as a Markov Decision Process (MDP) [2]. The MDP has different states that the environment can be in. It also defines an **action space**, from which the agent can take an action at a given step. The environment then transitions between the different states based on the actions, using the **transition function**. The environment *can* also provide a reward, depending on how good the action was using a **reward function**.
For all but the simplest learning tasks, the MDP behind the environment is not available to the agent: the agent has to learn about the MDP based on the observations it gets from interacting with the environment. The reward function is very useful, for example, it can be formulated in such a way that it promotes certain behaviours [2]. The agent does not have access to the MDP behind this environment.

[1] This figure is based on *Grokking Deep Reinforcement Learning* [2].

1.2.2 Deep Q-network (DQN) and Double Deep Q-network (DDQN)

Deep Q-networks were described in 2015 by Mnih et al. [3]. The article describes a general algorithm that can learn to solve many different tasks. An agent can learn to solve a sequential decision problem by defining the **Q-value function**. If the agent is able to learn the optimal Q-value function, it will be able to take the 'best' action at any given point. The 'best' action(s) is defined as the action that will lead to the highest expected reward at the end of the episode. Here it is beneficial to discuss the agent's 'decision horizon': should the agent aim to prioritise short-term rewards even if they lead to a lower final expected reward **or** should the agent prioritise long-term rewards at the expense of lower short-term rewards? This is crucial, as the decision here will implicitly promote some **policies** (what actions to take given the observations from the environment) than other policies. One way to define this is to use a discount factor (γ), to control the agent's horizon [4]:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + ...$$

where G_t stands for the cumulative reward, R_t stands for the reward at time-step t and γ denotes the discount factor. Effectively, the closer the value of γ is to 1, the more long-term rewards are prioritised; the closer it is to 0, the more short-term rewards are prioritised.
The architecture proposed by Mnih et al. uses a Deep Neural Network to learn the Optimal Q-value function: this is what the 'Deep' in Deep Q-network (DQN) refers to. The other crucial part of this architecture is the use of **experience replay** [3, 5], when training the agent. This involves storing what the agent observes in a replay buffer and when training we sample a batch from this replay buffer and use it to train the agent. Through this the agent will be able to learn from past experience. Each item in the replay buffer will be a tuple of the current observation, the action taken, the reward received and the next observation [6].
Mnih et al. also use a **target network** which is the same as the **online network**, with the online network copied over to the target network after certain steps [3]. In Deep Q-networks, the neural network's aim is to predict the following:

$$Y_t = R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \theta_t^-)$$

where Y_t is the target value, R_{t+1} is the reward, γ is the discount factor, $Q(s, a)$ is the Q-value function, s is the current state, a is the action taken and θ_t^- are the weights of the target Deep Q-network.
The Double Deep Q-network (DDQN) is a variation of the above and uses the following modified target instead:

$$Y_t = R_{t+1} + \gamma Q(S_{t+1}, \operatorname*{argmax}_a Q(S_{t+1}, a; \theta_t); \theta_t^-)$$

where the new θ_t represents the weights of the online Deep Q-network (DQN). The crucial difference here is that DQN uses the same values to select and to evaluate actions; a DDQN uses two different values. This can be seen in the equation above, where the argmax uses θ_t to select the best action while it uses θ_t^- when evaluating actions [7, 3].

1.2.3 Deep Q-network (DQN) vs. Double Deep Q-network (DDQN)

A major weakness of a DQN is that it utilises the same values to select and evaluate the best action. This biases the Deep Q-network towards overestimating the values associated with each action (i.e. overoptimistic values) [3]. The modifications made by a DDQN decouples the above, allowing a better evaluation of the agents current policy.

1.2.4 Exploration vs. Exploitation

The Reinforcement Learning agent faces a dilemma that is easy to relate to: the agent can either utilise the knowledge it has gained on how different actions lead to different rewards in order to take the best action or it can try out a new action it knows nothing or very little about. Relying solely on one of these strategies will put the agent at a severe disadvantage. The agent hopes that continuously taking the action it knows has the best reward will lead to a higher end reward. However, what if another **untested action** has **an even greater reward**? Conversely, what if taking the unknown action leads to a terminal state[2] or severely effects the magnitude of future returns?
This dilemma is formally called the exploration-exploitation trade-off [8]. There are many different training strategies to help the agent with this but they mainly fall into three different categories [9]:

1. Random exploration strategies: These encourage the agent to mostly execute the action it deems the best, however, at random the agent will explore other actions. In implementation this is controlled by a threshold called ϵepsilon along with a random number generator: the higher the epsilon the more likely the agent is to explore.

2. Optimistic exploration strategies: These cause the agent to choose the most uncertain actions, increasing the agent's knowledge about the different actions. Eventually, as the uncertainty decreases, the value that the agent assigns to each action will approach the real value.

3. Information state-space exploration strategies: These explicitly add the **uncertainty** associated with each state to the **state space** (i.e. an extra parameter). This causes the agent to treat the unexplored and explored states differently during training and on inference.

The ϵ-greedy policy (a random exploration training strategy) is useful due to its ease of implementation and widespread usage. A threshold called ϵ controls the rate of exploration and at each decision step:

1. The agent draws a value from a Pseudo Random Number Generator (PRNG)

2. If this value is less than ϵ, the agent will decide to take a random action from the action space [10].

3. The higher the value of ϵ the more exploration the agent will carry out and vice versa.

[2] A state in the environment which causes the episode to end. An MDP represents this state by having all transition arrows out of this state **pointing back to this state**.

1.2.5 Real World Applications of Reinforcement Learning

RL can be utilised in diverse problem domains, such as:

1. **Games:** DeepMind's AlphaGo was able to beat a world champion in the complex board game of Go [11]. RL has also been very successful on Video Games, ranging from Atari games [12] to more complex real time strategy games [13] (Figure C.4 shows an example of the game Go).
2. **Finance:** Portfolio management can be optimised in financial investing using RL [14].
3. **Robotics:** RL can be utilised in robotics where sequential decision making is required in complex environments [15].
4. **Self-Driving Cars:** Autonomous vehicles are required to make sequential decisions based on data from a wide variety of sensors (Section 1.2.7) and RL can be a highly effective aid [16].

1.2.6 Levels of Autonomous Driving

SAE International defines the following to classify the levels of automation in a vehicle [17] (figure C.3 shows a visual representation provided by SAE International):

- **[Assistance] Level 0:** The driver is fully responsible for driving the vehicle. No automation is present, however, assistive technologies, such as 'lane departure warning' and 'automatic emergency braking' can still be present.
- **[Assistance] Level 1:** The driver is aided by assistive technology in either steering control (e.g. staying in lane) or throttle control (e.g. 'adaptive cruise control'), however, the driver is fully responsible for driving the car.
- **[Assistance] Level 2:** The driver is aided by assistive technology in **both** steering control (e.g. staying in lane) **and** throttle control (e.g. 'adaptive cruise control')[3]. The driver still remains fully responsible for driving.
- **[Automation] Level 3:** When the automation feature is enabled, the driver is **not** responsible for driving the vehicle. At this level, the vehicle can automatically drive in 'limited conditions and will not operate unless all required conditions are met' [18]. The driver has to drive when these conditions are not met.
- **[Automation] Level 4:** This level of automation builds up on the previous level, removing the need for the driver to take over when conditions are not met. Driverless taxis that can operate on limited routes are examples of this level of automation.
- **[Automation] Level 5:** A vehicle with this level of automation can drive itself 'under all conditions' [18]. For example, the driverless taxi is no longer limited to specific routes.

[3] In Level 1, the driver is aided in either steering control or throttle control but not both **'at the same time'**: the driver has to actively carry out the other task.

1.2.7 Reinforcement Learning and Self-Driving Cars

Self-Driving Cars have the potential to redefine the way we travel, increasing business productivity and improving human life in general. However, due to the critical nature of the application domain and the huge potential size of the market, great care has to be taken to ensure that this technology is safe. The task of an AI agent can be defined as follows [19]:

1. **Perception:** The agent first has to use the observations it receives to determine crucial information about the car in the environment. This can include where the car is (localisation), what objects are present in the agent's 'field of perception' (contextualisation), what areas of the environment does the agent need to be concerned with (attention mechanism) etc. Humans and other biological agents carry these tasks out subconsciously due to the accumulated knowledge of how the world works but the agent cannot do this.[4]

2. **Planning:** Based on what the agent 'perceives' about the environment and its current goal, the agent has to plan a series of actions that will lead it to the goal. For example, to stop at a red traffic light, after detecting the traffic light the goal of the agent is to slow down on approach to the traffic light before coming to a complete stop at the traffic light. The agent has to plan a set of actions that will lead to this desired behaviour.

3. **Control**: The agent has to execute the action(s) it has decided upon. For example, controlling reducing the throttle value when approaching the traffic light to lower the speed of the vehicle.

Reinforcement Learning with Deep Learning can aid in **all** of the above stages. For example, the DNN in a DQN can learn to automatically extract features from the observations best suited for **perception**. RL is inherently concerned with Sequential Decision problems, making it ideal for the **planning** and **control** step. Furthermore, RL is an **online learning** approach where the agent can continuously learn and adapt to the environment once deployed; traditional Deep Learning uses an **offline learning** approach where the agent is trained on a fixed dataset (ground truth). This project aims to determine how suitable Reinforcement Learning can be on specific tasks for autonomous vehicles.

1.2.8 Alternative approaches

Some alternative approaches that can be used for self-driving cars are listed below:

1. **Imitation Learning:** In Reinforcement Learning, the main constraint is that the agent is not provided with a **ground truth** dataset. This allows the agent to explore different solutions with a possibility to discover novel solutions. An imitation learning approach provides the agent with a 'model answer' to the solution, constraining the agent's exploration. The benefits of this approach are that the agent will converge quickly to an optimal solution and the solution can more readily be implemented (as it would be similar to how a human would approach the task) [20]. The lack of exploration, however, is a major limitation (for example, DeepMind's AlphaTensor utilised an RL based approach

[4]Perception can be a very challenging task due to the real world being very **noisy.**

to find more efficient methods for matrix multiplication [21]). **Behaviour Cloning** is a promising imitation learning approach that can be applied to self-driving tasks [22].

2. **Inverse Reinforcement Learning (IRL):** In this approach, the agent is provided with **expert trajectories** (e.g. recorded when a human is driving a car). A reward function is then learnt from these expert trajectories which is then used to train the RL agent. This allows for more human like solutions to problems while still giving the agent some ability to explore. IRL is very powerful on tasks where it is difficult to define a good reward function, however, the final behaviour learnt by the agent is highly likely to be similar if not influenced by the expert trajectories. Additional time and resources have to be devoted for collecting expert trajectories, which depending on the problem domain can incur significant financial costs as well.

Chapter 2

Methods

2.1 Schematic Diagram of Implementation

Figure 2.1, shows a high level overview of the implementation. The agent receives observations from the environment which are then transformed by the perception blocks. Steering control and throttle control are handled separately as a simplification and to allow redundancy[1]. More detail about these perception blocks is provided in Chapter 3. The agent then plans the best actions to take (planning blocks) and executes them (control block), after which the inference loop is repeated again.

