University of BRISTOL

Traffic Management with Connected and Autonomous Vehicles through Deep Reinforcement Learning

Fermin Orozco

JUNE 10, 2021

Final year project thesis submitted in support of the degree of Master of Engineering in Electrical & Mechanical Engineering

Department of Electrical & Electronic Engineering
University of Bristol

DECLARATION AND DISCLAIMER

Unless otherwise acknowledged, the content of this thesis is the original work of the author. None of the work in this thesis has been submitted by the author in support of an application for another degree or qualification at this or any other university or institute of learning.

The views in this document are those of the author and do not in any way represent those of the University.

The author confirms that the printed copy and electronic version of this thesis are identical.

Signed:

Fermin Orozco

Dated:

10/06/2021

Abstract

Advancements in the infrastructure of Cooperative Intelligent Transportation Systems grant Connected and Autonomous Vehicles access to immense amounts of data regarding traffic conditions. Deep Reinforcement Learning can be applied in the context of traffic management to train a centralised agent to develop a decision-making policy which uses the detailed description of the traffic environment to select optimal actions for Connected and Autonomous Vehicles.

In this study, Proximal Policy Optimisation and Vanilla Policy Gradient Reinforcement Learning Algorithms are used to train a centralised agent to act as the acceleration and lane-changing controller for Connected and Autonomous Vehicles. Human-Driven, and high-priority vehicles will also be simulated alongside the Connected and Autonomous Vehicles using explicit controllers. The objective of the agent is to maximise the average velocity of vehicles in the network while minimising the delays incurred by high-priority vehicles in the network. The particular case of high-priority vehicles studied will be buses. The Flow framework is used to interface between the traffic simulation software and Reinforcement Learning libraries.

Three networks with varying degrees of complexity were designed. For the square bus lane network, a trained policy outperformed the baseline human-driven vehicle experiment with respect to the average speeds of vehicles by 13.5% in a fully autonomous environment. In the grid network design, at 100% CAV penetration the trained policy demonstrated an increase in the average speed of the high-priority vehicles of 37.4%, compared to the case modelled by exclusively human drivers. Finally, in the case of the complex urban road network, at 10% CAV penetration rates, an improvement of 7.8% was observed relative to the human-driven baseline experiment.

Acknowledgments

To Professor Robert Piechocki, thank you for introducing me to the field of RL. I am extremely grateful for your guidance and support throughout my research. To Professor Nick Simpson, I am thankful for your generosity and attentiveness over the past four years. To UC Berkeley for their contributions to RL research through Flow, as well as the generous and helpful support in their Flow slack channel: Thank you.

To my Mother, Maria, who taught me the beauty of kindness and empathy. From my Father, Rafael, who exemplified strength, I learned the importance of resilience. Your voices within will forever give me support and hold me accountable.

To Dulce, who showed me to avoid judgement, practise patience, and remain unwaveringly true to ones character. Of Rafael; to imagine without limits, to work tirelessly for your goals, and to live with child-like convictions.

To Majo, thank you for your contagious bravery and laughter. To Angel; my kindred spirit whose curiosity and perseverance inspire and amaze me. To Isa, for your inspiring strength, true selflessness, and constant support.

To Abid, for your stoic steadfastness and kindness. To Julia, for your confidence, honesty, and warmth. To Julien, for your amazing serenity. To Jordan, for your bravery. To Dario, for exemplifying benevolence and drive.

To all my family, and all the people I have had the pleasure of living with and learning from.

Thank you God.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Drivers in the U.K. lost an average of 115 hours per capita in 2019 due to congestion, which reportedly cost the economy around £6.9 billion [1]. A solution to the challenge posited by the inefficiency of the current transport infrastructure lies in developing a Cooperative Intelligent Transportation System (C-ITS) capable of safely managing traffic among all types of vehicles whilst also increasing traffic efficiency and vehicle throughput.

Projections for the market penetration of Autonomous Vehicles (AVs) show that by 2030, a quarter of the cars on UK roads will have a high level of automation (Level 4 or above, as defined by the Society of Automotive Engineers [2]), and every car in the UK will have some degree of connectivity [3]. Connected Autonomous Vehicles (CAVs) build from regular AVs by establishing a network through which information can be shared wirelessly, so that a singular CAV will have more information than it could obtain through its sensors alone. The C-ITS would provide the framework whereby connected vehicles would communicate with one another. In a traffic scenario where all vehicles are connected and form part of a C-ITS, then information regarding the velocity, position, and status of all vehicles would be available. Other vehicles, as well as adaptive infrastructure, would be able to dynamically change their behaviour to improve traffic efficiency and security.

Establishing a reliable and robust C-ITS will be paramount towards minimizing traffic congestion whilst also ensuring road fairness and safety. Cooperative driving is a subsection

within C-ITSs involved with transferring status and sensor data across vehicles so that they may intelligently coordinate their actions in complex traffic scenarios. For cooperative driving to be applied to CAVs on a large scale, a reliable traffic management system is required to coordinate CAV behaviour. Deep Reinforcement Learning (DRL) offers a method for developing a traffic controller which learns the optimal behaviour from an extensive set of experiences that can be obtained from simulations. And once the learned controller has been trained, it can be integrated into the C-ITS to direct CAVs based on the information obtained from all connected vehicles in the network. In order to reach the stage where a DRL approach can be implemented as a traffic management system, a thorough, robust, secure, fair, and efficient framework which is able to function in real-world scenarios must be built.

1.2 Thesis Objectives

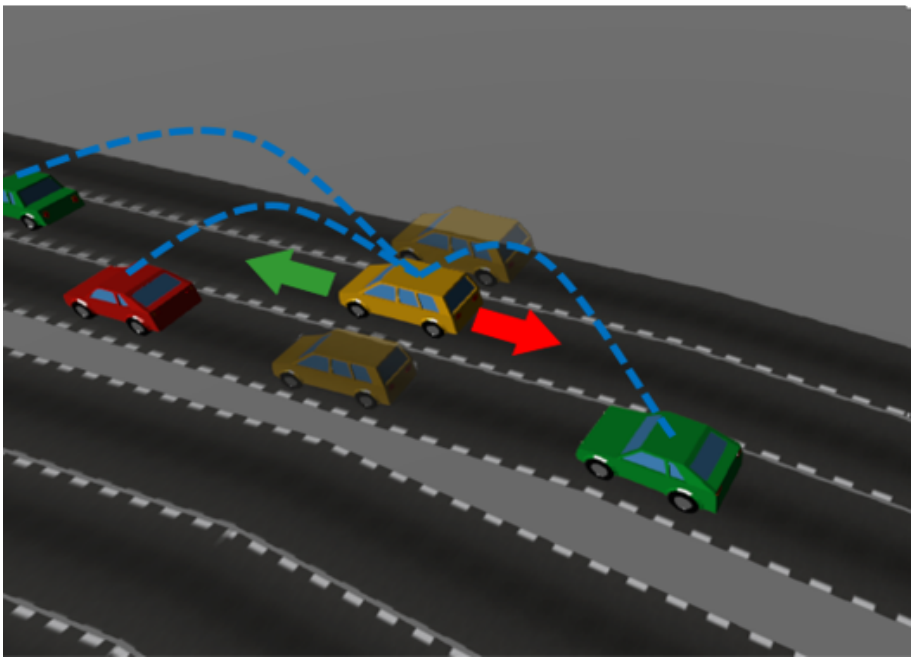

Figure 1.1: Interaction of CAV (yellow car) with non-RL controlled vehicles. Information received from other vehicles (blue lines) will be used to control the acceleration or lane-changing actions for the CAV. Image developed from the Sumo-Web3d simulation of the Bristol triangle network.

The aim of this research is to investigate a DRL approach to real-world traffic environments in situations where vehicles have different levels of priority. A plethora of research into DRL

algorithms used to maximise parameters such as traffic efficiency exists, but no research yet is aimed at developing policies which take into account the effects of cooperative driving on vehicles with high priority. High-priority vehicles can be defined by vehicles whose delay is more negatively impactful than for an average vehicle. Buses, ambulances, police, and fire brigades are some examples of what would constitute a high-priority vehicle. Through the field of cooperative driving, CAVs would be able to maximise their own velocity whilst ensuring that their actions do not negatively impact vehicles with high priority.

Figure 1.1 shows the process of how a CAV within a C-ITS would be able to perform lane-changing and acceleration actions based on the information it is receiving from the other connected vehicles in the network. A centralised agent would be able to control the action of CAVs to strategise driving behaviour and enable cooperative driving. The proportion of CAVs to other human-driven vehicles in a road network is known as the CAV market penetration.

As well as CAVs and human-driven cars, high-priority vehicles representing buses will also be modelled. Information regarding the high-priority vehicles will be available to the agent who will then aim to coordinate CAVs to minimise their associated delays. Interactions of the vehicles will be simulated at various degrees of CAV market penetration in a traffic simulation software. The collected set of experiences will be used to train the agents through DRL to develop a policy which will control CAV actions so that vehicles exhibit the desired behaviour.

The first objective of this thesis will lie in developing a Markov Decision Process for the DRL problem which aims to maximise the average vehicle velocity and minimise any delays to the high-priority vehicles, by developing a policy which acts as the learned acceleration and lane-changing controller for all CAV vehicles. The performance of the learned controller with respect to parameters such as average vehicle velocity, and high-priority vehicle delay will then be compared with traditional, explicit traffic controllers.

With the aim of implementing the learned policy in real-world traffic scenarios, the second objective of this research will be to design three different road networks with varying degrees of road complexity. The trained policies for each road network will be analysed, and the effects of the environment on the optimal policy performance will be discussed. Experiments

will be conducted at 10%, 50%, and 100% CAV market penetration to understand behaviour and performance changes at varying degrees of mixed autonomy environments.

The last objective of this research is to employ multiple DRL training algorithms to benchmark the learning rate and the trained policy for each scenario.

1.3 Contributions

The research presented in this thesis contributes to the field of Cooperative Driving for C-ITS' by developing an acceleration and lane-changing controller for CAVs, learned through DRL, which aims to minimise the delay of high-priority vehicles, whilst maximising the velocities of vehicles throughout the network. The designed Markov Decision Process is trained and tested in a variety of different networks, and its performance with respect to Key Performance Indicators such as high-priority vehicle delay, and average vehicle velocity is analysed. Vanilla Policy Gradient and Proximal Policy Optimisation DRL algorithms are used to benchmark the policy learning process, as well as the trained optimal policy. Further work regarding Multi-Agent approaches, high-priority vehicle modelling, and partially visible environments are suggested.

1.4 Structure Review

In this chapter, the application of DRL in the context of traffic management systems was motivated. The main objectives and contributions of the author's own research were then elaborated. Chapter 2 of this thesis will provide background to DRL which the rest of the work relies on, as well as discussing related research in the field of DRL applied to traffic management. In chapter 3, the proposed approach for the DRL problem will be presented. Chapter 4 details the experimentation method, as well as simulation and modelling descriptions. The results of the experiments will be presented and discussed in Chapter 5. Finally, in chapter 6 the method and results will be reviewed and areas for future research are suggested.

Derivation explanations and further details regarding the designed road networks are available in the appendix.

Chapter 2

Background

In this chapter, the theory of supervised learning and RL will be explained insofar as they relate to DRL. Section 2.1 will discuss the structures of Neural Networks alongside their training procedures. In section 2.2, the theory and notations used for RL will be introduced. Both topics will converge in section 2.3 to present DRL.

2.1 Supervised Learning

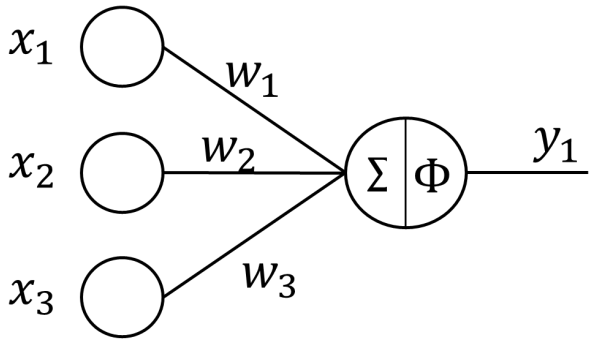

Figure 2.1: Structure of single-layer Perceptron.

A single-layer perceptron, as shown in Figure 2.1, was first presented in 1958 by F. Rosenblatt and is an example of an Artificial Neuron (AN) [4]. The single-layer perceptron applies an activation function, ϕ, to a sum of a set of weighted inputs and equates the result to an output y. The inputs, weights, and outputs can be expressed as vector sets using the notation x, w, and y, respectively. With the goal of minimising the error between the

current output and a desired output for a set of input data, the weights of the AN can be trained using Supervised Learning. The Widrow-Hoff Learning Rule, or Delta Rule [5], is a Supervised Learning method which can be used to train single-layer perceptrons with linear activation functions. The output of a single-layer perceptron is shown by

$$y_j = \Phi(\sum_i w_j i \;\; x_i) \tag{2.1}$$

where i is the number of inputs, and j is the number of outputs. For a linear activation function where $\phi = 1$, the output of the perceptron is a linear sum of the weighted inputs. A Cost Function, $E(w_{ji})$, which uses the mean squared error between the current output and the desired output is defined to quantify the error of the AN. The formula is given by

$$E(w_{ji}) = \frac{1}{2}(t_j - y_j)^2 \tag{2.2}$$

where t is the vector set of desired outputs. With the aim of using gradient descent optimisation on the Cost Function, the derivative of Equation 2.2 with respect to the weights of the perceptron can be calculated using the chain rule as shown below

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial y_j} \cdot \frac{\partial y_j}{\partial w_{ji}} = -(t_j - y_j)x_i = -\delta x_i \tag{2.3}$$

where δ is defined as the delta function, given a linear activation function. Gradient descent is an iterative optimisation algorithm used to minimise a function by taking repeated steps in the opposite direction of the local gradient so over time it will arrive at a local minimum of the function. For convex functions, gradient descent will arrive at the global minimum. The iterative equation for gradient descent is

$$a^t = a^{t-1} - \nabla f(a) \tag{2.4}$$

where a are the parameters of the function $f(a)$. By applying gradient descent optimisation on the cost function, and substituting the gradient of the cost function with respect to the weight parameters as shown in Equation 2.3 into the gradient descent formula in Equation 2.4, we obtain the equation

$$w_{n+1} = w_n - (-\delta x_i) = w_n + \delta x_i \tag{2.5}$$

Algorithm 1 Widrow-Hoff Learning Rule

1: Randomly initialise weights
2: Compute neuron output based on input data, $y_j = \sum_{j=0} w_{ji} x_j$
3: Compute the error function, $\delta_j = t_j - y_j$
4: **while** $\delta_j > 0$ **do**
5: Compute weight change, $\nabla w_{ji} = \eta(t_j - y_j)x_i$
6: Update weights through gradient descent, $w_{ji}^t = w_{ji}^{t-1} + \nabla w_{ji}$
7: Compute new y_j and δ_j from updated weights
8: **end while**

Algorithm 1 shows how gradient descent would be iterated to train the weights of a single-layer perceptron with a linear activation function until the error function is zero. The gradient descent algorithm has several hyperparameters, which are variables that can be modified manually which affect the training process. The learning rate hyperparameter, shown as η, dictates by how much the weights in the ANN can change with each update. Two other hyperparameters include the batch size, which controls the number of training samples processed before new weights are updated, and the epoch number, which controls the number of iterations over the input data. The batch size in Algorithm 1 would therefore be one, and the epoch number would be large enough so that the delta function is sufficiently minimized and approximates to zero. In the case of an AN whose activation function is not linear and therefore whose gradient is not equal to 1, the general formula of the delta function is given by

$$w = \dot{\Phi}(z_j)x_i\eta(t_j - y_j) \tag{2.6}$$

where $\dot{\Phi}$ is the derivative of the activation function, and z_j is the weighted sum of the inputs. A single-layer AN can be used as a function approximator for linear classification problems but is unable to find solutions for problems which are not linearly separable, such as implementing a XOR gate. To represent more complex functions, a multi-layer structure of ANs can be adopted. The interconnection of ANs over multiple layers produces a network known as an Artificial Neural Network (ANN). ANNs with more than one hidden layer are known as Deep Neural Networks (DNN). The multi-layer perceptron, shown in Figure 2.2, is an example of a feedforward ANN. Here, x is a set of the input data samples, h is a set of the output of the hidden layer nodes, y is a set of the final layer nodes output, v is a

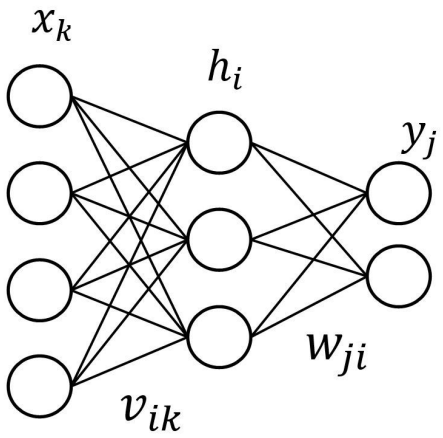

Figure 2.2: Structure of multi-layer Perceptron.

set of the weights connecting the input and hidden layer nodes, w is a set of the weights connecting the hidden layer and output nodes, k is the number of inputs, i is the number of hidden layer nodes, and j is the number of output nodes. Every node functions as explained before, summing all the values received, applying an activation function to the sum, and outputting the result through its connection. The outputs of the perceptrons on the first layer are passed as inputs to the perceptrons in the hidden layer and so on and so forth.

