HAW
HAMBURG

Bachelor Thesis
Lachlan George Mackay Townsend

Autonomous Vehicle Control using Deep Reinforcement Learning

FACULTY OF ENGINEERING AND COMPUTER SCIENCE
Department of Information and Electrical Engineering

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

HAMBURG UNIVERSITY
OF APPLIED SCIENCES

Hochschule für Angewandte
Wissenschaften Hamburg

Lachlan George Mackay Townsend

Autonomous Vehicle Control using Deep Reinforcement Learning

Bachelor Thesis based on the examination and study regulations
for the Bachelor of Engineering degree programme
Bachelor of Science Information Engineering
at the Department of Information and Electrical Engineering
of the Faculty of Engineering and Computer Science
of the University of Applied Sciences Hamburg

Supervising examiner: Prof. Dr.-Ing. Marc Hensel
Second examiner: Prof. Dr. Heike Neumann

Day of delivery: 30th August 2022

Lachlan George Mackay Townsend

Title of Thesis

Autonomous Vehicle Control using
Deep Reinforcement Learning

Keywords

Autonomous, Vehicle, AI, Deep Learning, Reinforcement Learning, Neural Networks

Abstract

In this thesis, an autonomous vehicle control system is developed using Deep Neural Networks, which are trained using reinforcement learning in a 3D simulation environment. The resulting system is implemented on a remote control car and tested in a real-world experimental environment.

Lachlan George Mackay Townsend

Thema der Arbeit

Autonome Fahrzeugsteuerung mit
Deep Reinforcement Learning

Stichworte

Autonom, Fahrzeug, KI, Deep Learning, Reinforcement Learning, Neuronale Netze

Kurzzusammenfassung

Diese Thesis befasst sich mit der Entwicklung eines autonomen Fahrzeugsteuerungssystems anhand von Deep Neural Networks, die mit Reinforcement Learning in einer 3D-Simulationsumgebung trainiert werden. Das System wird auf einem ferngesteuerten Modellauto implementiert und in einer realen Versuchsumgebung getestet.

Contents

List of Figures **vii**

List of Tables **ix**

Abbreviations **x**

1. Introduction **1**

2. Foundations **3**

2.1. Autonomous Vehicles . . . 3
2.1.1. Levels of Vehicle Autonomy . . . 3
2.1.2. Impacts of Autonomous Vehicle Technology . . . 4
2.2. Artificial Intelligence . . . 5
2.2.1. Machine Learning . . . 6
2.2.2. Neural Networks . . . 7
2.3. Reinforcement Learning . . . 12
2.3.1. The Markov Decision Process . . . 13
2.3.2. Policies and Value Functions . . . 14
2.3.3. Learning Methods . . . 15
2.3.4. Deep Q-Learning . . . 16
2.4. Digital Image Processing . . . 17
2.4.1. Canny Edge Detection . . . 17
2.4.2. Binary Thresholding . . . 20
2.5. Networking and Communication Protocols . . . 20
2.5.1. Protocols and Sockets . . . 22
2.5.2. Message Queuing Telemetry Transport . . . 23
2.6. Hardware . . . 23
2.6.1. Truggy Remote Control Car . . . 23
2.6.2. Raspberry Pi Single Board Computer . . . 24

2.6.3. Raspberry Pi Camera 25
2.6.4. PCA9685 Pulse Width Modulation Controller 26

3. Requirements Analysis 27
3.1. System Context 27
3.1.1. Experimental Environment 27
3.1.2. System Boundary 28
3.2. Stakeholders 28
3.3. Use Cases 30
3.3.1. Follow Lane 31
3.3.2. Set Steering 31
3.3.3. Set Throttle 32
3.3.4. Video Stream on/off 32
3.3.5. Control System run/stop 32
3.4. Restrictions 33
3.5. Requirements 33
3.5.1. Functional Requirements 34
3.5.2. Non-Functional Requirements 34

4. Concepts 35
4.1. Desired Outcomes 35
4.2. Operational Parameters 35
4.2.1. Normal Operation 36
4.2.2. System Failure Conditions 36
4.2.3. Real Time Operation 36
4.3. System Behaviour 37
4.4. Implementation Strategies 38
4.5. Foreseen Difficulties 40
4.6. Implementation Strategy Selection 41
4.6.1. Training Environment 42
4.6.2. Image Processing 42
4.6.3. System Design 42
4.6.4. Test Setup Within the Experimental Environment 43

5. Implementation 44
5.1. Training 44
5.1.1. Simulation Environments 44

5.1.2. Reinforcement Learning Setup 45
5.1.3. Comparison of Deep Neural Network Architectures 48
5.1.4. Controlling Variables 50
5.1.5. Image Processing and Training 51
5.2. Experimental Setup 54
5.2.1. Graphical User Interface 54
5.2.2. Experimental Environment 55
5.2.3. Communication for User Control 56

6. Results **57**
6.1. Training 58
6.2. Comparison of Training Results 60
6.3. Experimental Environment 61

7. Discussion **63**
7.1. Summary of Results 63
7.2. Review of Requirements 63
7.3. Limitations 65
7.4. Future Work 67

8. Conclusion **69**

Bibliography **70**

Appendix A. DNN Model Summaries **75**
A.1. Modified "DAVE-2" Model Summary 75
A.2. Open Source Donkey Gym Model Summary 76
A.3. Custom DNN Model Summary 76

Appendix B. Raspberry Pi Preparation Steps **78**

Declaration **80**

List of Figures

2.1. Neural Network structure . . . 8
2.2. Activation functions: sigmoid, tanh and ReLU . . . 9
2.3. Neural Network node . . . 10
2.4. Reinforcement Learning training cycle . . . 13
2.5. Canny non-maximum suppression using double threshold . . . 19
2.6. Canny edge detection example . . . 20
2.7. Truggy remote control car . . . 24
2.8. Raspberry Pi 4 Model B SBC . . . 25
2.9. Pi Camera Module . . . 25
2.10. PCA9685 pulse width modulation controller . . . 26

3.1. System context . . . 28
3.2. Use case diagram . . . 30

4.1. Processing and control pipeline . . . 37
4.2. State machine diagram . . . 38
4.3. Track zone classification diagram. . . . 43

5.1. Donkey Gym open source simulation environments . . . 45
5.2. Unsuitable track simulation environment . . . 45
5.3. "Waveshare" track simulation environment . . . 46
5.4. Reinforcement Learning activity diagram. . . . 47
5.5. Modified "DAVE-2" DNN architecture . . . 49
5.6. Open source Donkey Gym DNN architecture . . . 49
5.7. Custom DNN architecture . . . 50
5.8. Unprocessed RGB and Canny edge detection processed image frames . . . 51
5.9. "DAVE-2" DNN training outcomes using Canny edge detection image processing . . . 52

5.10. Unprocessed RGB and binary thresholding for image segmentation processed image frames 54
5.11. Binary thresholding for image segmentation processed frames (3D simulator and experimental environment) 54
5.12. Truggy control centre GUI 55

6.1. Training outcomes for modified "DAVE-2" DNN architecture 58
6.2. Training outcomes for open source Donkey Gym DNN architecture 59
6.3. Training outcomes for custom DNN architecture 60
6.4. Test outcomes using the trained Donkey Gym open source model in the experimental environment, plotted per success category. 62

List of Tables

3.1. Project stakeholders . . . 29
3.2. Use case: follow lane . . . 31
3.3. Use case: set steering . . . 31
3.4. Use case: set throttle . . . 32
3.5. Use case: video stream on/off . . . 32
3.6. Use case: control system run/stop . . . 33

6.1. Training results . . . 61
6.2. Braking time . . . 62

Abbreviations

2D Two Dimensional.

3D Three Dimensional.

AI Artificial Intelligence.

API Application Programming Interface.

CPU Central Processing Unit.

CTE Cross Track Error.

DNN Deep Neural Network.

DQN Deep Q-Network.

FPS Frames per Second.

GPU Graphics Processing Unit.

GUI Graphical User Interface.

HSV Hue Saturation Value.

IEEE Institute of Electrical and Electronics Engineers.

IoT Internet of Things.

IP Internet Protocol.

ISO International Organization for Standardization.

MDP Markov Decision Process.

ML Machine Learning.

MQTT Message Queuing Telemetry Transport.

NN Neural Network.

OSI Open Systems Interconnect.

PWM Pulse Width Modulation.

RAM Random Access Memory.

RC Remote Control.

ReLU Rectified Linear Unit.

RGB Red Green Blue.

RL Reinforcement Learning.

SAE Society of Automotive Engineers.

SBC Single Board Computer.

TCP Transmission Control Protocol.

UDP User Datagram Protocol.

1. Introduction

Machine learning has made enormous technological advances in the last two decades, developing from a field of scientific investigation to an established scientific and engineering tool [22, 16] that is expanding technological capabilities in applications such as autonomous vehicles, medical diagnosis and robotics [32].

As the transport industry embraces this technology and the emerging autonomous vehicle field matures, autonomous vehicles are becoming increasingly commonplace. Amongst other benefits, this technology stands to improve safety, efficiency and accessibility [42, 30].

With such promising benefits available from autonomous vehicle technology, it is important that development of this technology is supported by both industry and universities, to ensure high standards and affordable results so that it is available in as many vehicles as possible.

The aim of this thesis is to gather knowledge and hands on experience in the Reinforcement Learning field through the implementation of Deep Reinforcement Learning for the autonomous control of a remote control car, using affordable and accessible hardware components. This thesis places some restrictions upon the resources available for development, which are discussed in Section 3.4. The ultimate aim is that the vehicle should be able to autonomously navigate around a scale track, the definition of which is provided in Section 3.1.

In this thesis, Chapter 2 begins by providing an overview of relevant theoretical and practical concepts. Chapter 3 then details the system functionality, performs a stakeholder analysis and provides concrete system requirements. Chapter 4 follows by providing an overview of concepts involved in the development of the system, whereby desired outcomes and implementation strategies to achieve the requirements set out in Chapter 3 are discussed. It culminates in concrete decisions about the implementation strategy. These requirements are then used to structure the experimental setup, as well as the

implementation and training process, which is described and documented in Chapter 5. Chapter 6 presents the results achieved by the implemented system. Following the development of the system and the presentation of results, Chapter 7 then discusses the achieved results and provides an analysis in regards to the requirements outlined in Chapter 3. Suggestions for future work and systemic improvement are also made during this chapter. This thesis then culminates in Chapter 8, where the overall project and its development is summarised.

2. Foundations

This chapter presents the theoretical and practical foundations required to understand the processes and concepts that are used and explored within this thesis.

2.1. Autonomous Vehicles

The inclusion of autonomous technology into vehicles seeks to alleviate or even replace the requirement for human drivers [12, 38]. Autonomous vehicle control technology can include either electronic or mechanical devices, or both. Examples of early autonomous vehicle control systems focus on cruise control functionality such as speed, brake and lane control [12]. Modern technology, including the sophisticated computing abilities that are now available, is able to offer significantly more sophisticated and comprehensive abilities [12].

2.1.1. Levels of Vehicle Autonomy

As per the Society of Automotive Engineers [20], the following levels of vehicle autonomy are defined:

L1. Driver Assistance: The autonomous system controls one of either longitudinal (acceleration and braking) or lateral motion (steering), but performs no form of object or event detection [20]. L1 systems can operate adaptive cruise control or lane keeping assistance, but not both simultaneously. The human driver is responsible for the operation of the non-automated function [38].

L2. Partial Driving Automation: The autonomous system simultaneously controls both longitudinal and lateral motion, but performs no form of object or event detection [20]. L2 systems operate adaptive cruise control and lane keeping assistance

simultaneously, however the human driver must constantly monitor the situation and takeover vehicle control if necessary [38].

L3. Conditional Driving Automation: The autonomous system entirely performs the dynamic driving control task within specified operational and environmental limitations [20]. L3 systems allow the human driver to perform other activities such as using a mobile phone, however they must be prepared to takeover control at the request of the autonomous system [38].

L4. High Driving Automation: The autonomous system entirely performs the dynamic driving control task and has fallback behaviour for avoiding collisions and other undesirable events within operational and environmental limitations [20, 11]. L4 systems encompass highway pilots and fully autonomous systems that operate on closed campuses (geographically restricted areas of operation) [38]. This level of autonomy allows human drivers to be fully occupied with another task, such as sleeping, as the autonomous systems fallback to minimum risk conditions if and when needed [38, 11].

L5. Full Driving Automation: The autonomous system entirely performs the dynamic driving control task and has fallback behaviour for avoiding collisions and other undesirable events without limitation [20]. L5 systems can operate without a driver. People present in systems with L5 control activated are considered passengers rather than drivers [38].

2.1.2. Impacts of Autonomous Vehicle Technology

Vehicles with L2 autonomy, such as Tesla's Autopilot [21] and General Motors' Super Cruise [14] are already deployed on a global scale. Motivation for the development and rollout of this technology comes from many directions such as safety, efficiency and accessibility [12]. However, considering the exponential growth trend in autonomous vehicle research between 2012 and 2018 [12], both positive and negative impacts of this burgeoning technology should be considered. Whilst obvious effects are expected for vehicles and the systems and infrastructure with which they directly interact, flow-on effects are also anticipated at societal and transport system levels [12].

Negative traffic conditions such as accidents, oscillations and congestion are often caused by the random nature of human driving behaviour [12]. The traffic flow patterns that result are inefficient and have a strong negative impact upon vehicle emissions and the

environment [12]. Research has shown that autonomous vehicles, in particular connected autonomous vehicles, can reduce the occurrence of these negative traffic conditions in comparison to vehicles under human control [12, 41].

Traffic oscillation (spring effect) caused by merging near highway entry and exit points is the leading cause of highway congestion [43, 12]. Research by Zhou et al. [43] showed that the use of autonomous vehicles could alleviate this negative effect at merge points, resulting in improvements to congestion, safety, vehicle throughput and average merging speed [43, 12].

Lane changing is considered to be a risky manoeuvre due to the potential for collision with vehicles in front of and behind the manoeuvring vehicle in both its current and destination lane [12]. An inaccurate or flawed spatial assessment when performing a lane change can result in consequences ranging from traffic oscillation, to collisions and potentially loss of life [12]. By automating this procedure, research indicates that a reduction of 4-10% of all accidents caused by human error can be achieved [12].

As autonomous vehicle technology continues to become more commonplace, it will result in positive effects upon traffic flow, the capacity of existing infrastructure, vehicular operating cost, cost of travel and vehicular hours travelled [12]. A further propagation of this positive effect upon infrastructure planning and design, land use (reduced required parking spaces, increased required charging spaces), traffic safety, public health and the environment is also expected [12]. As Faisal et al. [12] note, there will be a great deal of pre-deployment planning, policy changes and infrastructure requirements in order to support the successful adoption of autonomous vehicle technology [12].

2.2. Artificial Intelligence

At its inception Artificial Intelligence (AI) was conceived as a research field to investigate the theorem that human learning and other intelligent behaviour could be wholly reproduced using machines [29]. It is from these beginnings that the concepts of Artificial Intelligence, Machine Learning (ML), Neural Networks (NN) and Deep Neural Networks (DNN) stem [29]. Early implementations of AI centered around the concept of modelling human behaviour in machines, however this concept has been modernised to form a more inclusive approach of solving extremely complex problems by any means,

instead of imposing artificial limitations upon systems to restrict solutions to that of human behaviour [8].

2.2.1. Machine Learning

Modern ML is the science of using a data driven approach to build self-improving algorithms and computational systems [22]. It finds itself at the intersection of computer science and statistics, whereby it uses repetitive training to optimise a particular performance metric [22]. There are three major paradigms of ML: supervised learning, unsupervised learning and reinforcement learning (RL).

Supervised learning focuses on function approximation, where the task is for a model to learn an output for a particular given input [22]. In this paradigm, training data takes the form of (x, y) pairs which are used to learn the mapping $f(x) \mapsto y$ or the probability distribution over y given the input x [22]. Practically speaking, these (x, y) pairs are sets of labelled training data, where the data is used as the input (x) to a model and the label is the desired output prediction (y) from the model [25]. Supervised learning can commonly be identified as either regression, where the output $(\hat{y})$ is continuous, or classification, where output $(\hat{y})$ values are labels which are assigned to particular categories [25]. These mappings can be implemented in many different forms, including but not limited to: decision trees, regression models, NNs and kernel machines [22]. This algorithmic diversity provides varying levels of computational complexity that can be selected depending upon the application and its requirements [22]. Common applications of supervised learning include classification tasks and medical diagnosis [22].

Conversely to supervised learning, unsupervised learning is used to analyse unlabeled data from which characteristic structural patterns of the data are learnt [22]. Often these characteristics are underlying and hidden within the data [25]. This is then used to perform clustering, where data is clustered given the absence of labels [22]. One huge advantage of unsupervised learning is that it removes the requirement of laboriously labeling data, which in itself opens opportunities for efficient use of enormous data repositories, which would not be possible using supervised learning [22]. Unsupervised learning is commonly used for tasks such as natural language processing and novel image generation [25].

Reinforcement learning is considered intermediate between supervised and unsupervised learning due to the information contained in training data [22]. RL sets out to learn

a policy (strategy) for an agent (commonly an NN or DNN) acting within a dynamic environment, where a variable reward is given for every action step [22]. The agent is trained in steps to choose an action for any given input state, where the objective is to maximise reward [22]. In contrast to both supervised and unsupervised learning, the actions taken by the agent are directly reflected in the environment and therefore the agent receives immediate feedback upon its actions [25].

2.2.2. Neural Networks

Computations in the human brain are performed by an enormous network of neurons, which communicate using electric signals sent via axons, synapses and dendrites [24]. As early as the 1940's, these neurons have been modelled as switches that are activated by input from other neurons [24]. This input is either amplified or dampened via weighting by the strength of the interconnecting synapse [24]. It is upon these biological foundations that the Artificial Neural Network (referred to in this thesis as NN) is based [24].

NNs are structures of layers of nodes, with a designated input and output layer and often hidden layers in between (see Figure 2.1) [24]. These layers are made up of artificial neurons (nodes), the earliest example of which was published by Frank Rosenblatt in 1958 and is known as a perceptron [37, 25]. The perceptron operates upon binary data only [25]. It receives input data from nodes in preceding layers to which it applies a weight. The resulting values are then summed. If this sum reaches or exceeds a particular threshold level, the perceptron is activated and its output is turned on, otherwise it remains un-activated and the output remains off [25].

Modern NNs commonly consist of nodes that can operate on non-binary data. In this case, the concept remains similar to that of the perceptron, however the node's output is continuous as opposed to the discrete binary function of a perceptron [24]. The behavioural characteristics of this continuous output are referred to as the node's activation function.

The model of an NN node shown in Figure 2.3 depicts inputs from the previous layer ($x_1 ... x_n$) and their associated weights ($w_1 ... w_n$). The total input to the node can be described using Equation 2.1 and the output of the node can be expressed using Equation 2.2 [24].

