Deep Reinforcement Learning in CARLA

by Keith Marange
Student ID: [F0305857]

Loughborough University

20COP327: Artificial Intelligence (AI)

Supervisor: Andrea Soltoggio
Submitted 25th August 2021 in Latex

Abstract

This project explores how to develop an autonomous vehicle using Deep Symbolic Reinforcement Learning. With simulations, the software CARLA is used to trial agents with different network architectures, hyper-parameters, and state representation. I will be taking inspiration from a variety of design philosophies utilized by autonomous vehicle companies; as well as emerging developments in deep learning and neuroscience studies into driving behaviours. The purpose of the experiments was to investigate how to accelerate the learning process and improve the agents' ability to generalize. The best model created used Proximal Policy Optimization on Actor-Critic architecture, Recurrent Neural Networks, Convolutional Variational Auto-Encoder with a custom colour space and Kalman Filtering on sensor measurements.

Acknowledgements

Acknowledgments to Dr Andrea Soltoggio my supervisor. And Damien Lopez, a data scientist for Decision Labs.

> Action choices are made based on value judgements. We seek actions that bring about states of the highest value, not the highest reward because these actions obtain the greatest amount of reward for us over the long run.
>
> ---
>
> *Richard Sutton*

Contents

1 Introduction

Artificial Intelligence is the pursuit by researchers to develop learning algorithms that can generalize better than conventional algorithms. The goal of AI is developing Artificial General Intelligence, having a computer being able to understand, reason and solve any task given. The sub-field of AI used in this project is reinforcement learning. In the hope that an autonomous vehicle can generalise well using the class of learning algorithms in that area.

1.1 Motivation

Reinforcement learning is a burgeoning area with many successes. A significant example of this success has come from the research firm Deep Mind for the board game Go. In 2016 they created an agent Alpha GO that beat the world champion Lee Seedol in a five-game series. (Wang et al., 2016) Notably, in the second game, the infamous move 37 by Alpha Go left spectators and punters extremely confused as the system proposed a 'novice' move. Turning out to be a very creative move that swept the rug from beneath Lee Sedol. Leading to an early victory for Alpha Go in game two.

Another significant motivator is the use of Reinforcement and Imitation Learning in robotics. Dr Pieter Abbeel is a researcher at UC Berkley who created the start-up Covariant. Using robots that have reinforcement learning algorithms to succeed in tasks that were not achievable with conventional

Figure 1: Lee Seedol(bottom right) is playing the game of GO against Alpha Go.(DeepMind, 2016)

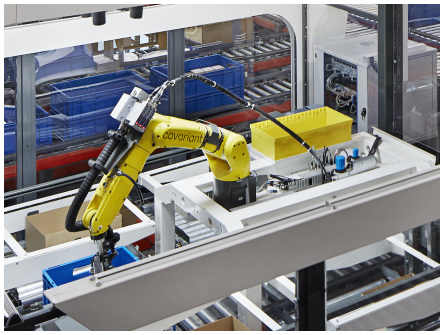


Figure 2: Covarient robot doing picking tasks

automation. In 2020, they were bought by the engineering company ABB. (PETTERSSON/COVARIANT, 2020)

One recent and notable development is Google's parent company Alphabet Group creating a robotics research group called 'Intrinsic'. Their objective is to explore robots interacting with each other to achieve tasks that require dexterity in handling. So that they can help grow the industrial robotics space. Using reinforcement learning, motion planning and controls engineering.I suspect they use intrinsically motivated agents. A paper covered in the literature review.

Exploring the use of deep reinforcement learning for autonomous vehicles enables me to learn skills for deep learning and improve my prospects to be a robotics engineer. All whilst researching an exciting problem in the world today.

1.1.1 Aims and Objectives

- Due to the three-month time-span of this project, we must first quickly develop a deep learning pipeline where initially a basic agent can be created. Then different strategies and hyper-parameters can be chosen to develop the agent over time.
- Understand the principles of deep reinforcement learning, effectively using Tensor-Flow to train an agent and see how well it can generalise. Additionally learning how to use the Linux system and required packages.
- Take inspiration from autonomous vehicle companies and their design philosophies for training and testing agents.
- Take inspiration from neuroscience papers to improve the environmental design.
- Run experiments to see if the research has paid off.

1.1.2 Project Background

Deep Symbolic Reinforcement Learning is used in this project to teach a car to navigate and achieve objectives set in its environment. Symbolic learning is involved as a Convolutional Variational Auto-Encoder is used to produce latent features of the environment. A probability distribution of the scene that forms the representation.(Sherburn, 2017) Reinforcement learning is used to teach an agent a collection of behaviours that enables it to drive conservatively on the road. We build the agent for expert driving behaviours such as lane changing and slowing down. During training, it learns to connect its perception (observation space) to action in line with its long-term goals. Aided by a value function that makes sure the policies actions are in line with achieving road objectives.

Alongside short-term dynamic goals, corrective behaviours are learned and exhibited that keep it in the lane. To achieve navigation objectives, the agent will develop a hierarchy of complex behaviours and sets of associated behaviours. These sub-personalities are called upon and exhibited as a deterministic chain of reflexes, an action outputted by the policy. This is the Actor in the network. Since with training, it matches the appropriate action to an observation. We use Proximal Policy Optimisation (PPO) as it is suited for a continuous action space. it works by optimisation the agents' current policy π_θ to the previous policy $\pi_{\theta old}$

This project uses CARLA, an open-source simulator for autonomous driving research. It uses the Unreal Engine to generate a world's whose environments can be changed in a plethora of ways. Ranging from road layout, pedestrian logic (multi-agent dynamics) and weather. The architecture of the car can be adjusted with different sensors as part of a suite. The representation of the environment by a camera to the agent is a new state for the agent. (Dosovitskiy et al., 2017) The challenge now is to use deep reinforcement learning to choose the optimal actions as it transitions from state to state.

The above figure is a screenshot of the heads-up display showing the vehicle navigating the road. In the top right is the dashcam that is used for agent's observations. The agent has decided to purposely drive on both road

Figure 3: Agent exhibiting odd behaviour

and pavement. This is an example of the behaviours we want to identify, punish and go through extinction. This project uses objectives to describe where to navigate in the environment. The value system held by the agent needs to identify what is best in the long term. The Critic in the network structure. This is as goal-directed attention affects the phenomenology of the world.(Peterson, 2021). The explanation of this behaviour is in the discussion about experiment 3. (van der Laan et al., 2017)

This project is based on a project on GitHub, where the investigated the use of a Convolutional Variational Auto-Encoder (CVAE) and Proximal Policy Optimisation (PPO) to train an autonomous agent. The code used is their architecture for the CVAE, their PPO class and how episodes were evaluated for reward offline. (Vergara, 2019) The reference project could steer and throttle only. I got it to steer throttle and brake.

My adaptation involves: creating an autonomous data collection script, where a town, environment and route can be set for a vehicle to record and collect data; using a recurrent neural network as part of the Actor-Critic architecture; environmental design of the sensor suite for the car and my own custom colour space. This colour space involves converting the RGB colour space to YUV. Then replacing the Y chroma component with the output of the depth camera that is in one dimension. Then at the end, an agent has its action space expand to include steering, throttling and braking. Then its observation space includes the latent features from the DUV, Kalman filter

measurements for acceleration in the latitudinal and longitudinal direction and yaw velocity.

1.2 Main Problems

Autonomous vehicles can be reduced to the paradigm of intelligent robots. Their main objectives being to sense plan and act.(Murphy, 2000)Behaviour arbitration is the main worry. Will the agent learn a set of actions to solve scenarios and not reach a terminal state. The hope of reinforcement learning is the agent learns a lot easier by itself with trial and error. Using a reward system to reinforce a sequence of random actions that exhibit the correct behaviour.

Other challenges include catastrophic forgetting, the network overwriting its weights when attempting to perform a new task.(Bansal et al., 2018) For this project, it could be the inability to generalise for different turns in the road in the new environment. Another challenge involved with the simulation is the Doppler shift occurring and adding perturbations to the environment.(Gupta et al., 2020)

Another significant difficulty that occurred was using the CARLA software itself. Taking me at least two weeks to get a packaged version running on the school computer. Then almost three weeks in trying to get other people's examples to work.

2 Literature Review

This literature review is inspired by the individuals I look up to and the people I have networked with. Then will be divided into four parts: deep reinforcement learning; current autonomous systems, perception (computer vision); and control theory; and related works. In March 2021, I connected with Damien Lopez. A deep reinforcement learning engineer, that works as an A.I scientist at the London data science firm Decision Labs. He has the same undergraduate degree as me, Mechatronics Engineering, and previously

worked in the car manufacturing industry. A great deal of help in increasing my confidence in this project and my career. Artificial Intelligence is inspired by many areas so a neuroscience paper will be covered in related works.

2.1 Deep Reinforcement Learning

2.1.1 Reinforcement Learning Overview

Scientists use reinforcement learning to optimise problems, by setting an agent in an unknown environment and rewards. Initially, the agent is taking random actions but through chance, when it happens upon a reward, it will remember that action. By mapping situations to actions, its goal is to maximise the numeric reward signal. He describes the paradigm in three aspects, sensation and a goal. One key feature is that the whole problem of a goal-directed agent interacting with an uncertain environment. (Sutton and Barto, 1998)

Any choice environment for initial training involves a trade-off between exploration and exploitation. The agent is given a target reward level and is encouraged to exploit what it already knows (the Critic). However, it could explore to attain plausible greater reward in the future. As an interactive agent, it has explicit goals (policy) and its actions influence its frame of the environment (state). Perfect for autonomous vehicles due to the interplay of the planning module and behaviour arbitration. Reinforcement learning is also inspired by biology and neuroscience, the psychological model of the mind and how neuromodulation affects reward uptake(Sutton and Barto, 1998).

From a mathematics frame, reinforcement learning has four aspects: policy; reward signal; value function; and a model. A policy is what behaviours are to be exhibited by an agent as it learns at a given time. It maps its frame of environmental states to actions that will be deployed in those states. It parallels Psychologies stimulus-response paradigm of rules and associations. On policy methods continually improve the policy that makes decisions. Off-policy methods evaluate one policy following another.

A reward signal is usually the goal of the agent, to increase its current level of experienced. The usual convention maximises reward. Modern deep reinforcement takes that idea to different levels of abstraction. Instead of maximising reward, it uses experience replays.(Andrychowicz et al., 2017) The reward it seeks is derived from the environment and can be viewed as short term intrinsic desirability. The reward gained is immediate and depends on the current action and state. This is the primary way of seeking reward by action.

A value function specifies the overarching objective of the learning agent. By predicting the rewards that can be possibly gained, the function estimates what different states rewards would be. A value judgement that uses the previous sequences of observations an agent makes in its lifetime. The long-term desirability reward estimate. Analogue to human individuals, choosing a certain religion or creed to guide the choices and the places they inhabit in life. To quote Richard Sutton from the same book, "Action choices are made based on value judgements. We seek actions that bring about states of the highest value, not the highest reward because these actions obtain the greatest amount of reward for us over the long run." (Sutton and Barto, 1998)

The model is used to mimic the behaviour of an environment, in this case, the deliberative planning element of an autonomous vehicle. A Markov decision model is used to stipulate the problem in the environment, then it is solved using a reinforcement learning method. In our case, Actor-Critic.

2.1.2 C.U.R.I.O.U.S

This paper was suggested by Mr Lopez in our networking call. This paper uses the paradigm of intrinsic motivation to get an agent to set its frame of the environment and establish its own goals. An open-ended environment. Coming up with a diversity of policies. From this, it discovers what is controllable in the environment and actively choose feasible goals themselves.

Theory
The learning algorithm CURIOUS stands for continual universal reinforcement learning with intrinsically motivated substitutions. My hope with this

paper is that intrinsic motivation for an agent parallels why humans learn to drive. People pay extra money for vehicles to save on time in their short life. This intrinsic motivation pushes people to pass their driving test. The CURIOUS algorithm in this paper is similar as overall learning progressed is biased by the agent's attention towards modules where it thinks it will maximise progress. Better yet, the use of universal value function approximators (UVFA) is used to address the infinite set of possible goals that will be seen in this open-ended problem. This method also uses hindsight experience replay to increase the probability of observing sparse rewards. (Sallab et al., 2017) Experiments were performed on a robotic arm with a gripper that learns to manipulate objects.

Model Architecture

The design philosophy is inspired by the intrinsically motivated goal exploration process framework. To enable autonomous continual learning, the researchers designed a modular multi-goal architecture. The learning architecture is Actor-Critic, the Actor is the action policy whilst the Critic estimates the Q values. It is then trained using a deep deterministic policy gradient (DDPG). An off-policy model-free learning algorithm that also uses experience replay. Using a discrete action space.

The representation is defined as modular goals, where the robot puts a cube in the environment. Selecting a module is modelled as a non-stationary multi-armed bandit problem. Learning progress is the derivative of the agents' competence in completing that module. Measured as the probability of success at a point in time. The agent learns to focus attention on modules with the greatest progress and ignore unsolvable modules. Its frame of the environment matures.

Method

Module and goal selection is sampled from the potential set of modules. Data collection is through the UVFA policy interacting with the environment. The learning policy updates its competence to solving the module. The Actor-Critic is updated during module and goal substitution. The agent computes its internal reward according to each transition. DDPG is used to update

the value function and policy of the agent.

Benefits
The framework inadvertently helps with catastrophic forgetting. When the agent detects good learning progress, it uses this form of bias to turn to another module. Essential learning modules in parallel. When it came to the experiments, the robot easily dealt with a variety of tasks. Even with additional noise in the form of distracting objects. The intrinsic motivation design philosophy helps the CURIOUS agent identify and accept hard tasks. This parallels humans getting bogged down in trivial detail when learning to drive.

The robot is given sensor perturbations, noise to its frame of the environment. It easily dealt with the task at hand. This is due to the hindsight experience replay mechanism that looks at past transitions and so recovers.

Gaps
The main disadvantage of CURIOUS is the slow learning progress estimator. It is what significantly delays the robot switching to different tasks.

2.1.3 Hindsight Experience Replay

This paper was again suggested by Damien Lopez. On the topic of Hindsight Experience Replay (HER). Its purpose is to transfer knowledge between goals. It uses an implicit curriculum where sample efficient learning is combined with an off-policy reinforcement learning algorithm. This simplifies the reward engineering as it's now sparse and binary. This type of reasoning is used to tell the agent that some sequence of actions only just missed the mark. By replaying memories of states, greater learning efficiency can be achieved. (Andrychowicz et al., 2017) In total HER is used to solve the problem spaces that have sparse extrinsic rewards.

The states are defined according to the pose of each segment of a seven-part arm. Holding angle and velocity values. Its goal is to reach the desired position in the state-space in the simulator, then actual world coordinates when deployed on the robot. The rewards are sparse and binary. The obser-

vations made are part of a policy that: looks at the current position of the gripper; the desired object; and the distance between the function. Grippers are defined with linear velocities. Then the objects have angular and linear velocities. The action space is four-dimensional. Three dimensions define gripper position in the next time step. The last dimension is the distance between the grippers. The strategy for this Markov Decision Process uses Hindsight Experience Replay, the final state of every episode is a goal. The reasoning is choosing which goals to replay after some sequence of observations.(Andrychowicz et al., 2017)

Benefits Ideas and Gaps
In training, the main method of HER is to replay experiences and use those sequences of actions to try and solve the problem at hand. The agent first experiences an episode in a state, storing the replay in a buffer with transitions. In the robot's case, the replaying past trajectories of its components moving in state space.