Algorithm 2 Backpropagation with gradient descent

1: Randomly initialise weights
2: **for** $m \in \{1, \ldots, M\}$ **do**
3: Introduce input sample, $x_k^m = s_k^m$
4: Compute hidden layer values, $h_i^m = \Phi(\sum_k v_{ik} x_k^m)$
5: Compute output, $y_j^m = \Phi(\sum_i w_{ji} h_i^m)$
6: Compute delta output function, $\delta_j^m = \dot{\Phi}(\sum_i w_{ji} h_i^m)(t_j^m - y_j^m)$
7: Compute delta hidden function, $\delta_i^m = \dot{\Phi}(\sum_k v_{ik} x_k^m)(\sum_j w_{ji} \delta_j^m)$
8: Compute weight changes, $\nabla w_{ji}^m = \delta_j^m h_i^m$, $\nabla v_{ik}^m = \delta_i^m x_k^m$
9: Update weights through gradient descent, $w_{ji}^t = w_{ji}^{t-1} + \eta \nabla w_{ji}^m$, $v_{ik}^t = v_{ik}^{t-1} + \eta \nabla v_{ik}^m$
10: **end for**

Introducing hidden layers into the architecture means that to update the weights of the hidden edges, the error of the hidden nodes must be calculated. Rumelhart et al [6] presented the idea of ANN training through backpropagation, wherein the errors of the output nodes are passed backwards to the hidden layers so that the weights of the hidden nodes can be

trained through gradient descent. The process is illustrated in Algorithm 2. The process of training through backpropagation involves three steps: Computing the output of the ANN based on the inputs, computing the delta function over all layers, then performing gradient descent optimisation for the weight parameters throughout the ANN. The batch size for this training algorithm would be M as the process is repeated M times, where M is the number of training data input samples. The epoch number would be one, as only a single training stage occurs for every set of input samples.

Calculating the weight changes and updating the parameters of an entire ANN for every individual sample in a large input training data set would require a lot of computational power and would mean a slower training process. Stochastic Gradient Descent (SGD) reduces the computing power and time associated with training an ANN by only computing the delta function and weight parameter updates of a single randomly selected training element [7]. This differs from the gradient descent algorithm implemented in Backpropagation Algorithm 2, as here the delta function and weight parameter updates were computed for all elements in the training set input samples. SGD also helps escape local minima when approximating non-convex functions [7]. Mini-batch SGD uses multiple randomly selected training elements, called mini-batches, to improve training accuracy over a single epoch. Adam, or Adaptive moment estimation, is an advanced optimisation algorithm which works similar to Mini-batch SGD, however it features an adaptive learning rate parameter to improve the learning rate over iterations [8]. Adam also computes an estimation of the mean and variance of the gradient to supplement the weight parameter optimisation process shown in Equation 2.4.

To calculate the delta function at the output and hidden layers the activation function must be differentiated. This is shown in stages 6 and 7 of Algorithm 2. Therefore, for Backpropagation to function it is essential that the activation functions used in the hidden layers have a derivative form. Furthermore, using non-linear activation functions within the hidden layers enhances the complexity of functions which an ANN can approximate.

Common non-linear activation functions used in ANNs are the sigmoid function and the hyperbolic tangent function, both of which are shown in Figure 2.3. One issue that arises when the derivative of the Cost Function with respect to the weight parameters in a layer is very small, is that the weight is only changed by a small amount. This effect is magnified to

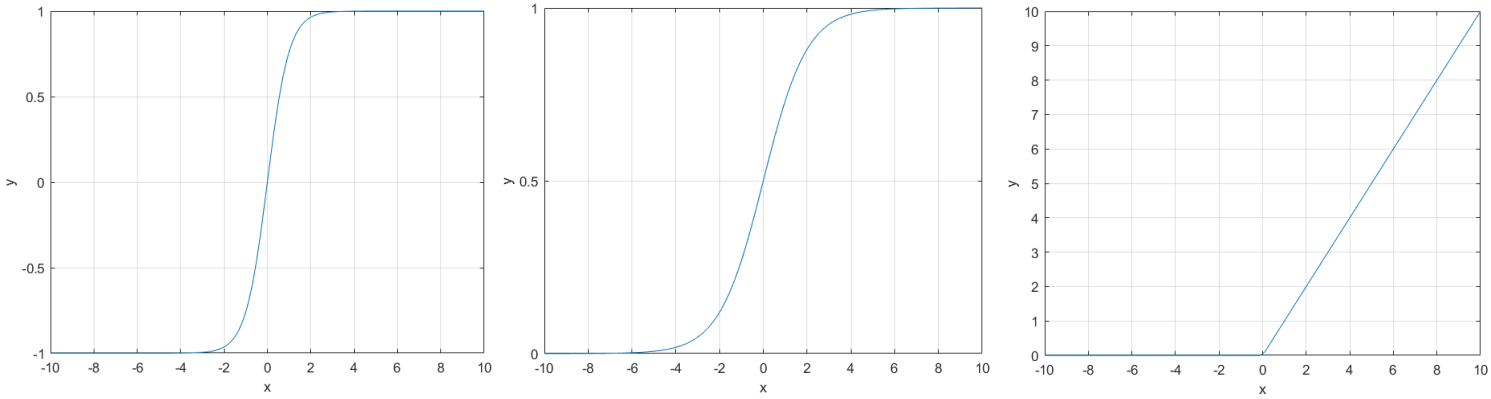

Figure 2.3: Graphs of Activation Functions Tanh, sigmoid, Relu from left to right.

previous layers when the delta function is backpropagated which means the training process is hindered due to the slower training process. This is known as the vanishing gradient problem, and the inverse of this problem – when the Cost Function derivative with respect to the weight parameters is very high and the effect is backpropagated – is known as the exploding gradient problem. The effect of the exploding gradient is that the training of the parameters becomes unstable. The main cause of the vanishing gradient problem is the activation function used for the hidden layer nodes. The outputs of the tanh and sigmoid functions saturate at 0, 1, and -1,1 respectively for extreme values of x. When there are multiple hidden layers in an ANN, Backpropagation computes the gradients using the chain rule and because the derivatives of the tanh and sigmoid functions fall to zero at the saturation points this has the effect of backpropagating an exponentially decreasing delta function.

Rectified Linear Units (ReLU) are neural network activation functions which are capable of lowering vanishing gradients because they do not saturate at extreme positive inputs. ReLUs also reduce the training time of an ANN due to the computational simplicity associated with calculating their derivatives. Although ReLU functions are composed of two conjoined linear functions, the non-linearity at the point of conjunction means that a sequence of layers utilising ReLU can approximate complex functions. Because of the non-linearity feature of ReLUs at zero, it means that at this point, the function is not differentiable, and to overcome this obstacle, the derivative at zero for ReLUs is defined as zero, so $\phi(0) = 0$. However, ReLU's then face the potential issue known as ReLU death, which occurs when the weights reach a large negative value so the gradient of the activation function for future training steps becomes zero since the output of a ReLU is constantly zero at negative inputs. Further

optimisations have been made to the ReLU, such as the Leaky ReLU, or the Scaled Exponential Linear Units, shown in Figure 2.4, which can apply a transfer function to negative inputs meaning ReLU deaths are no longer an issue.

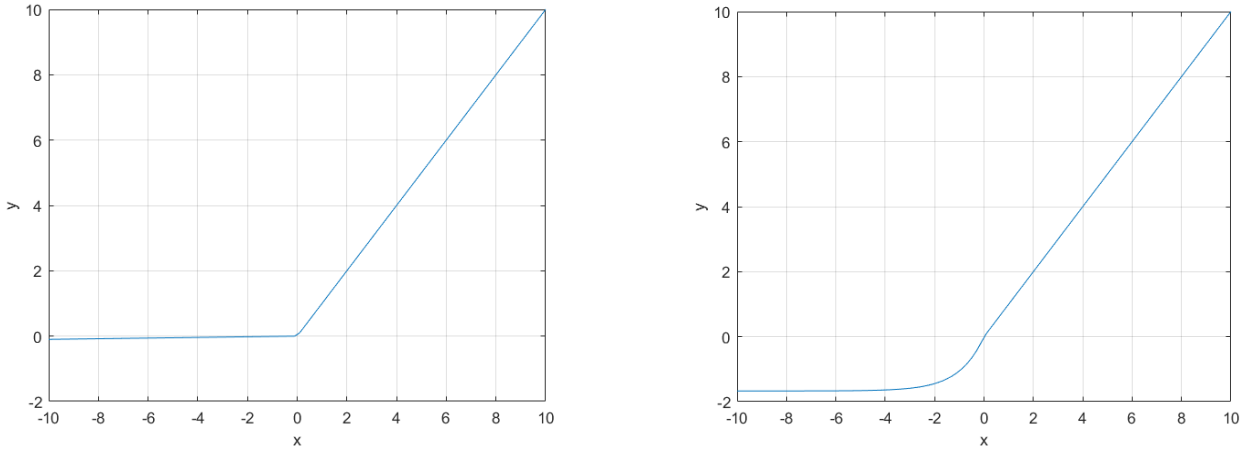

Figure 2.4: Graphs of Activation Functions Leaky ReLU and SELU.

Biased initialisation of the ANN's weights can cause the parameter optimisation process to converge to a local maximum for non-convex function approximations. This can be avoided by initialising weights to random values, which will lead to a more thorough exploration of the parameter space during training. In Algorithm 2, this example of Backpropagation initialises the weight parameters randomly. One way this would be implemented in practise would be by assigning a normal probability distribution to the weight variables with a mean of 0 and a variance of 1. Glorot et al [9] proposed an initialisation regime known as normalised initialisation which improved the performance and training efficiency of ANNs, where the variance of the weight parameters are scaled based on the number of neurons in the previous layer,n_{in}, and the number of neurons in the current layer n_{out}. Using normal distribution to map the weight parameters, this method would assert a mean of zero, and a variance given by the equation:

$$\sigma = \sqrt{2}\sqrt{\frac{2}{n_{in} + n_{out}}} \tag{2.7}$$

Overfitting occurs when an ANN has had its parameters tuned very finely to the training data, and as a result it is not able to generalise to input samples outside of the training data. When such an ANN attempts to process the test data, it would show worse perfor-

mance. Srivastava et al [10] show that by randomly disabling neurons in the hidden layers during a training step, overfitting is drastically reduced as neurons in a layer must learn to be as independently efficient as possible. This process is known as Dropout layering, and a hyperparameter called the dropout rate gives the probability of disabling a particular neuron during training. Having too much depth, meaning many hidden layers, also induces overfitting to the training set.

2.2 Reinforcement Learning

Reinforcement Learning (RL) is the branch of Machine Learning which relates to developing algorithms that can learn through experiences to choose actions that elicit a positive change in the environment. In RL, the agent is defined as the decision-making character, and the environment is the world that the agent can interact with. Through agent-environment interactions, the agent develops a decision-making policy which increases the probability of choosing good actions. A reward is used to give feedback based on if the action taken changed the environment in a positive or negative way.

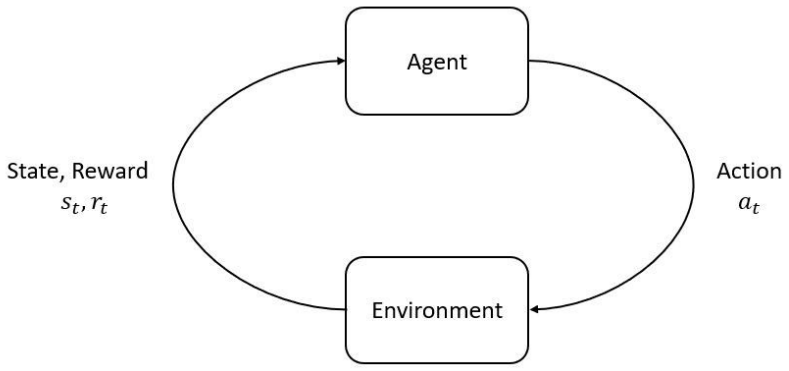

Figure 2.5: Reinforcement Learning agent-environment interaction diagram [1].

Figure 2.5 exemplifies the interaction between the Agent and the Environment in RL: At time t the agent will also observe the state of the environment, s_t, and obtain a reward, r_t. The reward will be based on features in the environment which changed due to the action taken in the previous time step, a_{t-1}. At the time step t, the Agent will choose an Action

[1]Diagram taken from OpenAI's Spinning Up RL resource [11].

a_t which will change the state of the environment and advance it to s_{t+1} at the next time step. A sequence of states are referred to as a trajectory, τ. If the observations that the agent makes of the environment completely describe the state of the environment then the environment is said to be fully observed, but if some states remain hidden to the agent, the environment is described as partially observable.

The Markov Decision Process (MDP) is used to formalise RL problems whose trajectory at the next time step depends only on the most recent state and action. This property is known as the Markov property. MDP is a tuple of four components, $\langle S, A, P, R \rangle$ which is used to formalise the RL problem.

The state space, S, is used to denote all observable states of an RL problem where $S = (s^1, \ldots, s^D)$, so S is the set of states of size D, and every element in the state space, s, describes the environment at a particular moment. Each individual element can hold a descriptive feature of the environment; in a traffic scenario an element of the state space could describe the velocity of a particular vehicle.

The action space, A, is used to denote all possible actions that an agent can take during its interaction with the environment that will advance the environment into a new state. $A = (a^1, \ldots, a^G)$, where G is the size of A, and $a \in A$. Environments can have action spaces whose elements are discrete or continuous. Applied in a traffic context, a could refer to an acceleration change action of a particular vehicle for a continuous action space, as acceleration is a continuous variable.

The reward function, R, returns a scalar value which depends on the current state of the environment, s_t, the action taken recently, a_t, and the state of the environment at the next time step, s_{t+1}, so $r_t = R(s_t, a_t, s_{t+1})$. Finally, the probability function, P, states the probability of an action $a \in A$, applied in state $s \in S$, transitioning the environment into state $s^{'}$, so $P(s^{'}|s, a)$. In a simulated traffic scenario, the reward function could return a scalar proportional to the average velocities of the vehicles at a time step, and the probability function would be the traffic simulation software, as it would be responsible for simulating the trajectory of the environment based on vehicle actions.

For a MDP, the agent will develop a policy to govern the decision-making process through

which the agent selects the actions to take. There are two types of policies: 1) Deterministic policies, which based on the current state will choose a clearly defined action, and 2) Stochastic policies, which based on the state will choose an action from a probability distribution. Given the stochastic nature of traffic simulation software and traffic in general, a stochastic policy would be more appropriate in this context. A stochastic policy, π, is denoted by

$$a_t \sim \pi_\theta(\cdot|s_t) \tag{2.8}$$

where θ are the parameters of the policy; in RL, the policy is often represented by a neural network hence θ represents the weight parameters which can be trained. The training process of the policy will be discussed later in this section. The goal of the policy is to maximise the cumulative reward. A function known as the infinite-horizon discounted return, $R(\tau)$ can be defined to represent the value of the cumulative reward which the policy aims to maximise:

$$R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t \tag{2.9}$$

where $\gamma \in (0, 1)$ is a discount factor which decreases the weight of future rewards, and ensures the convergence of the return function. The probability distribution over a trajectory will help understand how a stochastic environment progresses through states by acting based on a policy, and it is given by

$$P(\tau|\pi) = \rho_0(s_0) \prod_{t=0}^{T-1} P(s_{t+1}|s_t, a_t)\pi(a_t|s_t) \tag{2.10}$$

where $\rho_o(s_0)$ is the initial state distribution, T is the number of steps in the specified trajectory, and $P(s_{t+1}|s_t, a_t)$ is the probability that given a stochastic environment, an action a_t in state s_t will cause a state transition to s_{t+1}. From Equations 2.9 and 2.10, the expected infinite-horizon discounted return obtained by following a policy π from state s can be found. This is called the On-Policy Value Function, $V^\pi(s)$, and is defined as

$$V^\pi(s) = \int_\tau P(\tau|\pi, s_0 = s)R(\tau|s_0 = s) = \underset{\tau\sim\pi}{E}[R(\tau)|s_0 = s] \tag{2.11}$$

Similarly, the expected infinite-horizon discounted return obtained by first choosing a specific action a from state s, and afterwards act by following a policy π is called the On-Policy Action-Value Function, $Q^\pi(s, a)$:

$$Q^{\pi}(s,a) = \underset{\tau\sim\pi}{E}[R(\tau)|s_0 = s, a_0 = a] \tag{2.12}$$

Equations 2.11 and 2.12 could be combined to represent the advantage that an action a would have on the expected return compared to an action chosen through acting by the policy. This advantage function, $A^{\pi}(s,a)$, would be defined as

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s) \tag{2.13}$$

The optimal policy would maximise the expected infinite-horizon discounted return, and is what the current policy is attempting to approximate through RL. Similar to Equation 2.11, an Optimal Value Function, $V^{*}(s)$, can be defined as the expected return when the optimal policy as adhered to over a trajectory:

$$V^{*}(s) = \underset{\pi}{max}\ \underset{\tau\sim\pi}{E}[R(\tau)|s_o = s] \tag{2.14}$$

A plethora of RL algorithms exist, each providing a different method of training the policy to select actions. Some model-based RL algorithms are able to recreate the probability transition function so that the environment the agent is operating in can be completely and accurately modelled through agent-environment interactions. If the agent approximates the environment incompletely then the policy performance will be poor. In a traffic context, predicting the behaviour of all vehicles in a simulation based on one vehicles' action is a very complex computation and may overfit to the trained network so when new scenarios are presented the probability transition function may not generalise as the behaviour of vehicles changes completely. An alternative to model-based algorithms, are model-free algorithms. Although less-sample efficient, model-free algorithms train the policy exclusively from experiences between the policy and environment without having to construct a model of the probability transition function. Of the model-free RL algorithms, some - known as Q-Learning methods - train a policy indirectly by maximising the Bellman Equations for action-state pair values; Deep Q-Networks developed by Mnih et al [12], and Hindsight Experience Replay developed by Andrychowicz et al [13] are examples of Q-Learning algorithms. Because Q-Learning algorithms require an action-state pair value, they are more suited to discrete action spaces. The traffic scenarios that will be used in this thesis will require continuous action spaces as the acceleration of vehicles is not a discrete value, hence this will require another branch of model-free RL algorithms known as Policy Optimisation

methods. These RL algorithms train the policy directly via gradient ascent.

2.3 Deep Reinforcement Learning

Deep Neural Networks - covered in Section 2.1 - can be used in Reinforcement Learning to approximate objectives such as policies and value functions. This results in the field of Deep Reinforcement Learning (DRL). The two DRL algorithms which will be used to train the traffic management policy for this thesis, will be Proximal Policy Optimisation (PPO) presented by Schulman et al [14], and Vanilla Policy Gradient (VPG) presented in a Deep RL context first by Duan et al [15]. By studying the training process and developed policy of two different RL algorithms, their relative performance can be assessed and results benchmarked.