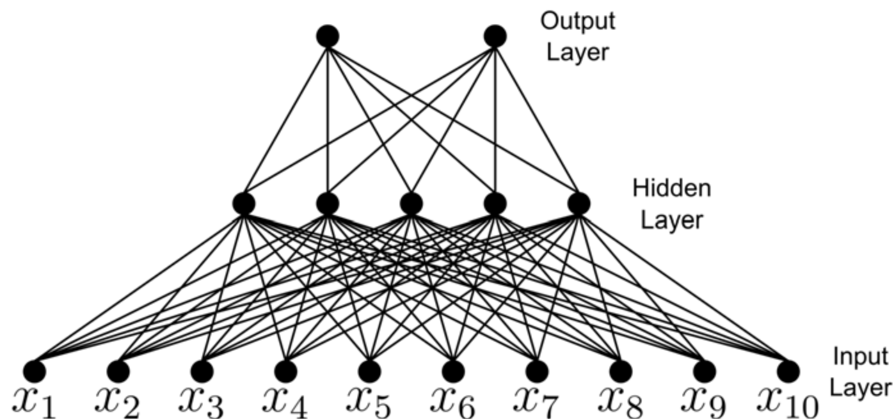

Figure 2.1.: Neural Network structure, adapted from [24]

$$\sum_{i=1}^{n} w_i x_i = w_1 x_1 + w_2 x_2 + ... + w_n x_n \tag{2.1}$$

$$a = \sigma \left(\left(\sum_{i=1}^{n} w_i x_i \right) - t \right) \tag{2.2}$$

In Equation 2.2 a represents the node's output, σ represents the node's activation function and t represents the node's threshold. The threshold value is of crucial importance to the functionality of an NN, as it is one of the key parameters which is adjusted during the training process [25]. A threshold value can also be referred to as a bias value, which is the inverse of the threshold value ($b = -t$).

There are a number of activation functions that can be used in NNs, with the most common being the sigmoid function, tanh function and the rectified linear unit function (often abbreviated as ReLU) [25, 24].

The most frequently used activation function, the sigmoid function, is defined in Equation 2.3 [25]. The smooth curve of the sigmoid function (see Figure 2.2) allows fine-tuning of a node's weights in order to achieve highly accurate nodal activation [25].

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.3}$$

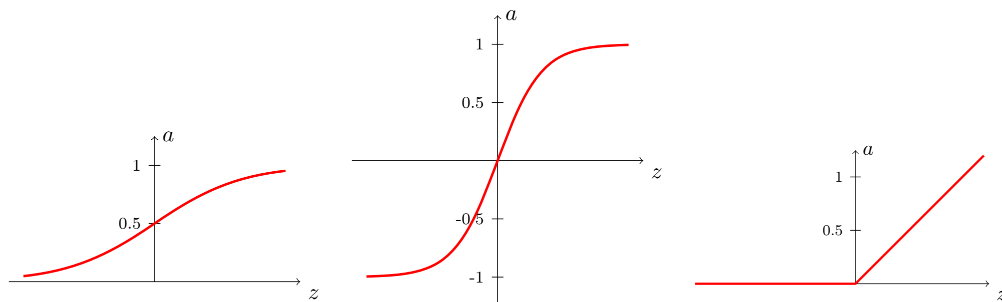

Figure 2.2.: Activation functions: sigmoid, tanh and ReLU.

The tanh function extends the output range of the sigmoid function, from $[0, 1]$ to $[-1, 1]$, whilst maintaining a very similar curve shape (see Figure 2.2) [25]. The tanh function is defined in Equation 2.4 [25].

$$\sigma(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{2.4}$$

Finally, the ReLU function is defined by Equation 2.5 [25]. The ReLU function has a strikingly different shape to both the sigmoid and tanh functions (see Figure 2.2), which is matched by very different behaviour. If the input to the ReLU function is equal to or less than zero, it returns zero, or else it returns the input value [25].

$$\sigma(z) = \max(0, z) \tag{2.5}$$

Figure 2.3 provides an example of an NN node, in place with an activation function (σ).

As previously established, nodes are grouped together into layers, the most simple of which is a fully connected (dense) layer in which nodes receive inputs from every node in the previous layer, as is depicted in Figure 2.1 [25].

A significantly more complex layer type is the convolutional layer, which is commonly used for image processing applications due to its ability to perform feature extraction via spatial pattern processing [25]. This is achieved through the use of kernels which perform convolutions whilst windowing across discrete positions of the input array [25]. Conversely to fully connected layers, convolutional layers do not have weights for every input, but rather each kernel contains a set of weights, which are re-used at each position

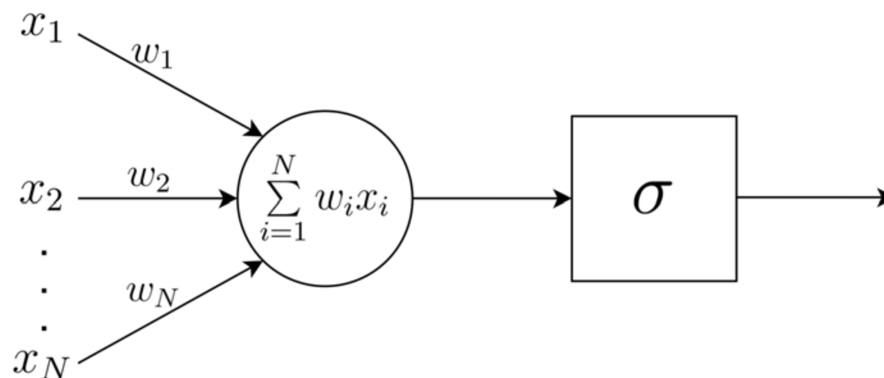

Figure 2.3.: Neural Network node, adapted from [24]

to calculate a weighted sum [25]. As the windowing process occurs, each kernel finds the position (or lack thereof) of specific features, causing activations. These activations are structured into a multi-dimensional activation map as the layer's output [25]. In order to make use of the output of a convolutional layer, another layer type is required, the flatten layer. Its purpose is to convert the multi-dimensional activation map that is output from a convolutional layer into a single-dimension array. This facilitates connection to a fully connected layer that can be used to represent a vector of classes [25].

An important subdomain of neural networks is Deep Neural Networks (DNNs), which refers to an NN with multiple hidden layers (usually three or more) [25]. This concept will be discussed further specifically in respect to reinforcement learning in Section 2.3.4

As a blank slate, an NN is not able to produce meaningful results, it must first be trained. The iterative training process involves making small changes to weights and threshold values, such that meaningful results can be achieved [24]. Various aspects of the training process are controlled through so-called hyperparameters, which are used to optimise the learning process [25].

Essential to training an NN is the random initialisation of nodal weights and threshold values [25], meaning that initial results produced by the NN are not meaningful. These weights and threshold values are then adjusted throughout the iterative training process and the accuracy of the results produced is measured using a cost function. Two common cost functions are mean squared error and cross-entropy cost, shown in Equations 2.6 and 2.7 respectively [25].

$$C = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.6}$$

$$C = -\frac{1}{n}\sum_{i=1}^{n}[y_i \ln \hat{y}_i + (1 - y_i)\ln(1 - \hat{y}_i)] \tag{2.7}$$

In both Equation 2.6 and Equation 2.7, for each given instance i, y represents the desired outcome (the label) and $\hat{y}$ represents the value produced by the NN (the prediction) [25].

Once calculated, the result of the cost function is then used to optimise the NN, that is, it enables the learning process during training with the goal of minimising the cost value [25]. To do this, all weights and threshold values are adjusted proportionally to their contribution to the cost value produced during inference. This process is known as backpropagation [25].

In its most simple form, the optimisation process constitutes a gradient descent process, in which each training iteration seeks to minimise the cost value, that is, it traverses a complex multi-dimensional function to find minima, one step at a time [25]. The step size is controlled by a hyperparameter called learning rate. A significant limitation of gradient descent is that local minima can prevent the process from finding the true global minimum [25]. As such, more sophisticated optimisation techniques such as stochastic gradient descent, which operates on sub-sets of training data, can be more effective. By using a so-called minibatch (sampled sub-set of training data), noise is introduced into the calculation of the gradient and as such local minima can be successfully traversed [25].

Further improvements beyond that of stochastic gradient descent can be made to the optimisation process through the introduction of a parameter called momentum. While there are several optimisation techniques that implement the calculation and use of momentum differently, the core concept is that the speed when moving on the cost curve is taken into account when preforming the current backpropagation step [25]. This enables the descent process to move past local minima, increasing the probability of finding the true global minimum [25]. A widespread method of optimisation that takes momentum into account is the so-called Adam optimiser [25].

As introduced in Section 2.2.1, the overarching aim of machine learning is to learn the mapping $f(x) \mapsto y$. In a macro sense, the training process works to fit the function

to the data, with the aim of minimising the cost function and hence maximising the accuracy of the NN. However, during this process a phenomenon called overfitting can occur. Overfitting describes the scenario when an NN fits too closely to the data upon which it is being trained [25]. On a practical level, this results in increasingly higher accuracy when inference is performed upon training data, however decreasing accuracy when inference is performed on data that is not included in the training data set [25].

2.3. Reinforcement Learning

As outlined in Section 2.2.1, reinforcement learning (RL) is the process in which a policy is learnt by an agent acting within a dynamic environment, where a variable reward is given for every action step [22]. RL stands out from other fields in ML because it addresses sequential decision making [13].

For the purpose of explicit clarity the following terms, essential to RL, are defined [25]:

- **Agent:** The algorithm (commonly an NN or DNN) being trained to perform a specific task.
- **Policy:** The resulting function that is learnt by the agent.
- **Environment:** The setting in which the agent exists and from which it collects information.
- **Action:** The result of the policy processing information collected from the environment.
- **Reward:** The quantitative feedback provided to the algorithm after performing an action.
- **State:** The resulting change in the environment after an agent performs an action (which is clarified in Section 2.3.1 to be synonymous with observation).

In essence, the RL agent learns an intended behaviour via trial and error. This is in contrast to other techniques which assume definitive environmental knowledge a priori [13]. Because of this characteristic, the RL agent can be trained via direct interaction with the environment, instead of requiring complete knowledge about and control over the environment [13].

The environmental interaction within the training cycle can be characterised as a discrete time stochastic control process [13, 40] such that the agent begins in an initial state ($s_0 \in S$) and gathers an initial observation ($\omega_0 \in \Omega$) [13]. For every time step (t), the agent takes an action ($a_t \in A$) [13]. Consequently the agent receives a reward ($r_t \in R$, given (s_t, a_t)), the state advances to the next state ($s_{t+1} \in S$) and the agent gathers a subsequent observation ($\omega_{t+1} \in \Omega$) [13]. This process is illustrated in Figure 2.4.

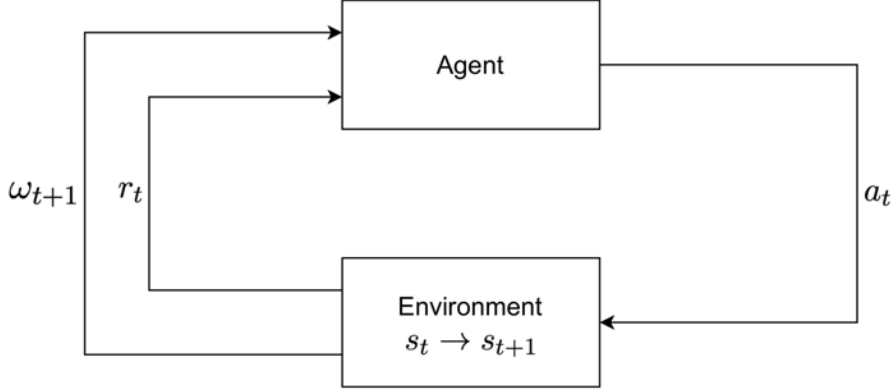

Figure 2.4.: RL training cycle, adapted from [13]

2.3.1. The Markov Decision Process

RL problems can be represented mathematically using the Markov Decision Process (MDP) [13, 40]. In order to define the MDP the Markov property must first be defined. It states that future states depend only upon the current observation, not upon any other historical observations [13, 40]. In order for a discrete time stochastic control process to have the Markov property it must fulfill both Equations 2.8 [13] and 2.9 [13].

$$P(\omega_{t+1} \mid \omega_t, a_t) = P(\omega_{t+1} \mid \omega_t, a_t, ..., \omega_0, a_0) \tag{2.8}$$

$$P(r_t \mid \omega_t, a_t) = P(r_t \mid \omega_t, a_t, ..., \omega_0, a_0) \tag{2.9}$$

Equation 2.8 states that the probability of an observation ω at time $t+1$, given the current observation and action, is equal to the probability of an observation ω at time $t+1$, given the full history of observations and actions.

Equation 2.9 states that the probability of a reward r at time t, given the current observation ω and action a is equal to the probability of a reward r at time t, given the full history of observations and actions.

The MDP can now be defined as the tuple (S, A, P, R, γ) [25] such that:

- **State space:** S,
- **Action space:** A,
- **Transition distribution:** P,
- **Reward distribution:** R,
- **Discount factor:** γ.

Both the reward and transition are probability distributions, meaning that depending upon the time step, the same state-action pair (s, a) may produce different results [25].

The discount factor, γ, is used to weight immediate rewards more strongly than rewards that require future time steps in order to be obtained [25]. This reinforces behaviour which recognises that immediate rewards are more likely to be obtained than those that are a number of time steps away [25].

Because a system that is an MDP is fully observable, it follows that the observation is the same as the environment ($\omega_t = s_t$) [13].

2.3.2. Policies and Value Functions

RL policies (π) can be either deterministic, where each action has a guaranteed resulting state and reward, or stochastic, where each action has a varying probability of a resulting state and reward [13]. In the search for an optimal policy ($\pi(s, a) \in \Pi$), the value function as per Equation 2.10 [26] is used. This means that for any given policy (π), the value of a state is the expected sum of rewards when starting from that state, whilst also taking into account the discount factor (γ).

$$V^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t r_t\right] \tag{2.10}$$

Given Equation 2.10, the optimal value function is defined in Equation 2.11 [13]. This denotes the policy that has a value function with the greatest return.

$$V^*(s) = \max_{\pi \in \Pi} V^{\pi}(s) \tag{2.11}$$

2.3.3. Learning Methods

As described in Section 2.3 and depicted in Figure 2.4, the agent learns a policy through iterative training with environmental data [25]. This training process can occur using a number of different structures, including online learning, offline learning, on-policy learning and off-policy learning [13, 40].

In an online learning setting, the agent interacts with and gathers experience directly from an environment as it is trained, which allows experience to be gathered in an exploratory (trial and error) manner [13]. Additionally, experiences can be stored in a replay memory for post-processing [25].

Offline learning strategies use batches of environmental data which were collected prior to training, hence there is only limited data available to the agent [13]. Because offline learning uses previously collected data, it does not allow for the possibility of environmental interaction [13].

On-policy learning utilises a single policy, which is used to initiate actions and is then adjusted using the reward issued as a result of that same action [13, 40].

In contrast to on-policy learning, off-policy learning utilises two policies [13]. The first, known as the main policy, is used for environmental interaction, from which actions are generated. The experience generated from this environmental interaction is stored within a replay buffer, which is then used by the second policy, known as the target policy, to calculate rewards [13]. The replay buffer contains historical experience, which, depending upon the replay buffer size, may have been generated from numerous policy iterations [13]. Because off-policy learning makes use of historical experience which can come from multiple previous policy iterations, it is more sample efficient in comparison to on-policy learning [13].

2.3.4. Deep Q-Learning

First introduced by Mnih et al. in 2015 [31], Deep Q-learning and Deep Q-networks (DQNs) use the Q-learning framework in combination with a DNN. Deep Q-Learning finds its foundations in Q-learning, a technique where a table of values is filled using one table position for every state-action pair [13]. Each value in the table represents the quality of an action to produce an outcome (reward) given the current state. Q-learning uses the Bellman equation of optimality (see Equation 2.12), whereby the agent pursues a greedy strategy by considering both the immediate reward and the discounted long term reward [26, 40].

$$V^{\pi}(s_t) = E\left[r_{t+1} + \gamma V(s_{t+1})\right] \tag{2.12}$$

Similarly to the state values defined in Section 2.3.2, Q-learning also defines an action value as per Equation 2.13 [26], which states that the value of any given state is equal to the maximum action value that can be obtained from the current state[1].

$$Q(s, a) = r(s, a) + \gamma \max_{a' \in A} Q(s', a') \tag{2.13}$$

Deep Q-networks address some shortcomings of Q-learning by employing two heuristic techniques, target networks and memory replay [13].

A target network, as described in off-policy learning (see Section 2.3.3), is not updated with every action taken by the agent. This limits the speed at which instabilities can propagate to a model during training [13].

Memory replay uses an epsilon greedy policy to collect experience and saves it into a replay buffer [13]. The epsilon greedy method is pivotal to DQNs, as during the early stages of training the results of Q-approximation are ineffective due to their uniformly distributed nature [26]. To combat this, the epsilon greedy policy selects between either a random action or an action derived from the Q-policy with a probability of ϵ [13, 26]. This allows random behaviour during early stages of training to expedite environmental exploration and efficient Q-policy refinement at the later stages of training [26].

[1] Note that Q is used to denote the value of an action, where previously V was used to denote the value of a state.

Once experiences have been collected using the epsilon greedy policy (see Equation 2.14), updates then occur using so-called minibatches (sets of tuples) [13, 26, 25], which are randomly selected from the replay buffer [13]. As described in Section 2.3.3, this helps to diversify the number of experiences sampled and hence makes the process more efficient [13, 26].

$$\arg\max_{a \in \mathcal{A}} Q(s, a) \tag{2.14}$$

2.4. Digital Image Processing

An image can be represented digitally in pixel positions by discretising (quantising) light values from a sensor [33]. A grayscale (black and white) image is represented in the form of a two dimensional (2D) array. A colour image is most commonly represented as a three dimensional (3D) array in either a red, green and blue (RGB) or hue saturation value (HSV) format [33].

Digital image processing (shortened to image processing) is the process of applying image transformations in the form of mathematical operations to the values in these arrays in order to produce a desired effect or outcome [33]. Image processing has many purposes, including but not limited to: image enhancement, compression, restoration and feature extraction [33].

2.4.1. Canny Edge Detection

Canny edge detection is an image processing technique which is used to perform feature extraction in order to find object boundaries (edges) within an image [9]. Naturally, objects come in an infinite number of size, shape and colour combinations, which, when captured in the form of a digital image, are discretised into a set of pixel values. When captured in colour, images can be converted to a grayscale format whereby the colour information is replaced by grayscale values which are used to visually represent pixels from full black to full white, with discretised gray steps in-between.