The key benefit of HER is how it can be added on top of a reinforcement learning algorithm. If we can identify driving behaviours whose learning motivation is extrinsic. It would be the perfect way to implement this policy.

2.1.4 Deep Reinforcement Learning framework, an overview

Summary
This paper provides a general overview of how to incorporate reinforcement learning into conventional autonomous vehicle algorithms. This system is an end-to-end Deep Reinforcement Learning Pipeline that also uses Recurrent Neural Networks. The design philosophy is divided into recognition, prediction and planning.

The recognition module of the pipeline is used to identify objects. To make sense of the multi-agent dynamics of these entities, the system tracks the objects into its 3D visualisation. The behaviours of agents from past observations are stored as latent hidden representations in a Recurrent Neural Network. These are used in internal modules for likely states of the environment. (Sallab et al., 2017)

The planning module is used to observe the environment dynamics and infer actions that will maximise safety. There is the use of lidar sensors, described as low dimensional due to the point clouds generated. Then cameras are used additionally in the sensor suite, described as high dimensional as a lot of features are generated. Negotiating lane changes in this planning module is analogues to the cognitive thought process humans go to as well. Since humans look at subtle behaviours and behavioural queues of other drivers when negotiating on the highway. The reinforcement learning module in planning parallels this train of thought by giving reward signals of good driving when observing other agents.

Method

The deep reinforcement learning framework can be divided into spatial and temporal aggregation. Spatial aggregation concerns raw input and representation. This module has two networks for sensors fusion and spatial features. The sensor fusion uses data from the camera and lidar sensors to update the state of the robot, localization. The sensor information includes orientation, position, velocity and acceleration. Alongside other raw data, these spatial features are fed as a raw input vector to a Convolutional Neural Network (CNN). The same CNN is then used to extract deep representations so that an attention filter in a kernel is used to create features of interest.

Temporal aggregation is used to take sensor readings over time to create a true state of the environment. The recurrence in this pipeline allows the handling of Partially Observable Markov decision making process scenarios when driving. The end of the pipeline is reinforcement learning planning. A deep q learning network is used whose outputs are the steering, gear, acceleration and brake values. The input to the network is the x-direction of the car and the track borders.

Benefits

The main benefit is that DRL enables an agent to learn as humans. According to that paper, removing replay memory lead to faster convergence. Meaning the agent selected more appropriate actions in its learning phase. There is

no detail of training or troubleshooting in its methodology.

2.2 Current Autonomous Systems

2.2.1 ChaufferNet - Google Waymo

Summary

The first paper for our Autonomous-Systems literature review will be Google Waymo's ChaufferNet. They use Imitation Learning, a form of Reinforcement Learning. They use a Recurrent Neural Network as the basis of the vehicle's representation. The vehicle perceives this representation from a top-down two-dimensional view. The use of imitation learning is to provide expert examples and exacerbate good experimental losses. For example, the agent staying in line on the road with a 1% error in testing, exacerbate those losses in the test data and feed it back to the model. At the end of the paper in the discussion, they even mention how the reinforcement learning framework is better. They formulate experiments in open and closed-loop evaluations, using twenty different driving scenarios. Open-Loop experiments are where the agent does not have their prediction loss available to them. Closed-Loop evaluation simply means the agent is driven forward and in a certain scenario. They include compensating for trajectory loss, slowing down for a slow car and deploying the model in real life. (Bansal et al., 2018)

Model Architecture

This model architecture has the core ChaufferNet model be a convolutional neural net that is the basis of the representation. Then a Recurrent agent network predicts successive points of driving trajectory and the bounding box of the vehicle in a spatial heat map. The two additional networks are the Road Mask Network and the Recurrent Perception Network (using the same representation). The first predicting drivable areas and the second predicts a spatial heat map of what other agents it thinks is in the area. (Bansal et al., 2018)

Losses

The losses used for the network are ranged and varied. The loss for the agent's velocity is measured in position, direction and bounding box predic-

tion. They also add dropout to the network so it will not cheat by extrapolating from the past. The example they use is learnt to respond to stop signs. Instead of stopping as that sign is detected, it will stop because it just so happened to decelerate in the past. They added on-road losses to prevent curb contact. Alongside geometry, loss to make sure the dynamics of the vehicle account for errors when driving according to calculating. Imitation learning is trained with expert's demonstrations. To increase and reflect the fidelity of the environment. They randomly dropped imitation losses for a random number of examples. Finally, auxiliary losses are the predictions of other agents. An imperative part. Losses are a heatmap with dynamic objects whose location is being predicted. (Bansal et al., 2018)

Good ideas

The novel idea of adding perturbations to expert driving seems to have yielded excellent results. The main problem in autonomous driving is predicting the trajectory of other agents. They dedicate an entire collection of auxiliary losses. Their approach is using a heatmap instead to model the dynamics of objects. Another key takeaway from the paper is adding dropout to the different neural networks. By augmenting training data or randomly dropping imitation losses.

Gaps

The drawbacks to this paper can be seen in its data strategy and the results of its experiments. Google collects data predominantly from one area and has a smaller sample size compared to Tesla. Having variance in your data set is fundamental in machine learning. (Russell and Norvig, 2009)The most alarming factor in the data is how they compensated for collision training. They outlined that the training data has no collisions as it 'does not generalize well'. Compensating by creating losses for concurrent bounding boxes of a predicted agent and ground truth of scene objects. This does not reflect the real-life consequence of a collision, death. A major flaw in the paper.(Bansal et al., 2018)

Conclusion

Google need variance in data if it is going to deploy these models commer-

cially with human lives at stake. From this paper, I learned the different losses that need to be added. Auxiliary losses are a way to define multi-agent dynamic objects and adding dropout to the network is needed.

2.2.2 Jesse Levinson - 2007 DARPA challenge winner, CTO of Zoox

In my own opinion, the company with by far the best business plan is Amazon's Zoox. A robo-taxi company lead by CEO Aicha Evans. Most notably is their co-founder and CTO Jesse Levinson PhD. Part of the winning team in the 2007 DARPA Urban Challenge. In short, his path planning algorithm predominantly used A* search. (Levinson et al., 2011)It is currently hard to find papers on Zoox's research but delving into Dr Levenson's research, he is quite LIDAR orientated. The paper below covers a review of his DARPA entry challenge. The design philosophy is a modular approach and has four main categories. Environment Framing, localisation, planning and control systems.

Summary

The environment is built up using a dual Xeon Linux system that interoperates Velodyne lidar data, four camera's, Bosch radars and SICK laser scanners. The laser is calibrated using an unsupervised basic optimisation algorithm, that builds up information into point clouds. Localisation is done by dividing the world map into orthographic grid representations. Whereby each grid has a probabilistic value of Gaussian distribution which holds the likelihood of objects existing or dynamic agents. Then Bayesian inference is used to reduce uncertainty and errors, by deducing which parts of the map are stationary and have reflective surfaces. Bayesian motion is used to track laser map and lidar data. Object recognition is done by segmenting Lidar scans and then applying a Kalman tracker.

2.3 Control Theory

2.3.1 Brief Introduction

Trajectory planning concerns the timing and planning of execution of the system's decisions. Control theory is used to calculate the lane changes. For example, merging into the traffic flow. At each step, the car follows the remainder of a previously formulated trajectory. Instead of constantly updating. It describes a trajectory in terms of lateral motion and longitudinal movement. The control systems are to dynamical model the car and account for the trajectory it is attempting. Model Predictive Control (MPC) can account for neighbouring vehicles dynamics as well as its own. A Proportional-Integral-Derivative (PID) controller is used to tune and dampen the response of steering the vehicle. Then a Linear Quadratic Regulator (LQR) controller is used to minimise the dynamics of the target's trajectory. The vision systems used are now outdated and of no use.

2.3.2 Modelling Lane Contention

Dhaval Shroff is a robotics engineer for Tesla motors. Working in their autopilot AI division. He has been published on three occasions. I am analysing his work on lane keeping and lane change to see how his design philosophy has benefited Tesla autopilot. This paper uses control theory to design various lane contention algorithms depending on the different scenarios of active vehicles.

Summary

Lane contention is the ability to track and detect where you are in the world space for robust lane changes. The control algorithm used is dynamic matrix control, specifically Model Predictive Control. Modelling is used to learn the behaviours of the car and adjusts for real-time changes in the vehicle's immediate dynamics. The guidance system is a wavefront-based planner. The environment is divided up into grids whilst the initial node and final node are knowns. The grid is made up of conductive material and a path based on entropy is created between them. The final grid node is of greater energy, so the vehicle tracks the temperature gradient to its final grid. When the

vehicle gets closer to the objective, accuracy therefore increases. (Parulekar et al., 2013)

Benefits, Ideas and Gaps
The full work of state-space equations and proofs can be found in the paper. The MPC's algorithm inputs are acceleration, the mass of the vehicle, inertia, steering control and peer position neighborship of vehicles. The output of the algorithm is the positional coordinates of the same neighbouring vehicles.

Model Predictive Control is a widely used control algorithm that should be able easier to implement into Carla. Advantages include how explicitly constraints are handled and that the model is directly used. The development time compared to a reinforcement learning algorithm is shorter as well.

Additional sensing requirements are required so a vehicle can be more self-aware, a direct comment from the paper. One major concern is the paper Does not present all problems presented in state-space equations.

2.4 Perception

This paper was chosen as Andrej Karpethy is the director of Artificial Intelligence at Tesla. The team lead at Tesla Autopilot. He seems to have a language and vision approach. To accustom ourselves to his frame of thinking and how computer vision works, we can look at a seminal paper he has produced from 2016. From watching the recent Tesla AI day, he add LSTMS to the architecture for Tesla-autopilot.(CVPR, 2021)

The second paper chosen is from the academic conference "Artificial Intelligence Technologies for Electric Power Systems", covering the area of identifying partial discharge. This particular paper was chosen as not only will it help in my explanation and uses of Convolutional Variational Auto-Encoders, it is related to my background in Mechatronics Engineering. Additionally aiding me in my short internship with the National Grid this coming September.

2.4.1 Dense-Cap: Fully Convolutional Localization Networks for Dense Captioning

Summary
The paper is a computer vision captioning task, where the AI system is required to identify and describe regions of interest with natural language. I am mainly interested in how they progressed from rich dense captioning of multiple objects in the frame to one significant label. This idea could be passed on to our perception task in autonomous driving: identifying important areas for safety.

Benefits, Ideas and Gaps
They design a fully Convolutional Localisation Network Architecture that can be fully trained end to end with one round of optimization. The input is a Tensor of activations. This identifies spatial regions of interest. A convolutional neural network to absorb raw data from the camera into a representation. This network contains 13 layers of 3 x 3 convolutions and 5 layers with 2 x 2 max pooling. It also uses a trained Recurrent Neural Network language model that generates natural language for the labels. (Johnson et al., 2015)

The novel dense localization layer processes regions and extracts matching activation's using bilinear interpolation. This layer interfaces with the recognition network of the Recurrent Neural Network. It extracts fixed-size feature representation for each variably sized region proposal. This uses a region of interest pooling layer with specific dimensions found in the original paper. Then a recognition network processes the features produced from the recognition layer. This outputs final scalar confidence for each proposed region and four scalars that encode coordinates for the region proposal. (Johnson et al., 2015)

2.4.2 Convolutional Variational Auto-Encoder

Summary
A Convolutional Variational Auto-Encoder (CVAE) is the use of neural

network architecture to map high dimensional input features to a lower-dimensional space, as a probability distribution. This project will use a pre-trained CVAE for the agent's representation, live video from the dash-board camera is passed through the CVAE, then to the agent's new state so it can make the appropriate action. (Zemouri et al., 2020)

Power Systems Background

Partial discharge is the phenomenon of localized electrical discharge in high voltage applications. It is dangerous and needs to be quickly identifying as it can lead to the breakdown of important power systems. They require a complex learning model, CVAE, for its use as a good prognosis tool. By identifying where partial discharge may occur in a power cable, they can adjust materials and make various repairs.(Zemouri et al., 2020)

Key Benefits

A CVAE makes use of convolutional neural networks (CNN's) and Variational Auto-Encoder (VAE). It has a symmetrical and reversed structure. Starting off with a CNN layer, the encoder then latent space. Following that is another latent space, decoder and a CNN. CNN's are excellent at pooling and prioritising vast amounts of information. For the researchers, it's 30 years of historical data taken from PD measurements in power systems. For this project, it is the use of the depth camera, semantic segmentation camera and lidar measurements. These measurements are required so the observation space contains entities and temporal data. (Zemouri et al., 2020)

The Variational Auto-Encoders are Auto-Encoders used as a generative models. An autoencoder is used to compress information. An equal number of units are used in the input and output layers. The encoderϕ takes in input data x and forms a latent representation z. represented by:

$$y = f_\theta(x) \tag{1}$$

Then the decoder θ decodes the latent representation z as a reconstruction of the input data.

$$x^{'} = h_\theta(z) \tag{2}$$

Where x' is the approximation of the original data. A Variational Auto-

Encoder is good a generative model as it combines Bayesian inference and Neural Networks to create a latent space of low dimensionality. Bayesian inference is the use of the Bayes theorem to update the probability for a current hypothesis.(bay, 1992) The inference is obtained by sampling the latent vector z with a prior specified distribution assumed to be gaussian. The resulting layer is made up of multiple elements z_i, which is the result of mean μ and standard deviation σ vectors:

$$z_i = \mu_i + \sigma_i \cdot \epsilon \tag{3}$$

Where z_i μ_i and σ_i are the i^{th} components of the mean and standard deviation vectors. This probability distribution is fed into the observation space of the agent. Specifically, the Actor in the network.

The latent space is obtained by the neural network optimising a loss function L:

$$L = L_{rec} + L_{KL} \tag{4}$$

Where L_{rec} is the reconstruction loss as the decoder fails to correctly regenerate original input data in the learning process:

$$L_{rec} = -E_{q\phi(z|x)}(\log(p_\theta(x \mid z))) \tag{5}$$

Where L_{rec} is the negative log-likelihood of decoder and encoders conditional probabilities. The final term L_{KL} is the Kullback-Liebler Divergence loss.

$$L_{KL} = \ KL(q_\emptyset(z|x\ )||p(z)) \tag{6}$$

It is a theoretical measure of the proximity between two densities q(x) and p(x). They are both asymmetric and non-negative. The term measures how close the conditional distribution density $q_\emptyset(z|x)$ of the encoded latent vectors are compared to a normal distribution of p(z).(Zemouri et al., 2020)

2.5 Neuroscience

2.5.1 Introduction

Reviewing my experience of learning to drive has encouraged me to look at neuroscience studies. As I think the two hardest objectives are learning to operate heavy machinery and then being comfortable and being familiar with neighbouring drivers. Learning to do any task comfortably means you are less nervous and accustomed to the task at hand. From a neuroscience perspective, you have created new pathways in the brain. (Gallistel and Matzel, 2013) So when experienced persons are driving, they are using those pathways and leaving spare cognitive load for more forms of reasoning and thinking. This comparative study observes novice and expert drivers using driving simulators. Then various measures of brain activity are recorded in all instances and compared for attention deficits.