2.3.1 Vanilla Policy Gradient

Policy Gradient Methods were first presented by Sutton et al [16] were the policy was represented using a function approximator whose parameters were updated via gradient ascent. Given Equation 2.8 where θ are the parameters of a function which approximates the policy π then through gradient ascent, θ can be updated in the manner

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\pi_\theta)|_{\theta_k} \tag{2.15}$$

where the current policy expected return is $J(\pi_\theta) = \underset{\tau \sim \pi_\theta}{E}[R(\tau)]$, similar to Equation 2.11. $\nabla_\theta J(\pi_\theta)$ can therefore be considered the policy gradient. An expression for the gradient-logarithmic probability for the policy gradient can be derived to be

$$\nabla_\theta J(\pi_\theta) = \underset{\tau \sim \pi_\theta}{E}\Big[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau)\Big] \tag{2.16}$$

where the term $\nabla_\theta \log \pi_\theta(a_t|s_t)$ refers to the log-likelihood of a particular action. The derivation for Equation 2.16 is shown by Achiam et al [11], and is included in the Appendix for the sake of completeness. Since the policy gradient expression shown in Equation 2.16 is an expectation, it can be estimated with a sample mean; for a set of trajectories, $\mathfrak{D} = \{\tau_i\}_{i=1,\ldots,N}$, collected by acting according to policy π_θ, the policy gradient can be estimated by

$$\hat{g} = \frac{1}{|\mathfrak{D}|} \sum_{\tau \in \mathfrak{D}} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau) \tag{2.17}$$

Based on the work discussed so far in this chapter, the algorithm for Vanilla Policy Gradient for training policies can be presented.

Algorithm 3 Vanilla Policy Gradient Algorithm

1: Initialise policy parameters θ_0, initialise value function parameters ϕ_0
2: **for** k=0,1,2,... **do**
3: Run the current policy π_θ and collect a set of trajectories, $\mathfrak{D}_k = \{\tau_i\}$
4: Compute the infinite-horizon discounted return as shown in Equation 2.9 for each point in the trajectory
5: Compute the advantage estimates, $\hat{A}_t$ using an advantage function estimation method based on the current value function $V_{\phi k}$
6: Estimate policy gradient using Equation 2.17, as

$$\hat{g_k} = \frac{1}{|\mathfrak{D}_k|} \sum_{\tau \sim \mathfrak{D}_k} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)|_{\theta_k} \hat{A}_t$$

7: Compute policy update through a gradient ascent algorithm
8: Fit value function by regression on mean-squared error over all trajectory points:

$$\phi_{k+1} = \underset{\phi}{\operatorname{argmin}} \frac{1}{|\mathfrak{D}_k| T} \sum_{\tau \sim \mathfrak{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2$$

ϕ parameters can be trained through a gradient descent algorithm
9: **end for**

Deep Neural Networks are commonly used to represent the policy and value functions in DRL, and they can be trained through gradient ascent algorithms (such as Adam) and gradient descent algorithms (such as Stochastic Minibatch Gradient Descent), respectively[14]. Because Vanilla Policy Gradient is an on-policy training method, the policy used to collect the set of trajectories affects the policy updates. This, coupled with the fact that as training ensues the entropy in the policy action choices decreases, means that the policy can converge at a local optima and not the global optima.

2.3.2 Proximal Policy Optimisation

First presented by Schulman et al [14], Proximal Policy Optimisation (PPO) is a DRL algorithm which is capable of handling discrete and continuous action spaces. Building on previous Trust Region Policy Optimisation (TRPO) methods [17], PPO offers increased stability during the on-policy training process. The training algorithm for PPO has a similar structure to VPG although it introduces new concepts to train the policy parameters [11]. There are two forms of PPO algorithms used: The first is PPO-Clip which clips the policy gradient function to prevent the policy from changing drastically over a training step, and the second is PPO-Penalty which uses a dynamic penalty to deter the policy gradient function and prevent major policy changes over a training step [11]. The variant of PPO which will be used for this thesis is PPO-Clip.

Schulman et al [14] present a surrogate advantage function, L, for PPO-Clip algorithms to develop the behaviour of PPO algorithms as they train their policy. The equation for the surrogate advantage function is given by:

$$L(s, a, \theta_k, \theta) = \min\Big(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}\hat{A}_t \ , \ g(\epsilon, \hat{A}_t(s, a))\Big) \tag{2.18}$$

where $\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}$ gives a measure of how much better the current policy, π_θ, performs with regards to the previous policy, π_{θ_k}. $\hat{A}_t$ is an estimation of the advantage function and the method required to obtain this will be discussed later in this section. Furthermore, the second component, g, inside the minimum function is defined as

$$g(\epsilon, \hat{A}_t(s, a)) = \begin{cases} (1+\epsilon)\hat{A}_t & \text{if } \ \hat{A}_t \geq 0 \\ (1-\epsilon)\hat{A}_t & \text{if } \ \hat{A}_t < 0 \end{cases} \tag{2.19}$$

where ϵ is a hyperparameter which governs the amount the policy can be changed before it is clipped. Analysing the equation above, when the advantage function for a state-action pair is positive, $\pi_\theta(a|s)$ increases, therefore the value of the new surrogate function can increase up until the cap enforced by $(1+\epsilon)\hat{A}_t$. Similarly, when the advantage function for a state-action pair is negative, then $\pi_\theta(a|s)$ decreases, hence the value of the new surrogate function can decrease up until the cap enforced by $(1-\epsilon)\hat{A}_t$.

While Policy Gradient DRL implementations such as VPG aim to maximise the policy expected return, $J(\pi_\theta)$, PPO algorithms instead aim to maximise the value of the surrogate advantage function, L, shown in Equation 2.18. Since maximising this equation is the aim of

the policy, in literature it is also called the objective function. PPO-Clip algorithms calculate policy parameter updates through stochastic gradient ascent typically [18]:

$$\theta_{k+1} = \underset{\theta}{\text{argmax}} \underset{s,a\sim\pi_{\theta_k}}{\text{E}}\Big[L(s,a,\theta_k,\theta\Big] \tag{2.20}$$

Because the expression above is an expectation, it can be estimated with a sample mean for a set of trajectories, similarly to Equations 2.16 and 2.17. Amalgamating the work explored thus far, the PPO algorithm is presented below.

Algorithm 4 Proximal Policy Optimisation

1: Initialise policy parameters θ_0, initialise value function parameters ϕ_0
2: **for** k=0,1,2,... **do**
3: Run the current policy π_θ and collect a set of trajectories, $\mathfrak{D}_k = \{\tau_i\}$
4: Compute the infinite-horizon discounted return as shown in Equation 2.9 for each point in the trajectory
5: Compute the advantage estimates, $\hat{A}_t$ using an advantage function estimation method based on the current value function V_{ϕ_k}
6: Update policy by maximising the PPO-Clip function

$$\theta_{k+1} = \underset{\theta}{\text{argmax}} \frac{1}{|\mathfrak{D}_k|T} \sum_{\tau\in\mathfrak{D}_k} \sum_{t=0}^{T} \min\Big(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}\hat{A}_t(s,a)\ ,\ g(\epsilon,\hat{A}_t(s,a))\Big)$$

through stochastic gradient ascent with Adam optimisation
7: Fit value function by regression on mean-squared error over all trajectory points:

$$\phi_{k+1} = \underset{\phi}{\text{argmin}} \frac{1}{|\mathfrak{D}_k|T} \sum_{\tau\sim\mathfrak{D}_k} \sum_{t=0}^{T} \Big(V_\phi(s_t) - \hat{R}_t\Big)^2$$

where ϕ parameters can be trained through a gradient descent algorithm
8: **end for**

2.3.3 Generalised Advantage Estimate

Both the VPG and PPO implementations require a method for estimating the advantage function, $\hat{A}_t$. As the algorithms for both of these methods allude to, an advantage estimate is needed for the policy objective optimisation. One infamous issue in the field of RL, is the sparse credit assignment problem [19], which alludes to the concept that if a series of

actions have small immediate rewards, but will eventually lead to a large reward, the dilution of the perceived reward over time means that the actions are deemed as weak, hence their likelihood in future iterations will be lower. By utilising accurate advantage estimation methods, actions which lead to high delayed rewards can be more fairly assessed. In their paper regarding advantage estimation, Schulman et al [20] present a generalised advantage estimation method suitable for on-policy training methods. The Temporal Difference (TD) residual of a value function approximation, V, with a discount factor, γ, is defined by

$$\delta_t^V = r_t + \gamma V(s_{t+1}) - V(s_t) \tag{2.21}$$

An introduction to the topic of TD learning can be found in [21]. From this, an estimation of the discounted advantage, $\hat{A}_t^k$, over k time-steps can be obtained through:

$$\hat{A}_t^k = \sum_{l=0}^{k-1} \gamma^l \delta_{t+l}^V \tag{2.22}$$

Finally, to Equation 2.22 shown above, Schulman et al introduce a hyperparameter, λ, which is bound between 0 and 1. At lower λ values there is a lower variance but a higher bias in the advantage estimate, and at high values values there is a higher variance and lower bias in the advantage estimate [22]. variance of the estimation vs bias. A high variance in the advantage estimation makes the policy training process less sample efficient, whilst a high bias can cause inaccuracies in the training process convergence, therefore a trade-off between the two is required when choosing the value for the λ hyperparameter.

2.4 Literature Review

In [23], a decision-making CAV control algorithm was developed for lateral and longitudinal movement based on proximity and velocity of surrounding vehicles. Furthermore, by utilising data available of all vehicles through the C-ITS, CAVs could exploit emergent behaviour such as platooning to maximise the vehicle efficiency and minimise the number of traffic conflicts associated with lane-changes or close rear-end proximities between vehicles. It was found that using their model, at 50% market penetration of CAVs the estimated traffic conflicts were reduced by 50-80%.

The authors in [24] investigated the impact of CAVs and AVs in large-scale, urban environ-

ments. Traffic in five major European cities are modelled using SUMO, and IDM is employed to model car-following behaviour in vehicles. Environments with AVs, CAVs, and HD cars are simulated, and parameters such as desired time headway, and minimum gap between vehicles are set to accurately represent the vehicle type modelled in each scenario. The differences in parameters such as desired time headway, or minimum gap between vehicles, illustrates the fact that CAVs are expected to drive closer to one another, and can accelerate at a faster pase. CAVs are also equipped with rerouting capabilities to better handle congestion. The results show that for all scenarios, CAVs outperform the baseline human-driven vehicles in key performance metrics such as average velocity, journey duration, and lower vehicle density. In particular, the introduction of CAVs in the simulated London urban road network increased the average vehicle velocity by more than three times.

In [25], Proximal Policy Optimization (PPO) is used as a DRL method to maximise the average velocity of vehicles at varying degrees of CAV penetration by acting as the learned acceleration controller. Mixed autonomy is achieved by using IDM to model human-driven cars, while the trained policy is used to control CAV actions. Three different learning strategies with multiple CAVs are explored and compared. Firstly, a shared single-agent learning method is used where the policy is optimized for a single CAV's states and actions. Secondly, a global joint cooperative learning method is used, where the agent's state and action spaces are defined for all CAVs in the system. A third learning strategy, identified as local joint cooperative learning, attempts to reduce communication cost by building a network of locally close CAVs. The training of the policy and the simulation of traffic was conducted using the Flow framework [26], which links the traffic simulation software SUMO and RL libraries. Mixed traffic flow is modelled as a discrete-time Markov Decision Process (MDP), defined from the number of agents, the state space, the action space, the initial state distribution, the transition model – which represents the environment dynamics - and the reward function. A parametrised RL policy is employed which outputs a probability distribution over the action space, so that when the agent is given a state, it will take the action with the highest probability to cause a reward, taken from RL policy.

A mixed autonomy model at a non-signalized intersection was simulated by the authors in [27]. The autonomous vehicle penetration rate varied from 10% to 100%, and the traffic simulation results are compared using spatio-temporal diagrams, average vehicle velocity, and fuel consumption of vehicles. The authors found that at 100% CAV market penetration, the

average vehicle speeds and delay times were improved by 1.38 times and 2.55 times, respectively. Vehicle emissions are modelled equally for all cars in this research, and limitations regarding the lack of electric or varied vehicle engine efficiency are discussed.

Vidali et al [28] use Deep Q Learning applied to traffic light management, allowing traffic lights to have dynamic duration and sequences which change depending on traffic conditions to optimize system performance metrics. In their proposed approach, the action space becomes the possible traffic light's phase and duration, the state space becomes the phase of the traffic light and the reward function for the proposed MDP is composed of the vehicle waiting times. Scenarios with high and low vehicle flow rates are modelled at an intersection. The agent in this work does not control vehicle actions, therefore it contributes mainly to C-ITSs in developing intelligent infrastructure. Velocity of vehicles are modelled using a Weibull distribution with a shape factor of 2. A novel reward function is presented which represents the cumulative total waiting time and the trained policy is compared to an alternative MDP whose reward function - obtained from literature - represents current vehicle delay times. Results show an improvement in the cumulative and average vehicle waiting times over the network when using the novel reward function for Deep Q Learning traffic light systems.

The authors in [29] use Deep Q Learning as a variable speed limit controller to reduce congestion in highways. Here, the action space for the centralised agent is are increments to the current speed limit. The action space is made state dependent so that an action is not very far away from its current state, as that would result in highly oscillatory vehicle velocities. Different parts of the modelled highway can also have different speed limits. The reward function is based on the vehicle delay due to congestion. The expected reward when executing an action in a state, gives its Q value. For each state and action, there exists a Q value. The optimal policy can be obtained from being the maximum possible Q values for all actions in the states. Results show that compared to the case where no variable speed limits are implemented, the learned policy outperforms the baseline study by nearly two times in terms of total vehicle delay times for road networks with high vehicle demand.

Wu et al [30] study the effects of centralised DRL agents which control CAV actions at varying degrees of market penetration on a variety of road network designs. The authors also benchmark hyperparameters for policy gradient DRL algorithms. Through the Flow frame-

work, acceleration and lane-changing centralised CAV controllers are trained using Trust Region Policy Optimisation to maximise the average velocity of vehicles in the networks. To test the policy's ability to generalise to scenarios different to the training configuration, the average velocity of vehicles obtained from following a learned policy are plotted over a range of vehicle densities. The results showed that the learned policy was able to generalise and produce stable traffic flow even for densities outside the training set.

Jang et al [31] trained a policy to control two CAVs which would lead a fleet of vehicles at a junction. Noise was introduced into the state space observations which increased the robustness of the trained policy. The policy was then transferred from simulation to a small scaled city design with appropriately scaled vehicles. The transferred noise-injected policy reduced average travel time by 5% compared to vehicles whose actions were controlled using solely the IDM model. As further research into this scaled city network the team at Flow developed the full scale road network design along with routing and lane-changing controllers, which can be found in [32].

In [30], the authors utilise the Flow framework to train a policy for a centralised CAV controller in mixed autonomy environments. A range of road layouts where studied and the performance metrics maximised by the policy varied from network to network: For the highway bottleneck network the reward function was based on vehicle throughput, for the grid network it was the average delay of vehicles, and for the merging network it was the average speed of vehicles in the network. The training process is benchmarked using Trust Region Policy Optimisation, PPO, Evolutionary Strategies, and Augmented Random Search DRL methods. To tune the parameters for the various training algorithms hyperparameter searches were conducted, and the authors made the hyperparameters available in [32]. The results show that all learned policies outperform the baseline IDM case with regards to the performance metric for each environment.

Chapter 3

Proposed Approach

In this chapter, the RL theory explained in section 2.2 is contextualised, and an MDP will be used to define the proposed approach. The road networks are also presented in section 3.2.

3.1 Problem Definition

Within a Cooperative Intelligent Transportation System where all vehicles present possess connectivity capabilities, data from the traffic flow would be accessible to a centralised system for processing. This data would include the information available to sensors on connected vehicles, such as individual velocities, positions, stopping times, and priority levels. Assuming that the wireless V2X (vehicle-to-everything) communication in the infrastructure is robust and accurate, then the traffic flow would be defined as fully observable. This can be modelled accurately by the simulation software where all vehicle information of the traffic flow is available during simulation. A centralised agent with access to this information would be able to manage the driving behaviour of all CAVs in the network with complete visibility of the environment. Contextualising the theory explained in section 2.2, a centralised traffic management agent would observe the state of the environment and would choose actions for the CAVs which it deemed as favourable. The traffic simulation software would receive the actions of the centralised agent and process this as an instruction used to transition the environment into a new state. After observing the reward of the chosen action, a DRL algorithm would be employed to maximise the probability of the centralised agent choosing CAV actions which lead to high rewards.

3.2 Environments

The proposed Markov Decision Process will be trained and tested in different road networks to observe the emergent behaviour of the learned controller at varying degrees of network complexities. There are two types of traffic flow: The first is uninterrupted traffic flow, where the flow is influenced exclusively by vehicle interactions and roadway characteristics, and the second is interrupted traffic flow, where the flow is mainly influenced by traffic control devices such as traffic lights. The behaviour of traffic flow under interrupted flow is much less affected by vehicle interactions or road geometry. Three custom networks were designed to assess the performance of the policy in interrupted and uninterrupted flows, as well as in varying degrees of road geometric complexity.