The process of Canny edge detection on a grayscale image is split into four distinct steps. Firstly, the image is processed using a Gaussian blur as per Equation 2.15 [39], where the σ variable is used to control the intensity of the resulting blur. The purpose of blurring

the image is to reduce the probability of false edge detections and the effect of noise upon the edge detection process [39].

$$B(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\pi\sigma^2}} \tag{2.15}$$

Edge detection is then performed upon the blurred image. As objects within a grayscale image will intrinsically have different intensities of gray (or may be full black or full white), edge detection can be achieved by assessing intensity differences between neighbouring pixels [17, 39]. This can be practically implemented by assessing gradients, using a partial derivative [17]. Equations 2.16 and 2.17 [17] detail how this is mathematically achieved for a given pixel $g(x, y)$.

$$|grad_x g(x, y)| = \left| \frac{\partial}{\partial x} g(x, y) \right| \tag{2.16}$$

$$|grad_y g(x, y)| = \left| \frac{\partial}{\partial y} g(x, y) \right| \tag{2.17}$$

However, as a digital image represents a set of discrete values, the partial derivatives previously described in Equations 2.16 and 2.17 must be approximated. The approximation for these operations to be used with discrete data is detailed in Equations 2.18 and 2.19 [17]. Using windowing, this process is repeated over the whole image [39].

$$\frac{\partial}{\partial x} g(x, y) \approx \frac{g(x+1, y) - g(x-1, y)}{2} \tag{2.18}$$

$$\frac{\partial}{\partial y} g(x, y) \approx \frac{g(x, y+1) - g(x, y-1)}{2} \tag{2.19}$$

With the gradient values now calculated, the results are compared to a primary threshold value, which is used to identify true edges only [17]. This process is often referred to as non-maximum suppression [39, 9]. It iterates over the array of gradient values on the x and y axes separately, whereby it selects values which meet or exceed the primary threshold. Values that do not meet the primary threshold, but are connected to values which do meet the primary threshold are also selected [17]. These values are then compared to a secondary (lower) threshold and any point which does not meet or

exceed the secondary threshold is discarded [17]. Values which neither meet or exceed the primary threshold nor are connected to a point which meets or exceeds the primary threshold are also discarded [17]. An example of this procedure is depicted in Figure 2.5, where the red curve is discarded because it has no point that meets the primary threshold and the entirety of the green curve is selected because it contains a maxima above the primary threshold and other connected values which exceed the secondary threshold. Figure 2.6 provides an example of Canny edge detection applied to a grayscale image. For comparison an example of Sobel edge detection, which does not implement non-maximum suppression and hence produces a result with false-positive edges, is also included in Figure 2.6.

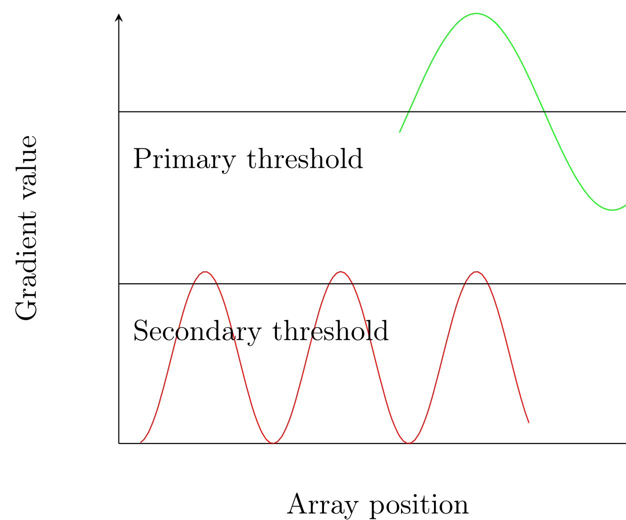

Figure 2.5.: Canny non-maximum suppression using double threshold

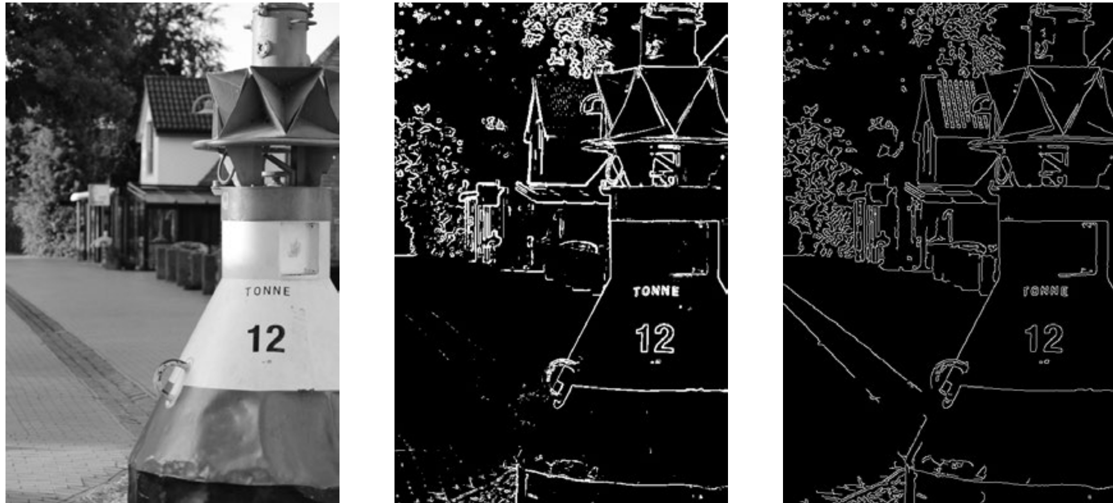

Figure 2.6.: Canny edge detection example: unprocessed grayscale image (left), Sobel edge detection without double threshold (centre) and Canny edge detection (right). Images courtesy of Professor Dr.-Ing. Marc Hensel.

2.4.2. Binary Thresholding

Binary thresholding is an image processing technique which is used to perform feature extraction for the purpose of segmenting images into pixels that meet a minimum threshold value and those that do not [5]. In essence it is a very coarse form of quantisation, which is performed as per Equation 2.20 [5], where t is the thresholding function, g is the grayscale pixel upon which it is operating and τ is the threshold value [5]. For pixels where the threshold is met, a logical true is produced (white) and for pixels where the threshold is not met, a logical false is produced (black).

$$t(g) = \begin{cases} \text{white} & \text{if } I \geqslant \tau \\ \text{black} & \text{if } I < \tau \end{cases} \tag{2.20}$$

2.5. Networking and Communication Protocols

In the field of computer science, a network is the concept of connecting two or more computers or devices via a link, either wired or wireless, which enables data to be transmitted and received (communication) [36]. Networks use the Open Systems Interconnect (OSI)

model as the foundation for their architecture. The OSI model was developed in 1984 by the International Organization for Standardization (ISO) as a framework through which inter-device compatibility could be ensured on both a hardware and software level [3]. The OSI model consists of 7 layers, which consecutively build upon each other to establish standards for a holistic communications system [3].

Layer 1, the physical layer, creates an electrical and mechanical connection, which provides the means for a raw stream of symbols (bits) to be transmitted and received [3].

Layer 2, the data link layer, provides mechanisms for error detection and defines telegrams/frames within the data stream [3]. This layer is often considered to contain two sub-layers, Medium Access Control and Logic Link Control. The former defines how the physical medium (layer 1) should be accessed, alongside implementing collision detection and/or avoidance, error detection, physical device addressing and multiplexing. The latter implements multiplexing of upper-layer communication protocols, which allow them to share the physical medium (layer 1) [3].

Layer 3, the network layer, takes the role of a network controller, which regulates message delivery in a multi-device network [3]. It combines messages and/or message segments into packets and adds routing information to ensure packets are communicated between the correct sender and recipient [3].

Layer 4, the transport layer, implements safeguards for message integrity between source and destination [3]. Common implementation methods include data segmentation, telegram acknowledgement and multiplexing between sources and applications [3]. In this layer, large pieces of data are split so that they can be delivered in separate telegrams. Implementations often include mechanisms to ensure that every piece of a message (telegram) has been successfully delivered and reconstructed at the receiver [3].

Layer 5, the session layer, provides users the control functionality required to establish, manage and terminate connections [3].

Layer 6, the presentation layer, adjusts message formatting to ensure it matches the application's requirements [3]. This process is somewhat akin to translation. The presentation layer includes the implementation of compression and encryption [3].

Layer 7, the application layer, communicates with the system and user applications to determine if supporting information or resources are required in order to service user requests [3].

The term network or networking, when used in relation to computers, commonly refers to communication via a cabled Ethernet connection or via a wireless WiFi connection. Ethernet communications are governed by a number of standards developed and published by the Institute of Electrical and Electronics Engineers (IEEE), including the 802.3 family for Ethernet and the 802.11 family (which draws from the 802.3 family) for WiFi [10]. Both Ethernet and WiFi are classified as layer 1 in the OSI model.

2.5.1. Protocols and Sockets

When connected via a network, devices can communicate using a multitude of different protocols. The most common is the Internet Protocol (IP), which is classified as layer 3 in the OSI model. The IP protocol provides mechanisms for device and message addressing as well as message routing, that is, it provides mechanisms that support the delivery of packets between devices [10]. Using the IP protocol, each device is assigned an address, which allows it to be located and identified on a network [10]. Not included in the IP protocol are mechanisms to ensure packets have arrived at the recipient, that packets are in the correct order when they arrive at the recipient and to specify which application running on the recipient should receive the packet [10]. These mechanisms are implemented in the transport layer (layer 4) via protocols such as the User Datagram Protocol (UDP) and the Transmission Control Protocol (TCP) [10]. Both UDP and TCP extend the features implemented by IP, however they do so in very different ways.

TCP is built around reliability. A TCP connection is established using a procedure commonly known as a handshake, after which data exchange can occur between two TCP ports [10]. The TCP protocol also includes a checksum to ensure the integrity of transmitted data. Additionally, it ensures that packets are correctly delivered to the receiver in the correct order, by sending an acknowledgement from the receiver to the sender for every packet [10]. Whilst this process ensures reliability, it also adds a great deal of overhead to the data transmission process. Additionally, if a packet is lost during transmission, the sender must re-transmit the lost packet while the receiver waits, which, in the case of streaming data, can cause significant delays [10].

Unlike TCP, UDP is significantly more simple. Whilst it retains a checksum to ensure data integrity, it does away with the features which ensure reliability, including the connection handshake and packet acknowledgement [10]. Whilst this removes the guarantee

of packets being delivered and arriving in the correct order, it provides a significant increase in transmission speed, especially in case of streaming data [10].

2.5.2. Message Queuing Telemetry Transport

The Message Queuing Telemetry Transport protocol (MQTT) is a lightweight TCP-based communications protocol, designed for passing messages between Internet of Things (IoT) devices [19]. It is based upon the concept of publishers, subscribers and brokers. A publisher is any device which produces information that is to be shared with other devices, a subscriber is any device which consumes information which has been published and a broker is the entity which facilitates this exchange of information [19].

MQTT is designed such that a single broker can be used for many different message types. To ensure messages are sent to the correct subscriber(s), they are categorised into topics [19]. In order to publish information to a topic, the publisher sends a message to the broker which includes its unique identifier (most commonly a unique device name), the topic to which the message should be published and the message itself [19]. Similarly, in order to receive messages, a subscriber must register itself with the broker using its unique identifier and the topic to which it wishes to subscribe [19]. For every message that the broker receives, it checks the topic and then forwards it to every subscriber which has subscribed to the same topic [19]. Due to the implementation of a broker, publishers and subscribers can dynamically connect and disconnect from each other, unlike direct TCP communication [19], making MQTT a very flexible communications protocol.

2.6. Hardware

This section introduces the hardware that will be used throughout this thesis. Key information that is required for the understanding of consecutive chapters is provided.

2.6.1. Truggy Remote Control Car

The Truggy remote control (RC) car is widely available and affordable, making it accessible for educational purposes, such as this thesis. The exact model made available for

this thesis is a Reely Dart Brushed Electric Truggy[2]. The Truggy is designed in such a manner that it can be easily modified to mount extra hardware. Professor Dr.-Ing. Marc Hensel provided the Truggy which is used for the system development associated with this thesis, which he has modified by removing the outer plastic casing and substituted an acrylic plastic board onto which hardware can be mounted. This is illustrated in Figure 2.7.

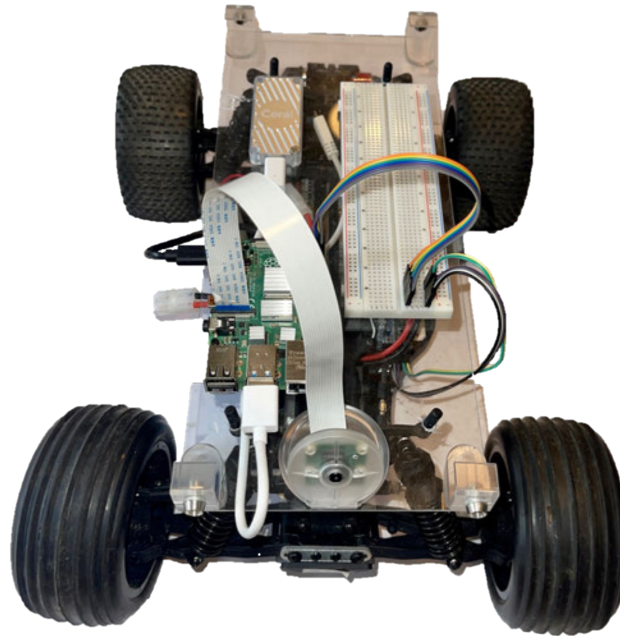

Figure 2.7.: Truggy remote control car

The Truggy RC car has an onboard battery pack that supplies a nominal voltage of 7.2 VDC [7], which is used in this thesis to power all onboard systems. The overall dimensions of the Truggy RC car are 420mm x 290mm x 160mm [7].

2.6.2. Raspberry Pi Single Board Computer

Designed specifically for the purpose of affordability, the Raspberry Pi Single Board Computer (SBC) provides a very accessible and capable platform upon which to implement the control system. For this thesis, a Raspberry Pi 4 Model B[3] with 4 Gigabytes of

[2]https://www.conrad.com/p/reely-dart-brushed-110-rc-model-car-electric-truggy-rwd-100-rtr-24-ghz-1405819
[3]https://www.raspberrypi.com

RAM and a Broadcom BCM2711 quad-core Cortex-A72 ARM CPU running at 1.5GHz is used [34]. Throughout the system design and development, the available hardware resources of the Raspberry Pi must be taken into consideration.

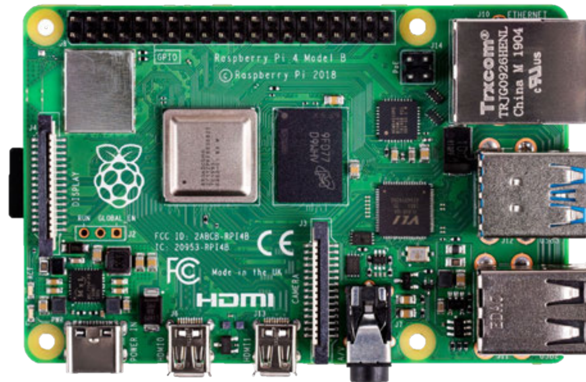

Figure 2.8.: Raspberry Pi 4 Model B SBC[4]

2.6.3. Raspberry Pi Camera

Following the Raspberry Pi ideology, the Pi Camera Module 2[5] is a low-cost camera specifically designed for connection and use with Raspberry Pi SBCs. It uses a Sony IMX219 image sensor with a resolution of 3280 $\times$ 2464 pixels, which is capable of producing still images at a resolution of 8 Megapixels or a video stream at 1080p 30FPS, 720p 60FPS or 480p 90FPS [35].

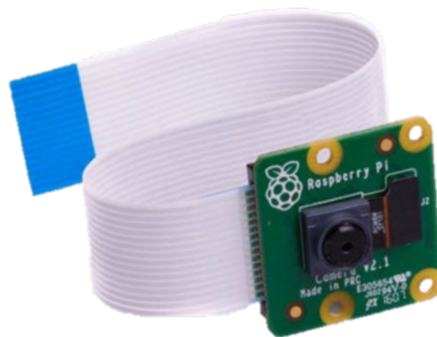

Figure 2.9.: Pi Camera Module (model 2.1)[6]

[4]Image source: https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-product-brief.pdf
[5]https://www.raspberrypi.com/products/camera-module-v2/
[6]Image source: https://www.raspberrypi.com/products/camera-module-v2/

2.6.4. PCA9685 Pulse Width Modulation Controller

The Truggy RC car comes equipped with a drive motor and steering servo which, under normal circumstances, connect directly to a wireless control module receiver. For the purposes of this thesis, the wireless control module is replaced with the Raspberry Pi SBC upon which the control system is implemented. Motors and servos are controlled using Pulse Width Modulation (PWM), which is in essence a constant square wave with a controllable width. When connected to a motor, the wave width is used to control the speed and direction in which it spins. When connected to a servo, the wave width is used to control the angle to which the servo is rotated. As PWM signals are continuous, generating them directly from the Raspberry Pi SBC would use all of the CPUs available processing time, meaning that no other operations or calculations could be performed while a PWM signal is being generated.

When the generation of PWM signals is required in parallel (simultaneously) to other operations or calculations, a PWM controller can be used. The purpose of a PWM controller, such as the PCA9685 PWM controller, pictured in Figure 2.10, is to produce a constant PWM signal based upon a control value that can be updated as frequently or as infrequently as the application requires. This relieves the CPU of the Raspberry Pi from the task of constantly generating the PWM signal, as it only needs to send a single value to the PWM controller if and when a change of PWM signal is required.

The PCA9685 PWM controller accepts control values via the I^2C protocol, a serial communications bus protocol [27].

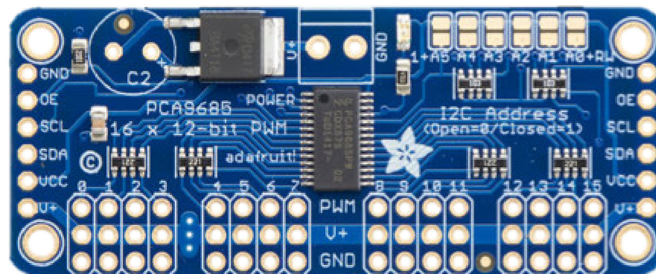

Figure 2.10.: PCA9685 pulse width modulation controller[7]

[7] Image source: https://learn.adafruit.com/assets/50181

3. Requirements Analysis

This chapter determines the requirements for the development of the autonomous control system. For this purpose, system context, stakeholders and use cases will be outlined, from which concrete system requirements will be determined.

3.1. System Context

The system context is used to outline environmental factors that can impact system operation and hence the development of the system. For clarity, a system boundary is used to define the functionality that will be developed.

3.1.1. Experimental Environment

For the purposes of this thesis, the experimental environment consists of a scale model track in an inside space, e.g., a university lecture room. The track is defined as a flat black road-surface with white line markings.

Lighting conditions of the experimental environment are kept as stable as possible by ensuring all available lighting fixtures are switched on during experimentation.

Background factors, such as stacked chairs and tables which may be detected by sensors, are to be expected. The control system should not be designed to take these factors into account and should in fact ignore them to the greatest extent possible.

3.1.2. System Boundary

As introduced in Chapter 1, this thesis aims to implement an autonomous control system to navigate an RC car around a scale track as defined in Section 3.1.1. In addition to the environmental factors outlined in Section 3.1.1, there are also a number of technical factors which directly affect the functionality of the control system. These technical and environmental factors are considered together in Figure 3.1 which provides an overview of the system context for the purpose of the development of the control system.