The hypothesis I briefly present here I hope to bring to the CARLA project. Examining scenarios where drivers are relaxed or decide to be more attentive, different algorithms can be used for the autopilot of the vehicle. We have already seen the use of control algorithms to plan the trajectory of a car. Additionally, we have seen the use of Hindsight Experience Replay to actuate a robot and the use of reinforcement learning in path planning. HER and the use of neural networks could be viewed as analogous to experienced drivers using new circuits in the brain for vehicle operation. By looking at the results of the study, we could end up having a hybrid algorithm for the autopilot in the CARLA simulator. If the results support this.

2.5.2 Comparative Study of Novice and Expert Drivers

In the study, psychophysiological measures are taken. Measurements deriving from Heart-Rate Variability (HRV), Galvanic Skin Response (GSR) and Electroencephalographic signals (EEG). This is so seven different comprehensive measures can be deduced: immersion, competence, negative effect, flow, tension, positive affect. In my own opinion, flow and immersion suggest the use of HER. Tension, negative affect and positive affect suggest the use of control algorithms. The subjects are placed in three different environments:

highway, city and racing track landscape.

Additional measurements produced from the simulator for each person and environment are average speed; the standard deviation of steering and gas (acceleration). The psychophysiological measures produced and used in the results are driver's accuracy (VG) and task demand (TD). VG is a relationship between performance and driver's vigilance. TD measure is a way of measuring the cognitive load on a brain. A value of less than one means the individual is in their comfort zone. Novice drivers will first be summarised, followed by expert drivers.

Novice

In summary, the paper supports my initial arguments and presents new ideas. Across all experiments, novice drivers had far higher VG measures and greater standard deviations in steering patterns. Meaning it took them more effort to focus on driving and they were less certain in making turns. In highway environments, they experienced tension and negative affect frequently. Having a tiresome experience that required more reasoning from them. (Ekanayake et al., 2013)

One surprising result is novice drivers learned and performed better in the racing track environment.(Ekanayake et al., 2013) Novice drivers were intrinsically more adaptive to this environment. This could be carried to possibly using Intrinsic Deep Reinforcement Learning methods to create a general autopilot agent for easy scenarios.

Expert Drivers

It was found that expert drivers on highways had the lowest uncertainty in steering. The standard deviation of steering was comparatively lower for novice drivers. When they went back for a second session in any environment, they experienced flow more as well. This means they used to experience more as opposed to constantly reasoning about the situation.(Ekanayake et al., 2013) Additional they can rapidly model-new situations and environments they just observed. This can be carried to the use of hindsight experience replay in CARLA.

In city and racing track landscapes, expert drivers had a comparatively higher standard deviation of steering, higher VG and lower task demands. The larger variance in steering indicates drivers fine-tuned their trajectories more in this environment. This could mean the use of more control theory algorithms for these driving scenarios. Expert drivers also had greater use of the accelerator, possibly indicating aggressive behaviour and reacting faster to a new change of state in the environment. In the experiments we view models with higher standard deviation in steering as a metric for consistent expert driving.

3 Design

3.1 Agent, Environment, Policy, Neural Network

The goal of reinforcement learning is to learn a policy. For this project, it is the set of actions taken to navigate a road and not crash. A policy is a function that maps states to actions. Approximated by a neural network with observation data from the environment, whose error is optimised by an optimisation algorithm. As the policy is updated at the end of a horizon, it is an online policy, using experiences.

At the start, the policy would take random actions measuring the reward received. In this case, reviewing an episode and deciding to update the policy or keep the current policy. On-policy paradigm. Rewards are used to define optimal policies in a Markov Decision Process. The objective of achieving an optimal policy is to maximise the total expected reward. (Russell and Norvig, 2009)A value function is involved to predict greater future rewards, another way of encouraging actions in the present state. In the context of this project, the agents' behaviour actuating the car is the policy, whose actions lead to new states being observed (in the Actor Network). The Critic is the value function that reviews actions.

The great advantage of this learning algorithm is the agent will not need

a model of the environment or a known reward function. Solving the field of autonomous driving with conventional logic or conventional supervised learning algorithms is not feasible. There is an infinite number of objectives to define for an algorithm and it will not be able to generalise in new environments and deal with counterfactual information in its observations. The counterfactual is when the current ego model changes the dynamics of its observation. [25] The use of on-policy reinforcement learning lets the agent decide itself.

3.2 The Partially Observable Markov Decision Process

For this project, the Markovian view of self-driving is an agent acting in the environment, whose states are the observations every time step. The actual truth is an agent's neural network is outputting a probability distribution in a separate script. This translates to actions on the car in the CARLA environment. This will be the current policy for actuating a car in the CARLA environment. The same script is only receiving observations through a CARLA camera object, that sends this stream of information to the Actor in the network. The agent cannot observe the entire environment. Even when adding Lidar to extend its 'Field' of view or a Kalman filter to track its location.

Therefore, we define our problem as a Partially Observable Markov Decision Process. Represented by the tuple (S, A, T, R, Z, O). Where:

- S is set of all possible environment states S_t at time t
- A is the set of all possible actions A_t at time t
- T is the transition function. A conditional probability $S * A * S \rightarrow [0, 1]$ Where $T(s_t, s_{t-1}) = p(s_t \mid a_t, x_{t-1}$ is the probability of ending in state S_t if the agent performs an action A_{t-1} in state S_{t-1}.
- R is the $SxA \rightarrow [0, 1]$ is the reward function where r(s,a) is the reward obtained by executing action u in state s.

- Z is the set of all measurements or observations Z_t at time t. Or all possible observations.
- O is the $SxAxS \rightarrow [0, 1]$ is the observation function, a set of conditional observational probabilities. Where $O(S_t, a_t, z_t) = p(z \mid a, s)$ Meaning the probability of observation z if action u is performed and the resulting state is s.(Ulbrich and Maurer, 2013) The conditional observation probabilities

In the context of our experiments, each variable will be explained. X includes: the observations from the camera that is a 3-dimensional matrix; the latitudinal steering of the car by whose space is [-1,1], throttling of the car whose space is [0,1]; the speed of the vehicle measured in kilometres per hour. Then in later experiments: a Kalman filter indicating the momentum of the car and coordinates in the environment. The Kalman filter will include the GNSS sensor, IMU sensor and LIDAR data.

3.2.1 Environment State S

A single state observation is made up of: camera observations latent features generated by the Variational Auto-Encoder; the longitudinal and latitudinal directions of the ego vehicle in the current time step, inferred by the sensor suite and speed of the car; and CARLA's notification telling the agent if it is in the lane or not.

3.2.2 Actions A, for Transition state S'

Set of actions A is our action space, steering and throttling. Over time, the neural network learns a policy that chooses optimal actions with an approximal solution method. It is forced to generalise with a limited subset space (only viewing the environment through a camera), making it a Partially Observable problem. With sequences of episodes and collected rewards, it uses Proximal Policy Optimisation to choose a better set of actions over time.

Where executing an action $U \in S$ with a system in the state $A \in S$ Is what will be called a policy $\pi : s \rightarrow a$. The goal of such a planning problem is to find a sequence of actions (policy) that maximises the expected reward

over the time horizon T. (Ulbrich and Maurer, 2013)

$$R_t = E(\sum_{\tau=0}^{T} \gamma^{\tau} * r_{\tau}) \tag{7}$$

According to Sergey Levine, the world of driving can be viewed as Non-Markovian,(Levine, 2018b) As no two roads are the same.Not even the same road at different times. In his research, he tackles the Non-Markovian problem of self-driving by establishing how the observations received are high dimensional and need to be converted to low dimensional latent space.A similar paper employs a Recurrent Neural Network and a Non-Markovian predictive coding.(Nguyen et al., 2021)So it is important to add augmentations to the dataset to reflect this nature. This project will use a Variational Auto-Encoder previously trained on world environments that are deployed in training to produce latent features of the environment.

The observation function is the Actor in the Actor-Critic network receiving observations of the environment and producing new actions for the Critic to infer and act upon.

3.3 Proximal Policy Optimisation

The learning method used is Proximal Policy Optimisation. A policy gradient method is used to train the network to optimise cumulative reward over time with the Neural Network in the Actor-network. The other main reason it is used is for the continuous action space. Steering is a range of values from [-1,1] and throttling is a range of values from [0,1]. This technique learns a parameterized policy that will select actions without consulting a value function. Where:

$$\pi(a \mid s, \theta) = P_r\{A_t = a \mid S_t = s, \theta_t = \theta\} \tag{8}$$

The probability that action is taken in the time given environment state s at time t with parameter θ. The value function can also be described as weight vector $w \in R^d$. This method maximises performance by learning a policy parameter based on the gradient of a scalar performance measurement

$J(\theta)$. Updating it's an approximate gradient ascent in $J(\theta)$ Updating it's an approximate gradient ascent in J:

$$\theta_{t+1} = \theta_t + a\nabla J(\theta_t) \tag{9}$$

Where $\nabla J(\theta_t) \in R^d$ is a stochastic estimate approximating the gradient of the performance measure concerning the argument θ_t . The policy can be parameterised if $\pi(a \mid s, \theta)$is differentiable concerning its parameters. This means the agent's policy can never become deterministic. This will hold as the autonomous driving problem is a non-deterministic environment. (Sutton and Barto, 1998) The environment is uncertain, the same state (field of view) is unlikely to ever arise again. When the agent trains then generalise in experiments, they are both stochastic process. When differentiated concerning its parameters, $\nabla\pi(a \mid s, \theta)$ is a column vector of partial derivatives concerning components θ

Since it is differentiable, the policy can be improved by following the empirical gradient by hill climbing. Evaluating the change in policy value for small increments in each parameter. Hopefully converging to local optimum space. For this project, it's the equivalent of the agent overcoming per se a left turn in the road. It requires a certain throttling and speed by the action space, with the representation being formed by changing π_θ. (Sutton and Barto, 1998)

3.3.1 Horizon - Hyper-parameters

For the case of stochastic policy $\pi_\theta(s, a)$ it is hard to compare hill-climbing changes $p(\theta)$ and $p(\theta + \triangle\theta)$ since the rewards received in each trial will vary quite widely. At the start of training, the agent will still be executing random actions. The solution is to run trials for a set number of episodes, measuring the variance and using it for an indication of improvement for the performance $p(\theta)$. For sequential cases, it is seen as:

$$\nabla p(\theta) \approx \frac{1}{N} \sum_{j=1}^{N} \frac{\nabla_\theta \pi_\theta(s, a_j))R_j(s)}{\pi_\theta(s, a)} \tag{10}$$

In the code, this is the number of steps to simulate per training step. Where a_j is executed in s on the j^{th} trial and $R_j(s)$

3.3.2 Why Continuous Action

This paper uses a policy gradient method for continuous action spaces. Policy representations in discrete action spaces are not how cars operate. The agent actuates the car in the environment by accelerating, applying torque to the wheels that react to the ground and leads to a resultant force in one direction. This can only be done by varying the throttle that allows this. For example, the car cannot suddenly go from 5 km/h to 15 km/h, the agent must learn what value the throttle to be is with a given state. Additionally, steering is a range of values. The other disadvantage for discrete action spaces is that from a theoretical vantage is that e-greedy leads to drastic changes in action output when there is a state change. (Sutton and Barto, 1998) An infinitesimal small change in θ can cause the policy to switch haphazardly from one action to another. The value of the policy will change discontinuously.Making the gradient based search difficult. (Russell and Norvig, 2009)

3.3.3 PPO continued

Policy parameterization for continuous action learns the statistics of a probability distribution of actions to take. For every different state, there is a probability density function consisting of the mean and standard deviation. It is a lot easier to learn the space for steering [-1,1] and [0,1] for throttling. The actions are chosen from a normal Gaussian distribution:

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} exp(-\frac{(x-\mu)^2}{2\sigma^2}) \tag{11}$$

Where μ, σ are the mean and standard deviation of the normal distribution. The value p(x) is the density of the probability at x. Whose integral must sum up to 1. From this, policy parametrization is approximating a function that chooses the optimal actions:

$$\pi(a \mid s, \theta) \doteq \frac{1}{\sigma(s,\theta)\sqrt{2\pi}} exp(-\frac{a-\mu(s,\theta))^2}{2\sigma(s,\theta)^2}) \tag{12}$$

This function approximator is to find the mean and standard deviation as it approaches the limit to that value that will choose the correct action given a state. The two parameterised function approximators are $\mu : S_x R^d \rightarrow R^+$ and $\sigma : S_x R^d \rightarrow R^+$ The policies parameterised vector is $\theta = [\theta_\mu, \theta_\sigma]^T$

In the context of this project every time there is a transition to a new state, the Critic's neural network is fed a representation of the environment and approximates a new mean and standard deviation, which leads to a set of actions for steering and throttling. (Sutton and Barto, 1998)

$$\mu(s,\theta) \doteq \theta_{\mu}^{\top} x_{\mu}(s)\, and\, \sigma(s,\theta) \doteq exp(\theta_{\mu}^{\top} x_{\mu}(s)) \tag{13}$$

3.4 Actor Critic

This is a schema where the policy gradient method learns both the policy and value functions. A temporal difference method where the structures are explicitly separated to represent the Actor being the policy structure used to select actions. Then the Critic is separated as it estimates a value function that Criticizes the Actors' action. Using temporal difference error as its error function.

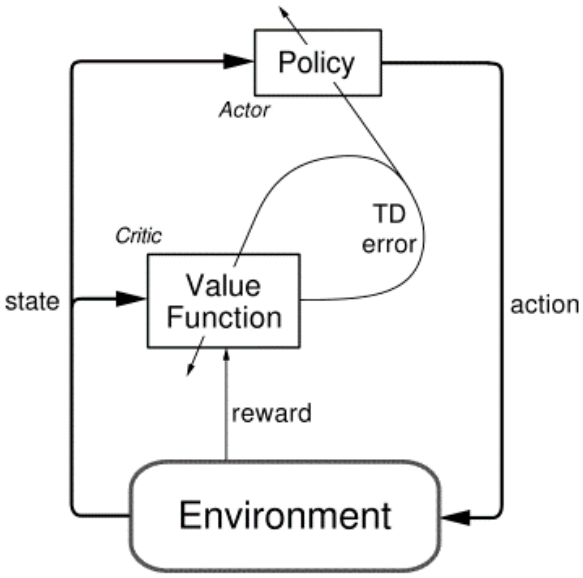


Figure 4: Actor Critic Overview (Sallab et al., 2017)

This affinity of two neural networks working together for the agent to operate in an environment. PPO is used to optimise both neural networks. Below is the pseudocode for how the networks' policy and weights are updated:

1. The input must be a differentiable policy parametrization $\pi(a \mid s, \theta)$ and differentiable state value parameterization $w \in R^d$, $\theta \in R^{d'}$ Sutton and Barto (1998)

   For each time step do the following in a single loop.

2. Run the policy and sample state action pairs for the current policy

$$A\ \pi(a \mid s, \theta),\ to\, get\, (s, a, s', r) \tag{14}$$

   Using the current policy gets the current state observation s, action a, new state s' and reward r for the transition.