3.2.1 Square Bus Lane

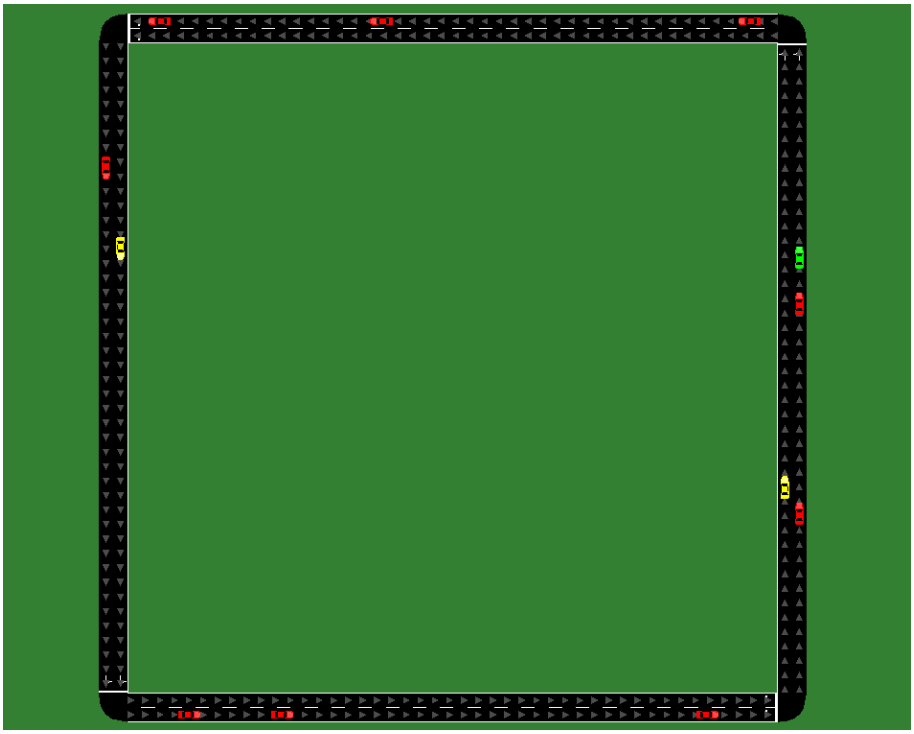

Figure 3.1: Square bus lane road network design with vehicles in initial configuration. Single CAV (green car) simulated against nine human-driven vehicles (red cars) and two bus vehicles (yellow vehicles). Placements show perturbation from uniform distribution using Gaussian distribution with standard deviation of 8 meters.

A square bus lane network was conceptualised to demonstrate the behaviour of the policy

under uninterrupted flow. The simple geometry and conditions allow for emergent behaviour to be visualised clearly. Figure 3.1 shows the designed simple square bus lane network. All roads have two lanes, and are all 150 meters in length. Buses will start on the inner lane - denoted as the bus lane - and all other vehicles will start on the outer lane. Similar to a real-world scenario, human driven vehicles are not allowed to change lanes onto the bus lane. CAVs will able to perform lane changing actions to avoid the congestion and increase their velocity, but only if this is not at the cost of an incurred delay to high priority vehicles. Initially non-bus vehicles are uniformly distributed along the outer lane, but a perturbation from the uniform distributed starting places is introduced. This perturbation can be modelled by a Gaussian distribution with a standard deviation of 8 meters. The fact that the starting positions for vehicles varies over each restart of the simulation improves the ability of the policy to generalise and avoid overfitting to the trained scenarios.

3.2.2 London Grid Network

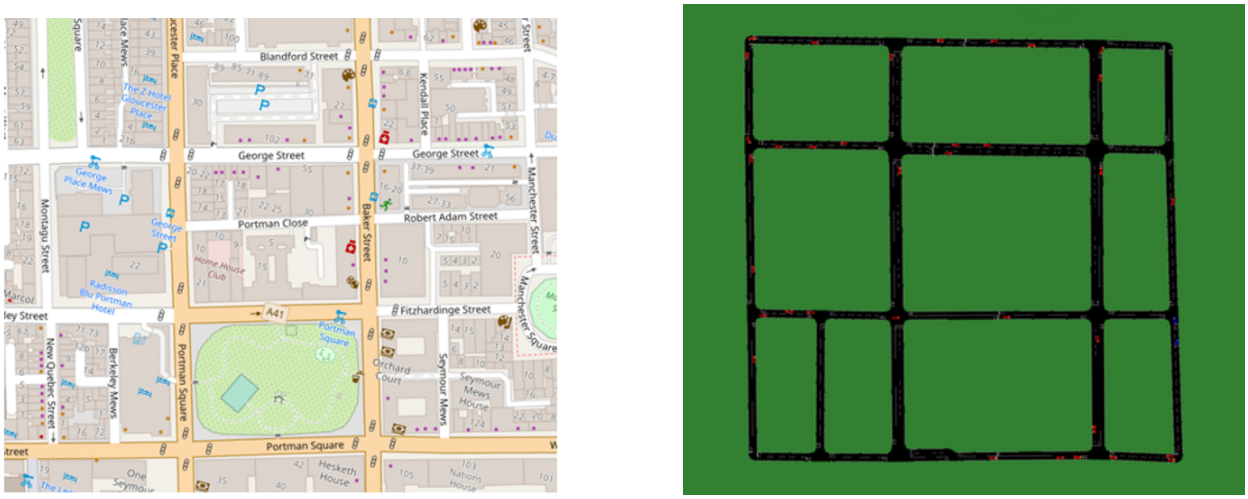

Figure 3.2: Side-by-side comparison of the grid road network in Portman Square, London and the grid road network designed in SUMO. Bus routes, traffic lights, and network plan details included in the appendix.

This network was designed to emulate an interrupted traffic flow scenario with square road geometry. Figure 3.2 shows the designed network next to the real-world grid network found in Portman Square, London. The precise road network plans were obtained by exporting the Open Street Map layout of the London Area onto SUMO via the netconvert tool [33]. When this network was simulated, issues within the traffic simulation caused vehicles to

stop suddenly due to complex road geometry. Important to note that the netconvert tool allows changes to be made to the produced network configuration file by parsing commands to simplify the geometry (*–geometry.remove*), and to merge junctions incorrectly produced by the netconvert tool (*–junctions.join*). The road network and geometry were further simplified to ensure vehicles would be able to explore the entire state space during training. Further details regarding the netconvert tool can be found in [33]. Static traffic lights were then placed at the inner nodes of the network. Actuated traffic lights which use induction loops to trigger phase changes when vehicles are at a close proximity were first implemented, however the training process failed to converge. This is likely due to the fact that the optimal behaviour for CAVs that is required to exploit actuated traffic lights is very complex, and may not have been explored fully during the training process. Similar to the square bus lane network, the initial positioning of vehicles is disturbed from a uniform distribution places using a Gaussian distribution with a standard deviation of 8 meters. The speed limit on all roads is 28.8km/hr (8 m/s). Furthermore, all the static traffic light details, road plans, and bus routes are shown in the Appendix for the sake of completeness. There are nine roads in this network with two lanes, and this will allow the agent to use cooperative driving and strategise to maximise the reward function.

3.2.3 Bristol Triangle

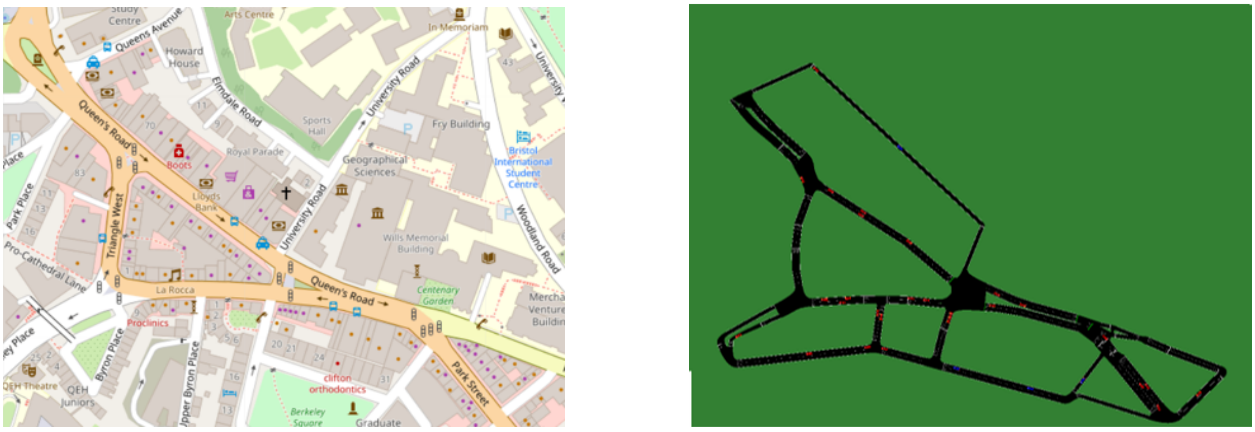

Figure 3.3: Side-by-side comparison of the triangle road network in Clifton, Bristol and the road network designed in SUMO. Bus routes, traffic lights, and network plan details included in the appendix.

The purpose of this network was the see how a trained policy would perform under interrupted flow conditions, with complex road geometry. Unlike the London grid network,

the Bristol triangle has curved roads, sharp turns, road merges, and large intersections. Aside from the netconvert simplifications states in the London grid network section, further manual road road simplification was required at the intersections to prevent vehicles from being unable to move due to improper road connections. Static traffic lights were placed at the major intersections, whose location and phase progression is elaborated in the appendix section. Starting position for vehicles were disturbed from uniform positioning using a Gaussian distribution with a standard deviation of 8 meters. The speed limit on all roads is 28.8km/hr (8 m/s).

3.3 Markov Decision Process

3.3.1 State Representation

Features from the environment should be chosen to embellish the state space and provide sufficient information to the agent for it to develop an optimal policy. Adding more detail to the state space than is necessary will however increase the complexity of the RL training process. One of the specific data features which would be required for the centralised agent policy would be the velocity of every vehicle in the network.

Another feature will be the priority of every vehicle in the network. This will allow the network to learn which vehicles' delay is more significant than others. To pass this feature into the state space, the coefficient of vehicle priority, c_i, for every vehicle is passed to the state space. Non-high priority vehicles will be assigned $c = 0$, and high priority vehicles will be assigned $c > 0$. The two cases which will be studied and compared further are when $c = 1$ and $c = 2$. This will aid in understanding how changing the importance of the high priority vehicles' delays affects the developed policy. How these values are factored in to the MDP will be explained further in section 3.3.3.

Another feature would be the position for all vehicles in the network. One way to track the positions of all vehicles on a small network would be by using a coordinate reference system and track each vehicles' x and y coordinates. The traffic simulation software used does not track the position of vehicles in a coordinate system however, instead only storing the edge - or road - the vehicle is on, the lane its traversing in,and the driving distance from the start of the current edge in meters [34]. The way that this information can be used to build a

coordinate system, is by ordering the edges based on the order that they are defined in the network configuration file, and then summing all the road lengths to produce a value for total road lengths. Since the vehicle's current edge is known, we can sum the lengths of all edges before the current edge and add this to the distance of the vehicle from the start of its current edge to obtain the distance in meters of the vehicle from the start of the first edge. This distance could then be normalised to the total road distances to ensure this state variable does not reach extreme values, as this would greatly increase the training time of the policy. The formula for this position variable could be defined as

$$p_i = \frac{\left(\sum_{j=0}^{t-1} l_j\right) + d_{i,t}}{\sum_{u=0}^{T} l_u} \tag{3.1}$$

where p_i is the position of vehicle i in the new global coordinate system, t is the position of the current edge among the ordered list of edges, l_j is the length of the edge in position j of the ordered edges, $d_{i,t}$ is the distance from vehicle i to the start of its current edge t, and T is the total number of edges in the network.

The lane index for every vehicle will also be used as a component within the state space. This is because two vehicles which are side-by-side on the same edge but on different lanes will return the same value for Equation 3.1, therefore to differentiate the state representation of both of these vehicles, the lane index of every vehicle must be expressed within the state space. The traffic simulation software orders lanes on every road in number based, where the rightmost lane is given the index 0, and subsequent lanes to the left have their lane index incremented by one [34].

Combining the individual features, the state space, S, can be defined as a vector of length n where n is the number of vehicles in the system, and each item is a tuple of the velocity of a particular vehicle, v_i, the priority of the vehicle, c_i, the position of the vehicle as obtained through Equation 3.1, p_i, and the lane index of the vehicle, l_i:

$$S := (v_i, c_i, p_i, l_i) \in \mathbb{R}^n \tag{3.2}$$

Given that the number of inputs of the NN which will be used to represent the policy must remain a fixed size, the state space must not change in size during the simulation.

For this reason vehicle inflows will not be simulated; they would require a dynamic state space to describe a varying number of total vehicles during training. For this reason all the aforementioned network layouts were designed with no incoming or outgoing roads.

3.3.2 Action Space

The action space will define the ways that the centralised agent will be able to control CAVs. Because the learned policy will effectively act as the acceleration and lane changing controller for CAV vehicles then the action space of the centralised agent will be a vector of length k, where each item is a length two tuple of an acceleration term (bounded by the maximum acceleration and deceleration) and a lane changing term which will take continuous and discrete forms, respectively. The action space will therefore be defined as

$$\begin{aligned} A_1 &\in \mathbb{R}^k_{[-a_{max}, a_{max}]} \\ A_2 &:= [-1, 1]^k \\ A &:= (A_1, A_2) \end{aligned} \tag{3.3}$$

where k is the number of CAVs, and $-a_max$ and a_max are the maximum vehicle deceleration and acceleration values, respectively.

3.3.3 Reward Function

The reward function is used to provide feedback to the agent regarding its choices of actions. Therefore the reward function will not only affect the training process of the policy, but also the behaviour of the optimal policy. Stemming from this, the reward function should be defined so that when it is maximised by the optimal policy, the emergent behaviour of the system has the characteristics of the desired behaviour. The objectives of the policy outlined in section 1.2 were to maximise the average vehicle velocity and minimise any delays to the high-priority vehicles.

The first component of the reward function could therefore be represented by the average velocity of the vehicles. This enforces the concept of fairness among non-high-priority vehicles; actions taken by CAVs to increase their velocity should not come at the expense of the velocities of other drivers. This component of the reward function is taken from the flow benchmark paper produced by Vinitsky et al [35], where they utilise it to develop a policy

which maximises velocity across ring, grid,and merging road networks:

$$R_1 := \max\Big(||v_{max} \cdot \mathbb{1}^n||_2 - ||v_{des} - v||_2, 0\Big) \tag{3.4}$$

where v_max is the maximum allowed velocity used to reward vehicles to drive at high speeds, and $v \in \mathbb{R}^n$ is a vector composed of the velocites of all vehicles in the network.
The second component of the reward function will be used to introduce a punishment for any delays attributed to high-priority vehicles. This component could be defined as

$$R_2 := c \sum_{e \in PV} \frac{v_{bus} - v_e}{v_{bus}} \tag{3.5}$$

where PV is the set of all vehicles with high-priority, and v_{des} is a hyperparameter used to define the desired operating velocity for high-priority buses. Furthermore, the coefficient of Equation 3.5, c, is the aforementioned coefficient of vehicle priority which is used to scale the punishment attributed to delays in high priority vehicles. To maintain a constant reward, a punishment incurred from a delay in a bus must be balanced out by a large increase in the average velocities of the non-priority vehicles. This coefficient will be subject to change depending on the types of high-priority vehicles being studied; police cars, ambulances, fire brigades would have a much larger coefficient to signify the much larger punishment that a delay should produce. Further policy training will be conducted for the cases where $c = 1$ and $c = 2$ to further assess the behaviour of the learned policy at different levels of vehicle priorities. The type of high-priority vehicles modelled during each case of these cases will be homogeneous, as only buses will be modelled during each simulation. Therefore c will not vary for high-priority vehicles during a simulation. Increasing c beyond 2 was found to cause the training process to become unstable for the road networks which modelled interrupted flow, as the delay to high-priority vehicles inherently caused by the traffic lights caused the negative punishment to be extremely large. This then led to the training policies to failing to converge. VPG and PPO will both be used to benchmark the learning rate and final reward of the two policy training algorithms.

Reward shaping is a concept in reinforcement learning which aims to provide further guidance to the learning agent to develop a desired optimal behaviour function [36]. This can be done by adding more components to the reward function to provide more feedback for optimal actions. Through efficient reward shaping, the policy training process can converge

more quickly and the sparse credit assignment problem can be overcome.

In the first attempts to train the MDP, one of the prevalent issues in the learned policy behaviour was that CAVs would excessively change lanes even when no advantage came from it. Introducing a scalar punishment for each CAV lane change action, this behaviour was drastically reduced. This first reward shaping component is defined as

$$R_3 := y\Delta LC \tag{3.6}$$

where y is the coefficient of the lane changing punishment, and ΔLC is the lane change actions performed by the agent from the previous time step in the simulation. The coefficient for the lane changing punishment, y was given the value of 10: A value higher than this could lead to the agent being discouraged from making lane-changing actions, which would then mean that the state space is not explored thoroughly.

Another issue which sometimes appeared during the training process was that some vehicles would sometimes remain stopped at a traffic light even when the light had turned green. It is not yet clear why this behaviour persisted during the training process, although it is suspected that these vehicles were gaming the reward function somehow and the policy could have converged to a local maximum during this. To combat this issue, a penalty was introduced to punish vehicles with low velocities. The formula for this component was obtained from literature, in [31], where the authors defined it as

$$\begin{aligned} R_4 &:= \sum_{c \in K} \beta_c \\ \beta_c &= \begin{cases} 0.1 & \text{if } v_c \geq 5 \\ 0 & \text{else} \end{cases} \end{aligned} \tag{3.7}$$

Combining Equations 3.4, 3.5, 3.6, and 3.7 would produce produce the total reward function:

$$R := R_1 - R_2 - R_3 - R_4 \tag{3.8}$$

Chapter 4

Experimental Setup

In section 4.1, the framework used to train policies from traffic simulations through DRL is presented. Section 4.2 will discuss the procedure of modelling human behaviour in the experiments. In section 4.3 a brief outline will be given of the experiments which will be conducted, and section 4.4 will present the hyperparameters used for the DRL training algorithms.