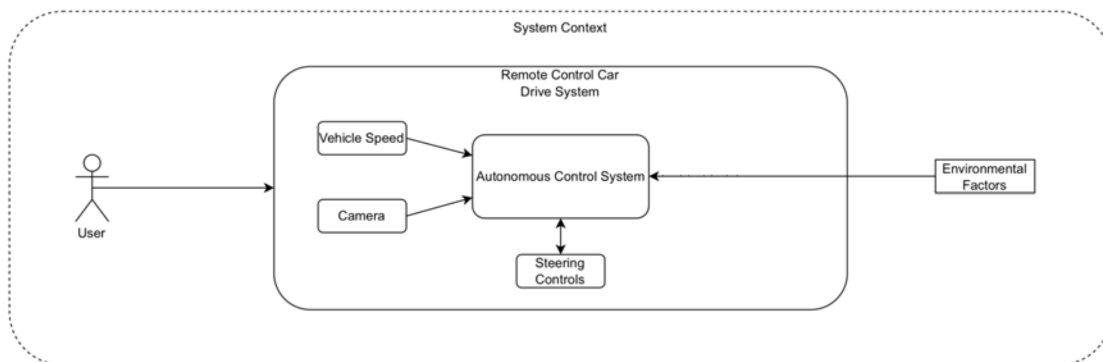

Figure 3.1.: System context

3.2. Stakeholders

Any individual or group who is deemed to have an interest in this project is considered to be a stakeholder. Their interest may be direct and actively involved during the course of the project or indirect and purely outcome based. Clear and efficient communication with stakeholders is a crucial part of successful project management and to this end a comprehensive list of stakeholders and interests has been compiled below and summarised in Table 3.1.

User: The end user expects a well-documented and easy to use system whose functionality matches the documentation.

First Examiner: Professor Dr.-Ing. Marc Hensel has an interest in expanding his knowledge base in the ML domain, specifically within the RL subdomain. Through a well documented record of knowledge (this thesis), he expects to be able to build

competencies and knowledge to help facilitate future industry partnerships. Additionally he has an interest in receiving a well-documented and reusable code-base.

Second Examiner: Professor Dr. Heike Neumann has an interest in widening her ML knowledge to include the RL subdomain.

HAW Hamburg: High quality scientific theses are published by HAW Hamburg in their online repository (REPOSIT), making the University itself a stakeholder in the overall outcome of this thesis.

Thesis Author: The author of this thesis has a very strong interest in building his technical competencies in the ML domain, especially by introducing the subdomain of RL. He also has a very strong interest in further developing his critical thinking skills and understanding of scientific method, through the production of a high quality thesis.

Successive Authors: Successive authors that use parts of this thesis for their own work have an interest in the quality of the project and its accompanying documentation.

Society: Greater society has an interest in the development and understanding of autonomous vehicle technology that is well researched, safe and reliable.

Table 3.1.: Project stakeholders

Stakeholder	**Area of Interest**
User	Overall system functionality and accompanying documentation.
First Examiner	Expanding scientifically acquired knowledge, overall system functionality, documentation and code-base.
Second Examiner	Expanding scientifically acquired knowledge.
HAW Hamburg	High quality scientific work for publication.
Thesis Author	Acquisition of technical competencies and production of high quality scientific work.
Successive Authors	High quality scientific work, accompanying documentation and code-base.
Society	Scientific research and development of autonomous vehicle technology.

3.3. Use Cases

Using the system boundary established in Section 3.1.2, this section identifies and describes the use cases relevant to the development of the autonomous control system. Figure 3.2 provides an overview of the system use cases, which are concretely defined throughout this section.

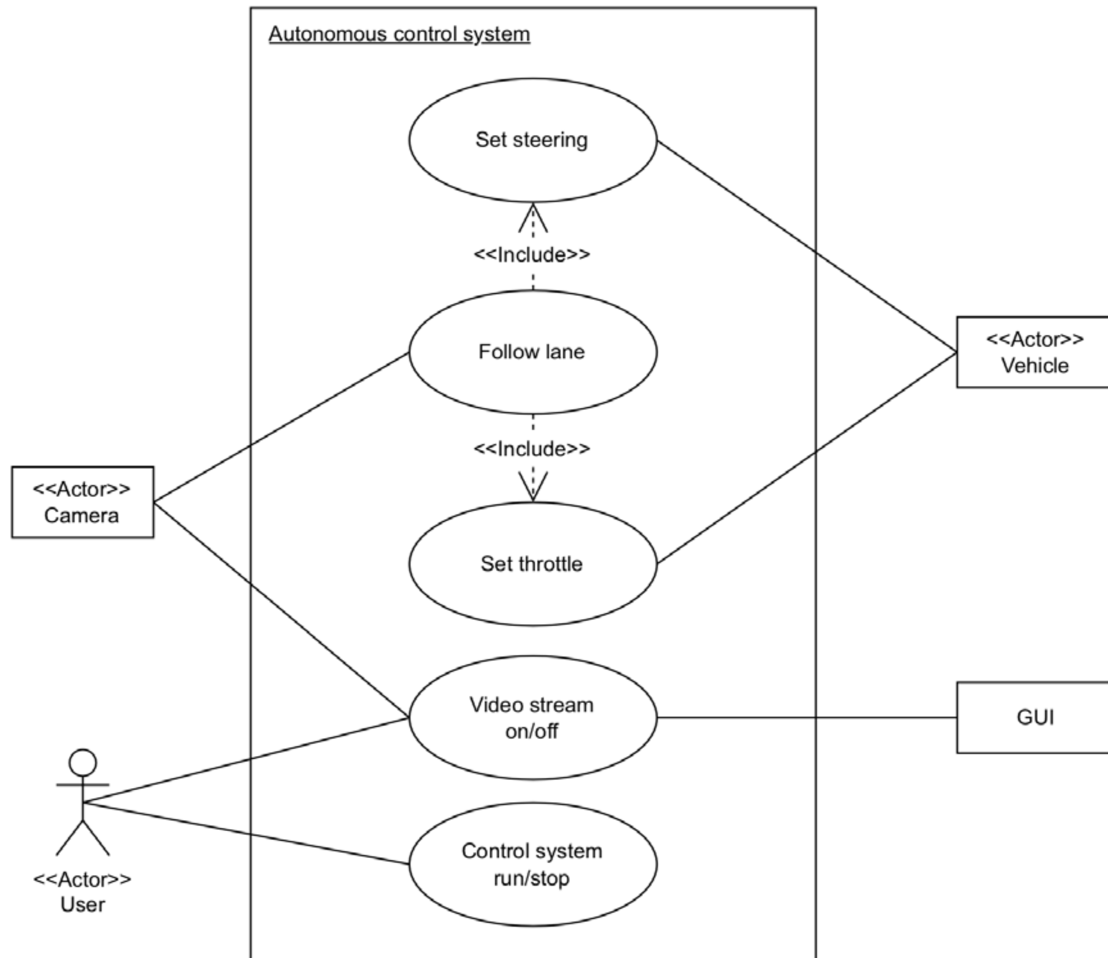

Figure 3.2.: Use case diagram

3.3.1. Follow Lane

The system ingests and processes camera sensor data in order to determine the steering angle required to stay on the track. The system can be set to run at any time by the user and will begin to produce control values once data has been ingested and processed.

Table 3.2.: Use case: follow lane

Name	Follow lane
Description	Camera sensor data is ingested and processed by the system to determine required steering controls.
Outcome	Correct steering angle determined in order to keep vehicle on track.
Dependencies	System is in a run state and sensor data is available.
Trigger	Automatically occurs once per execution cycle.
Behaviour	Sensor data is processed and steering control values are produced.

3.3.2. Set Steering

Processed data is used to determine the required steering angle, which is communicated to control hardware. Control begins after sensor data has been processed.

Table 3.3.: Use case: set steering

Name	Set steering
Description	Steering control values are sent to control hardware.
Outcome	Vehicle steering position is updated.
Dependencies	Control values produced and active connection to control hardware.
Trigger	Automatically occurs once per execution cycle.
Behaviour	Control values are sent to hardware and hardware responds accordingly.

3.3.3. Set Throttle

The user should be able to start and stop the movement of the vehicle.

Table 3.4.: Use case: set throttle

Name	Set throttle
Description	Start or stop vehicle movement via throttle control values based upon user commands.
Outcome	The system responds to user start or stop commands by setting the throttle control value accordingly.
Dependencies	User is connected to the control system and the system is in a run state.
Trigger	Asynchronous execution by user.
Behaviour	User triggers a start or stop command.

3.3.4. Video Stream on/off

The user can start or stop a video stream. This can occur at any time.

Table 3.5.: Use case: video stream on/off

Name	Video stream on/off
Description	The user can start or stop the video stream.
Outcome	System responds to the command sent by the user.
Dependencies	User is connected to the control system.
Trigger	Asynchronous execution by user.
Behaviour	User sends command and system applies command.

3.3.5. Control System run/stop

The user can start or stop the control system. This can occur at any time.

Table 3.6.: Use case: control system run/stop

Name	Control system run/stop
Description	The user can start or stop the control system.
Outcome	System responds to the command sent by the user.
Dependencies	User is connected to the control system.
Trigger	Asynchronous execution by user.
Behaviour	User sends command and system applies command.

3.4. Restrictions

There are a number of restrictions placed upon the development of the system due to the nature of the project requirements and the budgetary restrictions set in-place by HAW Hamburg.

The control system is to be implemented such that it can run on a Raspberry Pi 4 Model B Single Board Computer, using a Pi Camera as sensory input. The vehicle used for development is to be a Reely Dart Brushed Electric Truggy remote control car [7].

3.5. Requirements

In this section the outcomes of the system context, stakeholder analysis and use cases are used to establish concrete requirements for the system development.

Requirements are divided into two categories: functional requirements and non-functional requirements. Functional requirements are used to define requirements that directly affect system functionality and therefore the technical implementation. Non-functional requirements place requirements upon factors that affect project execution and design, but do not directly specify technical system functionalities. They influence project design and architecture decisions which may or may not have a flow-on effect upon technical implementation.

3.5.1. Functional Requirements

F1: The user should be able to start or stop the system at any time, as long as the system is powered on and the control system is connected.

F2: The system should receive and process camera sensor data in such a way that steering control values are produced.

F3: The system should determine steering control values based upon the road-markings present in the field of view of the camera.

F4: The system should react to a stop command in no more than the time or distance that a human driver would require.

F5: The user should receive feedback about the system's status.

3.5.2. Non-Functional Requirements

NF1: Onboard control system processing should occur using a Deep Neural Network.

NF2: The Raspberry Pi must communicate with the PWM controller via I^2C bus[1].

NF3: Road markings should be white on a black road-surface. The total road width should be equal to two vehicle widths.

NF4: The system should be implemented such that it can run on a Raspberry Pi 4 Model B Single Board Computer.

NF5: The Neural Network should be trained using reinforcement learning.

[1] The provided PCA9685 PWM controller can only communicate using I^2C bus.

4. Concepts

This chapter builds upon the analysis in Chapter 3 to identify and discuss desired project outcomes and potential implementation strategies to facilitate these. It then concludes by presenting a chosen implementation strategy accompanied by justified reasoning therefore.

4.1. Desired Outcomes

The ultimate aim of this thesis is to develop and train an agent in the form of a DNN model using Deep Reinforcement Learning that is capable of navigating an RC car around a track, the exact requirements of which are detailed in Chapter 3. There are a number of different approaches that could be implemented in order to achieve this. These will be identified and discussed in Section 4.4.

Adjacent to the desired practical outcomes of this thesis, there are a number of additional desired outcomes. From an educational perspective, as determined in the Stakeholder Analysis of Section 3.2, it is desired that the author builds his knowledge and technical competencies in ML, specifically within the subdomains of DNNs and RL. A key part of this process is documentation, both in the form of this thesis and the documentation accompanying the code-base. This allows the transfer of acquired knowledge to both examiners and to successive authors who may choose to build upon this thesis.

4.2. Operational Parameters

The control system is developed solely for the purpose of control of an RC car, for safety reasons the system is intended for use only under controlled conditions. The following section details the operational parameters of the system during various operational scenarios.

4.2.1. Normal Operation

The system is considered to be in normal operation when all systems and sub-systems are functioning as intended (including but not limited to hardware and software systems) and the operator has the ability to stop and start vehicle operation at any given time.

When the system is in normal operation it should be able to navigate around a scale model track autonomously.

4.2.2. System Failure Conditions

The system is considered to be in a failure state if any system or sub-system is not functioning as intended or if the operator loses the ability to stop or start vehicle operation. A failure state may occur as a result of a hardware (physical) or software malfunction.

Should the onboard battery deplete, the system will shutdown. This is because both the control and drive systems are powered by the same onboard battery.

In the event of disconnection or malfunction of the camera, the system should come to a halt. Without accurate and relevant camera data, the system will not be able to correctly determine control values for the steering system.

4.2.3. Real Time Operation

Due to the real time nature of controlling a moving vehicle, requirements considering the processing and control pipeline need to be carefully considered. Figure 4.1 provides an overview of the end-to-end processing and control pipeline, from the camera sensor to the final output from the servo controller.

In order to successfully navigate around the track, the system must ingest data, process it and produce a control output. Whilst operational, the vehicle is constantly moving, meaning that input data is only relevant to the control system for a short period of time. In order to ensure that the system is processing relevant input data, the system processing pipeline must be able to process data at a high enough rate.

Paramount to safety in terms of real time operation of a vehicle is the braking system. The German motoring association ADAC lists the average human reaction time for braking

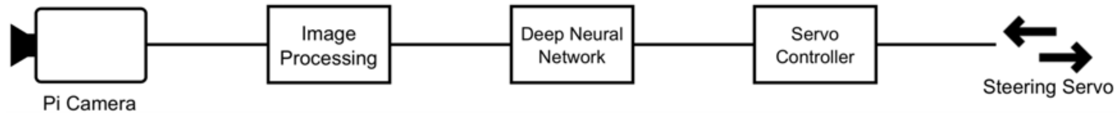

Figure 4.1.: Processing and control pipeline

as 0.8 to 1.2 seconds [1]. Additionally, they calculate the distance required to come to a full stop using the Faust formula, shown in Equation 4.1 [1].

Using Equation 4.2, where s represents speed, d represents distance and t represents time, the "worst case" distance travelled whilst reacting can be calculated using a vehicles initial (pre-braking) speed. By combining the results of both Equation 4.1 and Equation 4.2, the total distance travelled from the beginning of the reaction time until the vehicle has come to a complete stop can be calculated.

$$d_{stopping}(\text{m}) = \frac{\text{speed (km/h)}}{10} \cdot \frac{\text{speed (km/h)}}{10} \tag{4.1}$$

$$d = \frac{s}{t} \tag{4.2}$$

4.3. System Behaviour

To further clarify the system use cases established in Section 3.3, Figure 4.2 provides an overview of the expected system behaviour in the form of a state machine diagram. This diagram details the various operational states of the system as well as the conditions which need to be met in order to transition between them.

The system initially loads into a ready state, from which it can transition to a run state upon a user command or to a failure state if a sensor fails. From the run state, it can transition back into the ready state upon a user command or to a failure state if a sensor fails. Once in a failure state, the user must manually reset the system in order to run the system again (enter the ready state).

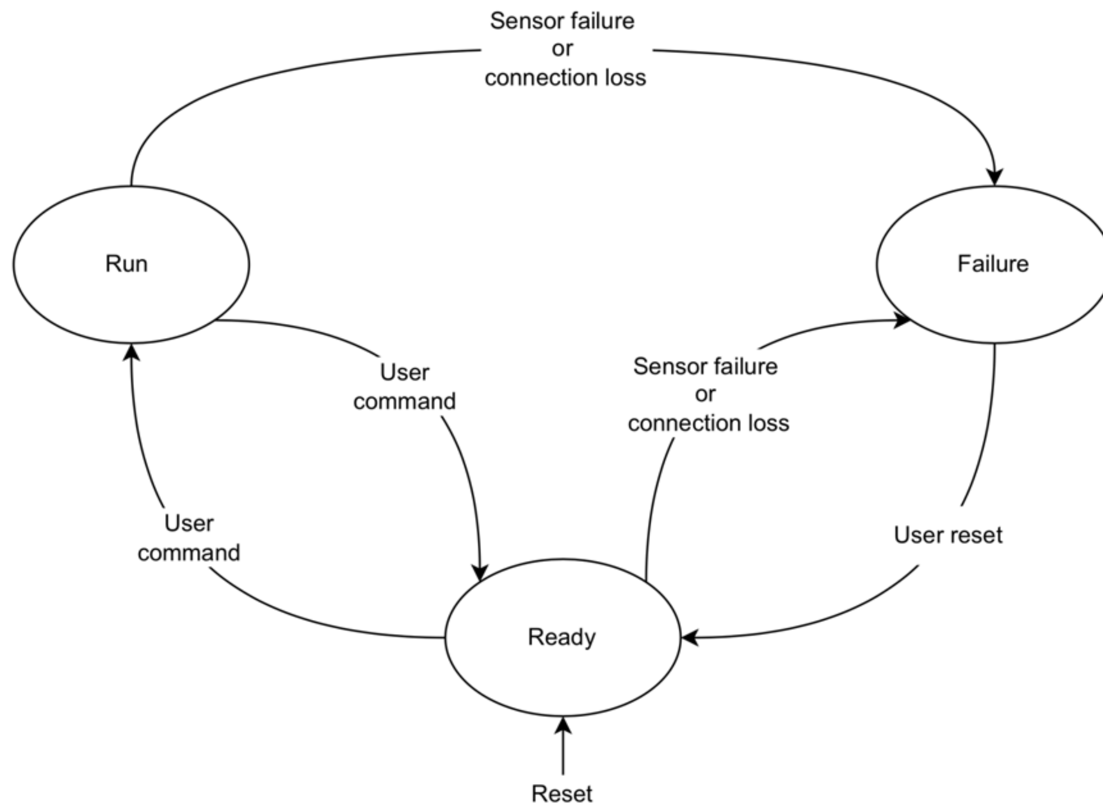

Figure 4.2.: State machine diagram

4.4. Implementation Strategies

In order to train an RL agent (DNN model), a very large number of repetitive episodes are required before it becomes useful [28], especially if training purely with visual input stimuli [2]. Performing such a large number of training episodes by hand would be both highly repetitive and impractical, therefore some degree of automation is required. This section discusses implementation strategies that could be used to fulfill the requirements of this thesis and is followed by the selection of a specific implementation strategy in Section 4.6.

There are three accessible possibilities for the automation of the episodic training that are implementable within restrictions placed upon this thesis: virtual simulation, physical simulation and a virtual-physical simulation combination. The Donkey Car open source

project[1] has an OpenAI Gym [6] wrapper called Donkey Gym[2] which is implemented in the Python programming language. Donkey Gym facilitates the training of an RL agent by providing a 3D simulation environment in which an agent can interact using an Application Programming Interface (API). Alternatively, Professor Dr.-Ing. Marc Hensel has built and made available for the purposes of this thesis, a physical simulator upon which an RC car can be mounted for training. Both of these training methods have advantages and disadvantages. It is therefore of great importance that these are carefully assessed before a final implementation strategy is chosen.