3. Update the Critics' Value Function using the targeted reward for future states

$$V_\phi^\pi\, using\, r\ +\ \gamma V_\phi^\pi(s') \tag{15}$$

   With the new Critic's value function V_ϕ^π being the sum of the current reward r and the future reward $\gamma V_\phi^\pi(s')$

4. Evaluate the policies action to see how good the selection (probability distribution) was.

$$A^\pi(s, a) = r(s, a), \gamma V_\phi^\pi(s_i^{'} - V_\phi^\pi(s')) \tag{16}$$

   With the temporal difference error of that action in the given state being $A^\pi(s, a)$. Where the current value function $V_\phi^\pi(s_i))$ represents the Critics' current approximation of how good the Actor's actions are. Then the expected value function reward being $\gamma V(s_{t+1)}$ (Levine, 2018a)

5. If the next state is terminal, initialise all the state value functions approximation (Critic) to zero, or don't update them.

$$V(S^{'}, w) \doteq 0 \tag{17}$$

6. Given that the current policy parameterisation is differentiable, search for a new gradient to update the policy gradient estimate, hence further approximating the function.

$$\nabla_\theta J(\theta) \approx \nabla \log \pi_\theta(a_i \mid s_i) A^\pi(s_i, a_i) \tag{18}$$

7. The policy gradient is a hill climb along the plane, so you update the policy by adding which direction to go.(Levine, 2018a)

$$\theta \leftarrow \theta + a\nabla_\theta J(\theta) \tag{19}$$

3.5 Multi-Layered Perceptron

For the first set of experiments, a Multi-Layer Perceptron is a neural network used for both the Actor and Critic. It is a feed-forward network whose interconnected neurons are used for a nonlinear mapping between data and its approximation. This is represented as an input vector and an output vector. The nodes are connected by weights and their output signal. Which is the sum of the inputs to a node modified by a nonlinear activation function. These nodes are superimposed on one another, layered on top of each other and each node having a connection to a node in the next layer.

It is feed-forward since the output error is mitigated by information from the input node passing through the weights.(Gardner and Dorling, 1998) It's objective to approximate an unknown function f that relates input vectors in X to the output vectors in Y.

$$Y = f(X) \tag{20}$$

Where X = [k x n] and y = [n x f]. The variable k is the number of input nodes and n is the number of training patterns. The function F is optimised with proximal policy optimisation using training data. This project has the Actor use a Multi-Layer Perceptron (MLP) to is a probability density function consisting of the mean μ and standard deviation σ. This project has an MLP with three hidden layers of sizes 500,300. A Rectified Linear Unit activation function is used for the first two layers and the final layer has no activation function. Since there is no activation, the output has a custom activation function to scale the value. A hyperbolic tangent function so their values end in the range of [0,1].

$$\mu = a_i^{min} + \frac{tanh(o_i) + 1}{2} * (a_i^{max} - a_i^{min}) \tag{21}$$

With a_i^i being the maximum and minimum value required. (Vergara, 2019)

3.6 Recurrent Neural Network

A Recurrent Neural Network is a neural sequence model whose network achieves immense performance on natural language processing tasks. The architecture of the neural network has hidden states so the previous outputs can be used as inputs. This is temporal space that makes it good at memorising sequences of input. It has a trick for temporal data, by memorising temporal patterns. An episode with multiples steps can have the policies actions viewed as temporal data. Perfect for the Critic. It would be many too many maps. My arguments for using RNN's are based on competitions in the autonomous vehicle space and my literature review.

3.6.1 RNN - Time series data and Control Theory

By viewing successful driving as a sequence of successful actions, reviewing input data is time-series data. Time series data is usually modelled with a RNN or with algorithms in the space of control theory. I have two arguments on why the driving problem is also timing series data, hence why recurrent nets are needed. From the results of a competition for autonomous driving and in my literature review where a model predictive control was used as a lane contention algorithm.

Lex Fridman is an MIT lecturer in the field of Artificial Intelligence. He presented a lecture on RNN's and their uses in autonomous driving. Highlighting how for a competition, individuals were given a video of a car's front view driving through a town. The data was just a sequence of images and a supervised learning task as they were given the angle and speed and torque for each image. The task was to predict the steering angle with the given dataset, the competition winners used RNN's to predict the steering angle on test data sets with greater accuracy.(Fridman, 2021b) This highlights how driving should be seen as time-series data, with RNN's being known as the best way to model that data.

As previously stated in my literature review, the researchers use control theory to aid a car in lane contention. The purpose of model predictive control is to tune the response of the controller to errors in its input. Its purpose is to achieve equilibrium, staying in the centre of the lane. It works by being given negative feedback in its control loop, which is its position relative to the lane. The researchers were successful in their experiments. The fact they specifically used a dynamic matrix control algorithm to tune the response, reiterates the idea of time series data as an input for the agent

3.6.2 Using the Comparative study of beginner and expert drivers

From my literature review, I summarised a paper that explains the neuroscience of humans learning to drive. With the paper classing beginner, intermediate and expert drivers and giving them a task in varying environments. The researchers would analyse the brain chemistry as individuals attempted the task with EECG analysis and other methods as previously explained. A significant piece of data was that expert drivers had a greater standard deviation of steering in cities compared to beginner drivers. Meaning they were comfortable and adept with steering more often.(Ekanayake et al., 2013) Due to vehicle ballistics, (Fridman, 2021a)This usually happens at lower speeds.

This gave me the idea of bringing memory into this reinforcement learning problem. As to do certain turns, an individual need to remember in the past what sequence of actions and observations helped you circumvent that task. This would be what the vehicle dynamics are in. Initially, I wanted to implement hindsight experience replay, a good idea but not within the scope. A recurrent neural network should suffice in replicating how to model the behaviours expert drivers exhibit.

3.6.3 Cancer AI research similarities

There was a paper abstracted away from my literature review on cancer research. They developed a novel and state of the art in method in cancer detection and optimal chemotherapy-scheduling. Their design philosophy is viewing incoming data from patients as time-series data. It is like this project as it is a model-free algorithm optimised for continuous control with Deep

Deterministic Gradient. (Tordesillas and Arbelaiz, 2019)Used to estimate the policy gradient. Additionally, they use the Actor-Critic method to make the optimal decisions, covered in detail in my literature review. To represent sequential past data, they also use a replay buffer. Their representation is very similar to this project's Variational Autoencoder giving latent features, they use natural language processing to describe the different states of tumours in question. In their experiments as well, they prove how DDPG with continuous action is better than a DQN network with discrete action.

Artificial Intelligence is the ideal way to go about search space. Nowadays using learning algorithms to implement the optimisation and statistics that underpin these methods. By viewing temporal data of medical scans as a search space, previously failed searches can be remembered to quickly converge at an optimal solution.

3.7 Convolutional Variational Auto-Encoder

A Variational Auto-Encoder is used to compress input data from higher dimensions to lower dimensions as a probability distribution. This is so important entities in the scene from the dashboard can be identified quicker and encoded in the observation space as:

$$s_t = \{z_0, z_1 ... z_{dim-1}\} \tag{22}$$

Where s_t is the current state in the Markov Decision Process and z is the output of the decoder. This project sets the number of latent dimensions to 64. The main advantage of a custom CVAE is it can act as a pre-trained vision network. For this project, images were collected using a logic-based car agent to drive in five different towns. This data further varies as different types of turns and roads were recorded, and the lighting and weather changed. Another advantage is its use for noise removal.

Having the state representation comprised of probabilities is like symbolic AI. In fact, there is an area of research called deep symbolic reinforcement learning. In one thesis, they investigate the use of a fully Convolutional Variational Auto-Encoder to have an agent learning the mapping from the

symbolic representation. (Sherburn, 2017) This symbolic representation is aided using a segmentation camera, where what typically would be RGB input, the scene is already segmented into 12 different classes of entities, which are identified with different colours.

This projects agent is choosing the best action, with a new state, the latent features of the CVAE. The state consisting of what the agent sees and current vehicles dynamics. A search space where it is finding the best actions to next take using proximal policy to find the best gradient for $\pi(a \mid s, \theta)$

4 Methodology

Scripts	Uses
view_spawns:py	Visualise spawn points across all maps. The same program allows you to see the route viewed taken by an agent with additional commands.
collect_driving_data:py	Set arguments for weather, spawn location, destination and town. So a basic agent can autonomously collect images from the environment.
train_ve:py	A script containing Variational Auto-Encoder class and TensorFlow code to train and validate a model.
train_net:py	Inherits environment-class from environment script, relevant PPO model from PPO script. To train an agent using reinforcement learning. Inheriting another class that records each episode and evaluates the cumulative rewards between horizons separately.
environment:py	Constructs CARLA agent with sensors then maps environment with waypoint and reward systems.
PPO model script, 4 different ones	These scripts have the PPO model classes. They differ as the Actor or Critic will have a Multi-layer Perceptron or a recurrent neural network.

4.1 Feature Engineering

This is the process of collecting training data (images) for the CVAE and adding augmentations to the dataset. Arguably more time was spent getting autonomous agents to drive around and collect in CARLA. This is as running the server in synchronous mode leads to frames per second (FPS) of 1. Then adding sensors to the car lead to a considerable drop in FPS. The FPS of the client would drop much lower than the server, leading the car to react slowly to situations and crash into objects. This was circumvented by not using the RGB camera and not giving output to the heads up display to view the progress of the car. You would have to check the agent did not crash by monitoring the collision sensors and reviewing the data outputted. If the car had crashed, a snapshot of the same obstacle would be the rest of the data.

4.2 Setting route for agent

Data were collected across five towns. The first step is to use the 'View_spawnspy' script that lets you visualise all spawn points on a map. As seen in the figure below.

Figure 5: Town 7 With Spawn Locations

Figure 6: Town 7 with routes visualised.

Then to view the route you want your logic agent to take, the script is run with the following additional commands and the result is below

$$"PythonView_spawns.py–route1,13–waypointsTrue–Towntown07" \quad (23)$$

To collect images, two cameras were used. One camera gave a semantically segmented view of the world scene. Dividing up the world into 12

different classes of entities. In addition to this, the lidar output of the camera was taken from the sensor and interlaced to the same field of view. The output of these images is set to a folder called 'interlaced'. The second camera returns the logarithmic depth of the current field of view. It acts as a depth sensor, with the images saved to a file called 'logDepth'. Great effort was taken to efficiently save multiple images in real-time to the computer, as it could take up too much bandwidth for the client-server collection. This would lead to a drop in FPS for the client and agent crashing on its route.

4.3 Navigation and Topological Path Planning

Typically, autonomous vehicles are set objectives on the plane of the map and are guided along the road using A* search. (Levinson et al., 2011) (Russell and Norvig, 2009) For this Deep Symbolic Reinforcement Learning project, the agent is set objectives by having markers set every 10 metres in the coordinate space. This means the agent at times tries to drive along the curb to reach its next objective.

It learns by reinforcement by being with negative rewards to stay in the lane, whilst trying to achieve its next objective.

4.4 Creating Two Representations and Data Augmentation

The first representation takes the original output of segmented images from the 'interlaced' folder. As seen in the figure the second output is a custom colour space. It takes the images from the 'logDepth' folder and converts them to binary, so the dimensions are reduced from three to one. Then the images from the 'interlaced' folder are converted to YUV format. YUV is a colour space whose colour encoding is one luma (Y) and two chrominance (UV) components. (Ibraheem et al., 2012) The luma component gives brightness whilst the two remaining dimensions are used for human perception, the entification that goes on in a scene. The Y component is swapped for the one-dimensional logarithmic image. Creating a custom DUV colour space as seen in figure 6. differences in the colour scheme are seen below. YUV colour space is converted using matrix multiplication.

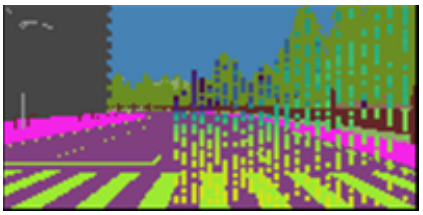


Figure 7: Segmented Output and Lidar Data

Figure 8: Custom DUV Output

4.5 Training CVAE model

The CVAE is pre-trained manually, so when deployed to the agent latent features of the environment are easily identified. In total, three different CVAE's need to be made, the description is below. For context, this report is written when experiments were finished.

CVAE	Experiments and Data
Segemented Images CVAE	Experiment 1, 5000 images
'DUV' Colour Space CVAE	Experiment 1 and 2, 5000 images
'DUV' Colour Space CVAE, Augmented Dataset	Experiment 3 and 4, 10000 images

These three different models have the same architecture. A visual representation of its architecture is below in figure 9, based on a VAE architecture design from a paper. (Asperti et al., 2021)

The input of the CVAE will be a normalised image of one-dimension 80x160. Each convolutional layer has: the number of filters labelled above; a kernel size of 4; a pooling stride of 2 and a non-linear Relu activation function. It uses binary cross-entropy as a loss function for attempting to reconstruct data in the training phase.

For augmenting the data, code was written by myself and taken from open-source websites. Where individuals post solutions for manipulating

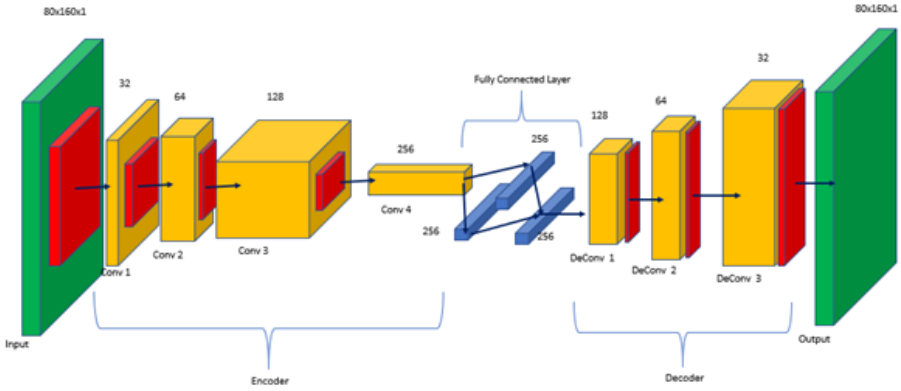
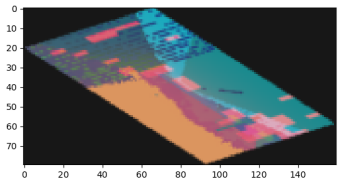
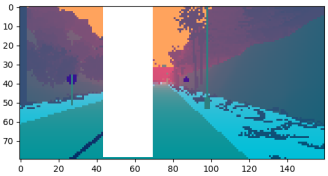


Figure 9: Convolutional Variational Auto-Encoder Architecture

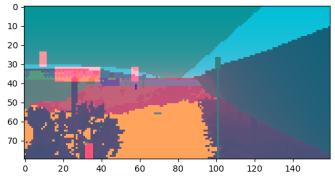
NumPy arrays. Images were manipulated in five different ways. Being randomly rotated; the image partially erased, flipped vertically, flipped horizontally, Gaussian blur added to it. Examples are below.