4.1 Simulating Traffic

SUMO is an open-source, microscopic, continuous traffic simulation software with large network capabilities [37]. Static configuration files describing the road network, vehicle demand modelling, and simulation parameters (such as vehicle types and traffic lights) are required for simulation. As a microscopic traffic simulator, SUMO allows for the behaviour of individual vehicles to be modelled using a variety of different explicit controllers. During a simulation SUMO is able to generate data regarding vehicles' velocity, emissions, battery usage, and positions among other vehicle-related information. TraCI is an Application Programming Interface (API) which liaises with SUMO to allow for dynamic re-routing of vehicles during the simulation [38]. Therefore through TraCI actions for vehicles can be chosen and rewards and states from the environment can be obtained during the simulation runtime. The Flow framework developed by Kheterpal et al at UC Berkeley [26] allows for RL libraries to interface with SUMO and TraCI so features of the traffic simulation can be used as parameters for custom Markov Decision Processes. Through Flow, policies can be trained with DRL libraries to behave as learned controllers for vehicle dynamics. Important

to note that all dynamic commands set after the experiment has initialised are conveyed from Flow to SUMO via TraCI.

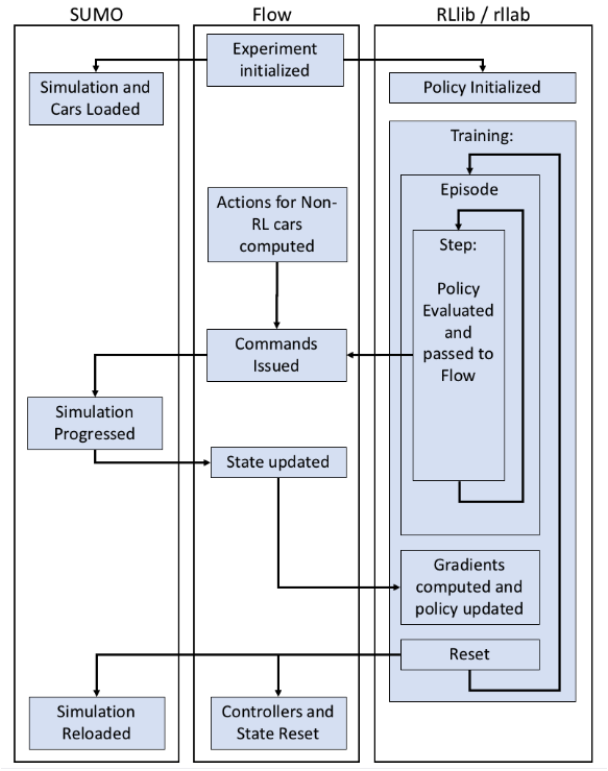

Figure 4.1: Diagram showing the interaction between SUMO and Flow, and how policies can be trained from experiments conducted in SUMO. Communication between SUMO and Flow is mediated by TraCI. [1]

Figure 4.1 shows the interface between SUMO and Flow during simulations and training. Observing the block processes which occur during a training episode, it can be seen that at every simulated time step the policy is evaluated using RLlib - a DRL library - to produce an action. This action is then passed through to SUMO where it is simulated. The environment states are updated based on this action, and this change can be observed by RLlib which utilises the state transition to train and improve the policy. The top branch of processes also show how simulations, experiments, and policies can be initialised and loaded through the Flow framework to replay trained policies and observe their behaviour.

[1] Diagram shown taken from the original Flow publication published by Kheterpal et al [26]

4.2 Modelling Human Behaviour

The behaviour of human-driven vehicles and buses in a variety of different networks and mixed autonomy conditions must be appropriately modelled using accurate microscopic modelling. Furthermore, the learned acceleration and lane changing controller will not be sufficient in providing the complete behaviour for CAVs; traffic assignment modelling is required to give CAVs route instructions.

4.2.1 Microscopic Modelling

Non-CAV vehicles will require explicitly defined controllers using models which can accurately represent the behaviour of human-driven vehicles. The two major classes of of microscopic traffic flow models are car-following models, and Cellular-Automaton models. Cellular-Automaton models discretise time as well as space, dividing road lengths into blocks to simplify the physics of vehicle interactions. While this lowers the computational complexity of the model, it causes a loss of accuracy. Car-following models on the other hand are continuous in time and space and model the dynamics of vehicles using ordinary differential equations. In their report comparing the different car-following models, Kanagaraj et al [39] found that the Intelligent Driver Model (IDM) [40] gives better estimates for vehicle dynamics than Gipps' [41], Das and Asundi [42], or Krauss' models [43], under steady state conditions. Furthermore, the authors in [44] found that IDM modelled vehicle interactions and trajectories more accurately than other-car following models at intersections, and similarly the authors in [45] found that the IDM produced significantly better vehicle acceleration profiles in a merging road network. The IDM is also used extensively in the field of DRL as both the modeller for human behaviour in mixed autonomy environments, and the sole controller for baseline cases of 0% CAV market penetration from which to assess the relative performance of a policy. For these reasons the IDM will be used as the explicit acceleration controller for bus and human-driven vehicles. The equations used in the IDM are:

$$\dot{v} = a\left[1 - \left(\frac{v}{v_0}\right)^{\delta} - \left(\frac{s^{\star}(v, \Delta v)}{s}\right)^{2}\right] \tag{4.1}$$

$$s^{\star}(v, \Delta v) = s_0 + \max\left[0, \left(vT + \frac{v\Delta v}{2\sqrt{ab}}\right)\right]$$

where v is the current velocity, v_0 is the desired velocity, s^* is the desired gap between vehicles, T is the minimum time headway, s_0 is the minimum gap, a is the acceleration, and b is the decceleration. SUMO comes pre-equipped with capabilities of modelling IDM, but requires values for these parameters.

Parameter	Value
Minimum Gap	2.5 $[m]$
Maximum Acceleration	2.9 $[ms^{-2}]$
Comfortable Deceleration	6.0 $[ms^{-2}]$
Emergency Deceleration	9.0 $[ms^{-2}]$
Desired Speed	28.8 $[kmh^{-1}]$
Minimum Time Headway	1.0 $[s]$
Acceleration Exponent	4.0 $[-]$
Step Duration	1.0 $[s]$

Table 4.1: SUMO required parameters for the IDM. Buses and human-driven vehicles will be modelled using the same baseline parameters.

Congestion in the simulations will be modelled by reducing the maximum speed by 15% for 20% of the human-driven vehicles. This will ensure even at 50% CAV penetration rate, congestion will still be modelled among human drivers. Modelling the velocity of human-driven vehicles using a Weibull distribution would improve the accuracy of congestion modelling, however when such a velocity distribution was implemented the mean reward returned by the policy failed to converge on occasions. This could be because the shape factor of the Weibull speed distribution requires precise tuning to ensure that the low velocity vehicles do not bottleneck simulations, and that the high velocity vehicles do not make car-following models volatile.

4.2.2 Route Assignment

Conventional route assignment modelling for vehicles requires trip generation data, which gives information regarding the frequency, origin, and destination of trips, the trip distribution model, which models the commuting fluxes between the origin and destination points, and the mode choice model, which offers information regarding the proportion of commuters travelling through a particular transport mode. The fact that the networks designed must be closed to prevent vehicle inflows and outflows from affecting the state and action space

size means that conventional transportation forecasting models can not be used for route assignment modelling. This will limit the accuracy of a direct comparison between the simulated experiments and real-world traffic; in the real-world, commuters can choose different paths to take so as to minimise the costs associated to their journey, and route assignment can be modelled based on Wardrop's principle of user equilibrium [46].

The authors in [31] utilise a route assignment algorithm for their RL traffic management system which chooses a random route for vehicles when arriving at a junction. In their work to train a policy to manage traffic over a city network, the learned policy was found to converge to a stable value and even outperformed the baseline IDM case on average. A similar routing controller will be used for the London grid and Bristol triangle networks which require route modelling, with adaptations made to route the buses around their pre-planned routes, and to also perform random lane-changes during the routing for human-driven vehicles at intersections. This will ensure sufficient exploration of the state space during training. Buses will not be given lane-changing parameters.

4.3 Experiments

For each of the road networks shown in section 3.2, policies will be trained to manage traffic at 10%, 50%, and 100% CAV market penetration. Furthermore, for each CAV market penetration rate two cases for the coefficient of vehicle priority, c, will be studied: One where $c = 1$ and one where $c = 2$. The relative difference between the two values of c will showcase how the optimal policy behaves when there are different levels of priority among the high-priority vehicles.

Baseline experiments will use the IDM holistically to model traffic at 0% CAV market penetration. Simulations will be run which utilise the policies so the average vehicle velocity and average high-priority vehicle velocity metrics can be presented. These simulations will be repeated 200 times, and their mean value and standard deviation will be documented. The delays for high-priority vehicles relative to their maximum free flow speed will also be measured for each case. During training, vehicles were disturbed from their initial uniform distribution by a Gaussian distribution to improve the ability of the policy to generalise by adding variety, and ensure adequate exploration of the state space. However to keep conditions constant during testing for all cases, the simulated configurations are changed to ensure

	Bus Lane				London Grid				Bristol Triangle			
Pen. Rate	0%	10%	50%	100%	0%	10%	50%	100%	0%	10%	50%	100%
HD Vehicles	10	9	5	0	20	18	10	0	20	18	10	0
CAVs	0	1	5	10	0	2	10	20	0	2	10	20
LP Vehicles	10	10	10	10	20	20	20	20	20	20	20	20
Buses	2	2	2	2	2	2	2	2	1	1	1	1

Table 4.2: Vehicle compositions for each network at each CAV penetration rate. CAV penetration rate is measured by the proportion of CAVs to HD (Human-driven vehicles), so buses or high priority vehicles not included. At 0% penetration rate, all LP (Low-priority) vehicles are controlled by IDM. Vehicle composition will remain the same for both cases of the coefficient of vehicle priority studied.

uniform initial distribution of all vehicles over the network. This way the performance of each policy developed for a network can be more accurately compared to one another.

Fundamental traffic flow theory defines vehicle density as the number of vehicles occupying a unit length of road. Since the number of vehicles for each network and the total road length is constant for each network, the vehicle density is also constant. For the bus lane, London grid, and Bristol triangle networks, the vehicle density is 0.0222, 0.00372, and 0.00600. These values lie within the range of vehicle densities observed in average urban road networks [47].

4.4 Hyperparameters

The initial hyperparameter values were based on the DRL optimisation searches in [35]. Minor parameter tuning was performed to improve the effectiveness of the trained policy. Parameters not specified in Table 4.2 take the value of the default parameters specified by the Flow framework, and these can be found in [48]. The rollout number gives the number of trajectories which will be sampled from the simulation, and the time horizon gives the length of the trajectories. Therefore, the duration of a training iteration is the product of the number of rollouts in a training iteration and the time horizon for each rollout. Important to note that in the event of a collision, the rollout responsible would be terminated. For 200 iterations where the time horizon is 600 and 10 rollouts, the number of simulated time steps is 1.2 million.

Parameters	Training Method	
	VPG	PPO
Time horizon	600	600
Number of rollouts	10	10
Gamma	0.99	0.999
Hidden layers	(128, 128, 128)	(128, 128, 128)
SGD iterations	20	20
Number of workers	8	8
Learning rate	0.0004	-
Lambda	-	0.97
Lambda (GAE)	-	0.97
Clip parameter	-	0.4
Iterations	200	200

Table 4.3: Specific hyperparameters used for the RLlib implementations of PPO and VPG training. Hyperparameters not specified assume default Flow framework values which can be found in [48]

Chapter 5

Results

5.1 Bus Lane Network

5.1.1 Training Performance

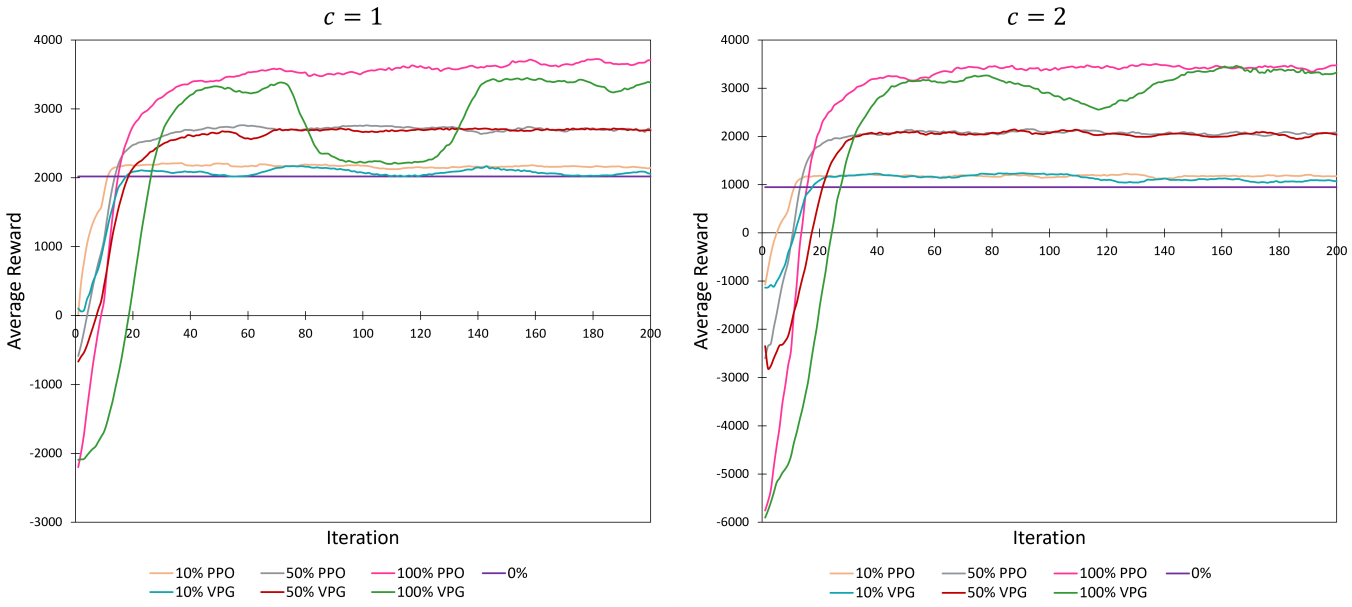

Figure 5.1: Average reward obtained over training iterations for policies implemented in the bus lane network. The two graphs represent the coefficients of vehicle priority $c = 1$ (left), and $c = 2$ (right). For each of the two cases, the training curves for 10%, 50%, and 100% CAV market penetrations are plotted using PPO and VPG training algorithms. The 0% line (purple) represents reward function returned from the Baseline IDM study.

For the bus lane network, Figure 5.1 shows the average reward observed during each iteration of training for the policies obtained through PPO and VPG over a range of CAV penetration rates. By iteration number 200, all curves had smoothed and flattened, indicating that the policy had converged.

The average reward per iteration returned by the converged policies increases with CAV penetration rates. Moreover, the policies trained using a higher coefficient of priority demonstrated a greater improvement in the final average reward over their respective baseline cases: For $c = 1$ in a fully autonomous network, the converged reward is 1.64 times greater than its respective baseline return, yet for $c = 2$, it was around 3 times greater. The numerical training data regarding the mean and the standard deviation of the reward values in the final 10 training iterations for every policy is included as Appendix A.6.

At 100% CAV market penetration for both cases of the vehicle priority coefficient, the VPG training curve shows a dip in the average reward returned at around iteration number 100. Both PPO and VPG are on-policy training algorithms so the experiences collected to train the policy are directly influenced by the policy used during those experience. This can lead to a cascading effect which induces instability when a bad policy change causes the next set of experiences to also be bad. PPO is more robust as it utilises clipping to limit policy changes and limit the cascading issue with on-policy DRL methods. The learning rate discrepancy between VPG and PPO is demonstrated by the fact that the initial gradient for the average reward obtained over training iterations is higher for PPO than for VPG policies. This effect appears more pronounced for the case of 100% CAV market penetration. The initial reward returned by the policy at the start of the training process is inversely proportional to the CAV market penetration rate. This is because a higher CAV market penetration requires a larger action space size which means that the untrained policy that has not yet learnt an accurate policy will make more sub-optimal actions, therefore incurring a higher punishment.

For both cases, at 10% CAV penetration rates the trained policies slightly outperform the 0% CAV baseline in terms of the average reward per iteration. For $c = 1$, the baseline scenario returns a reward of 951, while the PPO and VPG trained policies return a final average reward of around 1180 and 1072, respectively. And for the case of $c = 2$, the baseline scenario returns a reward of 2019, while the PPO and VPG algorithms return a final average reward of around 2144 and 2057, respectively.

The baseline study of 0% in both cases is constant throughout testing iterations, as in contrast with the training configuration, neither the vehicle initial configuration nor the network geometry are changed during testing. Therefore the reduction in baseline performances for $c = 1$ to $c = 2$ is solely due to the greater punishment produced by any delays to the priority vehicles. Interestingly however, at 100% CAV penetration rates the converged mean reward is around 3400 for both $c = 1$ and $c = 2$ in spite of the fact that when $c = 2$ punishments attributed to delays in high-priority vehicles are weighted twice as much.

Shown in the ensuing page, Figure 5.2 shows a series of charts documenting key traffic parameters obtained from the VPG and PPO policies for the varied set of mixed autonomy and priority level conditions. The baseline for both cases returns constant values with no standard deviation. This relates to the fact that for interrupted flow, the traffic flow is governed solely by road geometry and vehicle dynamics hence there is little entropy in the simple network. In contrast to this, the learned controller chooses actions based on a stochastic policy. Continuing to train the policy for more iterations would reduce the entropy of the agent's action space, consequently causing it to overfit to the training samples.

The two coefficients of vehicle priority were grouped together in the bar chart shown by Figure 5.2 to help compare the differences in the behaviour observed when the coefficient of vehicle priority is raised.