2.2 Sprint Methodology

Due to significant time constraints, a strict agile sprint methodology was used to manage this project: both the implementation and write up were worked on in frequent sprints. It was decided early in the project that the report would be written along with the project implementation, to ensure both the reliability of the report and to manage time effectively. Jupyter Notebooks were used to aid with this principle, as the implementation code and Markdown explanations could be written at the same time. Furthermore, descriptive commit messages were used to keep a record of major milestones and these proved very helpful in the evaluation stage of the project. Figure C.1, shows the sprint plan that was followed for this project.

2.3 Implementation Methodology

The following professional standards were followed in this project:

[1]If either one of the behaviour fails, there is still a level of control available that can provide safety. For example, if steering control fails but throttle control is still functioning, the agent can still safely stop the vehicle.

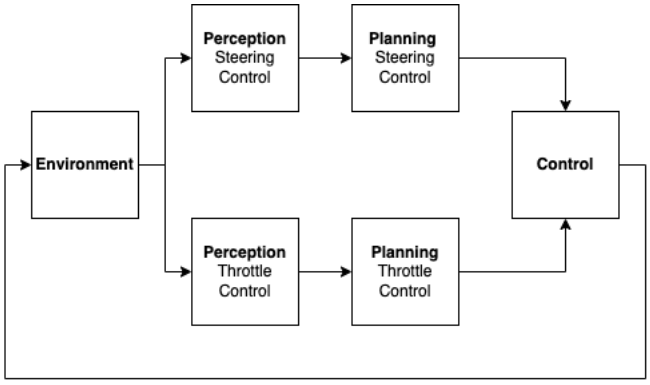

Figure 2.1: Schematic Diagram

- **Version control:** Frequent commits were made to the local repository and these commits were consistently pushed to the online code repository on GitHub. This ensured that both a detailed record and a backup of the project was always present. Furthermore, a less frequent backup of the entire repository was made on a private Google Drive folder as a further precaution.
- **Isolation of contributions:** External code used or adapted was properly cited in the Jupyter Notebooks to ensure proper accreditation (through code comments, comments in the markdown of the Jupyter Notebook etc.) For the Unity project, new or modified C# code was placed in a separate folder to distinguish this from external material (Footnote 25).
- **Object Oriented approach**: Python Classes were used to reduce repetitive code and to allow more higher level classes to inherit base code (logging, iterative saving of models when training etc). Figure
- **Common code library:** All the python code was isolated into the python module at the path **progress-notebooks/individual_project_library**. This aided during implementation because changes only had to be made in the base classes, which could then be utilised by other classes.
- **Metric reliability:** Sections 3.3.3 and 3.3.4 have details on steps taken to ensure that the metrics used for quantitative evaluation are reliable and reproducible.

Chapter 3

Results

3.1 Lane Control

3.1.1 Introduction

This chapter documents details about training the agent on the lane control task. The project utilises PyTorch Lightning for the implementation of the DDQN, as it provides more functionality than the base PyTorch package.[1]

Figure 3.1: Lane Control

It is good practice in industry to mostly use tried and tested implementations, as these are **more efficient** and **more secure.** For the project, however, implementing these architectures provided deeper insights and more fine-grained control.

3.1.2 Testing the DDQN Implementation: Lunar Lander

It is crucial that any non standard implementation is tested on baseline environments, as this reduces the time spent **debugging**: iterative testing is a crucial part of **modern agile development**, allowing implementation problems to be detected and fixed more easily.
The project uses the Lunar Lander environment provided by OpenAI gym.[2]
This environment is similar to the actual lane control task, as the agent has to choose a discrete action. This environment is 'simple', has widespread usage and it does not require a huge amount of time to train.
Figure 3.2 shows what the environment looks like. The agent has to use observations at each time step in order to decide which action to take. Observations are a **tuple of eight values** describing the coordinates, angle, linear velocities and angular velocity of the lander. There are 4 discrete actions that the agent can take.

[1] Please refer to the accompanying repository for code specific details. This repository contains Jupyter Notebooks that document the implementation details and document the sources used during the implementation. Chapter B has details about external code, modified or not, used as part of this project.

[2] The OpenAI Gym library provides many interesting default environments while providing a unified interface to interact with these environments. It is easy to make modifications and adapt for custom environments making it perfect for this project. The full details of the LunarLander environment can be found https://www.gymlibrary.dev/environments/box2d/lunar_lander/.

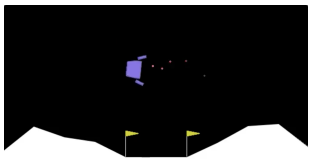

Figure 3.2: The LunarLander-v2 environment

The following videos show how the agent performs before, during and after training [3].

1. Before training, the agent is understandably not very good at its task: agent during training after 0 episodes [4].
2. After 500 episodes, the agent is considerably better than when it started: agent during training after 500 episodes [5].
3. After training for 1000 episodes, the agent has learnt a good policy for the Lunar Lander environment:agent after training for 1000 episodes [6].

3.1.3 Lane Control: Observations for the Agent

For the lane control task, the agent received an RGB image [7] and will be able to control the steering angle and throttle value to control the car. The DonkeyCar simulator, provides the agent with an RGB image as an observation. This emulates how a human sees a colour image of the world when driving. It is crucial to note that the real world is a lot more **chaotic** and **noise** is an ever present issue. However, this simplification allows for faster training and is required due to resource and time constraints [8].

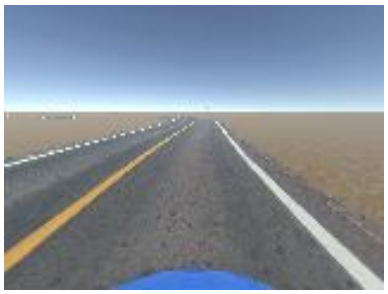

Figure 3.3: Observation from the environment as a RGB colour image

Figure 3.3 shows what this image looks like: the environment provides a matrix with dimensions (120, 160, 3) i.e. a 160 pixels wide and 120 pixels high RGB colour image. Training the agent on these **raw** observations did not lead to good performance, as the agent was not able to learn lane control in the allotted number of episodes for training (10,000): it

[3] Due to time constraints quantitative evaluation was not possible for the Lunar Lander environment.
[4] This video is also available in the repository provided under the **progress_notebooks/saved_videos/lunar-lander-ep-0.mp4.**
[5] This video is also available in the repository provided under the **progress_notebooks/saved_videos/lunar-lander-ep-500.mp4.**
[6] This video is also available in the repository provided under the **progress_notebooks/saved_videos/llunar-lander-ep-1000.mp4.**
[7] An image with three colour channels, red, green and blue.
[8] Please refer to the Jupyter Notebooks in the repository for more details.

seems that the observation size was too large and complex for the agent. Resizing the image to a smaller size of (25, 25), causes a significant reduction in the observation space.

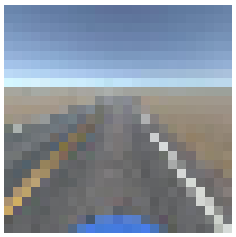

Figure 3.4: Resized RGB colour image

Figure 3.4 shows the resized image. Note that all the salient features in the image, namely the road, lane marking and sky are still recognisable by the human eye. Unfortunately, this did not lead to any significant improvement in agent performance.
One issue that can be seen in the resized image is that there is no emphasis on the important parts of the image. This is not a significant issue for humans, as humans can utilise experience in the real world to decide what parts of an image are important **for a specific task**, The agent does not have this knowledge to fallback on, therefore, making features more prominent will aid training.

Figure 3.5: Resized RGB observation colour image with adjustments made

Figure 3.5 shows that by changing the contrast and brightness level the lane markings can be made more prominent, as they are of a brighter colour. However, the above transformations still did not make significant changes to the agent's learning rate. Transforming to a single channel grayscale image reduces the observation space significantly, as shown in Figure 3.6 [9].

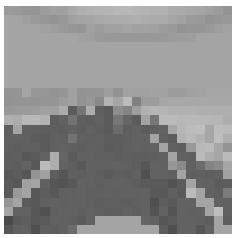

Figure 3.6: Grey-scale resized observation image from the environment

This change did lead to some improvements in the agent's training, however, they were not as significant as expected. One observation from the grey-scale image is that there is effectively a

[9] This is done by transforming the image to the HSV (Hue Saturation Value) image space and using the value channel.

lot of 'noise' in the image, i.e. many pixels are present which **do not directly aid the agent in its lane control task**.

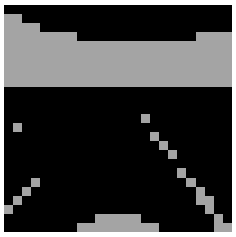

Figure 3.7: Grey-scale resized observation image from the environment, with a value threshold used to reject less bright pixels

In Figure 3.7, a threshold value is used to remove this noise, any pixel with a grey-scale value (i.e. brightness value) lower than this is set to 0 (black) and any pixel above this is allowed, reducing some of the impact of 'noise' on the performance. In figure 3.8, the sky is removed from the image as it is not directly conducive to the lane control task.

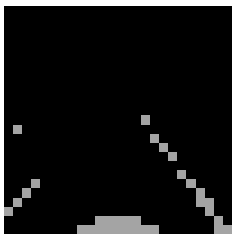

Figure 3.8: Sky removed from previous image

With these transformations, the agent was able to learn a good policy in an acceptable amount of time. However, the learnt policy was not very robust to change. Looking at figure 3.8, it is evident that the lane markings are visible. However, they are **not** very prominent and at significantly different road angles, there would be breaks in the lanes or other visual artefacts. This could be one possible reason for the worse performance at significantly different road angles.

Figure 3.9: Dilate function used to emphasise lane markings

In figure 3.9, the **dilate** operation provided by **OpenCV** is used to put more emphasis on the lanes. This made the the agent more robust to changes.[10].

[10]Transforming observations into a simpler form is called feature engineering.

3.1.4 Lane Control: Deep Q-network Architecture

The optimal Q-value function is approximated using a Deep Neural Network (DNN). Figure 3.10 shows what this neural network looks like.

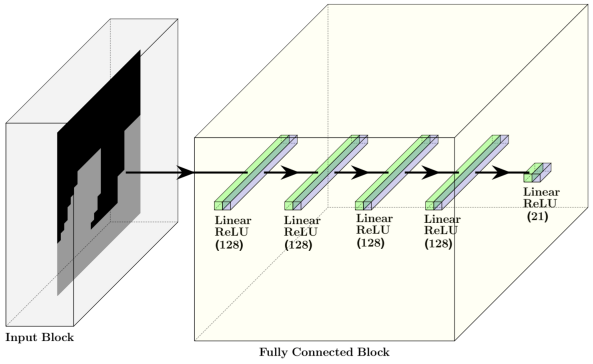

Figure 3.10: Deep Q-network architecture

The numbers under each rectangle show the number of **units** in each layer. The transformed image is **flattened** to a 1-dimensional vector before it is fed into the **Fully Connected Block**. Section 3.1.5 delves into more detail about why there are 21 units in the final layer. In the DNN architecture shown in 3.10 the absence of any **convolutional** layers is by design. Convolutional layers are very powerful, however, there power generally lies in their ability to be robust to change [11], and being able to deal with multiple objects. After the transformations applied in the previous section 3.1.3, a convolutional layer will not aid in this specific scenario.