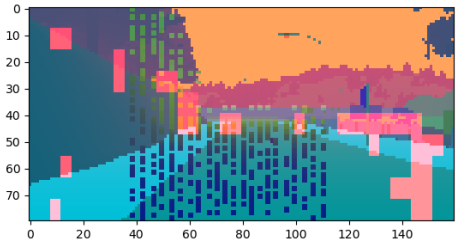
Randomly Rotated



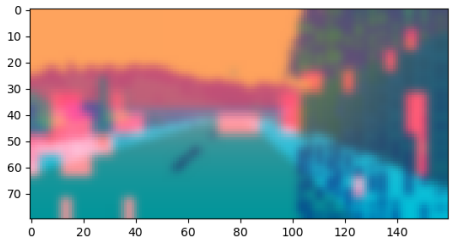
Random Erasing



Horizontally Flipped



Vertically Flipped



Gaussian Blur

Figure 10: Augmented Images

An additional 5000 examples were created for the third model from data augmentation. Using an 80-20 split for training and validation.

4.6 Training

Training is done by reinforcement learning. So an agent is initialised in a Carla environment at the start of the episode. The episode is also recorded whilst the agent is active seeking rewards, exhibiting behaviours that are actions for the current policy. During each episode, state, action and reward are recorded. During an episode, the current PPO model is given the current state and makes predictions using its current policy. These predictions are a tuple of values: throttle, speed and brake (in later experiments). These values are applied to the vehicle object in the Carla environment. Only experiment 4 uses brake in its action space.

The modules this script is dependent on:

Wrappers	Contains Carla object wrappers for the vehicle and world environment.
Reward functions	Systems of Rewards and Punishments
train_vae:py	Contains the CVAE classes
run_eval:py	Trains the model in a deterministic setting, using recorded episodes.
PPO	Contains the various versions of Actor-Critic networks
Environments	Contains world environment, vehicle objects with relevant sensors. Waypoints and objectives.

4.6.1 Training PPO Model - Stochastic Setting

Algorithm 1: Reset Environment. Destroying vehicle and camera objects. The cumulative reward is set to zero. The terminal flag is false. Construct PPO model, with action space and hyperparameters

Train PPO Model;
Input : Environment state, observations, PPO Model
Output: PPO model actions, rewards, new state
while *not at end of this document* **do**
    Take observation of the 'DUV' dashboard camera. Speed, throttle, brake, steering. Encode with CVAE.;
    PPO model with current policy makes predictions, giving continuous actions and value estimates.;
    Run the environment for one-time step, using current actions applied to the vehicle object.This gives a terminal state flag, new state and reward for actions.;
    Store state, actions reward onto a stack, indexed by their step number.;
    Check terminal flag if the episode was terminated.;
end
if *5 episodes have been run, record the next episode as a video for evaluation.* **then**
    Compute generalized advantage estimator (GAE) with rewards, values, discount factor and lambda value for GAE, for advantages.;
end
while *Update the old policy for 3 epochs* **do**
    Using the current data of states, taken actions, reward returns, and advantages (for an episode index) Train the PPO model.;
end

4.6.2 Evaluating PPO Model - Deterministic Setting

Training the model in the Carla environment to optimise its actions is a stochastic setting. As the agent explores the environment can't be deter-

mined from its field of view. Recording an episode allows us to train the model from a deterministic setting. As the video plays, the current model tries to predict what action should have been taken. Google's ChaufferNet from the literature review uses the same method. There is also the autonomous vehicle competition Lex Fridman talked about. (Fridman, 2021b) Then the error from that is returned as a cumulative reward. If the reward calculated from the deterministic setting beats its previous attempt. The model is saved. Therefore, the policy is updated.

Algorithm 2: Reset Environment. Destroying vehicle and camera objects. The cumulative reward is set to zero. The terminal flag is false. Construct PPO model, with action space and hyperparameters

```
Evaluate PPO Model;
  Input  : Current state from recorded video, terminal flag false,
           reward as 0
  Output: Total reward
while Recorded Episode is running do
    PPO model predicts and gives action with current observation.
     Using a video frame. ;
    This action is taken and a step is returned in the environment. ;
    This returns the new state (next video frame) reward and if the
     episode was terminal.;
end
```

4.7 Remaining Scripts

Reward Functions
The code is taken from the GitHub repo itself. It contains:

- Punishment for driving below a minimum speed. Leading to a terminal state
- Punishment for deviating too far from the centre. Leading to a terminal state.
- Punishment for too high of a speed. Leading to a terminal state

- A reward for going straight and above a minimum speed
- A reward for staying centred.
- A greater reward for staying at the target speed.

Environments

This script contains information on constructing a world in Carla. This includes: spawning the vehicle, associated sensors and initialising call-backs; setting objectives and waypoints; running a training step and resetting the environment.

- Connect to Carla server in asynchronous mode, loading town 7.
- Spawning vehicle with a logarithmic depth camera, semantic segmentation camera, lidar sensor, camera for the heads up display.
- A function that interlaces lidar points onto the field of view for the semantic segmentation camera. Used for the first experiment. Implementation took two days.
- A function that does a soft reset of the environment. Transporting the ego-vehicle and resetting rewards.
- A function that runs a step in the CARLA environment with actions from the PPO model. Actuating the vehicle. Returning the next encoded state.
- A function that draws a path to the next objective.
- A method that takes semantic segmentation and logarithmic depth camera observations into a buffer. Then combines for the final representation of "DUV". Several weeks.

Actor-Critic Networks

Depending on the experiment there will be different Actor-Critic networks. But there are similarities:

- One class containing the policy graph and one child class that has the PPO model class.

- Loads hyper-parameters.
- Loads the Actor, Critic or both with a Multi-Layer Perceptron or Recurrent Neural Network.
- Function to predict with current weights
- Function to save model and write to tensor board summary.
- Function to train and optimise the weights of the network.

Wrappers

Following the principle of object-oriented programming, the Carla world environment is spawned within a class. Then vehicle, sensors and custom sensors are added to it. Great effort was made on this script.

- Carla Actor base class. Given by CARLA project
- Collision and lane invasion sensor classes.
- Custom GNSS sensor. For Kalman filtering.
- Custom Inertial Measurement Unit sensor (IMU). For Kalman filtering
- Custom Lidar Sensor. For the first experiment.
- Vehicle and World base class. Given by CARLA project.

4.8 Table of Hyper-Parameters

Batch Size	How to divide up training memories
Learning Rate	As part of the hyper-parameter search for PPO, metric for how much change the model will go through with given error rate (Midilli and Parsutins, 2020)
Value Loss Scale	Scaling back reward values with given loss.(Midilli and Parsutins, 2020)
Clipping Parameter	Setting PPO gradient values to the expected range (Sutton and Barto, 1998)
Horizon T	How many steps into the future the PPO model will be concerned with (Midilli and Parsutins, 2020)
Discount Factor	A scale factor $R \in [0, 1]$ where the maximum values have the agent tend toward future rewards. (Sutton and Barto, 1998)
GAE parameter	Associating how good a particular set of actions was for a particular state (Midilli and Parsutins, 2020)
Entropy Loss Scale	Used to help exploration by encouraging stochastic policies. (Ahmed et al., 2019)
Number of Epochs	How many times to pass data as a vector through the Actor-network. (Russell and Norvig, 2009)

5 Experiments

5.1 Set Up

5.1.1 Overview of Experiments

Experiment	VAE Encoder	Description
Experiment 1	Semantic camera observations interlaced with lidar points vs 'DUV' colour space	Changing the representation. Segmented input against a custom YUV colour space that has a depth camera encoded. (DUV) To see who has the higher learning rate?
Experiment 2	DUV colour space	MLP vs RNN networks. Deployed in an unknown environment as well
Experiment 3	DUV colour space, Augmented Images	Changing network structure. Adding RNN to Actor, Critic, or both
Experiment 4	DUV colour space, Augmented Images	Representation is DUV. RNN added to the Actor-Critic architecture. VAE has an augmented dataset. Kalman filter of location added to observations. Action space expanded to add breaking. Prior experiment to see if the action space should be 3 dimensions. Adding heavy penalties for throttling and breaking. Or 2 dimensions with an action space of [-1, 1] to have throttling and breaking in continuous state-space.

5.1.2 Hyper-Parameters CVAE

Hyper-parameters	Parameters
Learning Rate	1e-4
Beta	1
Z dim	64
Batch Size N	50
Loss Function	Binary Cross Entropy

5.1.3 Hyper-Parameters Agent

Hyper-parameters	Parameters
Batch Size	64
Learning rate	1e-4
Value Loss Scale	1
Horizon	128
Discount Factor	0.98
GAE parameter	0.95
Entropy Loss Scale	0.01
Epochs	3

For this experiment, data were collected and recorded during training. Leading to 8 different infographics for each model trained. In addition to this, the model would be evaluated as it generalised in a town, to complete an objective of driving a single lap. The training results of the CVAE are in the appendix.

5.1.4 Infographics Chosen

Average Centre Lane Deviation
Y-axis is the latitude travel by an ego vehicle with the respect to the plane of road. How much it veers off an imaginary central line its lane. The x-axis is the step number. But it is measured every 5 steps. A single episode would be roughly 150 steps at the beginning of training. Taken when the model is evaluating. Specifically using the same PPO model against a screen recording to see how accurate its actions were. Higher standard deviation towards the end of training indicates expert driver behavior. (Ekanayake et al., 2013)

Average Speed
The average speed is given in kilometres per hour on the y axis and episode number on the x-axis. Taken from when the model is evaluating.

Distance travelled
Measures the distance travelled by the ego vehicle and not how far along the route it got. X-axis being metres and the y-axis being step number. Taken from when the model is evaluating.

Reward
Measures the total reward received against the episode number (x-axis). Taken from when the model is evaluating.

Action 0 – Standard Deviation of steering policy
Inspired by a neuroscience paper for the comparative behaviours of expert-novice drivers, this measures the standard deviation of steering action outputted by the Actor. Taken during the training phase of the model.

Action 1 - Standard Deviation of throttling policy
Again inspired by the neuroscience paper, a measure of the standard deviation of throttling outputted by the agent. Taken from the training phase of the model.

Training loss
When a model was updated with a zipped list of items indexed with its ob-

servation state and actions, what was the lost experience when training the model.

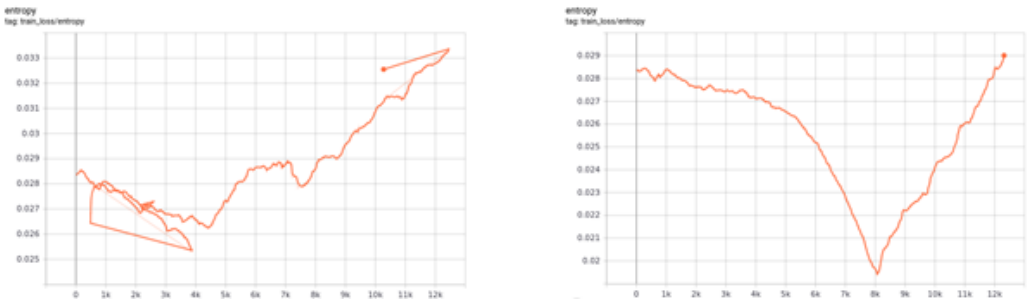
Entropy
This is the predictability of actions by the agent. (Ahmed et al., 2019)

5.2 Experiment 1

The objective of this experiment was to compare two different state representations by changing the input that the agent sees. This would lead to rough guidance as to what can accelerate the learning. For a fair experiment, only the colour space was changed whilst other variables were held constant: neural architecture (Multi-Layered Perceptron); Hyper-Parameters for both the CVAE and Actor-Critic; Carla environment, specifically sensor suite and reward system in the environment.

Some of the metrics are too erratic and noisy for comparison so only a few will be selected.

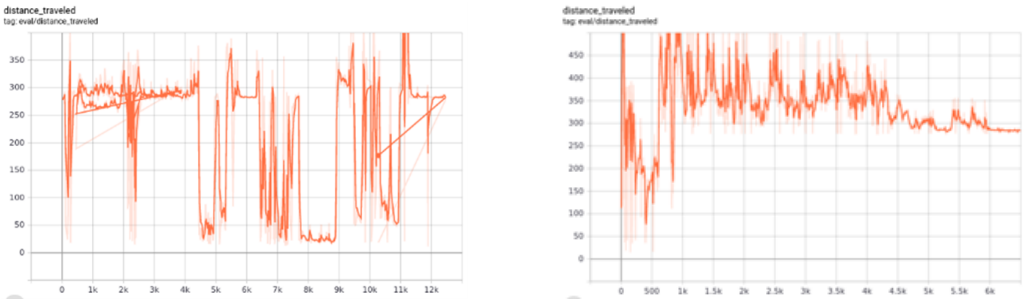
5.2.1 Results



Segmentation DUV

Figure 11: Entropy

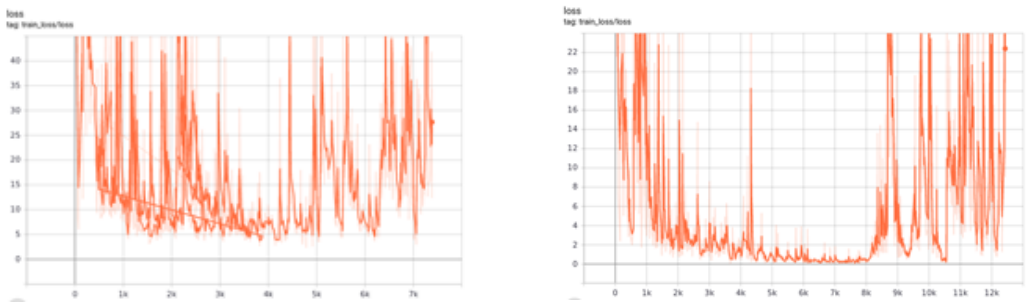
From the above figures, using a segmented representation led to a dramatic fall in the predictability of agents actions. Meaning the PPO found failed to find the gradient. There is an earlier switch in confidence at 4k steps compared to 8k steps for DUV.



Segmentation DUV

Figure 12: Distance travelled

This metric compares the distance travelled, the remaining 6k steps for DUV is missing, unfortunately. Not only does DUV on average maintain a greater distance travelled, but it also doesn't gain sudden losses in distance travelled with time. This could be to do with the policies being updated as it reaches further along the track. In the later stages, the segmentation agent may have learnt faster initially, but could not generalise well enough on unknown parts of the track.



Segmentation DUV

Figure 13: Training Loss

By taking the machine learning perspective and viewing training losses we can see a clear winner in terms of representation. For this, we can only look at a maximum of 7k steps. The DUV model could minimise its losses a lot more concerning time. The DUV model being better.

5.2.2 Summary

- The outcome of this experiment is that the DUV representation enables a PPO model to generalise faster. This will be useful in helping to run experiments quickly.
- My initial assumption was that DUV colour space could be harder to interpret since the first component of the 3D array is the depth camera. This theoretically increases the different number of states substantially. But as the scene is described more succinctly, it leads to quicker convergence.
- From the previous point, the entropy (performance) of 'DUV' is better.