The Donkey Gym 3D simulator allows for real time training with high-resolution time-steps that are very close to the continuous-time time-steps that the system receives once trained and deployed. However, this comes at the expense of training using image frames captured from a 3D simulation, not from the Pi Camera Module that the system receives once trained and deployed. Additionally, whilst the Donkey Gym offers multiple different training environments, none of them exactly match the track requirements of a black road-surface with white line markings and most feature a range of background objects and environmental features which may distract the DNN. This could result in longer training times (more episodes required) or in a DNN which places undesired consideration upon these factors when processing image frames to determine control values.

The physical simulator, on the other hand, trains the DNN using image frames captured from the same Pi Camera Module that is used for deployment. This greatly minimises the discrepancies between the data that the system receives during training and the data it receives once trained and deployed. Additionally, the physical simulator allows training to occur in the experimental environment, meaning the same road-surface can be used for both training and deployment/testing. Exactly matching the road-surface for both training and deployment/testing maximises the achievable accuracy for the DNN. Unfortunately, the physical simulator has some significant intrinsic disadvantages, the most prevalent of which is the speed of training. For every time-step during the training process, the vehicle must be positioned, an image frame must be captured and the DNN must process the image frame and determine the control system values. A reward value must then be calculated based upon the simulator position and a control system value determined by the DNN. Finally, based upon the control system value determined by the DNN, the relative vehicle position must be calculated and the process begins again by re-positioning the vehicle.

[1] https://www.donkeycar.com
[2] https://github.com/tawnkramer/gym-donkeycar

Across the thousands of episodes, each containing thousands of time-steps, the physical simulator is significantly slower in comparison to the real time speed of the 3D simulator. Additionally, as is often the case applications that requires physical automation, the results are limited by the accuracy of the hardware. To achieve the same high-resolution time-steps as the 3D simulator, the physical simulator must use high-accuracy encoders or stepper motors and be carefully calibrated before beginning the training process. Finally, thousands of training episodes take many hours to complete. As such, it is assumed that the simulator will run unsupervised for the majority of the training duration. If during this time a failure occurs, such as a belt slipping or a connector coming lose it is possible that the entire training time may be wasted. Although this could be considered to be unlikely, it is still a risk which must be considered.

Finally, combining both training techniques in a virtual-physical combination could prove beneficial by exploiting the initial training speed of the 3D simulator and the final quality of training in the experimental environment. However, once again this implementation option is accompanied by a significant disadvantage. By training in both the 3D simulator and the experimental environment, the amount of implementation is significantly increased. Additionally, rewards for the RL process are calculated based upon the position of the vehicle relative to the track. In order to implement reward calculation in the experimental environment, a significant amount of additional implementation would be required, including a high level of image processing.

4.5. Foreseen Difficulties

Taking into consideration the operational parameters and the characteristics of the 3D training environment and physical simulator, several areas have been identified that pose particular difficulties to the overall project outcomes. This section will identify these foreseen difficulties.

As previously outlined, using a 3D simulation environment enables fast and efficient training, which results in more training episodes than could be achieved in the same time using the physical simulator. It however also results in an agent trained for the 3D simulation environment, not the experimental environment. Whether or not the agent's performance in the experimental environment is equal or even comparable to the 3D simulation environment cannot be determined until training has been completed.

This means that there is significant risk of wasting time training agents in a simulation environment that are not functional in the experimental environment.

A potential strategy to minimise this risk is to perform some level of image processing to the frames captured in both the simulation and experimental environment. By performing this pre-processing step, image frames could be normalised, allowing for more uniform image compositions than could otherwise be achieved. This reduction of distracting input stimuli to the DNN reduces the negative effects of variable environmental factors upon the DNN, however also results in an overall reduction of valuable input stimuli.

Another difficulty of using a simulation environment for training is the transition from an exact training environment to a non-exact experimental setup. Initial testing of the RC car shows that the steering mechanism does not have a high degree of accuracy. These discrepancies between the simulator and experimental equipment directly affect the performance of a system trained in the 3D simulator when implemented in the experimental setup.

Finally, the complexity of the DNN directly affects the performance of the system in two distinct ways. Firstly, complex models with larger numbers of trainable parameters are often more accurate in comparison to simpler models, as they can learn more abstract interpretations of the input data. However, they come with penalties in inference speed and the amount of training required before they become useful. Performance requirements must be carefully balanced when considering the DNN architecture, in respect to the resources available on the Raspberry Pi.

4.6. Implementation Strategy Selection

Using the formal requirements set out in Section 3.5 and taking into consideration the operational parameters and available implementation strategies outlined throughout this chapter, an implementation strategy is now selected and the reasons justifying this choice presented.

4.6.1. Training Environment

The 3D Donkey Gym simulator is chosen as the training environment due to the speed with which training episodes can be executed and for its ability to provide high-resolution time-steps without the requirement of calibration and monitoring to ensure uniformity of actions within episodes. In addition, the disadvantages of training in the experimental environment further incentivise the use of the 3D simulator. Specifically, the amount of high-level image processing required in order to implement reward calculations in the experimental environment is beyond the scope of this thesis and outside of the amount of implementation realistically achievable within the time restrictions placed upon this thesis. Using the 3D Donkey Gym simulator restricts the implementation to the Python programming language. Due to the widespread popularity of Python for ML [25] and the availability of commonly used and well supported ML libraries such as Tensorflow[3] and Keras[4] for Python, the impact of this restriction is not considered to be negative.

4.6.2. Image Processing

As outlined in Section 4.5, the use of the 3D simulation environment requires image processing in order to minimise the difference in input stimuli between the 3D simulation environment and the experimental environment. For this purpose, Canny edge detection is selected, which processes image frames such that only object outlines in black and white are input to the DNN. This allows road-markings to be extracted from the camera sensor data, whilst the rest of the unrequired and potentially distracting background information is removed.

4.6.3. System Design

To fullfil the requirements of user control and system status feedback, a system design based on a client server model is selected. Communication between the client and server is to be entirely network based, using a WiFi connection, allowing Truggy to move freely. A local control computer with a graphical user interface (GUI) for user control is to be connected to a remote Raspberry Pi attached to the Truggy. Whilst a command line interface would be sufficient to fulfill the requirements set out in Chapter 3, using a GUI

[3]https://www.tensorflow.org
[4]https://keras.io

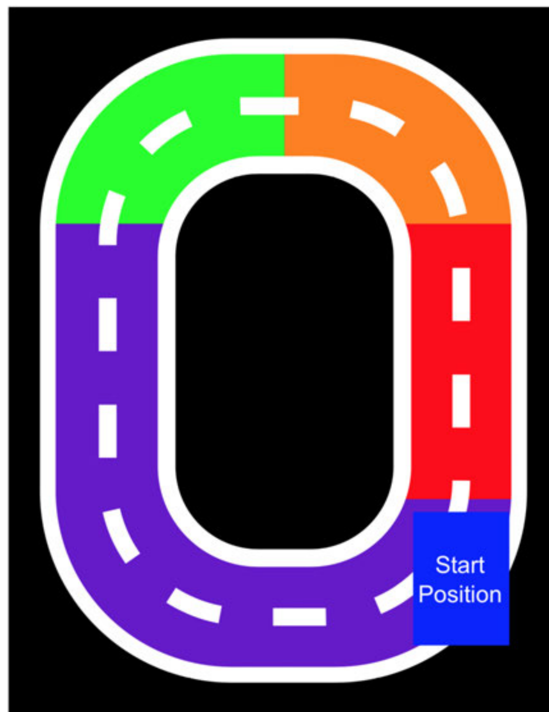

Figure 4.3.: Track zone classification diagram.

aims to provide a better user experience as key information such as the steering angle can be visually represented and is therefore more intuitive.

4.6.4. Test Setup Within the Experimental Environment

For the purpose of testing in the experimental environment, the track is divided into four zones, as per Figure 4.3. These zones are used to measure the degree of success to which navigation is performed during testing. If the vehicle leaves the track within the red zone, it indicates that the system was not able to successfully navigate a straight section of the track. If the vehicle leaves the track within the orange zone, it indicates that the system was successfully able to navigate within a straight section of the track, but not around a corner. If the vehicle leaves the track within the green zone, it indicates that the system was successfully able to navigate within a straight section of the track and around a single corner. If the vehicle leaves the track within the purple zone, it indicates that the system was successfully able to navigate within multiple occurrences of straight sections and cornered sections of the track.

5. Implementation

This chapter applies the concepts discussed in Chapter 4 to the Deep Reinforcement Learning implementation and experimental setup. It provides technical details as well as information about the scientific methods employed during the implementation process. Intermediate results which are required for decision making during the implementation process are presented, while the presentation of final results is reserved for Chapter 6.

5.1. Training

5.1.1. Simulation Environments

The Donkey Gym open source 3D simulator project [23] includes ten simulation environments, shown in Figure 5.1, each of which includes a different set of environmental factors. Unfortunately, the majority of the simulation environments use road-surfaces which are not suitable for image processing, due to a number of factors including glare caused by simulated road-surfaces, low level of contrast between road-surfaces and road-markings, ambiguous road-markings and other characteristics which cause undesired artifacts during image processing (see Figure 5.2).

For the previously outlined reasons, the selection of tracks is significantly limited, with only one out of the ten tracks providing an environment suitable for image processing. As such, the "Waveshare" track is selected for training purposes (see Figure 5.3). It features a high level of contrast between the road-surface and road-markings, making it suitable for use with image processing for feature extraction. Unfortunately, unlike other tracks, the "Waveshare" track does not have complex features, hence any agent (DNN model) trained on this track is suitable for use only in this very specific track design.

[1] Note the system processing pipeline is configured to run at a resolution of 80×80 pixel to maximise performance. This causes the pixelation visible in the figure.

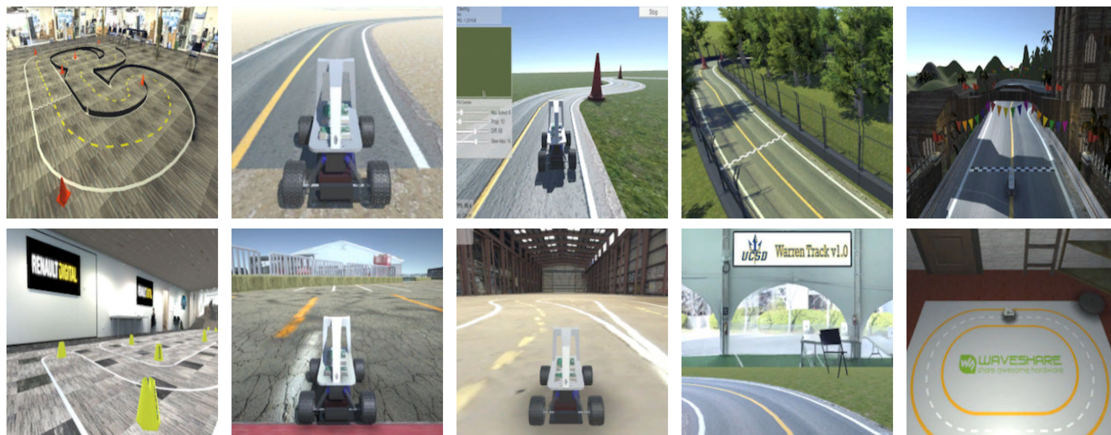

Figure 5.1.: Donkey Gym open source simulation environments [23]

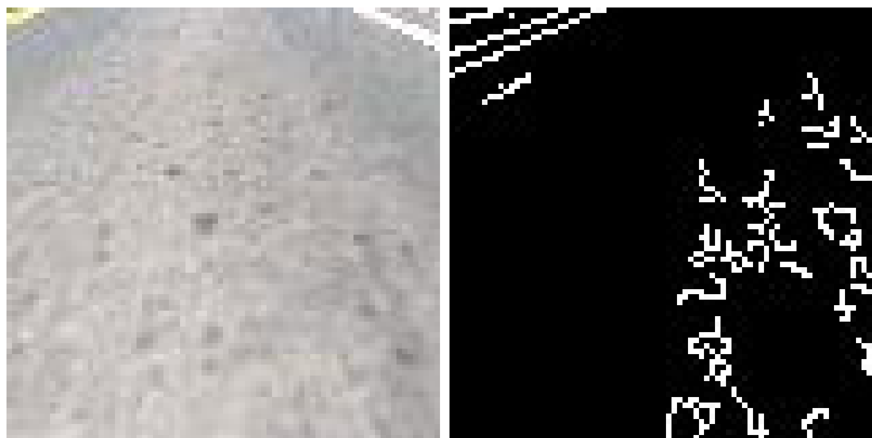

Figure 5.2.: Unsuitable track simulation environment[1] [23]

5.1.2. Reinforcement Learning Setup

The reinforcement learning process is implemented using online off-policy deep Q-learning in Python with the Tensorflow and Keras libraries. This particular skew of reinforcement learning is implemented in the form of two DNNs, a main DNN and a target DNN. The main DNN interacts directly with the simulation environment and collects experience which is stored in a memory replay deque, whilst the target DNN is used to calculate future actions and hence the maximum possible future reward.

Rewards are based upon the vehicle's lane position, labelled in the 3D simulation environment as cross track error (CTE). The closer to the centre of the right lane, the higher the reward. This is calculated using Equation 5.1, which is built-in to the 3D simulation

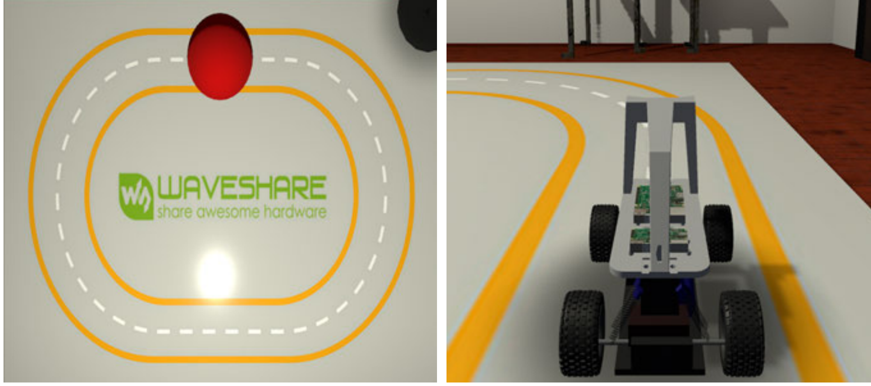

Figure 5.3.: "Waveshare" track simulation environment [23]

environment[2]. A CTE_{max} variable is used to control the maximum allowable deviation from the centre of the right lane, whereby exceeding this value terminates the episode. Experimentation with different CTE_{max} values produces no observable difference in the speed at which the agent (DNN model) learns or in the final training outcome. As such, the CTE_{max} is configured at the default value of $CTE_{max} = 5$.

$$r = 1 - \left| \frac{CTE_{current}}{CTE_{max}} \right| \tag{5.1}$$

Once a pre-determined amount of experience has been collected in the deque, it is used for training the main DNN with every action (step) that it makes. Training consists of a fit operation performed on the main DNN. Each training episode runs until the vehicle crashes into an object or until it returns a CTE value greater than CTE_{max}. The latter indicates that it has departed the track area, at which point a Boolean flag is triggered and the episode is terminated. At the end of a training episode, the weights from the main DNN are copied to the target DNN. Once the deque is full, old experience data is overwritten with new experience data, similar to the concept of a circular buffer.

An overview of the RL process in its entirety is provided in Figure 5.4.

[2]The built-in reward equation also takes speed into account, however seeing as this is set as a constant it is omitted from Equation 5.1.

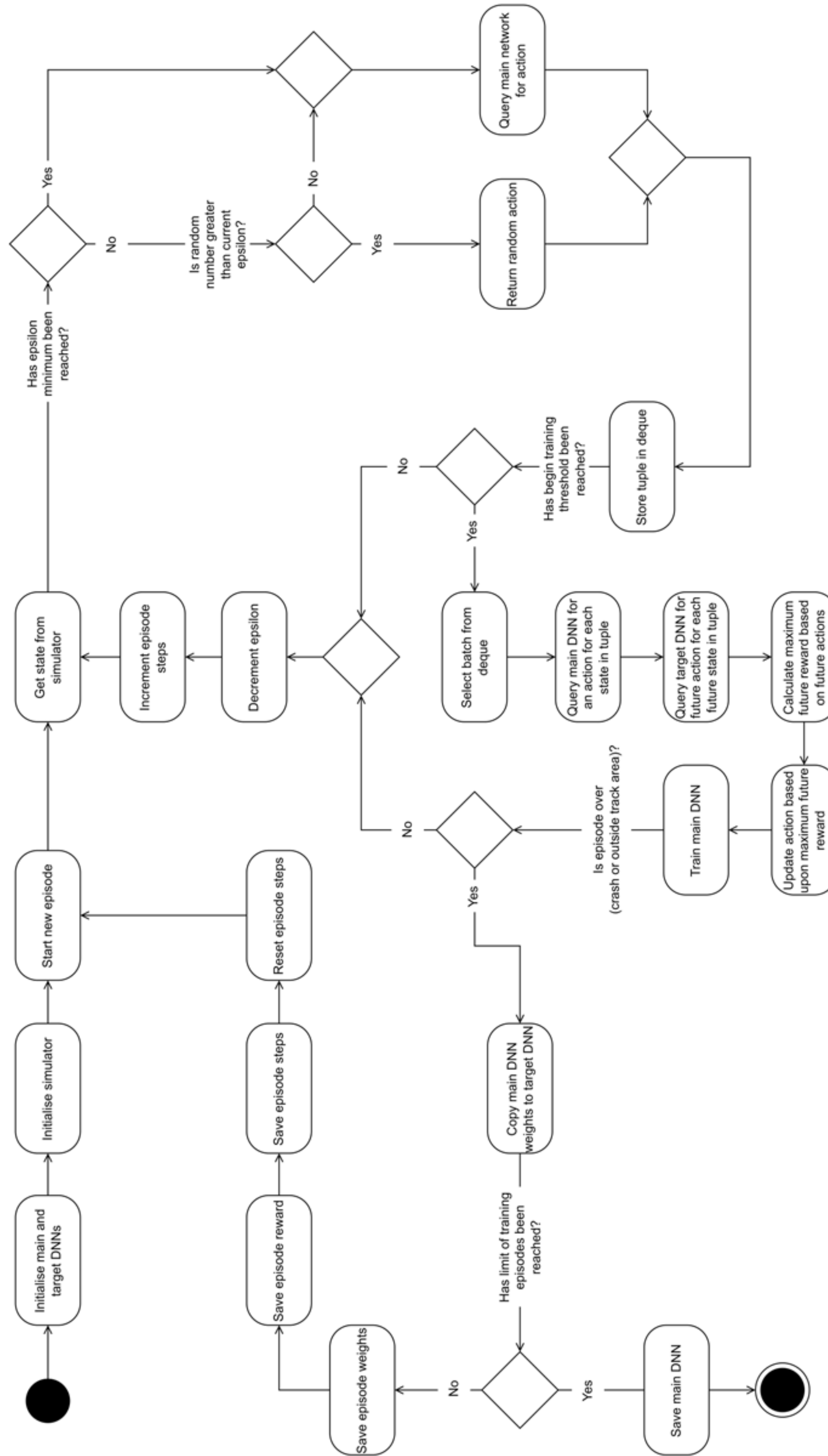

Figure 5.4.: Reinforcement Learning activity diagram.