3.1.5 Lane Control: Actions Space

In order to constraint the learning task a **fixed throttle** value has been used when training the **lane control** agent. A constant throttle was used in order to simplify the learning task for the agent, as otherwise the reward function would need to be too complex. The agent can choose from **21 different discrete steering angles**, ranging from -1.0 to 1.0 with a step of 0.1 between each discrete action in this range
[12].

3.1.6 Lane Control: Reward Function

The reward function used is based on the **cross track error**, which is a measure of how far the car is deviating from the **centre of the track**: the further the car is from the centre, the lower the reward. This reward function is not very complicated which makes it easy for the agent to learn a good solution to the task[13].

[11] Robust here refers to the ability to still recognise an object event after it is rotated, translated, deformed in the image or in different lighting conditions.

[12] Initially the action space was smaller and only had 5 discrete actions. However, the agent was not able to react proactively to steeper bends. After increasing the action space in granularity, the agent performed well even at steeper bends in the road.

[13] Section 3.2.8 has details about a reward function that is more complicated.

3.1.7 Lane Control: Qualitative Evaluation

The list below provides links to videos that show the agent at different stages.

1. Similar to the baseline Lunar Lander environment, the agent is not very good at lane control before training: agent before training[14].

2. After training, the agent becomes quite good at the lane control task: afent after training [15].

In order to evaluate the performance of the agent after training, different variations of the procedural road were generated. As can be seen from the video above, the track used deliberately has many steep bends and **sudden** changes in the road's angle to see how the agent performs in a complex environment. The policy that the agent has learnt is **robust** and the agent ensures that the car remains in its lane regardless of the unpredictable changes in the road's angles.[16].
There is, however, a notable issue with the current solution: the agent's movements are very **'jerky'** i.e. they are **both** very numerous and not smooth. This might not seem like a huge issue in simulation, however, too many jerky movements will harm the **physical components of a real car** and would **reduce the lifespan of these components**. Furthermore, this would not lead to a smooth experience for any humans in the car.

3.2 Red Traffic Light

3.2.1 Introduction

In the previous section, the agent was successfully trained on the Lane Control task. This chapter documents details about training the agent on a more complex behaviour, namely stopping at red traffic lights. The Lane Control behaviour was 'simpler' in the sense that the behaviour was mostly invariant: no matter where the agent was, the agent had to decide on the best steering angle to maximise the reward. However, the same is not true for the traffic light behaviour. This behaviour can be broken down into two distinct parts:

1. When there is no traffic light visible, i.e. we are outside the sphere of influence of a traffic light, the aim is to maintain a constant speed and to prioritise smooth throttle changes over abrupt and erratic ones.

2. When a traffic light is visible, the aim is to slow down while **approaching** the traffic light and then come to a halt when the agent is close to it.

Another major difference between the previous learning task and this one is the observation space: due to the nature of lane control, the agent was able to converge to a good policy using a lower dimensional representation of the observation. However, this cannot be done for the

[14]This video is also available in the repository provided under **progress_notebooks/saved_videos/lane-control-before-training.mp4 .**
[15]This video is also available in the repository provided under **progress_notebooks/saved_videos/lane-control-after-training.mp4 .**
[16]Just using a straight road when evaluating the agent would not have represented the real world accurately.

Figure 3.11: Traffic Light

current learning task, as the agent has to be able to learn to stop at traffic lights **irrespective of where it sees a traffic light**. Due to this constraint, the observations space will have to be larger and the agent will have to utilise spatial relationships between pixels in the image i.e. the image cannot be flattened.
Finally, the reward distribution for this learning task is not symmetric: the reward function has higher rewards for correctly slowing down and stopping at red traffic lights whereas the magnitude of the rewards for maintaining a constant speed are far lower. Due to this, using the same DDQN architecture we used from the previous section is not feasible.

3.2.2 Current Double Deep-Q Network Limitations

The DDQN reinforcement learning algorithm has several limitations that hinder its ability to **quickly** train on more complex environments and tasks. In order to remedy this, two more variants of Deep-Q Networks were implemented

1. Duelling Deep-Q Network
2. Prioritised Experience Replay Deep-Q Network

3.2.3 Duelling Deep-Q Network

The Duelling DQN utilises the fact that the Q-value function can be represented as the sum of:

1. The value of the current state (is the current state a good state to be in?). This is represented by the Value Function $V(s)$.
2. The advantage of an action, where this states the advantage of a carrying out a certain action over the other actions in the state. This is represented by the function $A(s, a)$.
3. Subtracted by the mean value of the advantage function for all actions in the state.

The sum of the above two functions gives us the Q-value function, $Q(s, a)$:

$$Q(s, a) = V(s) + A(s, a) - mean\_advantage\_for\_all\_actions\_in\_state$$

This decouples the learning process, allowing the agent to learn the value of each state independent of the action taken and vice-versa for the value of an action in a certain state.

3.2.4 Prioritised Experience Replay

Another major of the major limitation of the Double DQN[17] training algorithm, is in the method used to sample experiences from the replay buffer. The replay buffer utilises **random uniform sampling** when deciding what experiences to sample to train on. This is not a big issue for environments where the state space is not very large or for environments with little variation, for example, grid-worlds or the game of pong etc.
In our steering control task, the state space was limited due to the small width and height of the input black and white image. There was also not a great amount of variation in the environment from the perspective of what the agent observed about the lanes: regardless of how steep the bends in the road were, the variation in the transformed observation's feature space was relatively limited.
However, to teach the agent more complex behaviours the following need to be considered:

- The input image will be larger.
- The image will need to be either a greyscale (luminance) or colour (RGB) image, as the agent is unlikely to learn to detect a traffic light just from a lower dimensional image
- The environment varies a lot, hence, the optimal action to take will vary between different parts of the environment. For example, when the traffic light is not visible the optimal action is to maintain a constant speed that is both high but within the speed limit. However, when a traffic light is visible, the car will need to learn to do the converse, namely decelerating and coming to a halt at the red traffic light.

One variant of the Deep-Q Network utilises Prioritised Experience Replay. In this, a priority is associated with each experience based on how much information that experience tells us about the environment (positive or negative). This priority is then used as the probability of each sample being sampled for training from the Replay Buffer. The above modification seems insignificant at first sight, however, **it greatly increases the performance of the RL agent**. By sampling more important experiences (positive or negative) more often to train on than other experiences, the agent converges to a good policy more quickly.

3.2.5 Traffic Light: Observations for the Agent

The observation for this task is not heavily transformed: it is converted from a three channel RGB image to a single channel greyscale image. Other transformations are not required due to the use of Convolutional Neural Networks for this solution (Section 3.2.6). The observation space, hence, is just transformed from (120, 160, 3) to (120, 160). This reduces the complexity of the task for the agent while all the salient features in the image are still visible: no **structural** information is lost due to this transformation. Figure 3.12 shows a visual representation of this.

[17] Note that the abbreviation DDQN is not used here to avoid confusion between the similarly sounding Duelling DQN and Double DQN

Figure 3.12: Deep Q-network architecture for Traffic Light behaviour

3.2.6 Traffic Light: Prioritised Experience Replay DQN Architecture

The Deep Neural Network that approximates the Q-values for this task is more complicated than the architecture used for steering control. Here are a few reasons for this:

1. It is not easy to define the shape of the traffic light, especially such that it is invariant to transformations and distortions (perspective distortion etc). Furthermore, using traditional Computer Vision techniques for this task would lead to very "brittle" solutions, which would break by simple light changes, rotations etc. A Convolutional Neural Network (CNN) based solution is robust to transformations and distortions (to a certain extent).

2. A CNN based solution is computationally heavy while training, however, it is very efficient on **inference** (i.e. when using the **previously** trained model).

The following architecture made up of **three** distinct **chained** blocks is used:

1. The greyscale observation image is input directly into the **Convolutional Block**.

2. This is then connected to a **Fully Connected Block.**

3. Next we have two blocks that calculate the State Value and Action Advantage separately. Finally these two values are combined together as explained in 3.2.3.

Convolutional Block

Figure 3.13 shows the different layers inside this block. The transformed observation only has one channel: this is visually represented by the **Input Block** in Figure 3.13.
The next block receives the transformed observation as input for the first Convolutional Layer of the Deep Neural Network (DNN). This is a 2D convolutional layer (Conv2d) and it outputs 64 channels from the 1 channel input it receives. This is followed by a Max Pooling Layer (MaxPool2D) which allows the DNN to extract the most important information from the previous layer while also allowing the model to generalise more [23, 24]. After the MaxPool2d layer we have a ReLU activation layer which helps the model to learn more efficiently, converge more quickly and have a higher accuracy [25].
This sub-block (i.e. Conv2d followed by MaxPool2d followed by ReLU) is then repeated again: the second sub-block will look for higher level patterns on features in the image that the first sub-block extracted (feature extraction): this property of the above structure is one of the

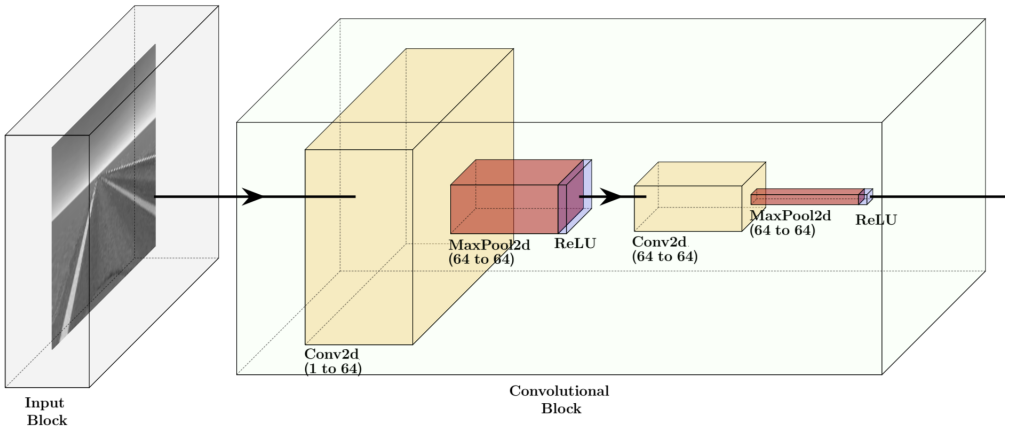

Figure 3.13: Convolutional Block Visualisation

primary reasons for its usage here [26]. Note the huge reduction in the dimensionality from the input observation to the final ReLU unit which allows training to be a lot more computationally efficient.