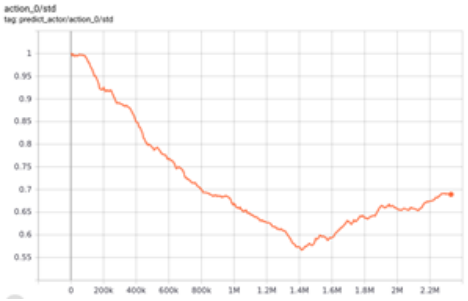
Infographic	Segmentation or DUV?
Average Center deviation (lowest)	DUV
Average Speed (Highest)	Segmentation
Distance Travelled (Greatest)	DUV
Reward Accumulated (Fastest to positive)	DUV
Steering, σ	Segmentation
Throttling, σ	Can't Say
Training Loss	DUV
Entropy	DUV

5.3 Experiment 2

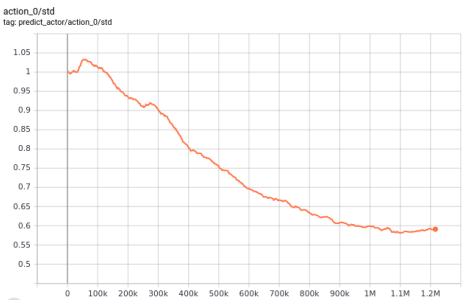
The objective of this experiment is to compare two different models for our agent and see what network the architecture should have. The decision-maker is again what gets our model to learn faster. The first model uses a Multi-Layered Perceptron in its network. Comprised of a series of dense layers. The second model uses a recurrent neural network in its architecture. I hypothesise that giving temporal space to an agent will instead remember a

sequence of actions when given a particular state. For example, if it steered too much and needs to stop from crashing into an object. This will be the situation in a Decision Process where it is given an unusual state that it would not usually expect. The model will also be deployed in an environment. Specifically, town 7. The representation used is DUV.

5.3.1 Results



Multi-Layered Perceptron

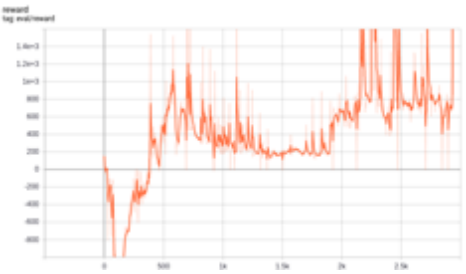


Recurrent Neural Network

Figure 14: Steering

By looking at the standard deviation of steering, we can see how confident the model is in selecting actions. We can only view up to 1.2 million steps. From our literature review, it was found that expert drivers had greater standard deviation for steering in most environments, helping them navigate cities better. The slope of the RNN is greater than MLP, indicating RNN is the better model. However at 400k steps, the standard deviation of the MLP model is 0.85 steps whilst for the RNN 0.8. Then at the of our data, the standard deviation at 1 million steps for MLP was 0.65 compared to 0.6 for RNN. The conclusion being that RNN is better.

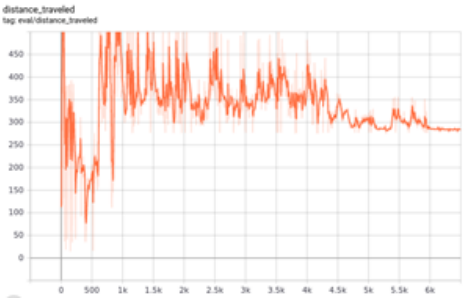
Multi-Layered Perceptron



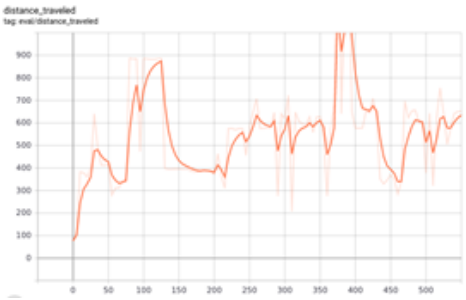
Recurrent Neural Network

Figure 15: Reward Accumulated

By looking at the standard deviation of steering, we can see how confident the model is in selecting actions. We can only view up to 1.2 million steps. From our literature review, it was found that expert drivers had greater standard deviation for steering in most environments, helping them navigate cities better. The slope of the RNN is greater than MLP, indicating RNN is the better model. However at 400k steps, the standard deviation of the MLP model is 0.85 steps whilst for the RNN 0.8. Then at the of our data, the standard deviation at 1 million steps for MLP was 0.65 compared to 0.6 for RNN. The conclusion being that RNN is better.



Multi-Layered Perceptron



Recurrent Neural Network

Figure 16: Distance Travelled

When the MLP model trained, I could say it took roughly 900 episodes to even stay in the lane. With RNN, it took 6 episodes. Another metric to show how much faster it is at learning is the distance travelled. The RNN infographic shows it took just over 45 episodes to travel 490 metres. Whilst it took over 700 episodes for the MLP network to maintain an average distance travelled of 400 metres.

Generalising in Town 7

This section of the experiment is to see how far the model can drive and complete objectives (locations in the map) in a single session. Unfortunately, I could only collect data for this experiment. When I attempted to deploy certain models in other environments, it completely failed.

Generalising in Town 7	Distance Travelled (m)	Deviation from Center (m)	Average Speed (km/h)
Multi-Layered Perceptron	13.6	0.548	9.28
Recurrent Neural Network	250	0.678	28.2

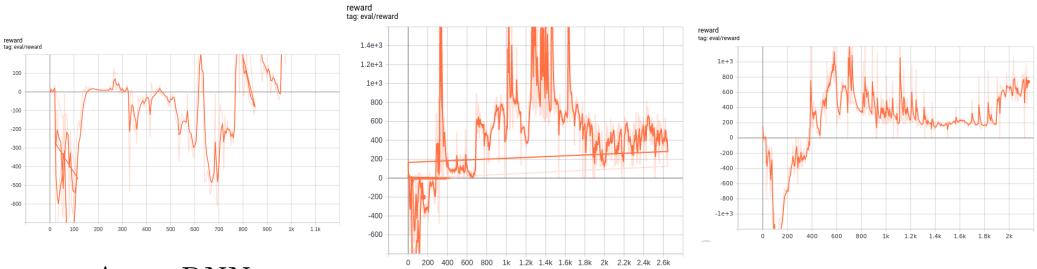
5.3.2 Summary

Infographic	MLP or RNN?
Average Center deviation (lowest)	Can't Say
Average Speed (Highest)	RNN
Distance Travelled (Greatest)	RNN
Reward Accumulated (Fastest to positive)	RNN
Steering, σ	RNN
Throttling, σ	MLP
Training Loss	Can't Say
Entropy	MLP
Generalising in Town 7	RNN

5.4 Experiment 3

The purpose of this experiment was to compare different neural architectures. By seeing if temporal space was needed in either Actor or Critic networks.

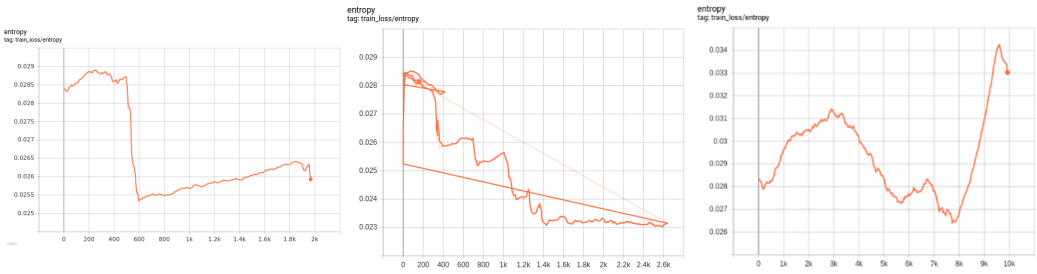
5.4.1 Results



Actor RNN Critic RNN Actor RNN Critic RNN

Figure 17: Reward Accumalated

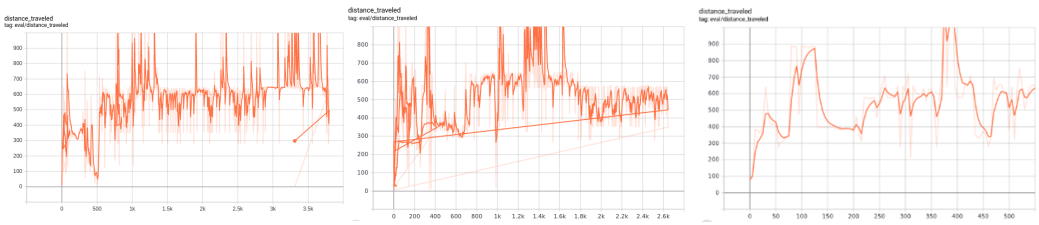
Surprisingly, the Critic RNN model learnt faster than the other two models. Reaching positive rewards in 300 episodes. The 'Actor RNN Critic RNN' model came in second place, maintaining positive reward 400 episodes in. The 'Actor RNN' model still learned faster than the model using an MLP network, but it struggles to maintain reward. In total, this could indicate the importance of the value function in encouraging a certain sequence of actions.



Actor RNN Critic RNN Actor RNN Critic RNN

Figure 18: Reward Accumulated

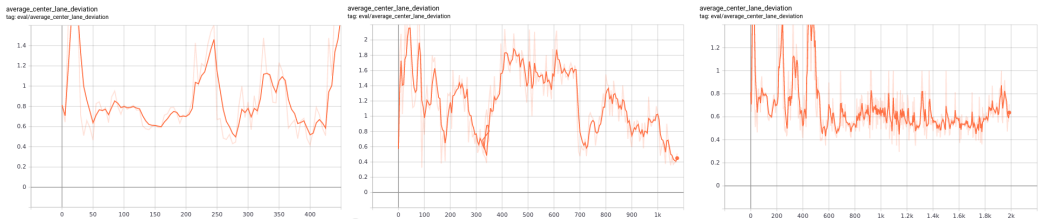
The results for entropy indicate that practically, having an RNN in both networks leads to better policies in the long run. As seen for the 'Actor RNN Critic RNN', the entropy value rises after 7900 episodes. Compared to the other networks that experienced sharp falls in entropy and no U-turns.



Actor RNN Critic RNN Actor RNN Critic RNN

Figure 19: Distance Travelled

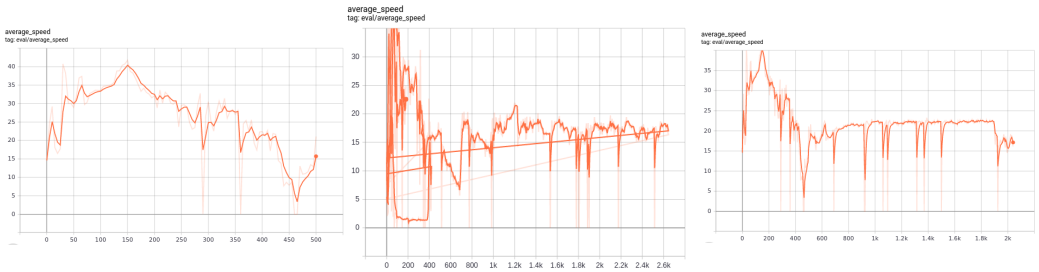
It is slightly hard to tell from this graph, but the 'Actor RNN' model could travel greater distances more reliably than the other network models. Unfortunately, this metric shouldn't be used as the length of episode data for the 'Actor RNN Critic RNN' model is not enough.



Actor RNN Critic RNN Actor RNN Critic RNN

Figure 20: Average Centre Deviation

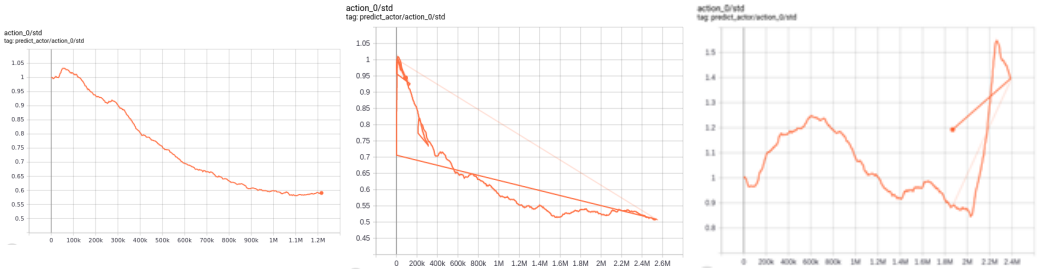
From these metrics, the 'Actor RNN Critic RNN' model learned to stabilise its vehicle a lot quicker with time. It managed to develop a behaviour associated with keeping control of the vehicle. This is why its deviation from the centre of the track is less erratic compared to its counterparts.



Actor RNN Critic RNN Actor RNN Critic RNN

Figure 21: Average Speed

This metric looks at the average speeds with given episodes. The graph with far less noise is the 'Actor RNN Critic RNN'. Suggesting it learned to traffic rules and appropriate behaviours for driving on the road. This can be inferred from the reward system that punishes traffic infractions.



Actor RNN Critic RNN

Figure 22: Steering

From looking at the standard deviation of steering, we can see how brittle the 'Actor RNN Critic RNN' model is. After 2 million steps, there is a sporadic jump in the standard deviation measure. Indicating the model started to forget previous sequences of actions and driving behaviours. This could be as if the model is near completion, its last task is to drive a lap of the environment 3 times. The steep faller in standard deviation by the 'Critic RNN' model indicates how important a strong value system is in informing actions by the Actor.

5.4.2 Summary

Infographic	Which Network
Average Center Deviation (lowest)	Can't Say
Average Speed (Highest)	Actor RNN Critic RNN
Distance Travelled (Greatest)	Can't Say
Reward Accumulated (Fastest to positive)	RNN
Steering, σ	Critic RNN
Throttling, σ	Can't Say
Training Loss	Critic RNN
Entropy	Actor RNN Critic RNN

- From this, we will have an RNN in both networks
- The peculiar results of the Critic RNN is explained further at the end

5.5 Experiment 4

The objective of this experiment was to reach further than the code this project was based upon. By adding braking to the action space. This can be done in two ways.

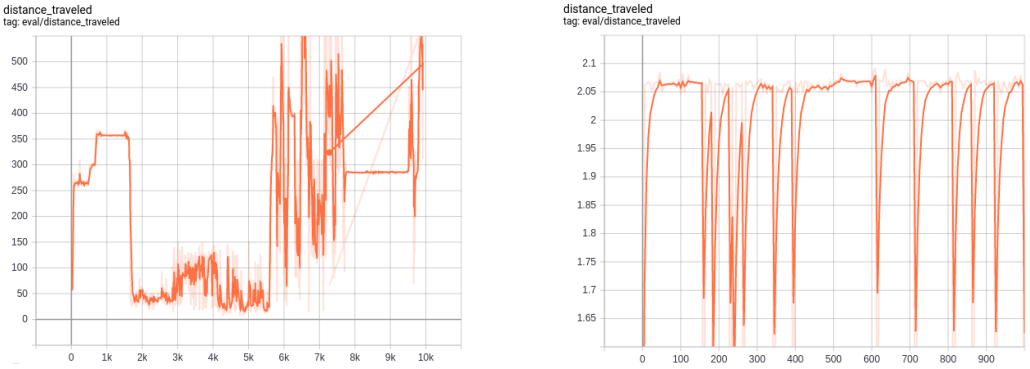
Model one action space is [-1,-1] [1,1]. Two spaces are wide. The first value for steering, giving it a range of [-1,1]. The second value in the action space is for braking and throttling. With positive values for accelerating the ego vehicle and negative values indicating the magnitude to brake at. Our previous experiments have shown how we can speed up learning so it can be done in this period.