5.1.3. Comparison of Deep Neural Network Architectures

Deep Neural Network architecture varies greatly depending upon the intended use case and desired performance characteristics. As briefly described in Section 4.5, due to the hardware limitations of this thesis, the architecture of the DNN model needs to be carefully balanced between performance, which directly relates to the type and number of layers, and the time required for the network to perform inference on the Raspberry Pi. For this reason, three DNN model architectures are compared: (i) a modified version of Nvidia's "DAVE-2" [4] DNN, (ii) a DNN which is provided as a part of the Donkey Gym open source project [23] and (iii) a custom DNN architecture developed using an iterative process as a mid-point between both of the aforementioned architectures.

The modifications made to Nvidia's "DAVE-2" DNN are based upon design recommendations from the Donkey Gym open source project [23]. The input layer is modified such that it receives an image of dimension 80×80 pixel $\times 4$. This is achieved by stacking the same grayscale image four times. The output layer is also modified from one single node to fifteen nodes, in order to coarsely quantise the number of steering angles which can be selected by the DNN. In total, the modified "DAVE-2" DNN has 7,705,027 trainable parameters, the majority of which are found in fully connected (dense) layers. Figure 5.5 provides an overview of the modified architecture.

Because the DNN provided as a part of the Donkey Gym open source project [23] is built to include these design recommendations, no modifications are required. In total it has 974,183 trainable parameters, which, like the modified "DAVE-2" DNN, are mostly found in fully connected (dense) layers. An overview of the model architecture can be observed in Figure 5.6.

The custom DNN is developed using an iterative process, whereby interest lies in finding an architecture with a total number of trainable parameters which lies between those of the modified "DAVE-2" DNN and the Donkey Gym open source DNN, in order to examine the effect of DNN parameter count upon both performance and inference time. The Donkey Gym open source DNN is used as a starting point, from which the number of layers is increased. With each increase in layers, the resulting DNN is trained for 100 episodes and the general performance of the model during and after the training process is compared to the previous iteration. This process is repeated until a noticeable decline in performance is observed, at which point the process is stopped and the last DNN model architecture before a decline in performance is selected. The result of this process

is the DNN architecture shown in Figure 5.7, with a total number of 2,786,647 trainable parameters.

Model summaries, which provide further information about each model architecture, can be found in Appendix A.

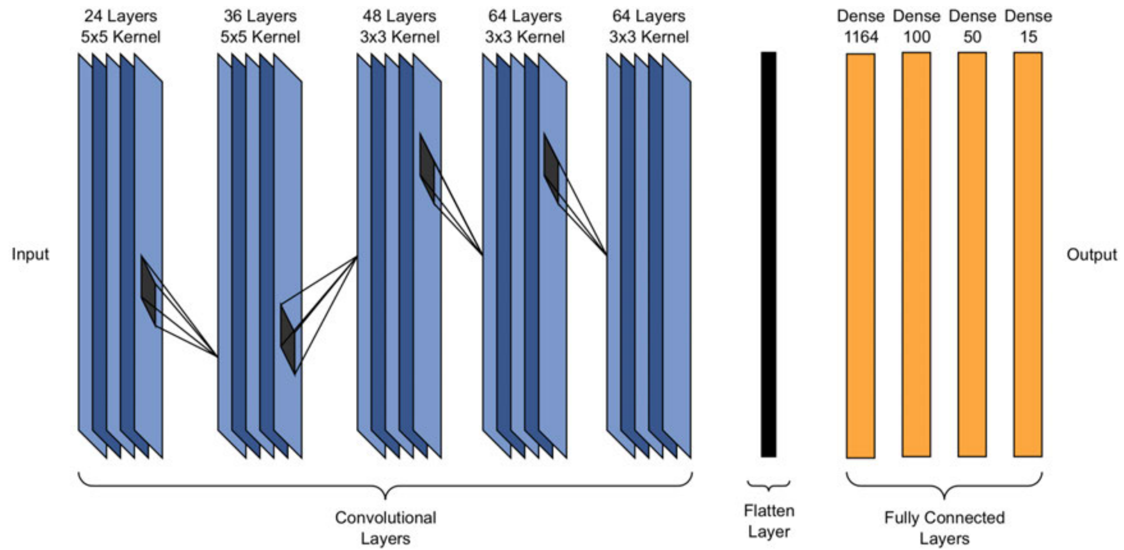

Figure 5.5.: Modified "DAVE-2" DNN architecture from Mariusz et al. [4], figure adapted from [4]

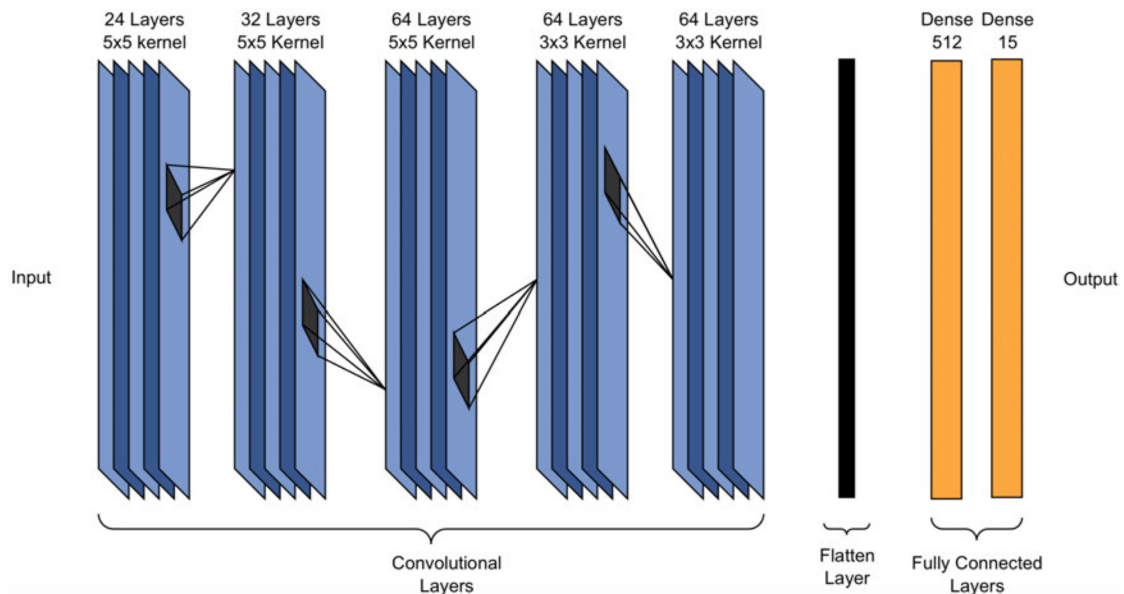

Figure 5.6.: Open source Donkey Gym DNN architecture [23], figure adapted from [4]

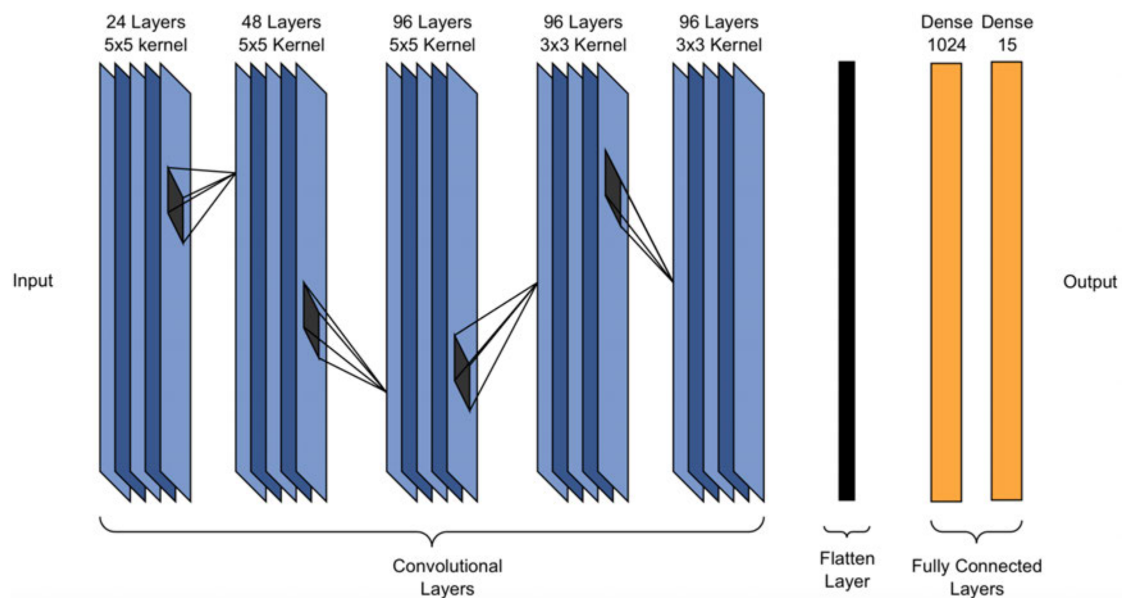

Figure 5.7.: Custom DNN architecture adapted, figure from [4]

5.1.4. Controlling Variables

As introduced in Section 4.5, the purpose of image processing is to control the environmental differences in input stimuli between the 3D simulator and the experimental environment.

An additional variable which must be controlled is the vehicle speed. An appropriate vehicle speed for training is experimentally determined using an iterative process. Due to the "Waveshare" track's dimension and shape, it is observed that using a medium to high speed (throttle value set between 30% to 40%) for training results in the agent learning an optimal policy of holding the steering control value almost constant, rather than dynamically adjusting the steering control value based upon environmental observations (camera sensor input). Due to this undesirable behaviour, it is experimentally determined that a lower vehicle speed (throttle value set to 10%) results in the vehicle spending more time on the straight sections of the track and hence the agent is forced to dynamically adjust steering values based upon the input stimuli, rather than the previously observed behaviour of holding the steering control value almost constant. Therefore the throttle value for training in the 3D simulator is set to 10%.

5.1.5. Image Processing and Training

As stated in Section 4.6.1, Canny edge detection is selected for image processing. The purpose is to extract only the road-markings from the camera sensor data, hence removing other undesired input stimuli. As outlined in Chapter 2, Canny edge detection uses two thresholds, one of which is used to determine if a pixel represents an edge and another which is used to discard pixels which are unlikely to represent a true edge. Using an iterative process, appropriate values for these two thresholds for image frames captured in the simulator are experimentally determined.

Figure 5.8 shows a comparison between an unprocessed RGB image frame and an image frame processed with Canny edge detection. It can be observed that the majority of the processed image frame is black. As the aim of image processing is to extract information about the position of the road-markings, information in the processed image frame is only contained within white pixels. By observing Figure 5.8, it can be determined that a high reduction in input stimuli occurs when processing with Canny edge detection.

The throttle control value is set to 10%, the begin training threshold is set to a minimum memory (deque) size of 100 and the minibatch (training batch) size is set to 64. The reward discount (gamma), which is used to weight immediate rewards more heavily than rewards in the future, is set to 0.99 and the epsilon decay, which controls the occurrence of random actions at the beginning of training, is set to decay over 10,000 steps.

Figure 5.8.: Unprocessed RGB (left) and Canny edge detection processed (right) image frames[3]

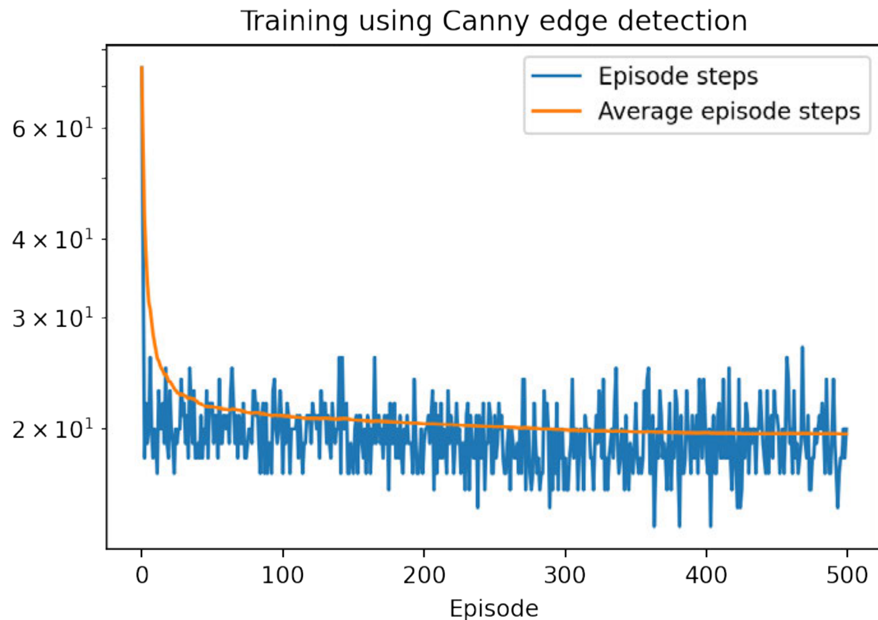

Figure 5.9.: "DAVE-2" DNN training outcomes using Canny edge detection image processing

Using these settings, the modified "DAVE-2" DNN is trained for 500 episodes using Canny edge detection for image processing, the results of which are shown in Figure 5.9. It plots the number of steps in each episode alongside a cumulative average calculation.

By observing the cumulative average episode steps, it can be clearly visually determined that the value decreases as the number of episodes increases. This behaviour is the opposite of the behaviour expected from a successful training process. This clearly indicates that the training is ineffective. No optimal policy has been learnt.

Once again observing Figure 5.8, the significant reduction in information contained within the Canny edge detection processed image frame is noted. Based upon this observation and the intermediate results, which indicate no effective behaviour was learnt during training, it is inferred that the unsuccessful training is due to a significant lack of input stimuli to the DNN, caused by the Canny edge detection image processing. It is clear that this image processing technique is not suitable for the purposes of this thesis. Another

[3] Note the system processing pipeline is configured to run at a resolution of 80 × 80 pixel to maximise performance. This causes the pixelation visible in the figure.

technique for image processing must be implemented. Due to this discovery, no further training is performed using Canny edge detection for image processing.

As outlined in Chapter 4, the desired result of image processing is to minimise the environmental difference between the 3D simulator and the experimental environment by extracting desired features and removing or ignoring undesired background features that may otherwise cause distraction. Considering the unusable results produced using Canny edge detection are deduced to result from a lack of input stimuli, one solution that follows is to choose an image processing technique that produces a level of input stimuli that contains significantly more information than Canny edge detection. For this purpose, binary thresholding for image segmentation is selected as a suitable candidate.

The training code is restructured to use binary thresholding as a means of image segmentation for the purpose of feature extraction of the road-markings. The binary thresholding threshold value is once again determined experimentally using an iterative process. Because the "Waveshare" track uses both yellow and white road-markings, the image must be processed twice, once to extract the outer lines and again to extract the centre line, the results of which are re-combined into a single processed image.

Figure 5.10 shows a comparison between an unprocessed RGB image frame and an image frame processed using binary thresholding for image segmentation. When comparing this to Figure 5.8, it can clearly be determined that a significantly higher amount of information (and hence input stimuli) is present in the image frame that has been processed using binary thresholding. Furthermore, Figure 5.11 shows a side-by-side comparison of an image frame extracted from the 3D simulator and an image frame from the experimental environment, both of which have been processed using binary thresholding. The features extracted from both environments using this image processing technique match very closely, hence confirming that this image processing technique provides a good level of normalisation between the two environments.

Again, the throttle control value is set to 10%, the begin training threshold is set to a minimum memory (deque) size of 100, the minibatch (training batch) size is set to 64, the reward discount (gamma), is set to 0.99 and the epsilon decay, is set to decay over 10,000 steps. Using these settings, each of the three network architectures is compiled and trained individually for 500 episodes in the 3D simulator using the revised image processing technique of binary thresholding for image segmentation.

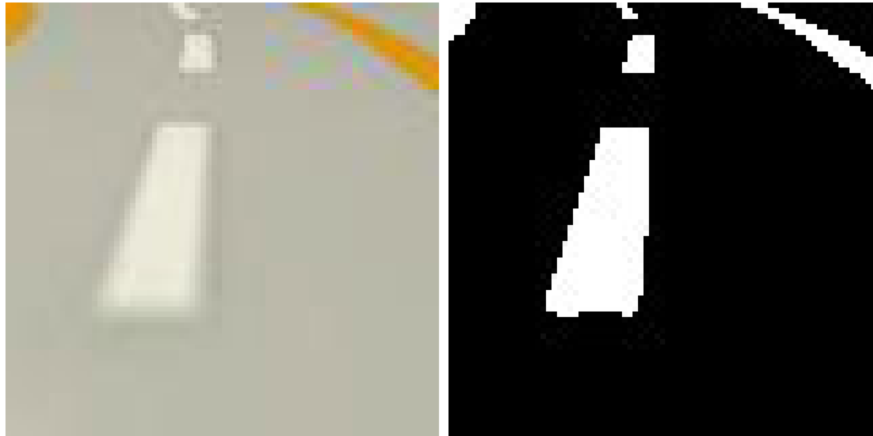

Figure 5.10.: Unprocessed RGB (left) and binary thresholding for image segmentation (right) processed image frames[4]

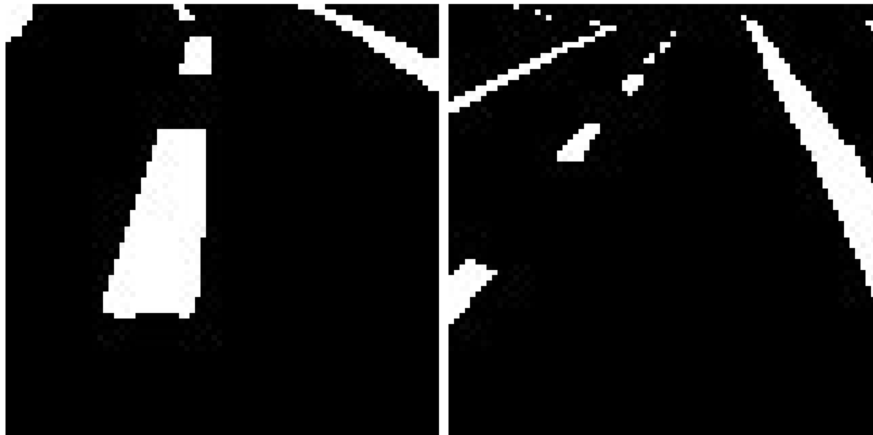

Figure 5.11.: Image segmentation using binary thresholding image processed frames captured in the 3D simulator (left) and experimental environment (right)[4]

5.2. Experimental Setup

5.2.1. Graphical User Interface

For the purpose of implementing the required user control, a Graphical User Interface (GUI) is implemented (see Figure 5.12). The GUI implements the essential user control

[4]Note the system processing pipeline is configured to run at a resolution of 80 × 80 pixel to maximise performance. This causes the pixelation visible in the figure.

requirements of being able to stop and start the system and receive feedback about the system status. It also provides some extra features, such as live RGB and image processed video feeds. The latter allows the user to monitor, in real time, the input to the DNN. Additional control is also implemented, which allows the user to toggle the video feed on and off in the occurrence of wireless network congestion to maintain an optimal system processing speed. Due to the large number of time critical operations that need to occur, logical sections of the control GUI code are split into threads to allow parallel execution.