Fully Connected Block

This block receives the output from the Convolutional Block as its input. It first **flattens** this input into a one dimensional vector [18]. This flattened input is then processed by this block. Figure 3.14 visualises the different units inside this block.

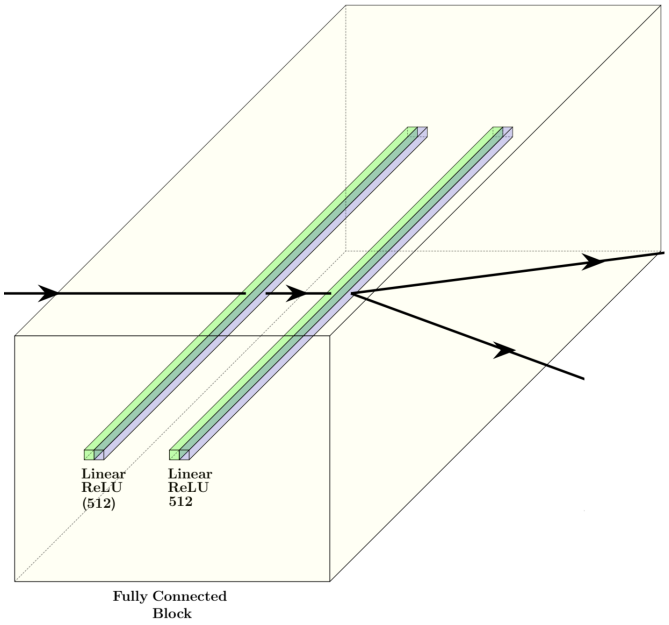

Figure 3.14: Fully Connected Block Visualisation

The Linear-ReLU units allow the DNN to understand higher level relationships on features extracted from the previous Convolutional Block. Note that the width and height of the blocks

[18] This is necessary for the Fully Connected Linear Layers.

have been increased for aesthetic purposes, however, the **depth** is accurate (i.e. how far the Linear layer's representation goes into the screen).

State Value and Action Advantage Blocks

As explained in 3.2.3, the architecture will estimate the State Value and Action Advantage separately. This is carried out by the next two separate block in the architecture. This is shown in Figure 3.15, however, note that these blocks are not linked: these are two separate blocks at the same 'level' in the DNN.

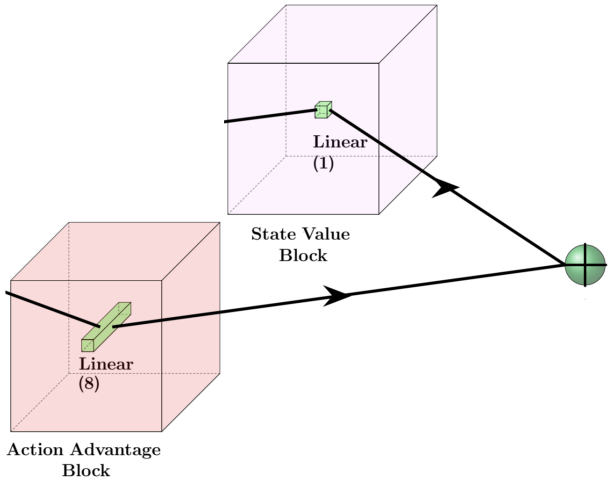

Figure 3.15: State Value and Action Advantage Blocks Visualisation

Note that these blocks neither have multiple repeated units nor non-linear activations (like the Linear-ReLU units before) **by design**. The final computational step is represented by the '+' symbol. This carries out the computation as explained in equation 3.2.3.

Visualisation showing all the different blocks combined

3.2.7 Traffic Light: Actions Space

The action space for this task consists of 8 discrete actions from the set of values $\{0.0, 0.01, 0.03, 0.04, 0.06, 0.07, 0.09, 0.1\}$. This corresponds to the **throttle** of the car that the

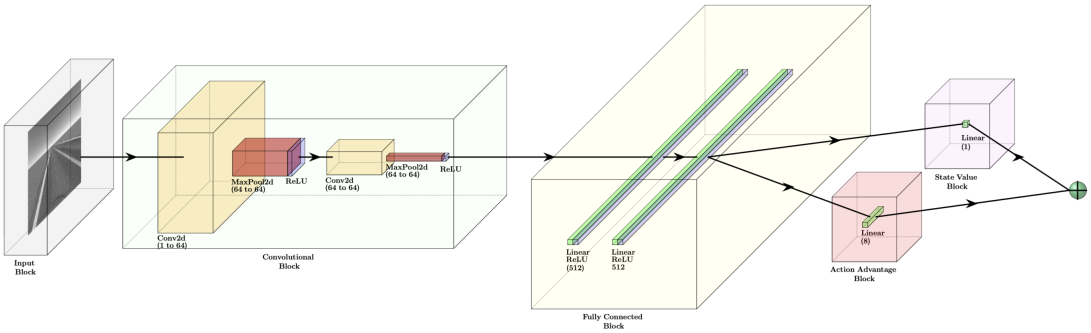

Figure 3.16: Full Traffic Light Deep Neural Network

agent controls [19]. The granularity of the action space is deliberately low to prevent 'jerky' movements [20].

3.2.8 Traffic Light: Reward Function

The reward function used for this task is a lot more complicated than the one used in Section 3.1.6. The following has more details about the reward function[21]:

1. The distance to the traffic light is **normalised**. Normalising the distance aids the training process and allows more flexibility. by dividing the distance of the closest traffic light with an **upper bound distance**. This is a hyper-parameter and is set to a constant value at the start of the training process and controls the **sphere of influence** of the traffic light. Otherwise the agent would have been influenced by traffic lights even **if they were very far away**, encouraging the agent to stop the car far in advance.

2. Increment or decrement the reward by a constant value, based on the distance to the traffic light and the current speed. Based on the distance to the traffic light, the desired behaviour might be for the car to:

   - Decelerate but continue moving (pre-emptive deceleration)
   - Decelerate to a complete stop at a red traffic light

3. A penalty is incurred for high speeds closer to the traffic light. The penalty is made by multiplying a **constant**, the **current speed** and **one minus the distance to the traffic light**.$1 - \text{distance\_to\_traffic\_light}$ is used because this will cause the penalty to increase the closer the agent gets to the traffic light and the speed remains high.

4. A reward is incurred based on whether the agent is decelerating when under the influence of a traffic light (1). This provides the agent an incentive to slow down when it is approaching the traffic light.

5. If the agent is not within the influence of the traffic light (1), the agent is penalised for not moving. After training for a few epochs without this, the agent decided that the best action to take was to not move at all because that incurred the least amount of penalty. This term encouraged the agent to not stop when it is far away from the traffic light.

6. Finally, the reward is smoothened by using $\exp^{\text{reward} \,/\, 1000}$. The division by a 1000 scales down the range of the reward function, as otherwise this was very high.

3.2.9 Traffic Light: Qualitative Evaluation

The video at this link shows the agent after it has been successfully trained for this behaviour[22]. The video[23] shows that the agent:

[19] It is important to note that the throttle values are related to the velocity, however, the velocity of a car is determined by other factors as well (drag, bends, tire conditions etc).

[20] Using a non-linear scale for the throttle value steps is likely to be even better[27].

[21] For a more detailed look at this, please look at the corresponding Jupyter Notebook in the code repository.

[22] This video is also available in the repository provided under the **progress_notebooks/saved_videos/stop-at-red-traffic-light.mp4.**

[23] The timescale of the video is higher than normal. This was done to speed up training and for collecting quantitative metrics in a shorter amount of time.

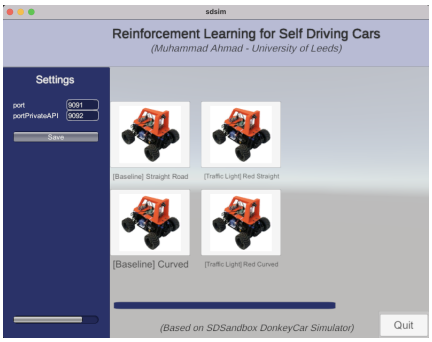

Figure 3.17: Modified simulator used during quantitative evaluation

1. Keeps moving the car when the traffic light is far away.
2. Slows down the car when closer to the traffic light.
3. Comes to a complete halt at the red traffic light.

Qualitative evaluation can be a useful tool to assess a solution, however, it **needs** to be taken into context with **quantitative evaluation**. Quantitative metrics can show issues that cannot be seen in qualitative evaluation and vice versa. Section 3.3 evaluates the fully trained agent using quantitative metrics.

3.3 Fully Trained Agent: Quantitative Evaluation

3.3.1 Introduction

The agent performs quite well on visual inspection on the Lane Control and Red Traffic Light environments (3.1.7 and 3.2.9). This chapter validates the agent's performance using **quantitative metrics**.

3.3.2 Evaluation Methodology

During quantitative evaluation, the agent tested on **three different custom environments** made for this project. The video at this link[24] shows an example of how the metrics were collected using the modified simulator and Jupyter Notebooks. These were[25]:

1. **[Baseline] Straight Road:** This environment has a finite[26] **straight** road and no traffic lights. This scene tests how well the agent performs on **steering control**. The agent should continue travelling down the road until the road ends [27].

[24] This video is also available in the repository provided under the **progress_notebooks/saved_videos/metric-collection-process-simulator-video.mp4.**

[25] These were modified versions of the original scenes, along with custom Unity C# scripts for this project. In the repository, the directory at the path **donkeycar-simulator-code/sdsandbox-master/sdsim/Assets/INDIVIDUAL-PROJECT** has these files. Under **INDIVIDUAL-PROJECT** default scenes (i.e. those provided externally) are under **INDIVIDUAL-PROJECT/Scenes/default_scenes** while those that have been made for this project are under **INDIVIDUAL-PROJECT/Scenes/custom_scenes**. Original, modified and custom scripts are within the **INDIVIDUAL-PROJECT/Scripts** directory.

[26] To aid metric collection.

[27] At this point the episode is terminated.

2. **[Baseline] Curved:** This environment has a finite **curved** road to assess the agent's performance when there are **steep changes in the road angle** [28].

3. **[Traffic Light] Red Straight:** This environment has a finite **straight** road. This scene tests how well the agent performs when there is a traffic light visible. The agent should: (a) Continue travelling down the road if the traffic light is far away. (b) Slow down when approaching the red traffic light. (c) Come to a full stop at the traffic light.

3.3.3 Collected Metrics

Table D.1 contains detailed descriptions of the different metrics collected for quantitative evaluation. The limitations of these metrics are also discussed in this table.

3.3.4 Reliability of Collected Metrics

In order to reduce the effect of **sample bias** on the metrics collected, the agent was evaluated for **1000 episodes** on each of the environments listed in Section 3.3.2; a large number of 'trials' increases the reliability and confidence in the metrics collected [29] and crucially on **any insights derived from these** [30].
The same python script was used to collect all the metrics to allow these results to be verified i.e. for reproducibility.[31] A detailed description and limitations of these metrics was also provided to allow **critical analysis** of the results and insights stated in this report.

