

An Investigation into Machine Learning Algorithms & Web Applications Applied to Autonomous Vehicles

by Adam Davies

Supervisor: Michael Dacey

Second Marker: Gordon Dickers

Project submitted as part of the requirements for the award of MSc Software
Engineering

University of Wales Trinity Saint David

Date: January 2020

Declaration Form and Form of Consent

Declaration

I, Adam Peter Davies declare that I am the sole author of this Project; that all references cited have been consulted; that I have conducted all work of which this is a record, and that the finished work lies within the prescribed word limits.

This work has not previously been accepted as part of any other degree submission.

Signed:

Date:

Form of Consent

I Adam Peter Davies hereby consent that my Project, submitted in candidature for the MSc Software Engineering degree, if successful, may be made available for inter-library loan or photocopying (subject to the law of copyright), and that the title and abstract may be made available to outside organisations.

Signed:

Date:

Abstract

Currently, the automotive industry is in a phase of rapid technological growth. Advances in various technologies such as Advanced Driver Assistance Systems (ADAS) and Artificial Intelligence (AI) have contributed to autonomous vehicles becoming a reality of a not too distant future. The primary principle of autonomous vehicles is to provide a machine with the full responsibility of driving. A question which arises is how can a machine perform human-like decisions? The answer is by implementing AI through a technique known as machine learning. Machine learning provides a computer system with the ability to learn from previous data and make logical decisions on new data, without having to be explicitly programmed.

This project outlines an investigation into the field of machine learning relating to autonomous vehicles. Additionally, the use of web applications in autonomous vehicles is investigated. The main aim of the project was to evaluate a test system simulating an autonomous vehicle, utilizing a machine learning model. Additionally, a web application was implemented to control and train the vehicle remotely.

The developed vehicle utilized two Convolutional Neural Networks (CNNs) which were used to predict speed and steering. Speed was predicted using a classification model and steering using a regression. The web application developed included authentication and provided remote control of the vehicle over HTTP. The system was evaluated on trained and unseen tracks, measuring the automation levels achieved and how well the vehicle drives. It was found that the vehicle achieved a high level of automation and compared favourably with similar studies. The vehicles performance was measured concerning speed and steering. It was found that the vehicle drives well on all tracks with results showing good driving throughout. There were positions on the track the vehicle struggled with, which were on sharper bends.

From the results achieved it is believed that CNN models excel at image processing and can be applied to autonomous vehicles in lane marking detection. However, a lot of training data and time is required to make the systems robust and accurate. The final system is regarded to drive considerably well on the majority of tracks, with it predicting the speed and steering effectively.

Acknowledgements

This project has challenged me intensely throughout its duration. The completion and success of this project would not have been possible without the continued support of many people. Firstly, I would like to take this opportunity to express deep gratitude to my supervisor Michael Dacey. He has challenged me to achieve the best out of myself and the project. With his guidance he directed me into interesting research areas which encouraged me to complete this project. His continued advice and support throughout has contributed to the success of this project. I would also like to thank my family, friends and work colleagues for their support and encouragement. Without support from everyone I would not have been able to achieve what I have.

Thank you.

Table of Contents

1	Introduction	1
1.1	Aim and Objectives	2
1.2	Report Structure	3
2	Background Information and Related Studies	4
2.1	Autonomous Vehicles	4
2.1.1	Levels of Automation	5
2.1.2	What is a Fully Autonomous Vehicle?	5
2.2	Machine Learning	6
2.2.1	What is Machine Learning?	7
2.2.1.1	Machine Learning Process	8
2.2.2	Supervised Learning	9
2.2.2.1	Classification	9
2.2.2.2	Regression	10
2.2.3	Unsupervised Learning	10
2.2.3.1	Clustering	10
2.2.3.2	Association	11
2.2.4	Semi-Supervised Learning	11
2.2.5	Reinforcement Learning	12
2.2.6	Deep Learning	13
2.2.6.1	What is an Artificial Neural Network (ANN)?	13
2.2.6.2	Feed-forward Neural Networks	14
2.2.6.3	Convolution Neural Networks (CNN)	15
2.2.6.4	Recurrent Neural Networks (RNN)	16
2.2.6.5	Training Neural Networks	16
2.2.6.6	Optimization	17
2.2.7	Evaluating Machine Learning Algorithms	18