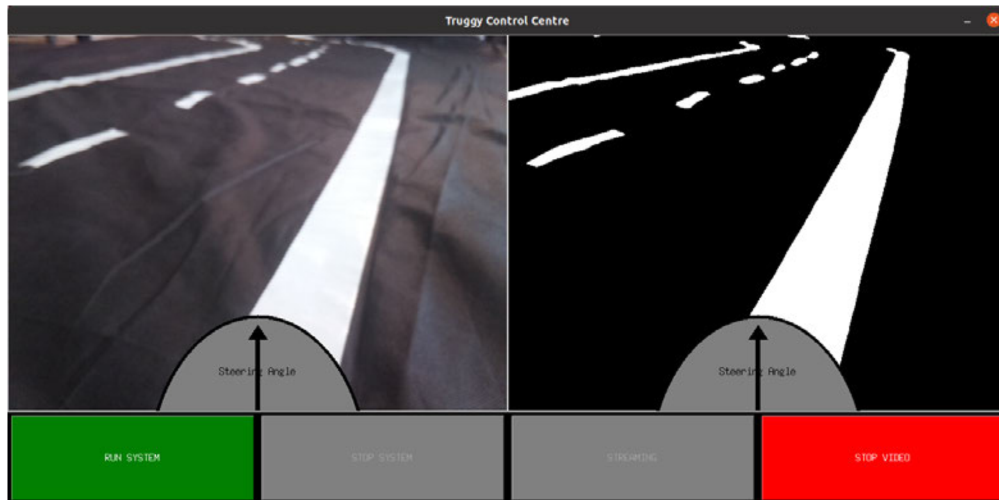

Figure 5.12.: Truggy control centre GUI, waiting in "ready" state

5.2.2. Experimental Environment

The trained agent (DNN) is implemented on the Paspberry Pi which is mounted on the Truggy RC car. As with the control GUI code, due to the large number of time critical operations that need to occur, logical sections of the vehicle control code are split into threads, to allow parallel execution.

The throttle control value is experimentally determined by visually comparing the speed of the vehicle in the 3D simulator to the Truggy in the experimental environment and matching them as closely as possible. The resulting speed is measured as 0.55 m/s.

5.2.3. Communication for User Control

Communication between the Truggy (Raspberry Pi) and control GUI (control computer) is established using a WiFi network connection. To ensure a low level of latency, video data is streamed from the Truggy to the control computer using the UDP protocol (see Section 2.5.1). Control values are communicated using an MQTT broker (see Section 2.5.1) running on the Raspberry Pi, with a unique topic for control values and steering angle alike.

6. Results

This chapter presents the final results obtained from the implementation process, which will then be discussed in Chapter 7. Both results from the 3D simulation environment as well as the experimental environment are presented.

To investigate the linear relationship between the amount of training and the resulting model performance, a Spearman rank correlation analysis is performed on the training results of each model architecture, at a significance level of $\alpha = 0.05$. Correlation analyses are used to denote the strength and direction of a relationship between two or more quantitative variables [18, 15]. The non-parametric Spearman correlation is chosen because the data is not normally distributed [15].

This analysis aims to provide an indication of how effective each DNN architecture is at learning a policy to control the vehicle on a per-episode basis. The correlation analysis results in a correlation coefficient r, with values in the range from -1 to 1 [18, 15]. A correlation coefficient of +1 indicates a perfect positive correlation, while a correlation coefficient of -1 suggests a perfect negative correlation [15]. Values close to +1 or -1 indicate a strong correlation [15]. Conversely, a correlation coefficient of zero indicates that there is no linear relationship between the two variables [15].

In the case of this thesis, the higher the correlation coefficient, the stronger the correlation between the amount of training and the DNN model's performance. Hence, the correlation analysis aims to provide a quantitative measure of how quickly the model is able to improve performance on a per-episode basis.

Additionally, the p-value (p) is calculated, which denotes whether the relationship is statistically significant [15]. This indicates how reliable the correlation analysis is, whereby a p-value of < 0.05 is considered significant.

6.1. Training

Figure 6.1 shows the results obtained from training the modified "DAVE-2" model architecture. It is observed that the number of steps per episode remains relatively stable until shortly before the 200th episode, after which higher peaks along with a steady increase in the cumulative average can be observed. A Spearman correlation test is performed on the episode number and the number of steps per episode for all episodes after the epsilon decay process is completed. This reveals a significant positive correlation of $r(328) = 0.619$, $p < 0.001$. The maximum number of steps recorded in a single episode during the training process using the modified "DAVE-2" model architecture is 900. After training, the model is allowed to run in the 3D simulation environment using the weights from the final training episode, achieving 3,376 control steps before crashing.

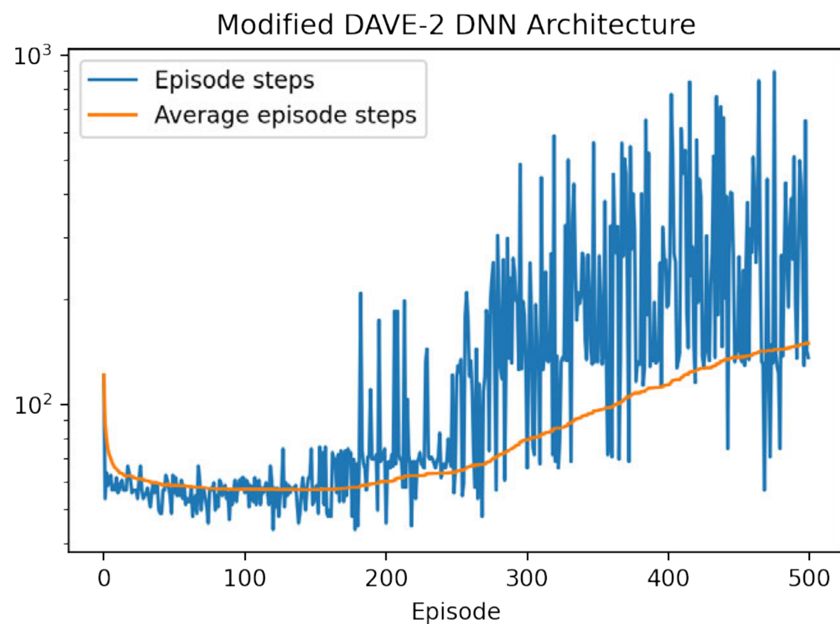

Figure 6.1.: Training outcomes for modified "DAVE-2" DNN architecture

Figure 6.2 shows the results obtained from training the Donkey Gym open source model architecture. It is observed that the number of steps per episode remains relatively stable until approximately the 200th episode, after which higher peaks can be observed, however low values remain similar to the earlier episodes. A stronger positive gradient can also be observed in the cumulative average in this area. Shortly after the 300th

episode a dramatic increase in the number of steps per episode is observed, which is strongly reflected in the cumulative average. A Spearman correlation test is performed on the episode number and the number of steps per episode for all episodes after the epsilon decay process was completed. This reveals a significant positive correlation of $r(341) = 0.755$, $p < 0.001$. The maximum number of steps recorded in a single episode during the training process using the Donkey Gym open source model architecture is 46,001. After training, the model is allowed to run in the 3D simulation environment using the weights from the final training episode. This is stopped after 24 hours without crashing, which equates to approximately 758,106 steps.

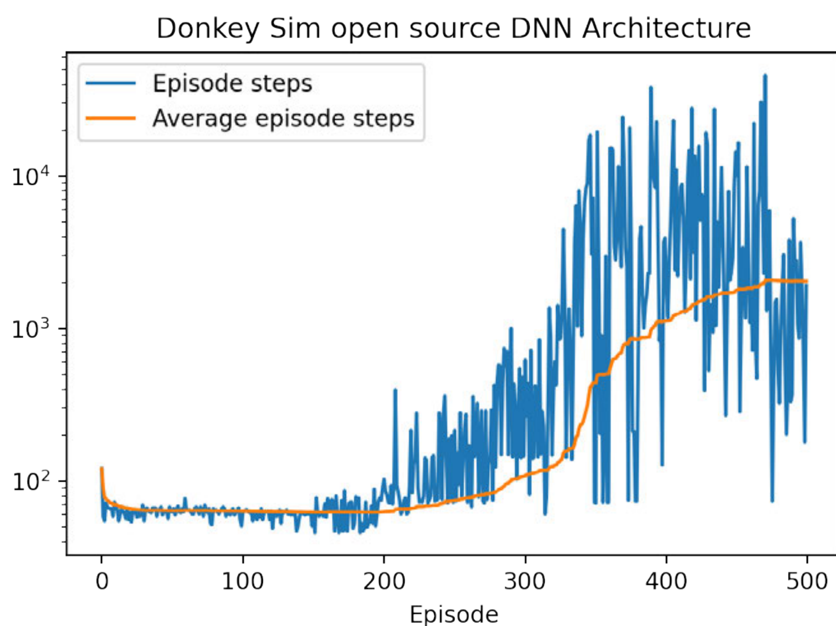

Figure 6.2.: Training outcomes for open source Donkey Gym DNN architecture

Figure 6.3 shows the results obtained from training the custom model architecture. It is observed that the number of steps per episode remains relatively stable until shortly before the 200th episode, after which higher peaks can be observed, in addition to a change in gradient to the cumulative average. A Spearman correlation test is performed on the episode number and the number of steps per episode for all episodes after the epsilon decay process is completed. This reveals a significant positive correlation of $r(327) = 0.655$, $p < 0.001$. The maximum number of steps recorded in a single episode during the training process using the custom model architecture is 27,766. After training,

the model is allowed to run in the 3D simulation environment using the weights from the final training episode. This is stopped after 24 hours without crashing, which equates to approximately 706,702 steps.

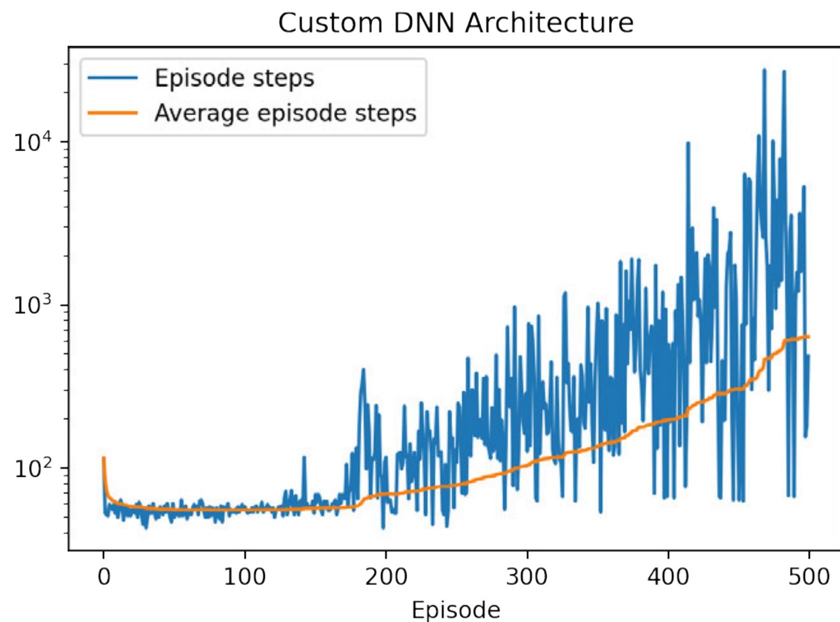

Figure 6.3.: Training outcomes for custom DNN architecture

6.2. Comparison of Training Results

As presented in the preceding sections of this chapter, each of the three model architectures produces different results during the training process.

Based upon the maximum control steps during the training process, the Spearman correlation test result and the maximum number of steps achieved in the simulation environment after training (see Table 6.1), the modified "DAVE-2" model architecture shows the weakest performance. The custom DNN model architecture shows better performance than the modified "DAVE-2" model architecture and the Donkey Gym open source DNN model architecture outperforms both the modified "DAVE-2" and custom model architectures. However regarding the post-training simulation run time, the Donkey Gym open source DNN model architecture ties with the custom model architecture, as both were stopped after running for 24 hours without crashing. Based upon its overall

Table 6.1.: Training results

Model architecture	Spearman correlation	Maximum steps during training	Maximum steps after training
"DAVE-2" model	$r(328) = 0.619$, $p < 0.001$	900	3,376
Donkey Gym open source model	$r(341) = 0.755$, $p < 0.001$	46,001	758,106 (no crash after 24 hrs)
Custom model	$r(327) = 0.655$, $p < 0.001$	27,766	706,702 (no crash after 24 hrs)

leading performance, the Donkey Gym open source model is selected for testing in the experimental environment.

6.3. Experimental Environment

Figure 6.3 shows the results of 100 trials using the trained Donkey Gym open source model architecture in the experimental environment. For each trial, the Truggy RC car was placed in the same starting position (see Figure 4.3) and the system was set to run using a button in the GUI. The graphs in Figure 6.3 depict the absolute number of trials that resulted in one of the three success categories (crash, straight, corner), whereby the graph colours match the track zones shown in Figure 4.3. The crash category indicates that the system was not able to successfully navigate a straight section of track, the straight category indicates that the system successfully navigated a straight section of track, but could not navigate around a corner and the corner category indicates that the system was able to successfully navigate both a straight section of track and around a corner[1]. From the 100 trials performed, 48 resulted in the vehicle leaving the track in the red zone (crash), 31 resulted in the vehicle leaving the track in the orange zone (straight) and 21 resulted in the vehicle leaving the track in the green zone (corner). Finally, the braking time is measured across 10 trials (see Table 6.2), resulting in a mean reaction time of $t_{stopping} = 0.78$s. The time recorded is measured from the moment the operator presses the stop button on the GUI to the moment the vehicle comes to a full stop.

[1]Note that the results do not include the purple zone shown in Figure 4.3, as no trial reached that area of the track.

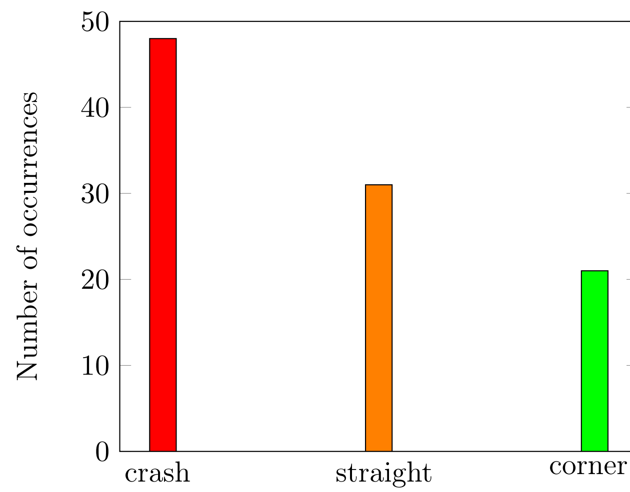

Figure 6.4.: Test outcomes using the trained Donkey Gym open source model in the experimental environment, plotted per success category.

Table 6.2.: Braking time

Trial number	**Time (s)**
1	0.93
2	0.76
3	0.89
4	1.08
5	0.49
6	0.63
7	0.54
8	0.71
9	0.89
10	0.91
Mean time	0.78

7. Discussion

This chapter discusses the results presented in Chapter 6. It also provides insights into the limitations of this thesis and offers suggestions for future work that may build upon this thesis.

7.1. Summary of Results

The training results presented in Chapter 6 show that RL is a viable and effective method for training a DNN for the purpose of autonomous vehicle control. All three network architectures demonstrated an ability to learn using RL and navigate the vehicle autonomously around the track in the simulation environment, with both the custom and Donkey Gym open source architectures demonstrating very good control abilities after training was completed, both of these model architectures were able to control the vehicle in the simulator for 24 hours without crashing or leaving the track.

Conversely, the results from the experimental environment show significantly poor performance, with 52% of all trials showing some level of control ability (i.e., successfully navigating a straight section or a straight section and a corner) and no trial successfully finishing an entire lap of the track.

7.2. Review of Requirements

This section presents the outcomes of the requirements, with each being categorised as either fulfilled, partially fulfilled or not fulfilled. A brief justification for the categorisation is provided for each requirement.

Functional Requirements

F1: The user should be able to start or stop the system at any time, as long as the system is powered on and the control system is connected. [FULFILLED]

The control GUI allows the operator to start or stop the system at any time, as long as the system is powered on and connected to the Truggy via WiFi.

F2: The system should receive and process camera sensor data in such a way that steering control values are produced. [FULFILLED]

The implemented system uses image frames captured from the Pi Camera Module 2 and processes them using binary thresholding for image segmentation. The segmented images are then processed using a DNN to produce steering control values.

F3: The system should determine steering control values based upon the road-markings present in the field of view of the camera. [FULFILLED]

The implemented binary thresholding for image segmentation is very effective at performing feature extraction of the road markings and removing distracting background features. This indicates that the DNN is receiving only information about the position of the road-markings, hence they are used for calculation of steering control values. This is visualised in Figures 5.10, 5.11 and 5.12.

F4: The system should react to a stop command in no more than the time or distance that a human driver would require. [FULFILLED]

As established in Chapter 4, the German motoring association ADAC lists the average human reaction time for braking as 0.8 to 1.2 seconds [1]. The RC car's mean braking time of 0.78 seconds suggests that the system outperforms a human driver, as it is able to bring the vehicle to a complete stop before a human driver would begin to brake.

F5: The user should receive feedback about the system's status. [FULFILLED]

The control GUI provides the user with information about the current operational status of the system and the current steering angle. It also displays streaming video in both the raw RGB format as well as the image processed format.

Non-Functional Requirements

NF1: Onboard control system processing should occur using a Deep Neural Network. [FULFILLED]

A DNN is used for the onboard processing of image frames in order to produce steering control values.

NF2: The Raspberry Pi must communicate with the PWM controller via I^2C bus. [FULFILLED]

Communication between the Raspberry Pi and the PWM controller is implemented using the I^2C bus protocol.

NF3: Road markings should be white on a black road-surface. The total road width should be equal to two vehicle widths. [FULFILLED]

The experimental environment was constructed using a black road-surface with white road-markings, with an overall road width equivalent to two Truggy RC cars.