3.3.5 Correlations between selected metrics for all environments

A correlation plot of the different collected metrics is shown in Figure 3.18. The following insights can be gained from this figure [32]:

1. The plots of **mean_episode_reward** with **episode_mean_speed** and **episode_mean_throttle_value** show a clear **negative linear relationship**. This suggests that the trained agent has learnt that both higher speeds and throttle values lead to lower rewards. The reasoning used here is that if the agent had **not** learnt this relationship, the scatterplot of these metrics would be randomly or uniformly distributed. Correlation plots can only show that there exists a relationship between **two variables** not **why the relationship exists** (causation), so some caution is needed when interpreting them. However, the strong linear relationships in this scenario do seem to suggest that the agent has successfully learnt these relationships..

2. A strong linear relationship can be seen between **episode_mean_speed** and **episode_mean_throttle_value,** which indicates that the agent has learnt that a high

[28] Such steep changes in angles are **not** very common in the real world, although they are prevalent in race tracks.
[29] Law of large numbers from statistics.
[30] Ideally, the agent would have been evaluated for even more trials and on even more diverse environments, however, time constraints prevented this from being done.
[31] The notebook at **progress_notebooks/6-See-the-agent-in-action-and-gather-metrics.ipynb** documents this process.
[32] Higher resolution version of these figures are provided in the repository in the directory at the path **progress-notebooks/resources/img.**

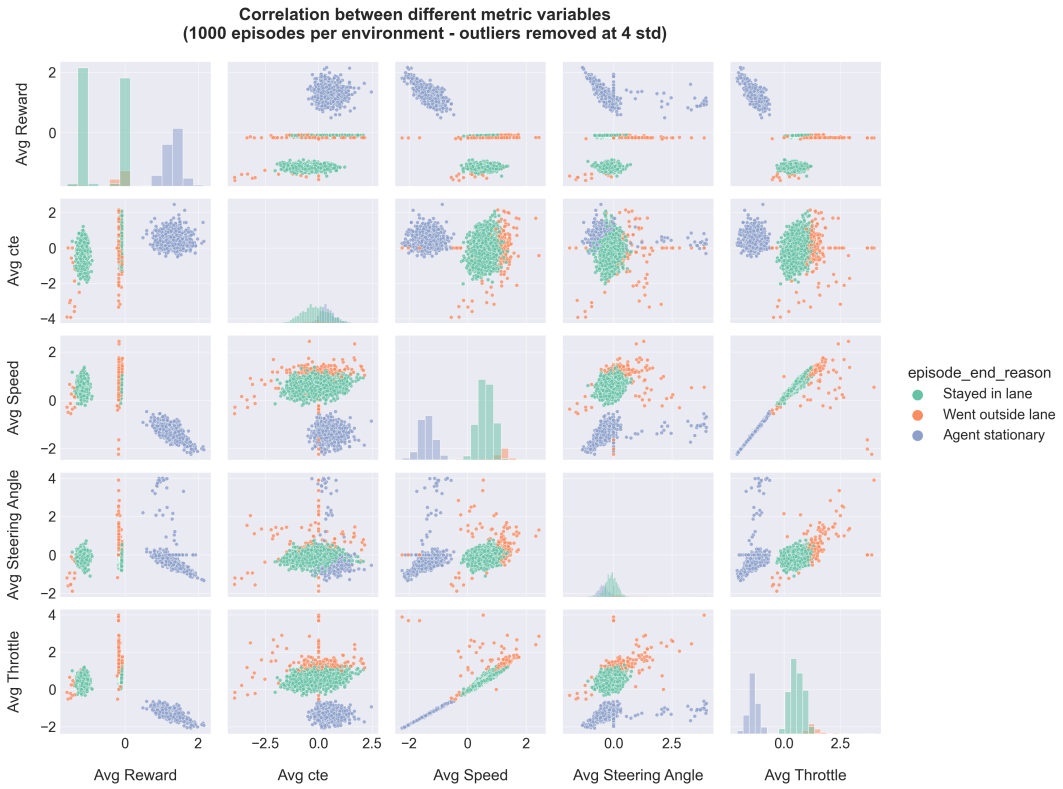

Figure 3.18: Correlation plot of **normalised** metrics with **outliers** further than 4 standard deviations from the mean removed.

throttle value leads to high speeds[33].

3. The plots of **episode_mean_steering_angle** with **episode_mean_speed, episode_mean_throttle_value** and **episode_mean_reward** are very informative:

   (a) The agent seems to prefer lower throttle values when steep steering angles are required [34], most likely when there are steep bends in the road. This seems like a good strategy, as a lower throttle leads to a lower speed which would give the agent **more control over the car at steep bends**, making it more likely to receive a higher reward.

   (b) The agent seems to go out of lane at steeper bends, where there is need for a higher steering angle coupled with a higher throttle value.

   (c) The episodes where the agent does **not** stop at a traffic light are where the agent had a **higher** mean throttle value. This suggests that failing to stop at red traffic lights is also due to the agent not slowing down adequately on approach to the traffic light instead of just due to the steep bends. Nevertheless, steep bends could **indirectly** lead to this behaviour: at steeper bends the visibility of the road is lower, which would cause the agent to detect the traffic light far later than it would on a straight road [35].

[33] The agent can directly control only the throttle not the speed.

[34] The **purple** agent stationary episodes are more spread out for both the mean throttle value and mean reward plots **in this column**.

[35] Further investigation is required for this to determine what is the major cause of failure: failure to adequately slow down or lower visibility at steeper bends.

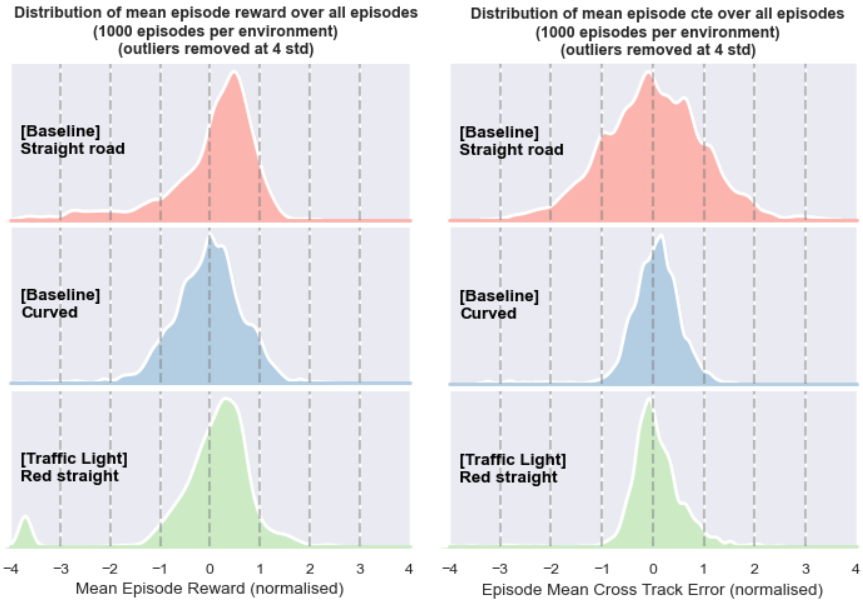

Figure 3.19: Ridge plot of selected normalised metrics for different environments. Each different coloured distribution is for a different test environment. Each figure shows the distribution of a different metric.

3.3.6 Ridge plots of selected metrics on different environments

Figure 3.19(a), shows the distribution of the **normalised average reward** per episode for the all the environments. The reward distributions are **skewed** towards positive rewards, which is an indicator of good performance on all environments. For the environment with a traffic light on a straight road (**[Traffic Light] Red Straight**), the small density near -4 is most likely due to penalties incurred for not stopping at the traffic light. Figure 3.19(b) also shows that the mean cross track error distributions for all environments have a mode around 0. This is a positive indicator of the agent's performance, as a cross track error closer to 0 indicates that the agent did not deviate a lot from the centre of the road.

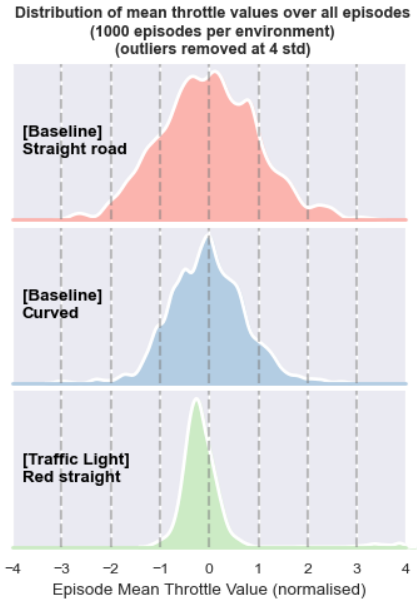

Figure 3.21: Ridge plot of selected normalised metrics for different environments (continued).

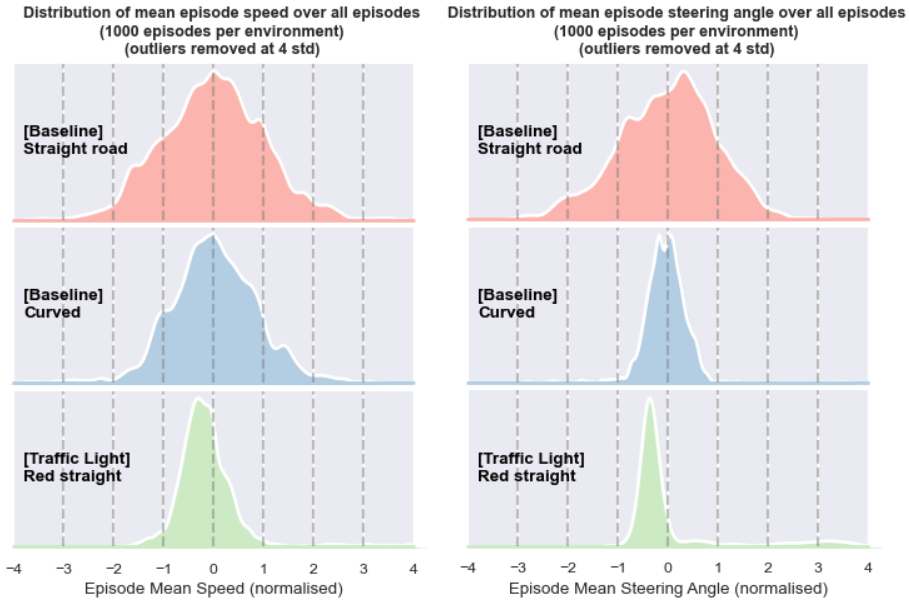

Figure 3.20: Ridge plot of selected normalised metrics for different environments (continued).