As seen by Figure 5.2, for both values of c the average vehicle speed increased with the CAV penetration rate. At the baseline IDM case, the average vehicle speed is 5.94 m/s. Taking the mean of the observed vehicle average speed for all the policies trained at each market penetration level, the learned controllers outperform the base IDM case by 1%, 4.9%, and 11.3% at 10%, 50%, and 100% CAV penetration rates, respectively.
In a fully autonomous environment, the policy trained through PPO which uses $c = 2$ showed an improvement in the average speed of vehicles in the network by over 13% in comparison to baseline case using the IDM. Increasing the coefficient of vehicle priority from $c = 1$ to $c = 2$ led to a notable increase in the average speed of vehicles at 100% CAV market penetration, particularly in the case for the VPG policy where it showed an improvement of 5%, or 0.31 m/s. In most cases, the policies developed for this network using PPO and VPG demonstrate similar traffic parameters, and show similar trends over the varying degrees of

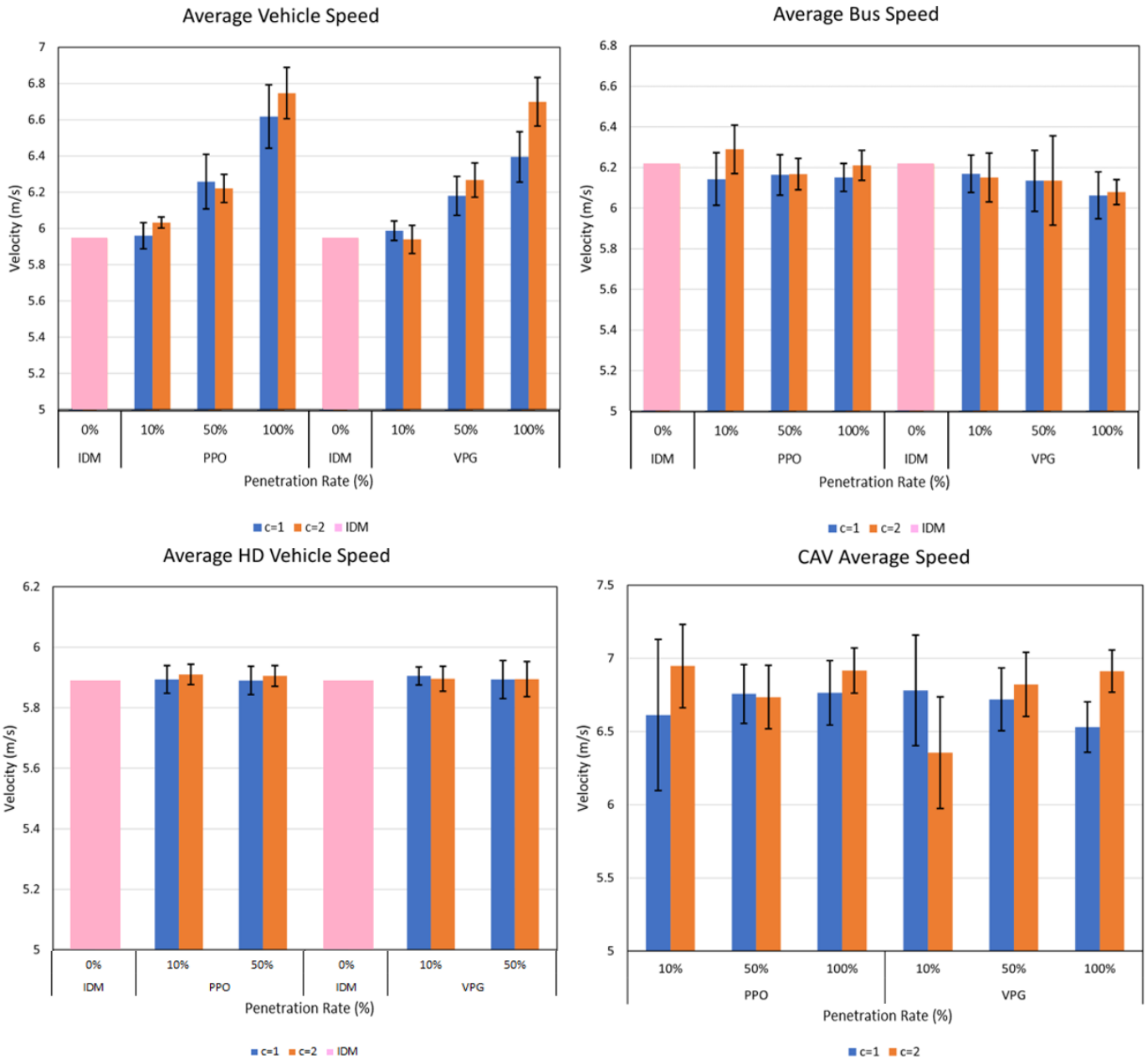

Figure 5.2: Performance of the learned policies for the bus lane network plotted for various traffic metrics. The average vehicle speed (top left), average bus speed (top right), average HD vehicle speed (bottom left), and CAV average speed (bottom right) metrics were averaged out from samples of 100 simulations of the trained policies, and error bars are used to show their standard deviation. The average vehicle speed metric is equally weighted over all the types of vehicles. The pink bar represents the baseline case, with only HD vehicles and buses.

mixed autonomy.

The velocity of the RL controlled vehicles was much faster than the human-modelled vehicles, reaching average speeds of up to 6.9 m/s while HD vehicles stay around 5.89 m/s for all simulations. This is to be expected, as the congestion modelled in human vehicles causes a bottleneck which can only be overcome by performing lane-changing actions.

Neither the average bus speed nor the average human-driven vehicle speed vary over the iterations as these parameters remain close to the baseline case for all scenarios. This is partly attributed to the simplicity of the network; their behaviour as modelled by the IDM would stay constant if not for the effects of the CAVs. Therefore even if the CAVs acted optimally the speed of buses would not increase.

An interesting result observed from a PPO policy is that at 10% CAV market penetration the average bus speed observed is higher than the baseline case. The standard deviation of the average bus speeds also shows that buses on multiple occasions exceeded the baseline speed of 6.22 m/s. For this network, the maximum speed for buses was hypothesised to be observed in the baseline case with no CAVs as there would be a minimum amount of congestion in the bus lane when there are no lane-switching vehicles. During testing simulations, the buses were initially uniformly distributed across the bus lane meaning they would start on opposite edges of the network. In the baseline simulation it was observed that the trailing bus would catch up to the leading bus due to car-following models wanting to reach a desired gap between leading and trailing vehicles [39]. When the vehicles caught up to one another, it would cause congestion in the bus lane, especially at corners where the trailing vehicle had to break early at corners to allow the leading vehicle to turn. This congestion would ensue for the rest of the simulation as buses were not permitted to lane change or overtake.

An emergent behaviour occasionally exhibited by the agent was that it would lower its speed when in front of a bus for a short period, then accelerate forwards and move on. Because lane-changing and priority vehicle delays incurs a punishment, if it led to no reward or lower punishment in the future, the agent would learn to avoid this action. If the trained policy did indeed converge to a global maximum instead of a local one, then this behaviour could be a sign of the agent gaming the car-following models to incur a small punishment in the present moment to avoid future unavoidable congestion in the bus lane. This would also

explain the higher average bus speed occasionally observed in mixed autonomy environments.

For this network, the policies developed through PPO obtained better traffic metrics than those trained with VPG.

As the wide error bars in the CAV average speed graph suggests, the variance of the average CAV speed metric is much higher relative to the variance of the other traffic parameters. However, the fact that the reward observed of the trained policies are smooth shows that although the CAV's acceleration actions have a larger range than the IDM vehicles, the centralised agent is strategising and employing cooperative driving to maintain a stable reward.

5.2 London Grid

5.2.1 Training Performance

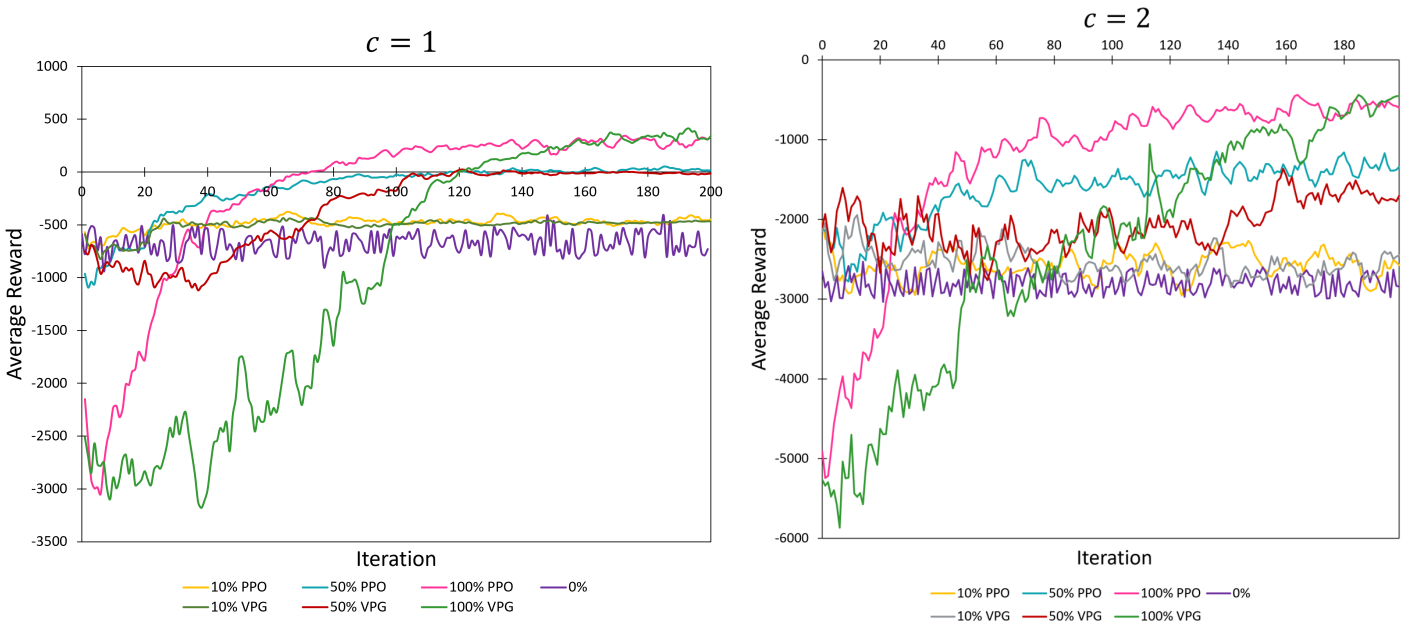

Figure 5.3: Average reward obtained over training iterations for policies implemented in the London grid network. The two graphs represent the coefficients of vehicle priority $c = 1$ (left), and $c = 2$ (right). For each of the two cases, the training curves for 10%, 50%, and 100% CAV market penetrations are plotted using PPO and VPG training algorithms. The 0% line (purple) represents reward function returned from the Baseline IDM study.

Figure 5.3 shows the training process for the London grid network policies. Unlike the training process for the bus lane network there is noise present in the average reward observed during training for all cases. Even the baseline case which utilises the IDM is subject to high-frequency oscillation in its average reward per iteration.

A factor which contributes to this is the routing controller. As previously mentioned, the traffic controller reroutes vehicles when they reach a junction to a new random destination. The unpredictability of the vehicle router means that consecutive simulations will be different, even when acting based on an explicit controller such as the IDM.

Another factor responsible for the changing conditions between training simulations are the traffic lights. Although their phases and duration are fixed, their yellow light duration is dynamic and dependent on vehicle path modelling. Because the routing algorithm used for path modelling acts randomly, then the traffic light behaviour will also be stochastic. The presence of entropy does not decrease the accuracy of the model, as it is well understood that traffic in the real world is not deterministic.

The policy trained in the 10% CAV penetration rate environment outperforms the baseline IDM model with respect to the observed reward function, and the average reward per iteration obtained by the converged policies increases with CAV market penetration.

Analysing the training process of the policies in Figure 5.3, it can be seen that the final average rewards are much lower than in the previous network, as the case where $c = 2$ reaches as far as -2641.8. This is because buses waiting at traffic light junctions will receive a large, negative reward. When the coefficient of vehicle priority is higher, then the punishment is greater. The policies shown still manage to converge in their objective to maximise the reward.

The policies trained using $c = 1$ have much smoother curves and are flatter at the end of the training process than the policies trained using $c = 1$. Table A.6, which states the mean and standard deviation of the final reward values for each policy, shows that all bar two of the trained policies exhibit lower standard deviation in their average rewards than the baseline IDM cases. This shows that the agent has developed a policy robust enough to maximise the reward function in a stochastic environment and inhibit noise in the reward signal.

Figure 5.4 shows the performance of the policies trained in the London grid network with respect to key traffic metrics. From the average vehicle speed graph it can be seen that increasing the proportion of CAV vehicles in the network substantially increases the average speed of vehicles. At the baseline IDM case, the average vehicle speed is 4.99 m/s. Taking the mean of the observed vehicle average speed for all the policies trained at each market penetration level, the learned controllers outperform the base IDM case by 3.6%, 9.7%, and 12.8% at 10%, 50%, and 100% CAV penetration rates, respectively.

In a fully autonomous environment, the policy trained through PPO which uses $c = 2$ showed an improvement in the average speed of vehicles in the network by over 13% in comparison to baseline case using the IDM. Increasing the coefficient of vehicle priority from $c = 1$ to $c = 2$ led to a notable increase in the average speed of vehicles at 100% CAV market penetration, particularly in the case for the VPG policy where it showed an improvement of 5%, or 0.31 m/s.
The positive effects of cooperative driving through DRL on high-priority vehicles are showcased by the average bus speed graph shown in the top right of Figure 5.4. In the base case with no CAVs present in the network, the average speed of the buses is 4.7 m/s. Averaging this traffic metric over both cases of c and both training algorithms shows an improvement in the average bus speed relative to the baseline case of 17.9%, 29.1%, and 34.63% at 10%, 50%, and 100% CAV penetration rates, respectively. Based on this, for a journey the length of a kilometer such an improvement would lower the delay of the high-priority vehicle at the aforementioned CAV penetration levels by 32.5, 48.4, and 55.2 seconds, respectively.

On average, the policies developed through PPO obtained higher values in the showcased traffic metrics than those trained with VPG. The two cases for the coefficient of vehicle priority assigned to the reward function - namely $c = 1$ and $c = 2$ - do not demonstrate a direct correlation with the average bus speed variable observed from the trained policies.

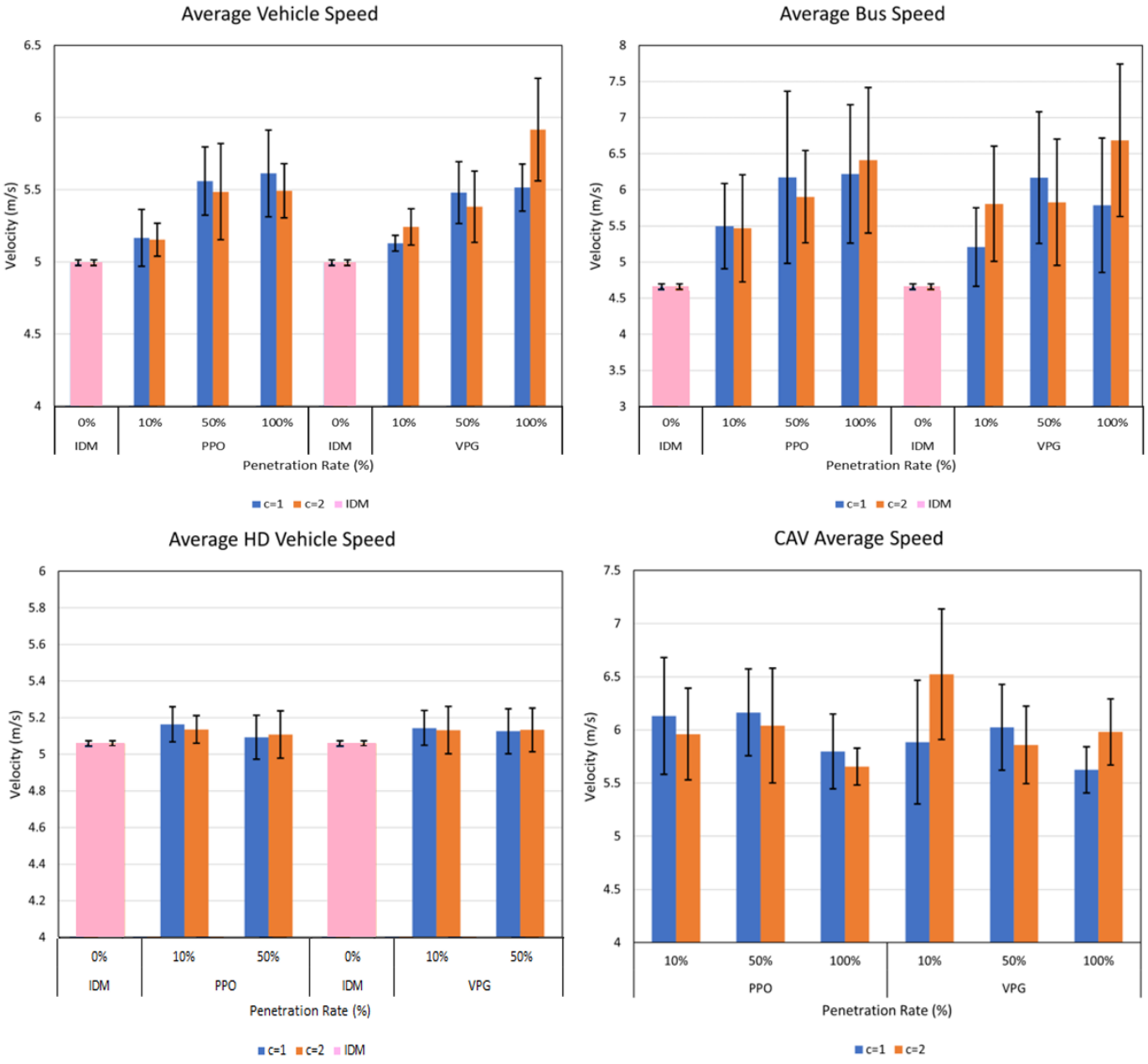

Figure 5.4: Performance of the learned policies for the London grid network plotted for various traffic metrics. The average vehicle speed (top left), average bus speed (top right), average HD vehicle speed (bottom left), and CAV average speed (bottom right) metrics were averaged out from samples of 100 simulations of the trained policies, and error bars are used to show their standard deviation. The average vehicle speed metric is equally weighted over all the types of vehicles. Finally, the pink bar represents the baseline experiment.