NF4: The system should be implemented such that it can run on a Raspberry Pi 4 Model B Single Board Computer. [PARTIALLY FULFILLED]

The system in the experimental environment runs on a Raspberry Pi 4 Model B. However, the DNN inference time in the training environment is measured at an average of 0.01836 seconds, whereas inference on the Raspberry Pi takes on average 0.75378 seconds. As the execution speed has a strong impact upon the overall performance of the system, this requirement has been categorised as only partially fulfilled.

NF5: The Neural Network should be trained using reinforcement learning. [FULFILLED]

The system was trained using reinforcement learning.

7.3. Limitations

Using the results established in Chapter 6 and the outcomes discussed in Section 7.1, limitations in the implementation and experimental setup are now discussed.

As addressed in Section 7.2, the execution speed of the system in the experimental environment, specifically the DNN inference time, is substantially slower than in the training environment. Because the system operates in a real-time environment, this limitation directly affects the system's ability to perform vehicle control effectively in the experimental environment.

Due to the requirement of image processing to minimise the difference in input stimuli between the 3D simulation environment and the experimental environment, a simulation environment was selected based upon the ability to extract desirable features from image frames, while ignoring undesirable and potentially distracting information. As introduced in Section 5.1.1, the Donkey Gym open source simulator [23] provides ten environments, however due to road-surface textures and unsuitable road-markings, only one environment was suitable for training the final model that is transferred into the experimental environment. Unfortunately this environment has a very simple design, which resulted in a model trained for use in a very specific environment.

Additionally, as discussed in Section 5.1.4, due to the simplicity of the 3D simulation environment, the throttle control value is set to 10% in order to prevent the DNN from learning an optimal policy that simply holds the steering control at a constant value. By lowering the throttle control value, the policy is forced to steer in a straight line for straight sections of the track, rather than holding a constant steering control value and cutting the corners of the track. A more complex 3D simulation environment that features both left and right turns of different radii would not require the throttle control variable to be manipulated in such a manner, in order to achieve the desired behavioural outcome.

A further limitation is the requirement for strong image processing in order to produce input stimuli for the DNN which are similar in both the 3D simulation environment and the experimental environment. One of the core capabilities of a DNN is to perform feature extraction, although this is limited by the complexity of the input stimuli (the higher the complexity, the more features can theoretically be extracted). The aforementioned image processing limits this complexity.

In the process of transferring the trained model from the 3D simulator to the experimental environment, another limitation is the accuracy of the Truggy RC car's steering mechanism compared to the 3D simulator. Actions in the 3D simulator are perfectly repeatable, whereas actions in the experimental environment do not have such high precision repeatability. The steering mechanism on the Truggy RC car has a sub-mechanism

that prevents damage to either the steering servo or the steering rack itself. Whilst there is obvious merit in this functionality, it prevents accurately repeatable steering control.

The most notable limitation is transferring a model trained in a perfectly repeatable 3D simulation environment into an imperfect real-world experimental environment. As shown in Chapter 6, the training results in a highly accurate and performant vehicle control system in the 3D simulation environment, however when implemented in the experimental environment, the same level of accuracy and performance is not achieved. Whilst image processing is used to normalise the input stimuli between the two environments, other variables such as steering mechanism accuracy are not controlled.

7.4. Future Work

Using the limitations set out in Section 7.3, recommendations for future work are now offered.

The most limitations of this thesis stem from discrepancies between the 3D simulation environment and the experimental environment. To address these limitations and allow the system trained in the 3D simulator to be directly implemented in the experimental environment, two approaches are conceptualised.

The first is to introduce a small level of error into the steering controls of the 3D simulator to simulate the uncertainty of the real-world experimental setup. This could be as simple as randomly adding or subtracting a percentage of the overall steering range from the DNN generated control value, hence preventing the 3D simulator from producing perfectly repeatable steering controls.

The second approach is to replace the steering mechanism on the Truggy RC car with a direct-drive mechanism that provides more accurate and repeatable steering control. With the implementation of this modification, it is expected that the increase in steering mechanism accuracy would result in an increase in performance of the control system in the experimental environment.

Alternatively, the limitations could be addressed by taking the model trained in the 3D simulator and using it as a baseline to be trained further in the experimental environment. Training the model in the environment in which it will be deployed and tested will produce

the most performant outcomes, however will require a large amount of implementation, especially in the image processing domain.

To address the execution speed limitation in the experimental environment, a hardware upgrade is required. Whilst DNNs can be run on a CPU (as is the case in this thesis with the use of the Raspberry Pi), the workload they produce is better suited for a graphics processing unit (GPU). By deploying the system on hardware which has a dedicated GPU, such as an NVIDIA Jetson which is designed for use with AI[1], it is expected that performance in the experimental environment will increase substantially.

Finally, due to the time constraints placed upon this thesis and the intrinsically long periods of time required to train a model using RL, only three models were compared. Future authors are strongly encouraged to use this thesis as a foundational framework to allow them to commence training more quickly in the hope that they can then allocate more time to model tuning.

[1] https://developer.nvidia.com/embedded/jetson-modules

8. Conclusion

This thesis investigated the use of reinforcement learning for the purpose of autonomous vehicle control. A reinforcement learning agent, in the form of a Deep Neural Network, was trained using online off-policy Deep Q-Learning. The resulting trained Deep Neural Network was then implemented on a Raspberry Pi 4 Model B Single Board Computer, alongside a comprehensive control and communications system that allows a user to remotely control and monitor the system using a Graphical User Interface.

To conclude, this thesis has fulfilled all five functional requirements and four out of five non-functional requirements, whilst one non-functional requirement was only partially fulfilled. The results achieved during training demonstrate a strong potential of reinforcement learning for use cases such as autonomous vehicle control. Three different Deep Neural Network architectures were compared, whereby the Donkey Gym open source model architecture showed the best performance. However, the lacklustre performance of the system in the real world experimental environment reinforces that all variables in the training and experimental environments must be carefully controlled, such that they match as closely as possible.

The author strongly encourages future authors to take this thesis as a foundation and build upon it by taking into consideration the presented limitations and considerations for future work.

Bibliography

[1] ADAC: *Bremsweg berechnen: Mit dieser Formel geht's.* 2021. – URL https://www.adac.de/verkehr/rund-um-den-fuehrerschein/erwerb/bremsweg-berechnen/. – Zugriffsdatum: 2022-08-16

[2] Arulkumaran, Kai ; Deisenroth, Marc P. ; Brundage, Miles ; Bharath, Anil A.: Deep reinforcement learning: A brief survey. In: *IEEE Signal Processing Magazine* 34 (2017), Nr. 6, S. 26–38

[3] Beasley, Jeffrey S.: *Networking Second Edition*. Pearson Education, Inc., 2009

[4] Bojarski, Mariusz ; Del Testa, Davide ; Dworakowski, Daniel ; Firner, Bernhard ; Flepp, Beat ; Goyal, Prasoon ; Jackel, Lawrence D. ; Monfort, Mathew ; Muller, Urs ; Zhang, Jiakai u. a.: End to end learning for self-driving cars. In: *arXiv preprint arXiv:1604.07316* (2016)

[5] Bovik, Alan C.: *The essential guide to image processing*. Academic Press, 2009

[6] Brockman, Greg ; Cheung, Vicki ; Pettersson, Ludwig ; Schneider, Jonas ; Schulman, John ; Tang, Jie ; Zaremba, Wojciech: Openai gym. In: *arXiv preprint arXiv:1606.01540* (2016)

[7] Conrad Electronic SE: *1:10 Electro-Truggy „Dart", 2WD, RtR Operating Instructions*. 2015. – URL https://asset.conrad.com/media10/add/160267/c1/-/gl/001405819ML02/manual-1405819-reely-dart-brushed-110-rc-model-car-electric-truggy-rwd-100-rtr-24-ghz.pdf. – Zugriffsdatum: 2022-08-21

[8] Dick, Stephanie: Artificial Intelligence. In: *Harvard Data Science Review* 1 (2019), jul 1, Nr. 1. – https://hdsr.mitpress.mit.edu/pub/0aytgrau

[9] Ding, Lijun ; Goshtasby, Ardeshir: On the Canny edge detector. In: *Pattern recognition* 34 (2001), Nr. 3, S. 721–725

[10] Dordal, Peter L.: *An Introduction to Computer Networks*. Loyola University, Chicago, 2021

[11] Emzivat, Yrvann ; Ibanez-Guzman, Javier ; Martinet, Philippe ; Roux, Olivier H.: Dynamic driving task fallback for an automated driving system whose ability to monitor the driving environment has been compromised. In: *2017 IEEE Intelligent Vehicles Symposium (IV)* IEEE (Veranst.), 2017, S. 1841–1847

[12] Faisal, Asif ; Kamruzzaman, Md ; Yigitcanlar, Tan ; Currie, Graham: Understanding autonomous vehicles. In: *Journal of transport and land use* 12 (2019), Nr. 1, S. 45–72

[13] François-Lavet, Vincent ; Henderson, Peter ; Islam, Riashat ; Bellemare, Marc G. ; Pineau, Joelle u. a.: An introduction to deep reinforcement learning. In: *Foundations and Trends® in Machine Learning* 11 (2018), Nr. 3-4, S. 219–354

[14] GMC: *Super Cruise Driver Assistance - Hands-Free Driving | GMC*. 2022. – URL https://www.gmc.com/connectivity-technology/super-cruise. – Zugriffsdatum: 2022-06-16

[15] Gogtay, Nithya J. ; Thatte, Urmila M.: Principles of correlation analysis. In: *Journal of the Association of Physicians of India* 65 (2017), Nr. 3, S. 78–81

[16] Grigorescu, Sorin ; Trasnea, Bogdan ; Cocias, Tiberiu ; Macesanu, Gigel: A survey of deep learning techniques for autonomous driving. In: *Journal of Field Robotics* 37 (2020), Nr. 3, S. 362–386

[17] Hensel, Marc: *Digitale Bildverarbeitung*. HAW Hamburg, 2022

[18] Holling, Heinz ; Schmitz, Bernhard: *Handbuch Statistik, Methoden und Evaluation*. Hogrefe Verlag, 2010

[19] Hunkeler, Urs ; Truong, Hong L. ; Stanford-Clark, Andy: MQTT-S—A publish/subscribe protocol for Wireless Sensor Networks. In: *2008 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE'08)* IEEE (Veranst.), 2008, S. 791–798

[20] Inagaki, Toshiyuki ; Sheridan, Thomas B.: A critique of the SAE conditional driving automation definition, and analyses of options for improvement. In: *Cognition, technology & work* 21 (2019), Nr. 4, S. 569–578

[21] Ingle, Shantanu ; Phute, Madhuri: Tesla autopilot: semi autonomous driving, an uptick for future autonomy. In: *International Research Journal of Engineering and Technology* 3 (2016), Nr. 9, S. 369–372

[22] Jordan, Michael I. ; Mitchell, Tom M.: Machine learning: Trends, perspectives, and prospects. In: *Science* 349 (2015), Nr. 6245, S. 255–260

[23] Kramer, T. ; Sokolkov, R.: *OpenAI Gym Environments for Donkey Car*. 2018. – URL https://github.com/tawnkramer/gym-donkeycar. – Zugriffsdatum: 2022-07-15

[24] Krogh, Anders: What are artificial neural networks? In: *Nature biotechnology* 26 (2008), Nr. 2, S. 195–197

[25] Krohn, Jon ; Beyleveld, Grant ; Bassens, Aglaé: *Deep learning illustrated: a visual, interactive guide to artificial intelligence*. Addison-Wesley Professional, 2019

[26] Lapan, Maxim: *Deep Reinforcement Learning Hands-On: Apply modern RL methods, with deep Q-networks, value iteration, policy gradients, TRPO, AlphaGo Zero and more*. Packt Publishing Ltd, 2018

[27] Leens, Frédéric: An introduction to I^2C and SPI protocols. In: *IEEE Instrumentation & Measurement Magazine* 12 (2009), Nr. 1, S. 8–13

[28] Li, Yuxi: *Deep Reinforcement Learning*. 2018. – URL https://arxiv.org/abs/1810.06339

[29] McCarthy, John ; Minsky, Marvin L. ; Rochester, Nathaniel ; Shannon, Claude E.: A proposal for the dartmouth summer research project on artificial intelligence, august 31, 1955. In: *AI magazine* 27 (2006), Nr. 4, S. 12–12

[30] Meyer, Jonas ; Becker, Henrik ; Bösch, Patrick M. ; Axhausen, Kay W.: Autonomous vehicles: The next jump in accessibilities? In: *Research in transportation economics* 62 (2017), S. 80–91

[31] Mnih, Volodymyr ; Kavukcuoglu, Koray ; Silver, David ; Rusu, Andrei A. ; Veness, Joel ; Bellemare, Marc G. ; Graves, Alex ; Riedmiller, Martin ; Fidjeland, Andreas K. ; Ostrovski, Georg u. a.: Human-level control through deep reinforcement learning. In: *nature* 518 (2015), Nr. 7540, S. 529–533

[32] Mohseni, Sina ; Pitale, Mandar ; Singh, Vasu ; Wang, Zhangyang: Practical solutions for machine learning safety in autonomous vehicles. In: *arXiv preprint arXiv:1912.09630* (2019)

[33] Petrou, Maria M. ; Petrou, Costas: *Image processing: the fundamentals*. John Wiley & Sons, 2010

[34] Raspberry Pi Trading Ltd.: *Raspberry Pi 4 Model B Product Brief*. 2021. – URL https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-product-brief.pdf. – Zugriffsdatum: 2022-07-15

[35] Raspberry Pi Trading Ltd.: *Raspberry Pi Camera Algorithm and Tuning Guide*. 2021. – URL https://datasheets.raspberrypi.com/camera/raspberry-pi-camera-guide.pdf. – Zugriffsdatum: 2022-07-15

[36] Robertazzi, Thomas G.: *Introduction to computer networking*. Springer, 2017

[37] Rosenblatt, Frank: The perceptron: a probabilistic model for information storage and organization in the brain. In: *Psychological review* 65 (1958), Nr. 6, S. 386

[38] Shladover, Steven E.: Connected and automated vehicle systems: Introduction and overview. In: *Journal of Intelligent Transportation Systems* 22 (2018), Nr. 3, S. 190–200

[39] Song, Renjie ; Zhang, Ziqi ; Liu, Haiyang: Edge connection based Canny edge detection algorithm. In: *Pattern Recognition and Image Analysis* 27 (2017), Nr. 4, S. 740–747

[40] Sutton, Richard S. ; Barto, Andrew G.: *Reinforcement learning: An introduction*. MIT press, 2018

[41] Van Arem, Bart ; Van Driel, Cornelie J. ; Visser, Ruben: The impact of cooperative adaptive cruise control on traffic-flow characteristics. In: *IEEE Transactions on intelligent transportation systems* 7 (2006), Nr. 4, S. 429–436

[42] Yao, Qiangqiang ; Tian, Ying ; Wang, Qun ; Wang, Shengyuan: Control strategies on path tracking for autonomous vehicle: State of the art and future challenges. In: *IEEE Access* 8 (2020), S. 161211–161222

[43] Zhou, Mofan ; Qu, Xiaobo ; Jin, Sheng: On the impact of cooperative autonomous vehicles in improving freeway merging: a modified intelligent driver model-based

approach. In: *IEEE Transactions on Intelligent Transportation Systems* 18 (2016), Nr. 6, S. 1422–1428

A. DNN Model Summaries

A.1. Modified "DAVE-2" Model Summary

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 40, 40, 24)        2424
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 20, 20, 36)        21636
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 10, 10, 48)        43248
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 10, 10, 64)        27712
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 10, 10, 64)        36928
_________________________________________________________________
flatten (Flatten)            (None, 6400)              0
_________________________________________________________________
dense (Dense)                (None, 1164)              7450764
_________________________________________________________________
dense_1 (Dense)              (None, 100)               116500
_________________________________________________________________
dense_2 (Dense)              (None, 50)                5050
_________________________________________________________________
dense_3 (Dense)              (None, 15)                765
=================================================================
Total params: 7,705,027
Trainable params: 7,705,027
```

```
Non-trainable params: 0
```

A.2. Open Source Donkey Gym Model Summary

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 40, 40, 24)        2424
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 20, 20, 32)        19232
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 10, 10, 64)        51264
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 5, 5, 64)          36928
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 5, 5, 64)          36928
_________________________________________________________________
flatten (Flatten)            (None, 1600)              0
_________________________________________________________________
dense (Dense)                (None, 512)               819712
_________________________________________________________________
dense_1 (Dense)              (None, 15)                7695
=================================================================
Total params: 974,183
Trainable params: 974,183
Non-trainable params: 0
_________________________________________________________________
```

A.3. Custom DNN Model Summary

```
Model: "sequential"
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 40, 40, 24)        2424
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 20, 20, 48)        28848
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 10, 10, 96)        115296
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 5, 5, 96)          83040
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 5, 5, 96)          83040
_________________________________________________________________
flatten (Flatten)            (None, 2400)              0
_________________________________________________________________
dense (Dense)                (None, 1024)              2458624
_________________________________________________________________
dense_1 (Dense)              (None, 15)                15375
=================================================================
Total params: 2,786,647
Trainable params: 2,786,647
Non-trainable params: 0
_________________________________________________________________
```

B. Raspberry Pi Preparation Steps

1. Create an x64 image for the pi using Raspberry Pi Imager.
2. Open the IP configuration file:

```
sudo nano /etc/dhcpcd.conf
```

3. Add the following lines to the end of the file to set a static IP address:

```
interface wlan0
static ip_address=10.0.0.10/24
static routers=10.0.0.1
static domain_name_servers=10.0.0.1
```

4. Save the file by pressing:

```
ctrl x
y
enter
```

5. Install Conda:

```
cd Downloads
wget https://github.com/conda-forge/miniforge/releases/
latest/download/Miniforge3-Linux-aarch64.sh
bash Miniforge3-Linux-aarch64.sh
cd ..
```

6. Create a Conda environment:

```
conda create --name tf_truggy python=3.7
conda activate tf_truggy
```

7. Install the required dependencies in the Conda environment:

```
pip install https://github.com/bitsy-ai/
tensorflow-arm-bin/releases/download/
v2.4.0-rc2/tensorflow-2.4.0rc2-cp37-none-linux_aarch64.whl
conda install -c conda-forge opencv
pip3 install adafruit-circuitpython-servokit
pip3 install imutils
pip3 install paho-mqtt
sudo apt install ffmpeg libsm6 libxext6  -y
sudo apt install mosquitto mosquitto-clients -y
```

8. Edit the mosquitto configuration file to allow remote connections to the mosquitto mqtt server:

```
cd /etc/mosquitto
sudo nano mosquitto.conf
```

9. Add the following lines to the end of the file:

```
listener 1883
allow_anonymous true
```

10. Save the file by pressing:

```
ctrl x
y
enter
```

11. Enable the Pi Camera interface and I2C bus:

```
sudo raspi-config
Select: Interfacing options
Set the Legacy camera option to: enabled
Set I2C option to: enabled
```

12. The Raspberry Pi is now ready to run the control system.

Declaration

I declare that this Bachelor Thesis has been completed by myself independently without outside help and only the defined sources and study aids were used.

City | Date | Signature