3.3.7 Summary

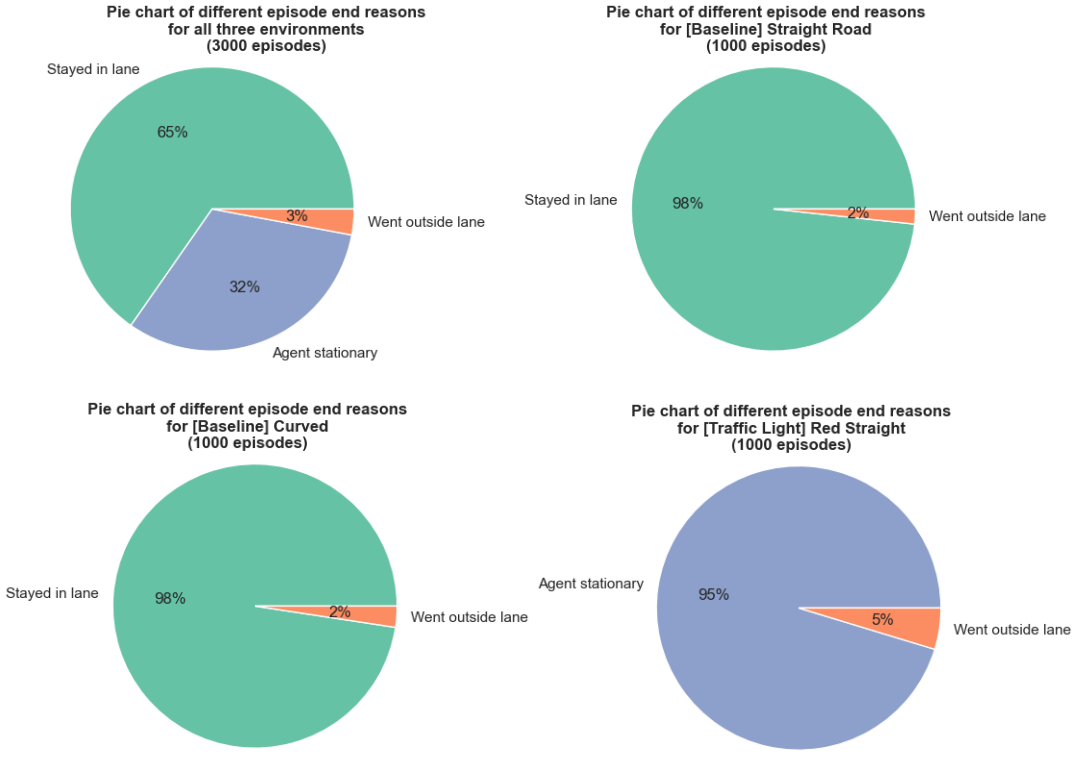

Figure 3.22: Pie charts showing episode end reasons for different environments.

As can be seen in Figure 3.22(a), out of 3000 episodes[36] 97% of episodes terminated in a successful state [37]. Constraining this aggregate visualisation by the different environments provides deeper insights into the agents strengths and limitations:

- The agent performs exceptionally in the **[Baseline] Straight Road** (Figure 3.22(b)) environment, as the agent stayed in the lane for 98% of the 1000 evaluation episodes. This provides evidence for the robustness of the agent's 'Stay in lane' behaviour.
- Similarly, on the **[Baseline] Curved (** (Figure 3.22(c)) environment the agent stayed in lane for 98% of the 1000 episodes. This environment had many steep bends to challenge the agent's lane control behaviour, therefore, a failure rate of only 2% demonstrates that the agent can successfully handle complex environments with very steep bends in the road.
- For the stopping at red traffic light (**[Traffic Light] Red Straight**) environment, the complex reward function specifically gave the agent rewards and penalties to slow down on approach and finally stopping at the red traffic light. Figure 3.22(d) shows that the agent successfully stops at the traffic light in 95% of the 1000 episodes. Figure 3.19(a), also showed that the reward distribution for this environment is skewed towards positive rewards. Both of these show evidence that the agent has been successfully trained for this complex behaviour.

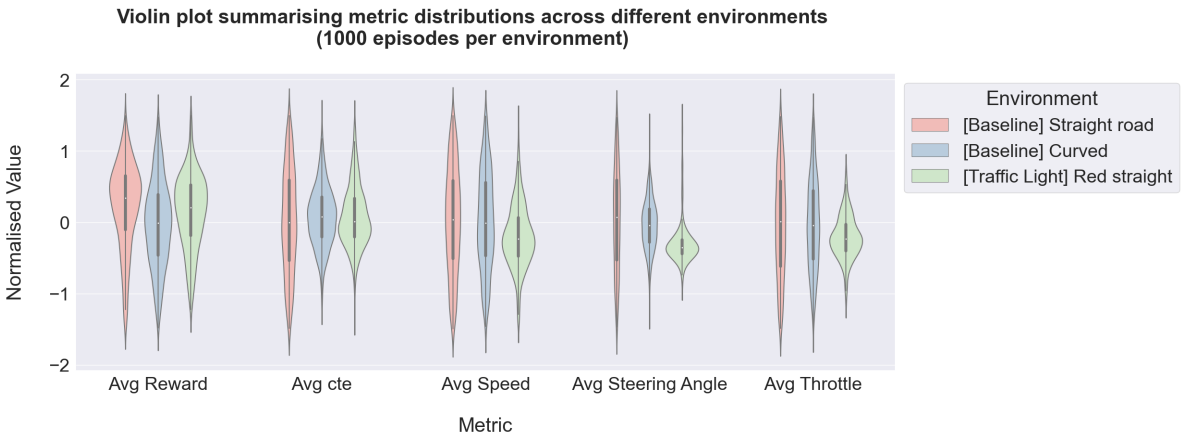

Figure 3.23: Summary visualisation: violin plot showing the distribution of different metrics for the three environments.

On the whole, the agent performs effectively on the different test environments. Table D.3, calculates summary statistics for all episodes over the different environments. Appendix D provides tables with summary statistics for the different evaluation environments. Figure 3.23, uses a violin plot to summarise this data visually [38]. The following insights can be gained from these statistics:

[36] Episodes are also referred to as trials in some academic literature.
[37] Here 'Stayed in lane' and 'Agent stationary' are the successful episode end states.
[38] The violin plot has a **box and whisker plot** inside each 'violin' to show the quartiles while the distribution of values are shown by the 'violin'.

- The standard deviation of the average rewards is a bit high. Ideally, the skew of the reward distribution would have been towards even more positive rewards and the distribution would have a lower spread.
- The agent consistently managed to keep the average cross track error across the episodes low, as the standard deviation is very small relative to the range for this metric. This implies that the agent on the whole performs **very nicely on the steering control task**.
- Mean episode speed, mean episode steering angle and mean episode throttle value all have a **relatively low standard deviation**, implying that the agent is applying a **consistent learnt policy**. Otherwise, these standard deviations would have been quite high.
- Finally, the standard deviation of the closest traffic light distance [39] on episode termination is **very low relative to the range for this metric**: this shows that the agent consistently stops at the red traffic light. Also note how small the minimum value, implying that even in episodes where the agent overshoots the traffic light, it does not travel quite far.

[39] This metric was not shown in the Figure 3.23 because data was only present for **[Traffic Light] Red Straight** environment. The other environments did not have traffic lights. Please refer to Table D.3 for the data referred to here.

Chapter 4

Discussion

4.1 Conclusions

On the whole, the implementation provides evidence that Reinforcement Learning with Deep Learning can be very successful for autonomous driving tasks. Section 3.3 shows that the final trained agent is performs very well on both the lane control and stopping at red traffic light behaviour. Figure C.1, shows that the project initially aimed only to focus on the lane control behaviour, however, the project was also able to train the agent successfully on the more complex traffic light behaviour.[1]

4.2 Ideas for future work

The following list shows some ideas on how to extend this project in the future:

- Train the agent on more complex car control behaviours such as:
  - Do not stop at Green Traffic Lights
  - Avoid static obstacles such as parked cars
  - Avoid dynamic obstacles such as pedestrians or moving cars
  - Night time driving
- Make a small robotic prototype using Raspberry pi etc. to test the agent in the real world. DonkeyCar simulator provides a similar interface for this for a physical toy car.
- Explore more advanced Reinforcement Learning Architectures.
- Explore Inverse Reinforcement Learning techniques to train the agent on more human like behaviours when driving cars.
- Implement the alternative approaches mentioned in 1.2.8. Compare and contrast this with the Reinforcement Learning approach used in this project.

[1] Critical evaluation of the implementation is provided in the previous chapter.

Index

Bibliography

[1] Schmidhuber, J. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[2] Morales, M. *Grokking Deep Reinforcement Learning*, chapter 2, pages 36–37. Manning Publications, 2020.

[3] Mnih, V., Kavukcuoglu, K., Silver, D., A. Rusu, A., Veness, J., G. Bellemare, M., Graves, A., Riedmiller, M., K. Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. Human-level control through deep reinforcement learning. *Nature*, 2015.

[4] Morales, M. *Grokking Deep Reinforcement Learning*, chapter 2, pages 58–59. Manning Publications, 2020.

[5] Lin, L. Reinforcement learning for robots using neural networks. 1992.

[6] Attaiki, S. The genesis of beating atari games. 01 2019.

[7] Hasselt, H. Double q-learning. In Lafferty, J., Williams, C., Shawe-Taylor, J., Zemel, R., and Culotta, A., editors, *Advances in Neural Information Processing Systems*, volume 23. Curran Associates, Inc., 2010.

[8] Kaelbling, L.P., Littman, M.L., and Moore, A.W. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4(237-285), 1996.

[9] Morales, M. *Grokking Deep Reinforcement Learning*, chapter 4, page 102. Manning Publications, 2020.

[10] baeldung. *Epsilon-greedy q-learning*. [Online]. 2000. [Accessed 12 February 2023]. Available from: https://www.baeldung.com/cs/epsilon-greedy-q-learning.

[11] Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, Jan 2016.

[12] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. *Playing atari with deep reinforcement learning*, 2013.

[13] Sethy, H., Patel, A., and Padmanabhan, V. Real time strategy games: A reinforcement learning approach. *Procedia Computer Science*, 54:257–264, 2015. Eleventh International Conference on Communication Networks, ICCN 2015, August 21-23, 2015, Bangalore, India Eleventh International Conference on Data Mining and Warehousing, ICDMW 2015, August 21-23, 2015, Bangalore, India Eleventh International Conference on Image and Signal Processing, ICISP 2015, August 21-23, 2015, Bangalore, India.

[14] Jiang, Z., Xu, D., and Liang, J. *A deep reinforcement learning framework for the financial portfolio management problem*, 2017.

[15] Polydoros, A.S. and Nalpantidis, L. Survey of model-based reinforcement learning: Applications on robotics. *Journal of Intelligent & Robotic Systems*, 86(2):153–173, May 2017.

[16] Kiran, B.R., Sobh, I., Talpaert, V., Mannion, P., Sallab, A.A.A., Yogamani, S., and PÃ©rez, P. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 23(6):4909–4926, 2022.