Model two would have an action space [-1,0, 0] , [1, 1, 1]. The steering range is [-1,1] and throttling and braking being separate, having the range of 0 to 1. This would be supplemented by a reward system that punishes

actions if the network wants to accelerate and throttle.

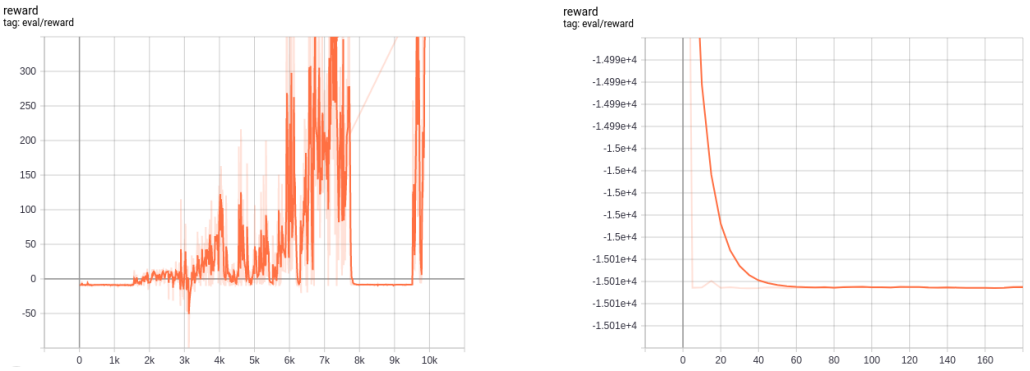
Unfortunately, this form of experiment failed when attempting to build the second model. After two days of training, it didn't even learn to go further than 3 metres.

5.5.1 Failed Experiment Results



Action Space of 2 Action Space of 3

Figure 23: Distance Travelled



Action Space of 2 Action Space of 3

Figure 24: Reward Accumulated

As seen above, 'Model 2' could barely go further than 2 metres. Then the negative reward accumulated became three-fold that of the negative reward accumulated by the first model, in the first 200 episodes. The feedback loop relating to trying new policies and the reward system in place may have led

to this collapse. Especially the fact there is a rule that stipulates to terminate and penalise if there was no movement.

By looking at the reward graph, there is a dramatic increase in learning since the agent can now brake. Now we know how to accelerate training, we can create another model that uses two different environments in its learning and has changed parameters.

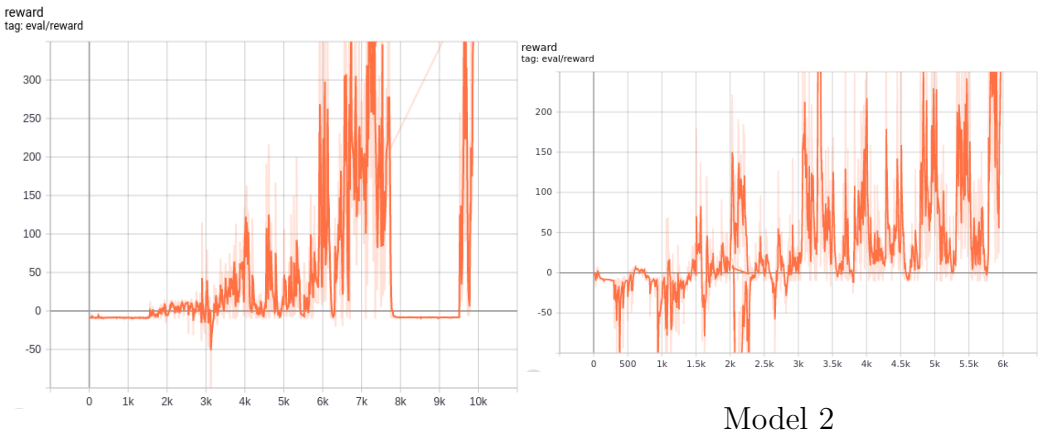
5.6 Experiment 4 - Revision

This experiment will interleave training between two environments. The first environment, Model 1, is the one used previously for experiments, with the objective being to complete a route. The second environment has objectives designed to have the agent complete a route in the town. Additionally, the Hyper-Parameters will be tuned to encourage a greater learning rate. So the learning rate was increased and the discount factor was reduced. To compensate, the clipping parameter was increased to prevent an exploding gradient. Since as the learning rate is increased, so does the step size. The evaluation number was doubled, so a greater range of policy implementations can be reviewed. Before the network updates.

5.6.1 New Hyper-parameters - 4.1

Hyper-parameters	Parameters
Batch Size	32 Halved
Learning rate	1.2e-4 Increased
Value Loss Scale	1 Unchanged
Horizon	128 Unchanged
Discount Factor	0.90 Decreased
GAE parameter	0.95 Unchanged
Entropy Loss Scale	0.01 Unchanged
Epochs	3 Unchanged
Evaluation	10 Increased

5.6.2 Results



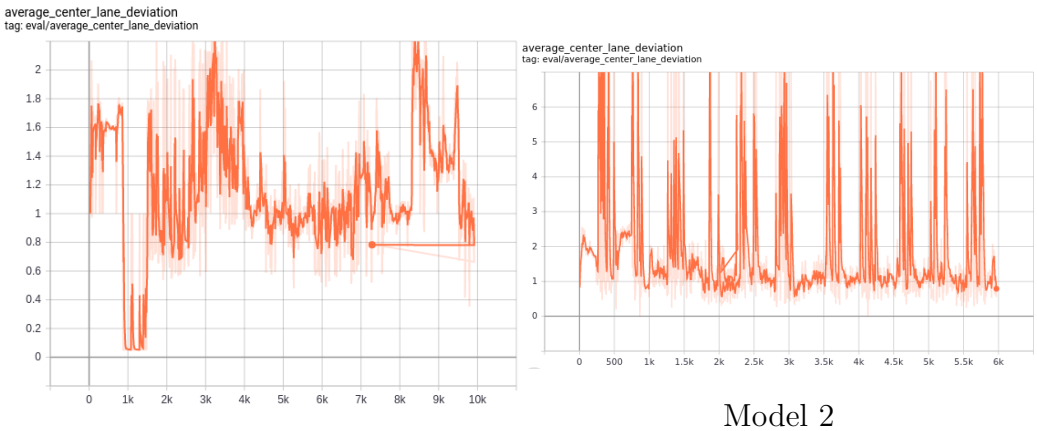
Model 1

Model 2

Figure 25: Reward Accumulated

The results for the second model seem cyclical, but its because the environments would switch. The first model seemed very conservative in its policy. As for 2000 episodes it had negative reward. It paid off as there was

a steady increase in the reward accumulated. Breaking through and consistently accumulating reward after 3500 episodes. The second model was a lot more exploratory in nature. As it was willing to gain negative reward for quite a while at the start. It consistently accumulated positive reward after 3000 episodes. This could indicate that interleaving training with my method is a good way to deal with catastrophic forgetting.



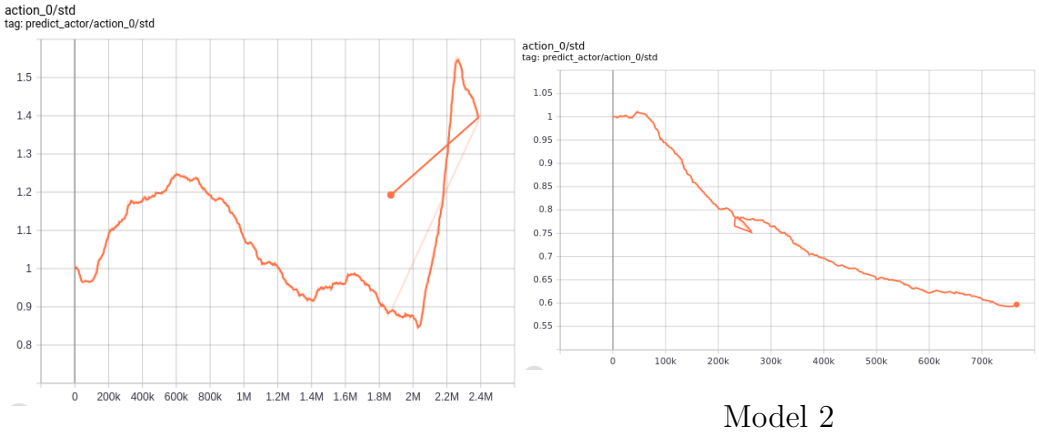
Model 1

Model 2

Figure 26: Average Deviation from Centre

This may not be a good metric as further analysis is required to approximate the 'centre lane deviation' for Model 2. However by looking at the steady state areas and its trend, the average distance increases. Furthermore the deviation is less than the first model. Indicating that it handles lane contention better and consistently.

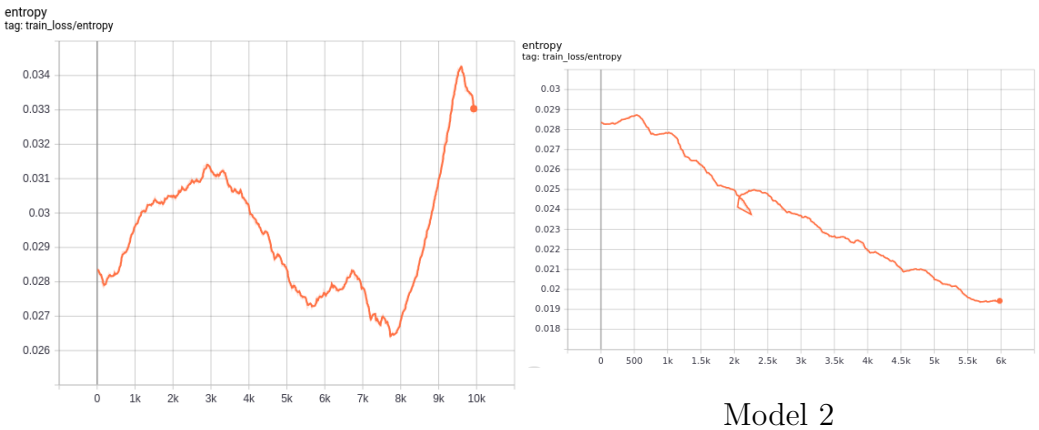
When training the model, it frequently failed after approximately 100 episodes. Upon investigation, the policy was the problem. It seemed that the change in learning rate affected the ability to search for a new gradient. The exception handling, I wrote pointed to the policies action being out of bounds. So the learning rate and learning rate decay was changed back.



Model 1 Model 2

Figure 27: Standard Deviation of Steering

By viewing only the first 700k steps, Model 2 is far more effective at learning due to the steady downward trend. A fall in standard deviation indicates greater confidence.



Model 1 Model 2

Figure 28: Entropy

Model 2 has a downward trend in entropy, meaning the predictability of the agent's actions become more certain overtime. This occurred despite the environments being switched. In other metrics you can see the change in the results as the environment switches.

5.6.3 Summary

Infographic	Which Network
Average Center Deviation (lowest)	Model 2
Average Speed (Highest)	Model 1
Distance Travelled (Greatest)	Model 2
Reward Accumulated (Fastest to positive)	Model 2
Steering, σ	Model 2
Throttling, σ	Model 2
Training Loss	Can't Say
Entropy	Model 2

- From successive experiments, I have found out a way to quickly train a car to steer throttle and break.
- From this experiment, my method of interleaving training is unconventional, but it clearly works.

6 Discussion

6.0.1 Achievements

Using Principles from Neuroscience
Andrew Ng is a leading researcher in the field of Artificial Intelligence who emphasizes the importance of data-centric AI. The key takeaway is that to improve a model, more focus should be on the quality and variance in data. (Press, 2021) From the neuroscience paper, it was found that expert drivers learn to focus on fewer objectives and are less conscious of the environment when driving. Instead focusing on the behaviours of surrounding cars. These principles were brought forward in my literature review and project. Proposing an intrinsically motivated agent who over time decides to focus on fewer objectives.(Colas et al., 2018) Due to the time scale of this project, a different

approach was taken. This was implemented in the project by creating a custom colour space 'DUV' that lead to a faster training time. Compared to the project this was based upon that used the RGB colour space. (Vergara, 2019)

From the same neuroscience paper, it was noted that learning to drive was easier for beginner drivers in landscape environments as it would be fun to do so. Being intrinsically motivates goal-directed individuals (beginners) to explore all sorts of policies to drive as fast as they can. Doing so has them learn the dynamics of the car faster. This was translated into this project by using the routeing environment, which reflects the meandering landscapes in the neuroscience paper. As seen from the results it accelerated learning.

An amalgamation of Hindsight Experience Replay (HER)
One of my hypotheses is that agents should instead remember sequences of action to correctly exhibit the required behaviours for certain manoeuvres. Leading me to explore HER as RL agents instead remember sequences of actions from the past. This implementation would not be possible due to the scope of the project. Instead, a recurrent neural network was implemented to reflect the idea of remembering sequences of actions. From experiments 2 and 3, it led to great success.

Accelerating learning
By investigating how Hyper-Parameters affect the Reinforcement Learning process, experiment 4 showed that learning was accelerated. (Ahmed et al., 2019) This allowed us to train and contrast multiple models for this project.

6.0.2 Missed Objectives and Difficulties

The biggest concern with our experiments is that many of the models could not generalise when tested in new environments. The objective of experiments 1 to 3 is to show how I could accelerate learning which I did. But further testing of the models by deploying them in an environment with navigation objectives would have helped to understand significantly.

Another missed objective stated in my project preparation report was that I would attempt to make the architecture of an RL agent in CARLA

as modular as possible. To try and reflect Google Waymo's design philosophy (ChaufferNet) (Bansal et al., 2018) As I have learned more about RL, I released how wrong I was, and it was an unfeasible task for this timespan. DRL is synonymous in the field of AI as to how black box its algorithm is. For example, I thought I could separately have perception and localisation modules. But for DRL, you just feed it into the observation space and over time it works it out itself.

One significant difficulty during training was with the last model. As it interleaved training with two different environments and reward systems. The frames per second would drop from 30 to 3, leading to frequent crashes. As seen below in figure 29.

Figure 29: Low Frames Per Second

The model could not be left to train without frequent supervision. So the server had to be manually restarted.

Another missed objective was evaluating models with driving scenarios, inspired by the Google Chauffernet paper. Utilising the loss functions they devised. Then additionally adding people and basic traffic rules into training. My focus was developing a basic DRL agent to showcase my skills and effort this summer.

6.0.3 Limitations

One significant limitation is that our analysis of how to accelerate learning is based on looking at the training data. To further discuss limitations, it is best to look at other academic papers to see how they evaluated their deep learning models. The models could not be formally tested. When deployed in an environment they simply crashed.

Methodology

The purpose of the CVAE was to use the principles of symbolic learning to reduce the high dimensional representation of our environment. Instead of the agent viewing a matrix of numbers it would observe latent features. Probability distributions of the image space. In experiment one, we chose to change the image input instead of the structure of the network.