2.2.7.1	Metrics	18
2.2.7.2	Evaluating Autonomous Vehicles	19
2.2.8	Technologies	20
2.2.8.1	Scikit-Learn	21
2.2.8.2	TensorFlow	21
2.2.8.3	PyTorch	21
2.2.8.4	Summary	22
2.2.9	Challenges in Autonomous Vehicles	22
2.3	Web and Mobile Applications in Autonomous Vehicles	22
2.3.1	Existing Systems	23
2.3.2	Challenges	23
2.3.3	Proposed Application	24
2.3.3.1	Database	24
2.3.3.2	Server	25
2.3.3.3	Client	25
2.4	Related Studies	26
2.4.1	Machine Learning	26
2.4.2	Web/Mobile Applications	27
2.5	Chapter Summary	28
3	Methodology	29
3.1	Research Methodology	29
3.1.1	Philosophy	30
3.1.2	Approach	30
3.1.3	Strategies	31
3.1.3.1	Document Analysis	31
3.1.3.2	Controlled Experimentation	31
3.1.4	Time Horizon	31
3.1.5	Data Collection	32
3.1.5.1	Automation Levels	32
3.1.5.2	Vehicle Performance	33
3.2	Development Methodology	33
3.2.1	Requirements	34
3.2.2	Design	34
3.2.3	Implementation	35
3.2.4	Testing	35

3.3	Chapter Summary	36
4	Design	37
4.1	System Requirements	37
4.2	System Overview	39
4.3	Hardware	40
4.3.1	Onboard Computer	40
4.3.2	Chassis	41
4.3.3	Camera	42
4.3.3.1	Motor Driver Controller	42
4.3.4	Hardware Schematic	43
4.4	Software	44
4.4.1	Web Application	44
4.4.1.1	Database	45
4.4.2	API and Client	46
4.4.2.1	Authentication	46
4.4.2.2	Validation	47
4.4.3	Vehicle System	48
4.4.3.1	Manual Control	48
4.4.3.2	Training Control	49
4.4.3.3	CNN Architecture	50
4.4.3.4	Autonomous Control	52
4.4.4	Full System	52
4.4.4.1	Package Diagram	53
4.4.4.2	Deployment Diagram	53
4.5	Chapter Summary	54
5	Implementation	56
5.1	Implementation Process	56
5.2	System Overview	56
5.2.1	Web Application	57
5.2.2	Vehicle System	58
5.3	Authentication	58
5.4	Vehicle Controller	60
5.4.1	Manual Control	60
5.4.2	Training Control	61

5.5	Custom CNN	62
5.5.1	Training	62
5.5.2	Optimization	64
5.5.2.1	Number of Convolutional Layers	65
5.5.2.2	Training Optimizer	66
5.5.3	Autonomous Driving	66
5.5.4	Final Architecture	67
5.6	Chapter Summary	68
6	Results and Analysis	69
6.1	Convolutional Neural Network (CNN) Performance	69
6.1.1	Classification Model	69
6.1.1.1	Classification Accuracy	70
6.1.1.2	AUC - ROC	70
6.1.2	Regression Model	72
6.2	Vehicle Performance	73
6.2.1	Automation Level	73
6.2.2	Driving Performance Results	74
6.2.2.1	Speed	75
6.2.2.2	Steering	77
6.3	Chapter Summary	80
7	Conclusion and Future Work	82
7.1	Future Work	84
	Reflection	85
	References	87
	Appendix A - Data Collection Summary	100
	Appendix B - Use Case Descriptions	102
	Appendix C - Project Plan	104

Table of Figures

2.1	Autonomous vehicle sensor layout	6
2.2	Machine learning process	9
2.3	Reinforcement learning state diagram	12
2.4	Perceptron Network	14
2.5	Max pooling depiction	16
3.1	The research onion	30
4.1	Use case diagram based around the full system	39
4.2	Initial system block diagram based around the full system	40
4.3	Raspberry Pi 3 B+	41
4.4	Sseed Studio robot chassis	42
4.5	Fish Eye camera module	42
4.6	TB6612FNG motor driver IC	43
4.7	Final hardware schematic	44
4.8	Web application three tier layered architecture overview	45
4.9	Entity relationship diagram of the web applications database	46
4.10	Sequence diagram for web application depicting JWT authentication	47
4.11	Sequence diagram for highlighting validation for registering a vehicle	48
4.12	Sequence diagram for vehicle system depicting manual vehicle control	49
4.13	Activity diagram for vehicle system depicting training vehicle control	50
4.14	NVIDIA Pilot net CNN architecture	51
4.15	Activity diagram for vehicle system depicting the main auto-drive loop	52
4.16	Package diagram showing the full systems package structure	53
4.17	Deployment diagram of the full system	54
5.1	Final hardware implementation	58
5.2	Manual control dashboard layout	60