[17] International, S. Human-level control through deep reinforcement learning. *SAE Standard*, 2021.

[18] International, S. Sae levels of driving automationâ¢ refined for clarity and international audience.

[19] of Toronto, U. *Self-driving cars specialization: Introduction to self-driving cars.*

[20] Hussein, A., Gaber, M.M., Elyan, E., and Jayne, C. Imitation learning: A survey of learning methods. *ACM Comput. Surv.*, 50(2), apr 2017.

[21] DeepMind. Discovering novel algorithms with alphatensor. 2022.

[22] Saksena, S.K., B., N., Hegde, S., Raja, P., and Vishwanath, R.M. Towards behavioural cloning for autonomous driving. In *2019 Third IEEE International Conference on Robotic Computing (IRC)*, pages 560–567, 2019.

[23] Krizhevsky, A., Sutskever, I., and Hinton, G.E. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, page 1097â1105, Red Hook, NY, USA, 2012. Curran Associates Inc.

[24] Zeiler, M.D. and Fergus, R. *Visualizing and understanding convolutional networks*, 2013.

[25] He, K., Zhang, X., Ren, S., and Sun, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034, 2015.

[26] Simonyan, K. and Zisserman, A. *Very deep convolutional networks for large-scale image recognition*, 2015.

[27] Quiter, C. *Smooth operator: Human-like self-driving with reinforcement learning.*

[28] Sutton, R.S. *Reinforcement learning : an introduction.* Adaptive computation and machine learning. MIT Press, Cambridge, Mass, 1998 - 1998.

Appendix A

Self-appraisal

A.1 Critical self-evaluation

Theoretical architectures (DDQN, PER etc.) were successfully implemented as part of this project and the final trained agent is very successful at the two behaviours it was trained on. The agent can successfully handle steep bends in the road, without going outside the lane while successfully stopping for the red traffic light.
However, the environments used for this project do not look very realistic. This limitation was not addressed due to resource constraints. Even though the agent was evaluated for 3000 episodes, ideally the agent should have been evaluated for a greater number of episodes to ensure reliability of the metrics. The agent should have been evaluated on an even larger number of environments to stress test the implementation.

A.2 Personal reflection and lessons learned

The workload in the final semester was very high. As such, I attempted to follow a consistent routine and contribute frequently to the repository. The decision to actively write the final report along with the implementation was a very good idea, as this alleviated pressure near the end. One lesson learnt is that in the future, less time should be spent on background research and more devoted to the write up.

A.3 Legal, social, ethical and professional issues

A.3.1 Legal issues

Autonomous Driving vehicles poses significant challenges for the legal community For example, it needs to be decided in law who bears the liability and has to cover the potential costs from a crash involving autonomous vehicles. Major car manufacturers have provided disclaimers about the potential dangers of their new autonomous technologies, pushing liability on the consumer. However, as the technology develops further towards Level 5 automation, can the driver in the vehicle be held legally responsible **even though they have a limited decision making capacity** in Level 4 and Level 5 autonomous vehicles?
Conversely, placing the liability solely on the shoulder of those who built the autonomous vehicles would deter investments in this field and cause these autonomous vehicles to be very expensive (to offset the potential damages from the producer having full liability for any damages).

A.3.2 Social issues

There is a lot of fear present in society currently about Artificial Intelligence in general with job displacements being a common fear. Without question, these new technologies will be highly disruptive for some occupations, however, these same technologies can increase productivity and create new industries and occupations. For example, during initial Information Technology 'revolution' there were fears of huge job losses in the short term. However, eventually these technologies led to increased productivity and the creation of totally novel industries (digital artists, game developers etc.)

A.3.3 Ethical and Professional issues

The autonomous vehicle industry is relatively new. As such there are unique and critical issues that need to be addressed, for example:

- There is an urgent need for:
  - standardisation in safety testing.
  - standardisation in professional training provided to employees who develop self driving cars. Continuous professional training needs to be provided to practitioners to prevent safety issues.
  - standardisation in the physical components used for these autonomous vehicles, so that the industry is more resilient to temporary economic shocks to the world economy .
  - optimise the global supply chain for this new industry, to ensure that the autonomous vehicles are affordable for all segments of society not just a select few.
- Monopolies should be prevented in this new industry as:
  - Healthy competition between rival enterprises is the fuel for innovation, without which the industry can stagnate.
  - An enterprise with a huge monopoly in this new market can charge extortionate prices for its products without challenge.

Appendix B

External Material

External:

- OpenAI gym
- PyTorch Lightning
- Unity (simulator)
- 3D models:
  - The car, procedurally generated road and the 'desert' plane were provided by DonkeyCar simulator (under MIT permissive license)

External source but **custom** modification made for the project:

- gym_donkeycar (permissive MIT license license)
  - The original Unity C# code was modified to allow custom values to be transferred to the python API (e.g. to transfer traffic light state and data to the python code to calculate the reward when training the agent)
  - The original Python API was modified to allow custom values to be transferred to the OpenAI gym environment

Appendix C

Further Figures

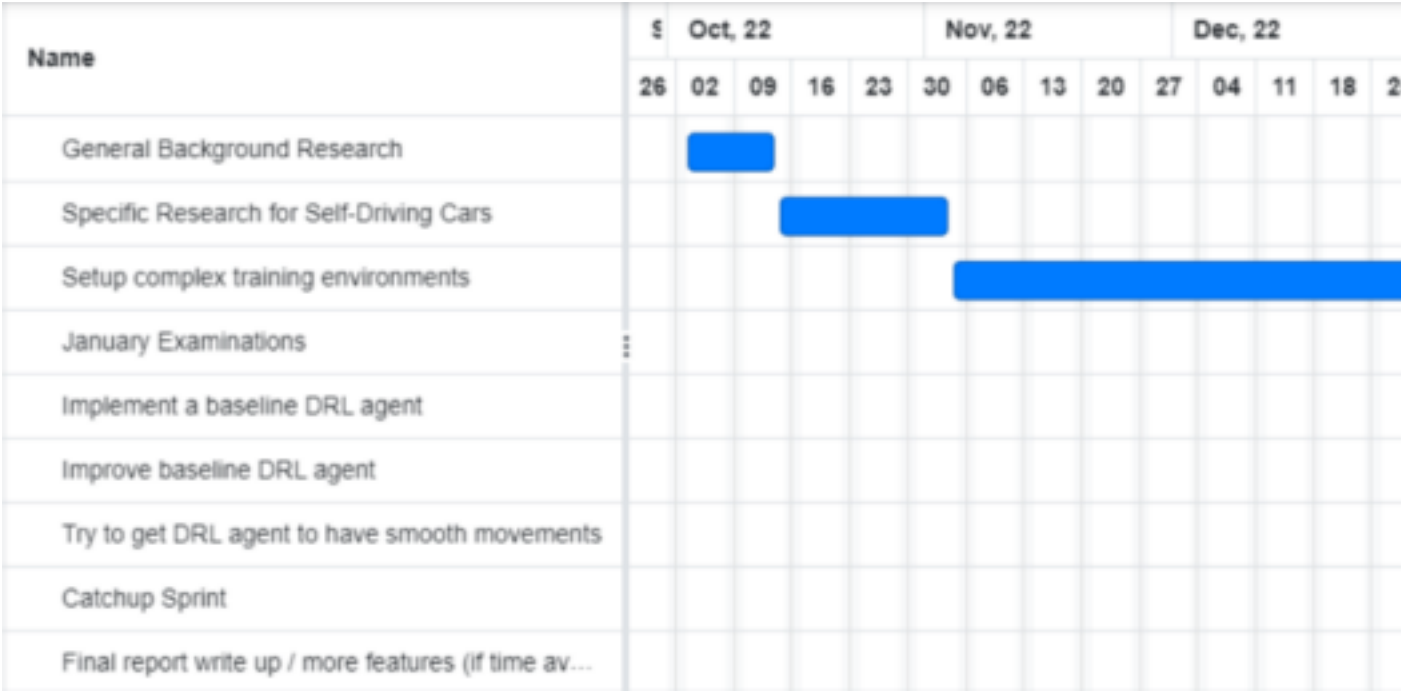

Figure C.1: Agile sprint plan.

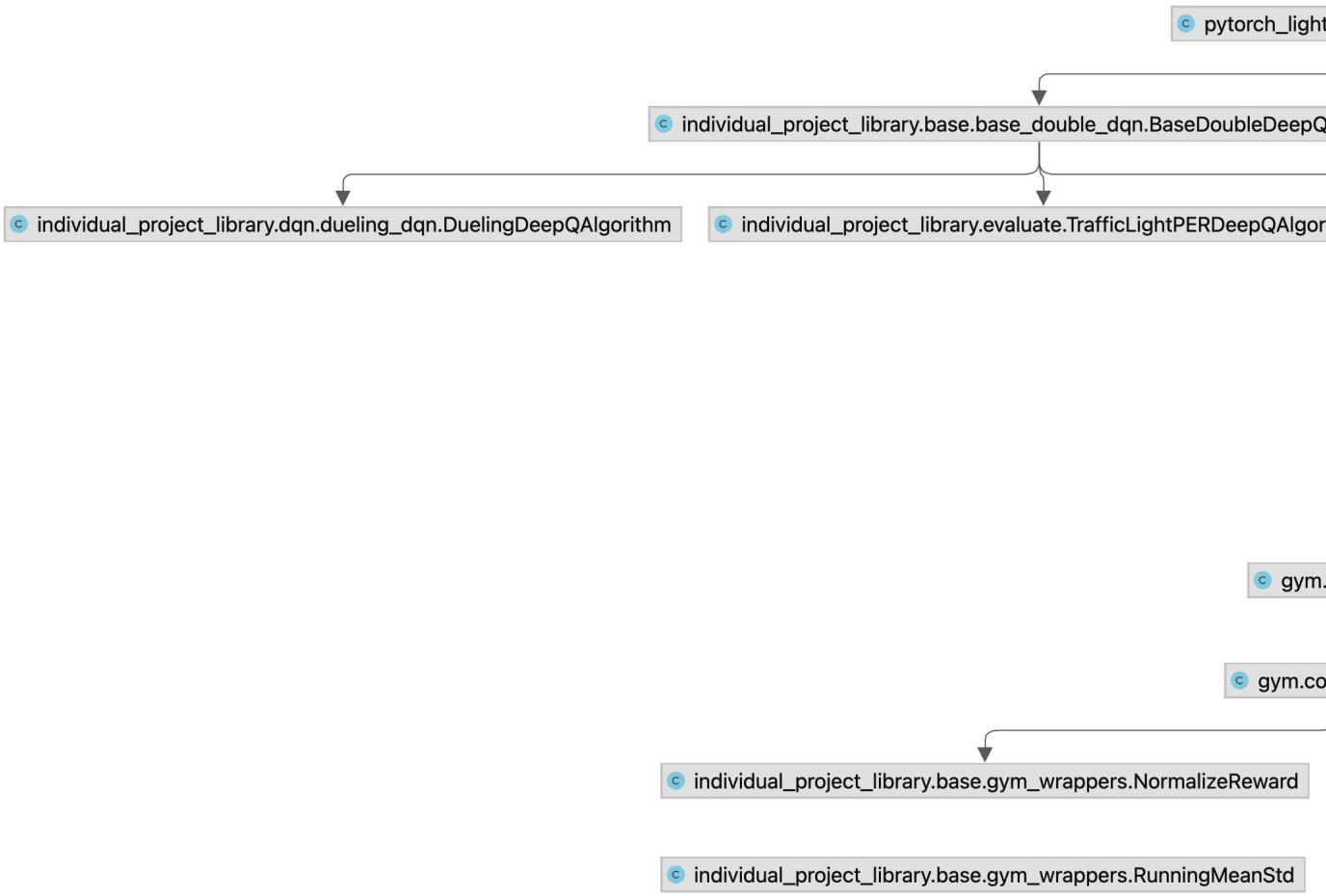

Figure C.2: Simplified high level overview of the Python code classes.