There is a paper that extensively argues how Variational Auto-Encoders only require a certain size for their encoder. By looking at variational autoencoders, they deduce that increasing the network size of encoders leads to diminishing returns. By identifying a property called a generative map. If this map is strongly invertible, it can be inferred that only a smaller encoder network is required. (Pareek and Risteski, 2021) This principle should be applied to our CVAE. By performing experiments to investigate its generative map, we can hopefully reduce the size of the network. This should lead to smaller time consumption of learning for the CVAE and efficient code.

Our Deep Reinforcement Learning agent utilised Recurrent Neural Networks to implement the policies action and to approximate the value function. It significantly sped up learning but the different network structures were not evaluated concisely. There is a paper that uses DRL and RNNs in a different problem space, image denoising, but performs experiments far better. They create a benchmarking system for the DRL image denoising scheme by using two different training strategies to judge how effective the model is. (Zhang et al., 2021) In short, With their proposed architecture they train it with stochastic reward. Then another altered model that is mainly RNN based is trained with a deterministic mean squared error. In the end, their experiments confirmed that the novel DRL solution achieved more reliable denoising results. This same principle can be applied to this

project's experiments. By using a recorded video of a PPO agent acting over time. Then using an RNN to train on the same data. Similar to how Google's ChaufferNet evaluated its RNN's.

Experiments
The biggest limitation of experiments is how there were no basic scenarios ran. Such as lane changing, roundabouts and performing a Michigan left. This would be a good judgement even with no NPCs in the environment.

In the third experiment where different neural architectures were investigated. The architecture that had an RNN in the Critic (value function approximator) exhibited dangerous behaviours. Like the figure just inserted. The car would frequently drive along the road and pavement, despite receiving the negative reward. I suspect it started to cheat the reward system by knowingly taking the shorter path to its next objective. The value system was biased by the knowledge of a greater of completing objectives quickly as opposed to doing it safely. The Critic was just a more powerful network, in the future alternative designs should be investigated. That would not have temporal space but still, increase its ability as a value function approximator.

In experiments 3 and 4, when inspecting the agent training, a dangerous behaviour was created by the agent. For long stretches of roads, it would decide to drive on both the pavement and road. This happened frequently in experiment 4 when only the Critic in the network had a Recurrent Neural Net. This is a classic case of misaligned objectives, between myself the programmer, and the RL agent. As its objective is lane contention with the road and not to crash. But it chose to use the contrast in colour space between pavement and road. This could be as the Critic with its recurrent neural network optimises its value function a lot faster than the Actor. So it's biased it with inappropriate advice on how best to achieve lane contention. The Critic is aware of the negative reward received for going off road, but it knows it can achieve a checkpoint objective a lot of faster if it tells the Actor to use the contrast to guide itself. This leads to greater reward as efficiency is in the reward system. The principle of using contrast in colour space was used in my undergraduate project to build a line-follower drone.

6.0.4 Future Work

Learning action representations
The action space of our final model in experiment two was 2 spaces wide. With one action being used to both throttle and brake. In future, we should try to better associate punishments and rewards with actions. To have a larger action space where throttling and braking are separate. There is a paper that attempts to get an agent to learn action representations like the idea just proposed. By decomposing a policy into a low dimensional space of action representations. Their algorithm is called policy gradients with representations for actions. (Chandak et al., 2019)

Hierarchical learning
Using DRL to develop an autonomous vehicle has the agent develop and learn a collection of different behaviours. These can be viewed as sub-policies whose individual behaviours are exhibited with certain state observations. To make sure certain behaviours are learned and executed for scenarios, there is a paper that proposes a supervisory idea. They create a Hierarchical Program-Triggered Reinforcement Learning Agent, where several RL agents are trained in a structured program for different manoeuvres and scenarios. (Gangopadhyay et al., 2021) This allows you to override actions exhibited by the policy of the RL agent for safety purposes. Then makes the entire DRL agentless black-box as its actions can be selected.

Multi-task reinforcement
Attempting to teach various scenarios in a structure could easily lead to catastrophic forgetting. To supplement this approach, another paper presents the idea of multi-task reinforcement learning. With the different scenarios collated together as joint training of multiple tasks. (Teh et al., 2017) By searching for common behaviours between tasks, the learning becomes a lot more stable. As opposed to constantly destroying and creating different environments as I did to interleave training in experiment 4.
As mentioned in the limitations, when RNN was only used in the Critic network it leads to the learning of dangerous behaviours. As mentioned in my literature review, Hindsight Experience Replay could be used to better inform the Actors' policy. (Andrychowicz et al., 2017)

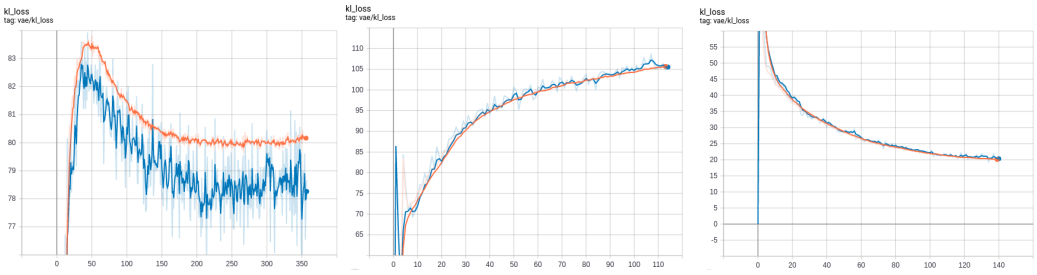
7 Conclusion

In this thesis, I have covered a diversity of topics that have helped me develop an agent for an autonomous vehicle. I was not able to get one that can generalise, but through experiments, I discovered how to accelerate learning. Additionally, I have improved my Python programming and Linux skills which will help me significantly in future.

In the context of the future of self-driving, I am bullish on Google Waymo as I feel their design philosophy is too narrow, a weak dataset and is driven more by monetary gains as opposed to ground-breaking research. Self-driving is an open problem, there is no right way to go about it yet. I feel Tesla headed by Andrej Karpathy has the best attitude and methods to be the first in the world to solve this problem.

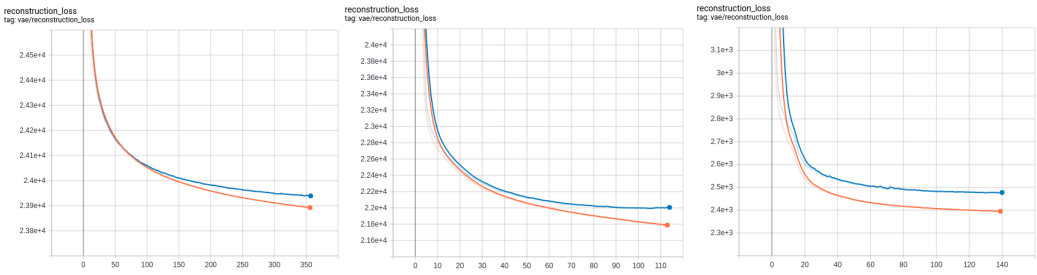
8 Results Appendix

8.1 CVAE losses



Segmentation 'DUV' 'DUV' Augmented

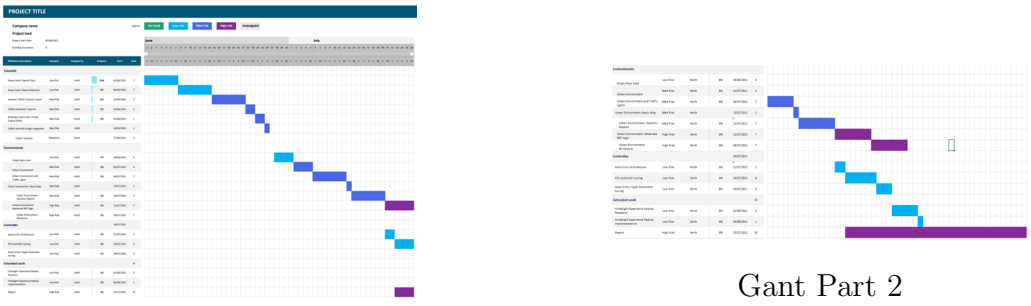
Figure 30: KL Loss



Segmentation 'DUV' 'DUV' Augmented

Figure 31: Reconstruction Loss

8.2 Gantt Charts



Gant Part 1 Gant Part 2

Figure 32: Updated Gant Chart from Project Specification Report

Bibliography

(1992). *Nature of Bayesian Inference*, chapter 1, pages 1–75. John Wiley Sons, Ltd.

Ahmed, Z., Le Roux, N., Norouzi, M., and Schuurmans, D. (2019). Understanding the impact of entropy on policy optimization. In Chaudhuri, K. and Salakhutdinov, R., editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 151–160. PMLR.

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, P., and Zaremba, W. (2017). Hindsight experience replay. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, page 5055–5065, Red Hook, NY, USA. Curran Associates Inc.

Asperti, A., Evangelista, D., and Piccolomini, E. L. (2021). A survey on variational autoencoders from a greenai perspective. *CoRR*, abs/2103.01071.

Bansal, M., Krizhevsky, A., and Ogale, A. S. (2018). Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst. *CoRR*, abs/1812.03079.

Chandak, Y., Theocharous, G., Kostas, J., Jordan, S. M., and Thomas, P. S. (2019). Learning action representations for reinforcement learning. *CoRR*, abs/1902.00183.

Colas, C., Fournier, P., Sigaud, O., and Oudeyer, P. (2018). CURIOUS: intrinsically motivated multi-task, multi-goal reinforcement learning. *CoRR*, abs/1810.06284.

CVPR (2021). Andrej karpathy (tesla): Cvpr 2021 workshop on autonomous vehicles.

DeepMind (2016). Alphago vs lee seedol. [Online; accessed August 24, 2021].

Dosovitskiy, A., Ros, G., Codevilla, F., López, A. M., and Koltun, V. (2017). CARLA: an open urban driving simulator. *CoRR*, abs/1711.03938.

Ekanayake, H., Backlund, P., Ziemke, T., Ramberg, R., Hewagamage, K., and Lebram, M. (2013). Comparing expert and novice driving behavior in a driving simulator. *IxD and A*, 19:115–131.

Fridman, L. (2021a). Lex Fridmanjim keller: Moore's law, microprocessors, and first principles — lex fridman podcast #70. Accessed: 2021-05-05.

Fridman, L. (2021b). Mit 6.s094: Recurrent neural networks for steering through time. Accessed: 2021-05-05.

Gallistel, C. and Matzel, L. D. (2013). The neuroscience of learning: Beyond the hebbian synapse. *Annual Review of Psychology*, 64(1):169–200. PMID: 22804775.

Gangopadhyay, B., Soora, H., and Dasgupta, P. (2021). Hierarchical program-triggered reinforcement learning agents for automated driving. *CoRR*, abs/2103.13861.

Gardner, M. and Dorling, S. (1998). Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. *Atmospheric Environment*, 32:2627–2636.

Gupta, A., Khwaja, A. S., Anpalagan, A., Guan, L., and Venkatesh, B. (2020). Policy-gradient and actor-critic based state representation learning for safe driving of autonomous vehicles. *Sensors*, 20(21):5991.

Ibraheem, N. A., Hasan, M. M., Khan, R. Z., and Mishra, P. K. (2012). Understanding color models: A review. *ARPN Journal of Science and Technology*, pages 265–275.

Johnson, J., Karpathy, A., and Li, F. (2015). Densecap: Fully convolutional localization networks for dense captioning. *CoRR*, abs/1511.07571.

Levine, S. (2018a). Actor critic, lecture 2.

Levine, S. (2018b). Supervised learning of behaviours, lecture 2.

Levinson, J., Askeland, J., Becker, J., Dolson, J., Held, D., Kammel, S., Kolter, J. Z., Langer, D., Pink, O., Pratt, V., Sokolsky, M., Stanek, G., Stavens, D., Teichman, A., Werling, M., and Thrun, S. (2011). Towards

fully autonomous driving: Systems and algorithms. In *Proceedings of IEEE Intelligent Vehicles Symposium (IV '11)*, pages 163 – 168.

Midilli, Y. E. and Parsutins, S. (2020). Optimization of deep learning hyperparameters with experimental design in exchange rate prediction. In *2020 61st International Scientific Conference on Information Technology and Management Science of Riga Technical University (ITMS)*, pages 1–4.

Murphy, R. R. (2000). *Introduction to AI robotics*. MIT Press.

Nguyen, T., Shu, R., Pham, T., Bui, H., and Ermon, S. (2021). Non-markovian predictive coding for planning in latent space.

Pareek, D. and Risteski, A. (2021). The effects of invertibility on the representational complexity of encoders in variational autoencoders. *CoRR*, abs/2107.04652.

Parulekar, M., Ramesh, V., Joshi, P., Padte, V., and Shroff, D. (2013). Modelling the lane contention algorithm for control in intelligent vehicular systems. *AASRI Procedia*, 4:43–49.

Peterson, J. (2021). Jordan Peterson 2016 personality lecture 10: The psychobiology of traits. Accessed: 2021-05-05.

PETTERSSON/COVARIANT, M. (2020). A covariant robot at a knapp-powered warehouse outside berlin. [Online; accessed August 24, 2021].

Press, G. (2021). Andrew ng launches campaign for data centric ai.

Russell, S. and Norvig, P. (2009). *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, USA, 3rd edition.

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

Sherburn, D. (2017). On the feasibility of using fully-convolutional variational autoencoders to advance deep symbolic reinforcement learning. Master's thesis, Imperial College of Science, Technology and Medicine Department of Computing.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, USA.

Teh, Y. W., Bapst, V., Czarnecki, W. M., Quan, J., Kirkpatrick, J., Hadsell, R., Heess, N., and Pascanu, R. (2017). Distral: Robust multitask reinforcement learning. *CoRR*, abs/1707.04175.

Tordesillas, J. and Arbelaiz, J. (2019). Personalized cancer chemotherapy schedule: a numerical comparison of performance and robustness in model-based and model-free scheduling methodologies. *CoRR*, abs/1904.01200.

Ulbrich, S. and Maurer, M. (2013). Probabilistic online pomdp decision making for lane changes in fully automated driving. In *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*, pages 2063–2067.

van der Laan, L. N., Papies, E. K., Hooge, I. T. C., and Smeets, P. A. M. (2017). Goal-directed visual attention drives health goal priming: An eye-tracking experiment. *Health Psychology*, 36(1):82 – 90.

Vergara, M. L. (2019). Accelerating training of deep reinforcement learning-based autonomous driving agents through comparative study of agent and environment designs. Master's thesis, NTNU. url= https://github.com/bitsauce/Carla-ppo.

Wang, F.-Y., Zhang, J. J., Zheng, X., Wang, X., Yuan, Y., Dai, X., Zhang, J., and Yang, L. (2016). Where does alphago go: from church-turing thesis to alphago thesis and beyond. *IEEE/CAA Journal of Automatica Sinica*, 3(2):113–120.

Zemouri, R., Levesque, M., Amyot, N., Hudon, C., Kokoko, O., and Tahan, S. A. (2020). Deep Convolutional Variational Autoencoder as a 2D-Visualization Tool for Partial Discharge Source Classification in Hydrogenerators. *IEEE Access*, 8:5438–5454.

Zhang, R., Zhu, J., Zha, Z., Dauwels, J., and Wen, B. (2021). R3l: Connecting deep reinforcement learning to recurrent neural networks for image denoising via residual recovery.