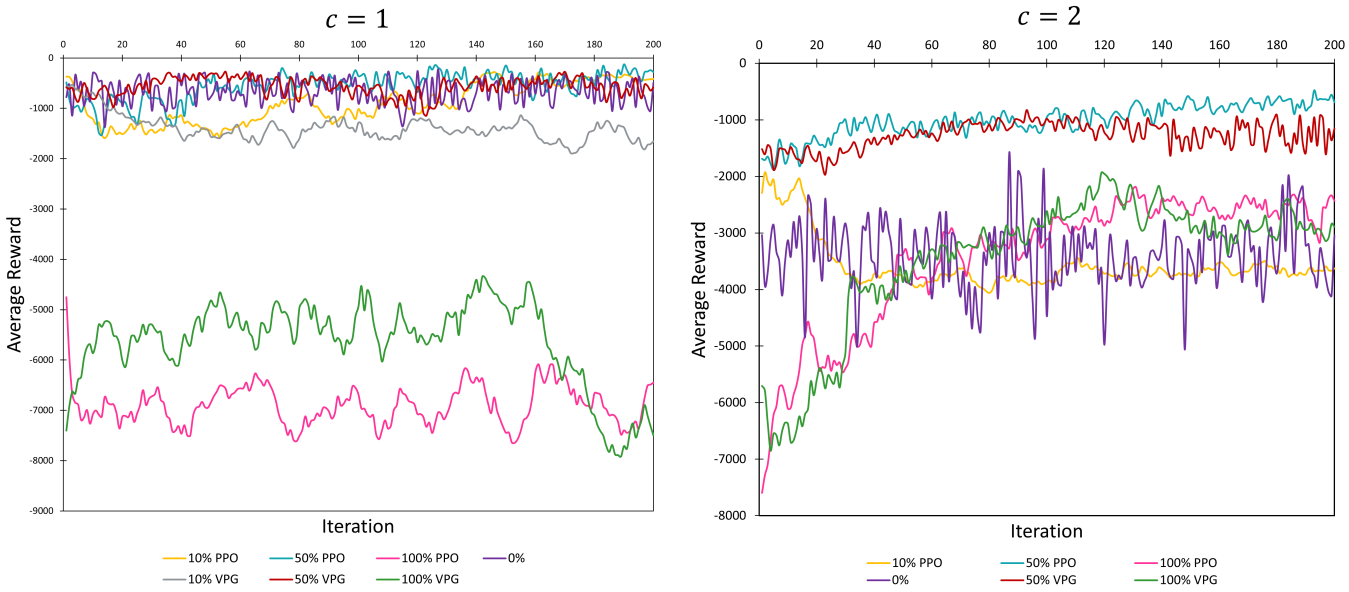

Figure 5.5: Average reward obtained over training iterations for policies implemented in the Bristol triangle network. The two graphs represent the coefficients of vehicle priority $c = 1$ (left), and $c = 2$ (right). For each of the two cases, the training curves for 10%, 50%, and 100% CAV market penetrations are plotted using PPO and VPG training algorithms. The 0% line (purple) represents reward function returned from the Baseline IDM study.

5.3 Bristol Triangle

5.3.1 Training Performance

Finally, for the Bristol triangle network, Figure 5.5 shows the average reward returned during the training of the policy using PPO and VPG for a range of CAV penetration rates. Multiple trained policies failed to smoothen or increase their average reward as they approached iteration number 200. In the case of $c = 1$, the policies appear to stagnate completely in their performance. At 100% CAV penetration rate, the policies initially returned an extremely low reward of around -7000, and continued to oscillate around this point.

The training curve for the policies using $c = 2$ showed marginally better performance; the policies at 100% market penetration rose above the IDM baseline, and the average reward returned by the policies in 50% mixed autonomy environments were higher substantially higher than the baseline. The 10% CAV penetration rate policy fell below even the mean IDM return.

The interaction between the lane-changing controller and the routing algorithm caused many

issues during simulation. At the complex junction structures found in the network, the lane-changing controller would, on occasion, perform actions which meant their routed path was inaccessible. If the routing algorithm assigned a route based on the connections of the CAVs current lane, and then the CAV changed lanes to one that was no longer connected to its assigned path, the vehicle would halt indefinitely. The network was simplified, but the interactions between complex road structures and the CAVs were intricate. The increased oscillations shown by the IDM baseline case in $c = 2$ acts as evidence that the state space exploration process is very sensitive.

Furthermore, attempts to train the VPG policy at 10% market penetration failed due to insufficient memory allocated to the RLlib processes. Out of all the experiments, this scenario was the only one which posited hardware issues. For this scenario, only the PPO trained policy is documented.

Figure 5.6 shows the performance of the policies trained in the Bristol triangle network with respect to key traffic metrics. Based on the training processes for the policies shown in Figure 5.5, it is likely that the policies developed are not well optimised, and therefore the traffic parameters obtained from them do not represent their optimal behaviour. The developed policies do show behaviour which maximises the reward function, as they all exhibit higher average bus speeds than the baseline case. For the VPG policy functioning at 50% mixed autonomy the average bus speeds observed are nearly 20% larger than the average speed in the baseline case, which is 5.35 m/s.

Unlike the policies trained in the other networks, the observed average vehicle speed for all cases is inversely proportional to the market penetration rate, bar the policy using VPG and $c = 2$. Furthermore, the IDM baseline experiment showcases a higher average vehicle speed than in any other case with mixed autonomy. The average speed of the buses does appear to increase with the CAV market penetration rate, performing better than the baseline case.

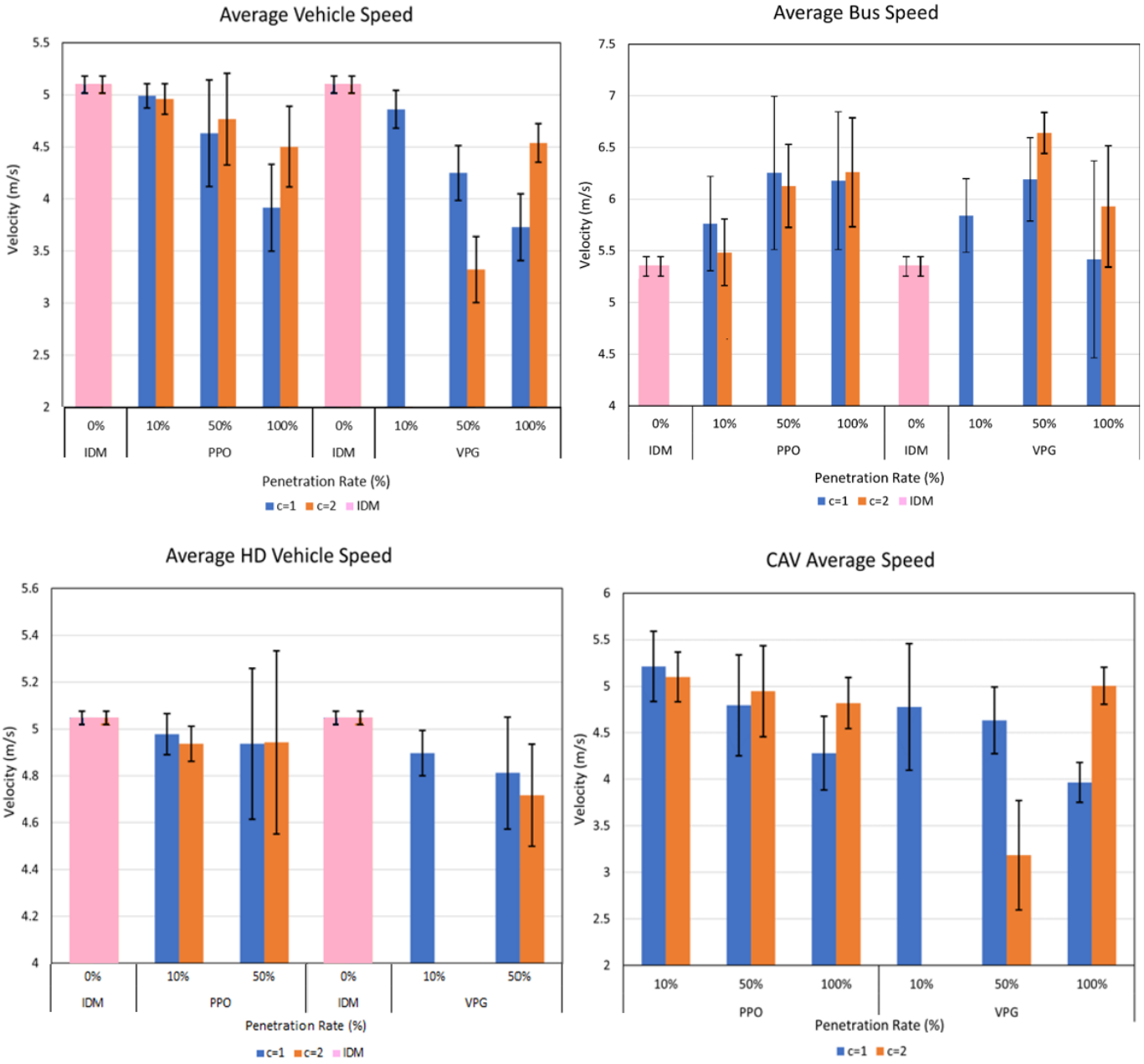

Figure 5.6: Performance of the learned policies for the Bristol triangle network plotted for various traffic metrics. The average vehicle speed (top left), average bus speed (top right), average HD vehicle speed (bottom left), and CAV average speed (bottom right) metrics were averaged out from samples of 100 simulations of the trained policies, and error bars are used to show their standard deviation. The average vehicle speed metric is equally weighted over all the types of vehicles. Finally, the pink bar represents the baseline experiment.

Chapter 6

Discussion

6.1 Discussion

In this thesis, the proposed approach for an agent which maximises the average vehicle velocity and minimises delays to the high-priority vehicles was realised. Three road networks were designed with varying degrees of road complexities. Two DRL algorithms, namely PPO and VPG, were used to train the policies for each network at varying degrees of CAV market penetration. They were also used to benchmark each other's training process, and corroborate observed relationships between traffic parameters, market penetration rates, and the coefficients of vehicle priority.

In the bus lane network, RL vehicles would be able to intelligently overtake the modelled congestion among human-driven vehicles by performing lane-changing actions onto the bus lane as long as this did not incur an unnecessary delay on the buses. Doubling the coefficient of vehicle priority, which has the effect of scaling the punishment given to high-priority vehicle delays, had the effect of increasing the average vehicle speed in the fully autonomous environments. So although the CAVs would consider the effects of delays on buses, they had no way of significantly improving the driving conditions for high-priority vehicles compared to their baseline flow when no CAVs were present.
The complex geometry in the London grid network allows for CAVs to perform intelligent lane-changing actions. The results of the simulated optimal PPO and VPG policies in the grid network show that as the CAV penetration rate rises, so does the average vehicle and bus speed in the network.

The third network studied possessed road structures more complex than in the previous scenarios. The policy training curves, as shown in Figure 5.5, do not appear to have converged by iteration number 200. Issues encountered between the routing algorithm used and the lane-changing capabilities of the CAVs cause issues which may slow the learning process for the agent. The trained policy showed worse average vehicle speed than the base case, but showed a higher average bus speed which scales with CAV penetration rate.

Improvements in the modelling of congestion among human-driven vehicles would improve the applicability of the developed solution to real-world problems. Furthermore, a routing controller able to dynamically route vehicles based on the lowest cost path, or other modelling theory would improve the accuracy of the simulation.

Although the average reward curve for many policies appears to converge over training, it is not known whether or not the policies have converged at the optimal solution. Unnecessary lane-changing behaviour is characteristic of all the policies simulated. Intuitively, the optimally behaving policy would not need to unnecessarily perform multiple lane-changing actions during a simulation given that a scalar punishment is associated with it.
A suggested area for future work lies in implementing a Multi-Agent RL approach, where instead of a centralised agent choosing actions for all the CAVs, a Multi-Agent approach would set each individual CAV as the agent with its fixed action space, interacting with other agents to learn and develop a policy shared by other agents. This approach also tackles the challenge of centralisation with regards to CAV control. Although difficult to implement, a Multi-Agent approach would allow for inflows and outflows of vehicles to be simulated during the training of a policy.
Furthermore, the performance of the trained policies were not tested in different scenarios other than the one's they were trained in. The ability of an agent to generalise to conditions outside of the training shows that it did not over fit to the scenarios it was trained on. Future work could be aimed at testing the trained policies in road networks with varying degrees of complexity where high-priority vehicles other than those simulated during training are present.
Developing a model for vehicles with dynamic coefficients of vehicle priority that can change the behaviour of CAVs when activated would aid in modelling more complex emergency scenarios, such as police cars and ambulances.

Appendix A

Appendix

Derivation of Policy Gradient Equation

The Derivation along with a more detailed guide can be found in [11].

First helpful fact, probability of a given trajectory $\tau = (s_0, a_0, \ldots, s_{T+1}$, from π_θ:

$$P(\tau|\theta) = \rho_0(s_0) \prod_{t=0}^{T} P(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$$

Second helpful fact, Log-Derivative Trick:

$$\nabla_\theta P(\tau|\theta) = P(\tau|\theta)\nabla_\theta log P(\tau|\theta)$$

Third helpful fact, Log-Probability of a Trajectory:

$$log P(\tau|\theta) = log \rho_0(s_0) + \sum_{t=0}^{T} (\log P(s_{t+1}|s_t, a_t) + \log \pi_\theta(a_t|s_t))$$

Fourth helpful fact, Gradients of Environment Functions:

Environment is not a function of θ, therefore gradients of $\rho_0(s_0)$,$P(s_{t+1}|s_t, a_T)$ and $R(\tau)$ are zero.
Fifth helpful fact, Grad-Log-Prob of a Trajectory:

$$\nabla_\theta \log P(\tau|\theta) = \cancel{\nabla_\theta \log \rho_0(s_0)} + \sum_{t+0}^{T} (\cancel{\nabla_\theta \log P(s_{t+1}|s_t, a_t)} + \nabla_\theta \log \pi_\theta(a_t|s_t)) = \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)$$

The complete derivation using these facts, is given by

$$\nabla_\theta J(\pi_\theta) = \nabla_\theta \underset{(\tau\sim\pi_\theta)}{E}[R(\tau)], \text{Expand Expectation}$$
$$= \nabla_\theta \int_\tau P(\tau|\theta)R(\tau), \text{Bring gradient under integral}$$
$$= \int_\tau P(\tau|\theta)\nabla_\theta \log P(\tau|\theta)R(\tau), \text{Log-derivative trick}$$
$$= \underset{\tau\sim\pi_\theta}{E}[\nabla_\theta \log P(\tau|\theta)R(\tau)], \text{Return to expectation form}$$
$$\therefore \nabla_\theta J(\pi_\theta) = \underset{\tau\sim\pi_\theta}{E}[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)R(\tau)], \text{Expression for grad-log-prob}$$

Temporal Different (TD) learning is a method originally described by Richard S. Sutton [49] to estimate a variable that depends on future values of a signal. More information regarding TD is given in [21]:

Phases for Traffic Lights and Junction Positions in London Grid Network

```
#Top Left Inner Node
phases_S_15 = [
    {"duration": "42", "state": "rrrGgggGgrrrGGg"},
    {"duration": "3", "state": "rrryyyyyyrrryyy"},
    {"duration": "42", "state": "GGgrrrrrrGGgrrr"},
    {"duration": "3", "state": "yyyrrrrrryyyrrr"}
]

#Top Right Inner Node
phases_S_17 = [
    {"duration": "42.00", "state": "rrrGGgrrrrrrGgggGg"},
    {"duration": "3", "state": "rrryyyrrrrrryyyyyy"},
    {"duration": "42.00", "state": "GGgrrrGgggGgrrrrrr"},
    {"duration": "3", "state": "yyyrrryyyyyyrrrrrr"},
]

#Bottom Left Inner Node
phases_S_10 = [
    {"duration": "42", "state": "rrrGgggGgrrrGGg"},
    {"duration": "3", "state": "rrryyyyyyrrryyy"},
    {"duration": "42", "state": "GGgrrrrrrGGgrrr"},
    {"duration": "3", "state": "yyyrrrrrryyyrrr"}
]

#Bottom Right Inner Node
phases_S_12 = [
    {"duration": "42", "state": "rrrGGgrrrGgggGg"},
    {"duration": "3", "state": "rrryyyrrryyyyyy"},
    {"duration": "42", "state": "GGgrrrGGgrrrrrr"},
    {"duration": "3", "state": "yyyrrryyyrrrrrr"}
]
```

Figure A.1: Static Traffic Light Phases for London Grid Network.

Software Used in Final Year Projects

- **All software** used in your project work must be declared and attributed to a source – even if you have written it entirely yourself from scratch.
- Below is an example of what is required. You will need to modify this for your own project requirements.
- For the Project Thesis, include this information as an Appendix in a similar format

Note:

- For all software algorithms/code the original source of the code together with details of modifications made by the student must be stated in the code header.