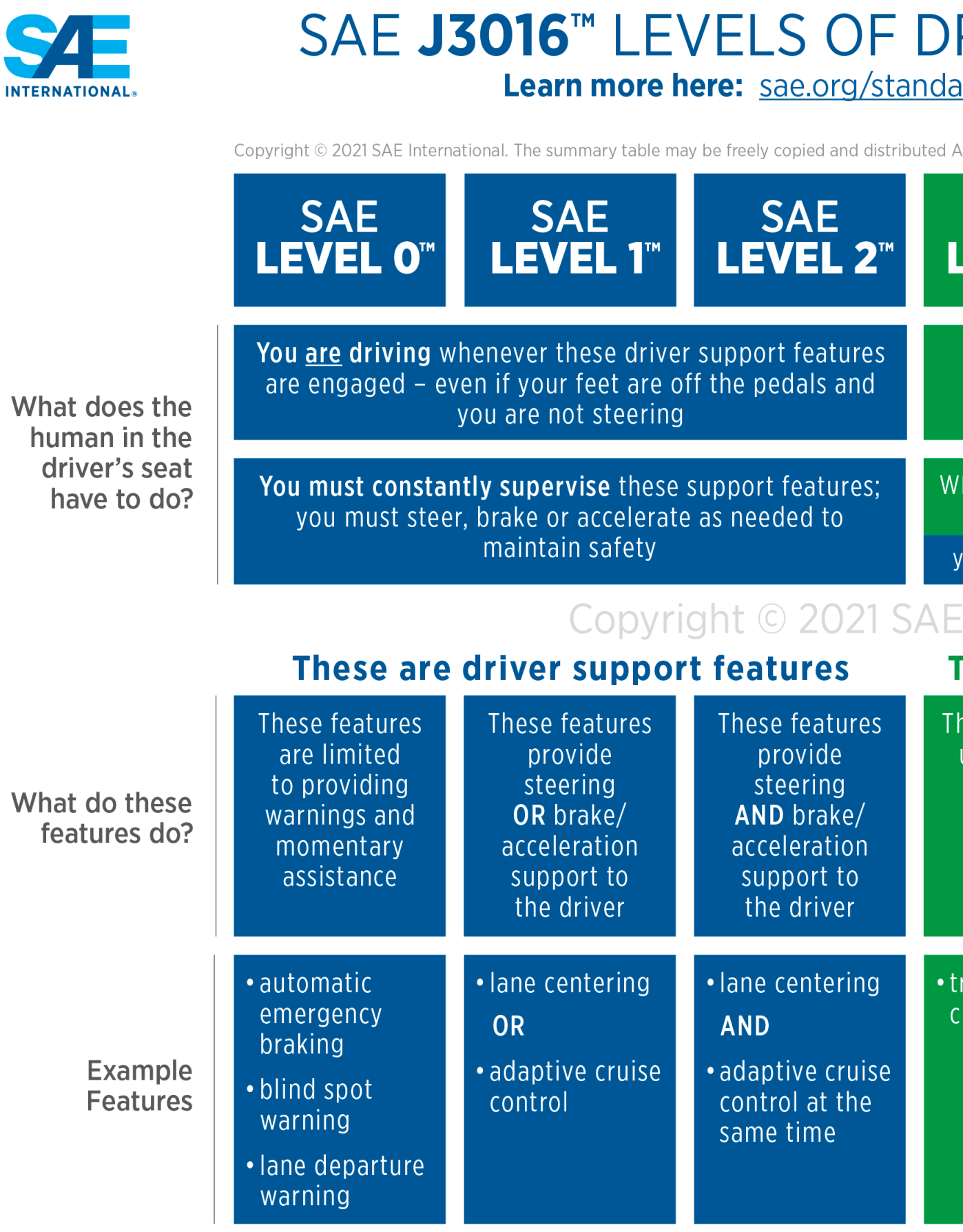

Figure C.3: SAE Levels of Driving Automation (link).

Figure C.4: Game of Go. DeepMind's AlphaGo performed amazingly at this game.

Appendix D

Tables

Metric	Description	Remark
steps in episode	Number of steps i.e. iterations in in the episode.	This refers to the number of times the agent carried out **inference using the trained model** in an episode. [1]
mean episode reward	The average reward at each step (D.1) of the episode.	This metric might **not** tell the full picture, as the reward distribution could be **skewed within the episode** e.g. more rewards could be gained near the end of an episode and none at the start. Due to time and resource constraints, however, it was decided not to address this limitation.
episode mean cte	The average **cross track error** (cte) at each step of the episode.	This metric shows how far the agent deviates from the centre of the track.
episode mean speed	The average **speed** of the vehicle at each step of the episode.	It is crucial to note that **speed is a scalar** whilst **velocity is a vector**. For this metric, the magnitude of the velocity i.e. speed was used. Velocity being a vector gives more information about how the speed is split between different directions. However, for this project it is sufficient to utilise **only the speed of the vehicle for evaluation**, as using a three dimensional vector velocity would add unnecessary complication to the evaluation stage without adding considerable value.
episode mean steering angle	The average **steering angle** of the vehicle at each step of the episode.	Steering angle is controlled directly by the agent

[1] According to Sutton, episodes are subsequences of the 'agent–environment interaction ... such as plays of a game, trips through a maze, or any sort of repeated interactions. Each episode ends in a special state called the terminal state, followed by a reset to a standard starting state or to a sample from a standard distribution of starting states. Tasks with episodes of this kind are called episodic tasks.' In a footnote, it is stated that 'Episodes are often called "trials" in the literature.' [28]

Metric	**Description**	**Remark**
episode mean throttle value	The average **throttle** of the vehicle at each step of the episode.	The agent controls the throttle value of the car and the throttle value affects the speed of the car, however, the throttle and speed of a car are not equivalent. There is a delay before the throttle prevails over the car which could be due to resistance, bends in the road, steering angle etc.
traffic light state	Metric value can be: 1. **empty:** no traffic light present in the scene. 2. **r:** closest traffic light was red when the episode terminated. 3. **g:** closest traffic light was green when the episode terminated.	Having other traffic light states allows for easy extension of the project for future work on more complex behaviours .
closest traffic light distance	This records the distance of the closest traffic light when the episode was terminated. The value is **not an average.**	This is not an **average** by design, as the average in this case would not provide any insights which could not be gained from the other metrics collected. This metric shows how close to the traffic light did the agent stop. A **negative** value indicates that the agent overshot the traffic light while a **positive** value indicates that the agent stopped before the traffic light.

Metric	**Description**	**Remark**
episode end reason	Metric value can be: 1. **'Stayed in lane':** This value indicates **successful** termination of a **Baseline** environment episode. 2. **'Went outside lane':** This value indicates **unsuccessful** termination of an episode, regardless of the environment. 3. **'Agent stationary':** This value indicates **successful** termination of a **Traffic Light** environment episode. 4. **'Did not stop at traffic light'**: This value indicates **unsuccessful** termination of a **Traffic Light** environment episode.	

Table D.2: Detailed description of collected metrics.

	mean episode reward	mean episode cte	mean episode speed	mean episode steering angle	mean episode throttle value	closest traffic light distance
count	3000.0	3000.0	3000.0	3000.0	3000.0	3000.0
mean	1.272	2.809	0.755	-0.104	0.071	0.047
std	6.589	0.082	0.161	0.194	0.017	0.069
min	-9.131	1.127	0.395	-0.470	0.036	0.000
25%	-5.960	2.779	0.562	-0.177	0.050	0.000
50%	0.662	2.819	0.828	-0.135	0.077	0.000
75%	8.804	2.848	0.874	-0.092	0.083	0.136
max	15.484	3.300	1.147	1.900	0.147	0.242

Table D.3: Summary statistics for metrics over all environments (rounded to 3.d.p).

	mean episode reward	mean episode cte	mean episode speed	mean episode steering angle	mean episode throttle value
count	1000.0	1000.0	1000.0	1000.0	1000.0
mean	0.641	2.826	0.882	-0.106	0.085
std	0.107	0.046	0.043	0.052	0.005
min	0.144	2.547	0.734	-0.260	0.071
25%	0.609	2.797	0.852	-0.140	0.082
50%	0.670	2.826	0.882	-0.104	0.085
75%	0.707	2.857	0.910	-0.072	0.088
max	0.843	2.980	1.034	0.292	0.113

Table D.4: Summary statistics for baseline straight road environment (rounded to 3.d.p).

	mean episode reward	mean episode cte	mean episode speed	mean episode steering angle	mean episode throttle value
count	1000.0	1000.0	1000.0	1000.0	1000.0
mean	-6.249	2.762	0.834	-0.127	0.077
std	0.716	0.099	0.045	0.122	0.005
min	-9.131	1.127	0.395	-0.47	0.062
25%	-6.585	2.741	0.812	-0.16	0.075
50%	-6.246	2.770	0.834	-0.132	0.077
75%	-5.960	2.797	0.860	-0.103	0.080
max	0.258	2.909	0.975	1.900	0.137

Table D.5: Summary statistics for baseline curved road environment (rounded to 3.d.p).

	mean episode reward	**mean episode cte**	**mean episode speed**	**mean episode steering angle**	**mean episode throttle value**	**closest traffic light distance**
count	1000.0	1000.0	1000.0	1000.0	1000.0	1000.0
mean	9.424	2.839	0.549	-0.079	0.050	0.140
std	2.497	0.070	0.093	0.306	0.012	0.034
min	0.142	1.988	0.396	-0.363	0.036	0.000
25%	8.806	2.823	0.507	-0.211	0.045	0.136
50%	9.905	2.840	0.530	-0.181	0.048	0.146
75%	10.774	2.863	0.563	-0.135	0.050	0.156
max	15.484	3.300	1.147	1.337	0.147	0.242

Table D.6: Summary statistics for straight road with a red traffic light environment (rounded to 3.d.p).

Table D.7: Snapshot of CSV file that recorded the metrics for evaluation.