Filename/ Algorithm/ Package	Supplier/Source/ Author/ website	Use/Modifications made/ Student written
Flow Anaconda Environment	Flow, developed by UC Berkeley. https://flow.readthedocs.io/en/latest/flow_setup.html	Environment used as it includes RLlib, TraCI, SUMO, tensorboard. Added CUDA 10.0.130.1, cuDNN v7.4.2, and tf-gpu 2.0.0
Minicity Router	https://github.com/flow-project/flow/blob/master/flow/controllers/routing_controllers.py, particularly only the minicity router class	Adapted to include bus routes for different networks, see Thesis section 3.3
Visualiser	https://github.com/flow-project/flow/blob/	Code supplied by <supervisor> - nmodified to return average velocities for all vehicle types
Training	Python,	Student-written features for *DeptCode.m*
Bus lane network, London grid network, Bristol triangle networkhttps://github.com/flow-project/flow/blob/	https://github.com/flow-project/flow/blob/	Designed and coded by student.
Bus lane environmenthttps://github.com/flow-project/flow/blob/	https://github.com/flow-project/flow/blob/	Environment designed by student, some components taken from literature (see Thesis section 3.3).
Tensorboard		
Initial configuration	*https://github.com/flow-project/flow/blob/*	*Each network had a custom configuration*
Baseline experiments	*https://github.com/flow-project/flow/blob/*	*Stable baselines*
Experiment	*https://github.com/flow-project/flow/blob/*	*Modified to collect data over iterated experiments*

Figure A.2: Software Declaration.

```
def specify_nodes(self, net_params):
    """See parent class."""
    nodes = [
            {'id': 'S_1', 'x': 427.41, 'y': 152.54},
            {'id': 'S_2', 'x': 474.97, 'y': 167.35},
            {'id': 'S_3', 'x': 529.01, 'y': 184.61},
            {'id': 'S_4', 'x': 568.76, 'y': 198.26},
            {'id': 'S_5', 'x': 599.64, 'y': 208.86},
            {'id': 'S_6', 'x': 672.62, 'y': 233.28},
            {'id': 'S_7', 'x': 728.35, 'y': 248.69},

            {'id': 'S_8', 'x': 392.56, 'y': 255.93},
            {'id': 'S_9', 'x': 441.89, 'y': 270.94},
            {'id': 'S_10', 'x': 494.53, 'y': 286.99},
            {'id': 'S_11', 'x': 567.58, 'y': 311.32},
            {'id': 'S_12', 'x': 638.80, 'y': 334.57},
            {'id': 'S_13', 'x': 691.32, 'y': 351.68},

            {'id': 'S_14', 'x': 351.16, 'y': 377.83},
            {'id': 'S_15', 'x': 453.14, 'y': 409.62},
            {'id': 'S_16', 'x': 482.10, 'y': 418.32},
            {'id': 'S_17', 'x': 599.19, 'y': 454.76},
            {'id': 'S_18', 'x': 649.29, 'y': 470.00},

            {'id': 'S_18b', 'x': 326.46, 'y': 453.53},
            {'id': 'S_19', 'x': 428.96, 'y': 485.11},
            {'id': 'S_20', 'x': 479.64, 'y': 500.82},
            {'id': 'S_21', 'x': 573.69, 'y': 530.52},
            {'id': 'S_22', 'x': 624.15, 'y': 545.25},
    ]
```

Figure A.3: Junctions in SUMO are called nodes, and unlike vehicle positions, nodes do utilise Cartesian coordinates. These are the coordinates for all nodes in the network. Edge geometry can be obtained from these values; edges connect nodes together.

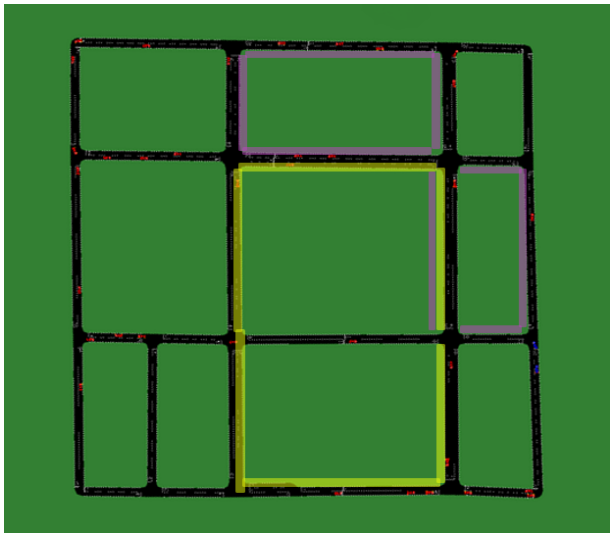

Figure A.4: Yellow and purple Lines show the periodic bus routes for the London grid network.

			Bus Lane		London Grid		Bristol Triangle	
			Mean	STD	Mean	STD	Mean	STD
1	0%	IDM	2019.21718	0	-668.0270771	100.4946013	-662.12	247.0498948
	10%	PPO	2156.842986	8.52382382	-458.6970696	26.92512907	-392.7877762	40.68238128
		VPG	2056.420642	21.40892868	-471.9923326	5.60301229	-1507.326976	116.5469752
	50%	PPO	2698.050077	12.68676596	26.62963265	10.71824438	-325.0637422	332.6465345
		VPG	2700.758152	8.924665454	-14.55479508	4.980084607	-619.8195285	116.5469752
	100%	PPO	3675.009967	28.23665156	280.0728671	34.43536216	-7016.595546	332.6465345
		VPG	3316.516724	47.17778163	345.3398795	29.8886996	-7492.724348	285.6974704
2	0	IDM	951.394	0	-2808.04	113.7786428	-3392.88	582.2591262
	0.1	PPO	1187.147573	9.436083905	-2641.753191	145.9754659	-3675.699172	52.2143091
		VPG	1078.493598	14.068927	-2546.020108	113.8413908		
	0.5	PPO	2066.090788	14.63101345	-1328.825193	87.78435346	-675.409337	92.95448482
		VPG	2018.380618	28.33814702	-1680.655815	77.62691739	-1259.587927	229.9169783
	1	PPO	3423.230489	35.81824928	-561.7392002	28.57808648	-2583.147576	228.8010094
		VPG	3336.427515	16.40494555	-543.5197591	81.48821596	-2675.156014	261.0197252

Figure A.5: Means and standard deviations for the last 10 iterations of the training process for every policy.

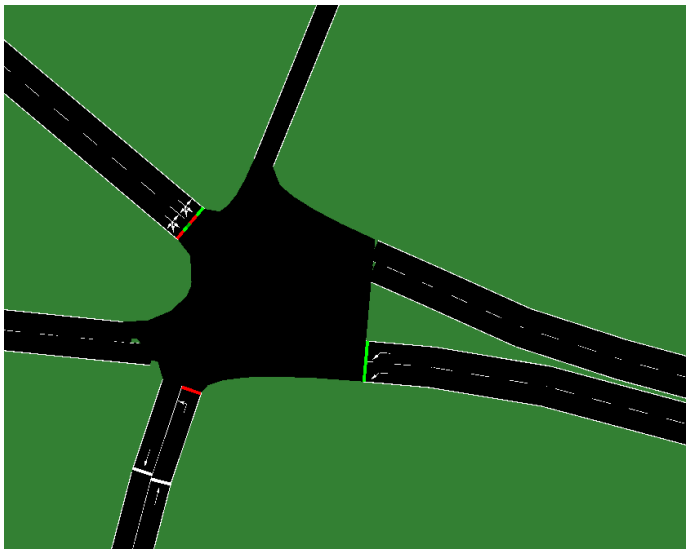

Figure A.6: Problematic Bristol triangle junction.

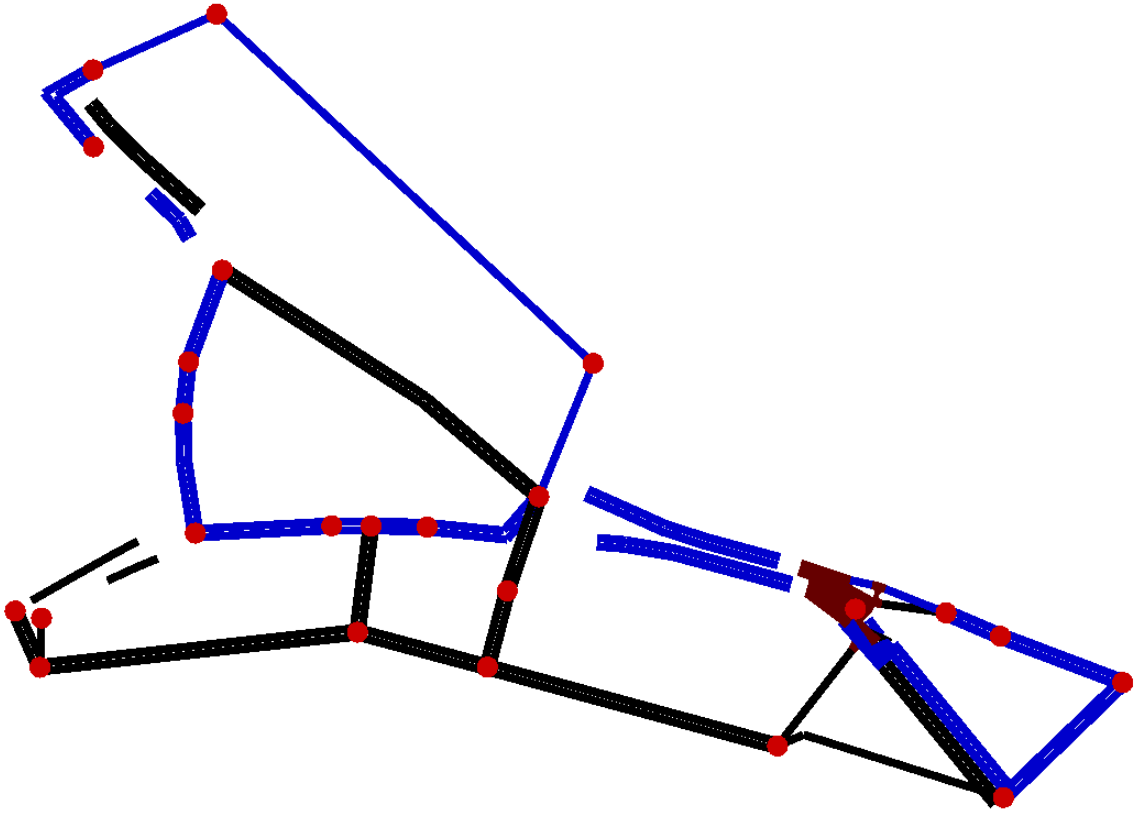

Figure A.7: Bus Route for Bristol triangle.

Bibliography

[1] Richard Ratcliffe. Inrix global traffic scorecard: Congestion cost uk economy £6.9 billion in 2019, Mar 2020. URL `https://inrix.com/press-releases/2019-traffic-scorecard-uk/`.

[2] Ian Shergold, myles wilson, and Graham Parkhurst. The mobility of older people, and the future role of connnected autonomous vehicles a literature review. sept 2016. 09 2016.

[3] John Leech, Gerard Whelan, and Mukarram Bhaiji. *Connected and Autonomous Vehicles - The UK Economic Opportunity*. KPMG, 2021.

[4] In The Brain and F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization.

[5] B. Widrow and M. E. Hoff. Adaptive switching circuits. 1960.

[6] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, October 1986. doi: 10.1038/323533a0. URL `https://doi.org/10.1038/323533a0`.

[7] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, 2010.

[8] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[9] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. 2010.

[10] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[11] Joshua Achiam. Spinning Up in Deep Reinforcement Learning. 2018.

[12] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. 2013.

[13] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. 2018.

[14] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. 2017.

[15] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. 2016.

[16] Satinder Singh Yishay Mansour Richard S. Sutton, David McAllester. Policy gradient methods for reinforcement leanring with function approximation. 1999.

[17] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization. 2017.

[18] Nicolas Heess, Dhruva TB, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, S. M. Ali Eslami, Martin Riedmiller, and David Silver. Emergence of locomotion behaviours in rich environments. 2017.

[19] Marvin Minsky. Steps toward artificial intelligence. 1963.

[20] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-Dimensional Continuous Control Using Generalized Advantage Estimation. *arXiv:1506.02438 [cs]*, October 2018.

[21] A. G. Barto. Temporal difference learning. *Scholarpedia*, 2(11):1604, 2007. doi: 10.4249/scholarpedia.1604.

[22] Tom Breloff. Deep reinforcement learning with online generalized advantage estimation. 2016. URL `http://www.breloff.com/DeepRL-OnlineGAE/`.

[23] Alkis Papadoulis, Mohammed Quddus, and Marianna Imprialou. Evaluating the safety impact of connected and autonomous vehicles on motorways. *Accident Analysis Prevention*, 124:12–22, 2019. ISSN 0001-4575. doi: https://doi.org/10.1016/j.aap.2018.12.019. URL `https://www.sciencedirect.com/science/article/pii/S0001457518306018`.

[24] Ioannis Mavromatis, Andrea Tassi, Robert J. Piechocki, and Mahesh Sooriyabandara. On urban traffic flow benefits of connected and automated vehicles, 2020.

[25] Haoran Wei, Shawn Liu, Lena Mashayekhy, and Keith Decker. Mixed-autonomy traffic control with proximal policy optimization. pages 1–8, 12 2019. doi: 10.1109/VNC48660.2019.9062809.

[26] Cathy Wu, Aboudy Kreidieh, Kanaad Parvate, Eugene Vinitsky, and Alexandre M Bayen. Flow: A modular learning framework for autonomy in traffic, 2020.

[27] Duy Quang Tran and Sang-Hoon Bae. Proximal policy optimization through a deep reinforcement learning framework for multiple autonomous vehicles at a non-signalized intersection. *Applied Sciences*, 10(16), 2020. ISSN 2076-3417. URL https://www.mdpi.com/2076-3417/10/16/5722.

[28] A. Vidali, L. Crociani, Giuseppe Vizzari, and S. Bandini. A deep reinforcement learning approach to adaptive traffic lights management. In *WOA*, 2019.

[29] Erwin Walraven, Matthijs T.J. Spaan, and Bram Bakker. Traffic flow optimization: A reinforcement learning approach. *Engineering Applications of Artificial Intelligence*, 52:203–212, 2016. ISSN 0952-1976. doi: https://doi.org/10.1016/j.engappai.2016.01.001. URL https://www.sciencedirect.com/science/article/pii/S0952197616000038.

[30] Cathy Wu, Abdul Rahman Kreidieh, Kanaad Parvate, Eugene Vinitsky, and Alexandre Bayen. Flow: Architecture and benchmarking for reinforcement learning in traffic control. 10 2017.

[31] Kathy Jang, Eugene Vinitsky, Behdad Chalaki, Ben Remer, Logan Beaver, Andreas Malikopoulos, and Alexandre Bayen. Simulation to scaled city: Zero-shot policy transfer for traffic control via autonomous vehicles, 2019.

[32] Flow-Project. flow-project/flow. URL https://github.com/flow-project/flow.

[33] Pablo Alvarez Lopez, Michael Behrisch, Laura Bieker-Walz, Jakob Erdmann, Yun-Pang Flötteröd, Robert Hilbrich, Leonhard Lücken, Johannes Rummel, Peter Wagner, and Evamarie Wiessner. Microscopic traffic simulation using sumo. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 2575–2582, 2018. doi: 10.1109/ITSC.2018.8569938.

[34] Introduction. URL https://sumo.dlr.de/docs/Simulation/Distances.html.

[35] Eugene Vinitsky, Aboudy Kreidieh, Luc Le Flem, Nishant Kheterpal, Kathy Jang, Cathy Wu, Fangyu Wu, Richard Liaw, Eric Liang, and Alexandre M. Bayen. Benchmarks for reinforcement learning in mixed-autonomy traffic. 87:399–409, 29–31 Oct 2018.

[36] Andrew Y. Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. pages 278–287, 1999.

[37] Michael Behrisch, Laura Bieker-Walz, Jakob Erdmann, and Daniel Krajzewicz. Sumo – simulation of urban mobility: An overview. *Proceedings of SIMUL*, 2011, 10 2011.

[38] Axel Wegener, Michał Piórkowski, Maxim Raya, Horst Hellbrück, Stefan Fischer, and Jean-Pierre Hubaux. TraCI. 2008. doi: 10.1145/1400713.1400740.

[39] Venkatesan Kanagaraj, Gowri Asaithambi, C.H. Naveen Kumar, Karthik K. Srinivasan, and R. Sivanandan. Evaluation of different vehicle following models under mixed traffic conditions. *Procedia - Social and Behavioral Sciences*, 104:390–401, 2013. ISSN 1877-0428. doi: https://doi.org/10.1016/j.sbspro.2013.11.132. URL https://www.sciencedirect.com/science/article/pii/S1877042813045230.

[40] Martin Treiber, Ansgar Hennecke, and Dirk Helbing. Congested traffic states in empirical observations and microscopic simulations. *Physical Review E*, 62(2):1805–1824, Aug 2000. ISSN 1095-3787. doi: 10.1103/physreve.62.1805. URL `http://dx.doi.org/10.1103/PhysRevE.62.1805`.

[41] P.G. Gipps. A behavioural car-following model for computer simulation. *Transportation Research Part B: Methodological*, 15(2):105–111, 1981. ISSN 0191-2615. doi: https://doi.org/10.1016/0191-2615(81)90037-0.

[42] A. K. Das and J. Asundi. A simple explicit model approximating the relationship between speed and density of vehicular traffic on urban roads. *Int. J. Crit. Infrastructures*, 8:195–204, 2012.

[43] S. Krauß. Towards a unified view of microscopic traffic flow theories. *IFAC Proceedings Volumes*, 30 (8):901–905, 1997. ISSN 1474-6670. doi: https://doi.org/10.1016/S1474-6670(17)43936-X.

[44] Laura Bieker-Walz, Michael Behrisch, Marek Junghans, and Kay Gimm. Evaluation of car-following-models at controlled intersections. 10 2017.

[45] Vase Jordanoska, Igor Gjurkov, and Darko Danev. Comparative analysis of car following models based on driving strategies using simulation approach. *Mobility and Vehicle Mechanics*, 44:1–11, 12 2018. doi: 10.24874/mvm.2018.44.03.01.

[46] Jose Correa and Nicolas Stier-Moses. *Wardrop Equilibria*. 02 2011. ISBN 9780470400531. doi: 10.1002/9780470400531.eorms0962.

[47] Mahyar Amirgholy and H. Gao. Modeling the dynamics of congestion in large urban networks using the macroscopic fundamental diagram: User equilibrium, system optimum, and pricing strategies. *Transportation Research Part B: Methodological*, 104:215–237, 10 2017. doi: 10.1016/j.trb.2017.07.006.

[48] Eugene Vinitsky. Commit: 0f45f80, 2018. URL `https://github.com/eugenevinitsky/ray`.

[49] Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3 (1):9–44, August 1988. doi: 10.1007/bf00115009.