5.3	Training track layouts	63
5.4	Image processing sequence	64
5.5	Results from convoloutional layers optimization	65
5.6	Results from training optimizer optimization	66
5.7	Final CNN architecture	68
6.1	Effect of classification accuracy over training time	70
6.2	Effect of classification accuracy over training time	71
6.3	Effect of regression loss over training time	72
6.4	Unseen test track layouts	73
6.5	Unseen test track layouts	74
6.6	Speed results achieved on track one	75
6.7	Speed results achieved on track six	76
6.8	Steering results achieved on track one	77
6.9	Steering results achieved on track three	78
6.10	Steering results achieved on track six	79
6.11	Dataset distribution of steering labels	80

Table of Listings

5.1	JWT authentication filter function	59
5.2	Training control endpoint function which collects training data	62
5.3	Grid search functionality implementation	65
5.4	Predication loop implemented for autonomous driving	67

Chapter 1

Introduction

The idealization that people are poor drivers is well-reported in mainstream society [1, 2]. While this thought is often over-generalized, over 90% of traffic collisions are a result of human error [3]. The majority of these accidents are caused through lack of focus and judgement, which is nothing unexpected as driving is a task fraught with rapid changes in surroundings and the environment, with us having to perform instinctive decisions.

Currently, the automotive industry is in a phase of rapid technological growth. Advances in various technologies such as Advanced Driver Assistance Systems (ADAS) and Artificial Intelligence (AI) have contributed to autonomous vehicles becoming a reality of a not too distant future. The primary principle of autonomous vehicles is to provide a machine with the full responsibility of driving. A question which arises is how can a machine perform human-like decisions? The answer is by implementing AI through a technique known as machine learning. Machine learning provides a computer system with the ability to learn from previous data and make logical decisions on new data, without having to be explicitly programmed.

Machine learning is growing rapidly with subfields such as Deep Learning catapulting the field. Autonomous vehicles are heavily reliant on machine learning to make adaptive decisions, making it currently one of the most exciting and growing applications of the field. Additionally, these growths continue to maintain a beneficial effect on the development of autonomous vehicles. Prior to autonomous vehicles being released they must be 100% reliable, with the algorithms requiring a large amount of training to be able to operate safely. Even current autonomous vehicle projects are still being road trained with various machine learning models still being researched to date [4].

The advantages which autonomous vehicles potentially bring could be extraordinary, with increases in safety and efficiency [5]. Alternative advantages of autonomous vehicles are increases in both consumer comfort and convenience. These advantages have been enhanced considerably over the last decade in the automotive industry, with the introduction of ADAS and more re-

cently interactive web and mobile applications. Highly sophisticated applications exist currently and provide consumers with the ability to park a vehicle or send a vehicle to a provided location. Such applications are expected to continuously grow and become more complex with increased automation in vehicles.

The purpose of this project is to explore and investigate the use of machine learning algorithms in the field of autonomous vehicles developing a small-scale simulation of an autonomous vehicle. The work focuses on both steering and speed control through appropriate machine learning models. The vehicle will be trained on small-scale tracks replicating a road. To aid in training the vehicle a web application will be developed to interact with the vehicle as a trainer and controller. The ultimate goal of the project is to evaluate the implemented vehicle in terms of automation and how well it drives on various tracks. Although work in this field is well documented, which is understandable given autonomous vehicles current status in a developing field, the work is still considered relevant. The goal is to add views to the current developments of machine learning and automation vehicles with a desire that this research is to be applicable as the technology becomes closer to reality.

1.1 Aim and Objectives

The project aims to investigate and evaluate the effectiveness of a machine learning algorithm used to develop a small scale autonomous vehicle. Additionally, the use of web applications in autonomous vehicles is to be investigated, with a web application developed to control and train the vehicle. To achieve the stated aim the following objectives were set:

1. To investigate the research literature around machine learning algorithms and web applications used in autonomous vehicles, summarized in a literature review.
2. To design and develop a small scale autonomous vehicle utilizing appropriate machine learning model to control the speed and steering of the vehicle.
3. To design and develop a web application to aid in the control and training of the vehicle remotely.
4. To test the implemented vehicle based on metrics found through literature, collecting and recording results.
5. To analyse findings collected from performed tests, comparing results with related studies found from researched literature.

1.2 Report Structure

The report is structured with five main chapters. Chapter two overviews all related background information to the project including a discussion on current literature found. The third chapter overviews both the research and development methodologies undertaken. Outlined in chapter four is the design process for the proposed system. Subsequently, chapter five outlines the implemented system based on the design, highlighting key implementation details. Chapter six of the report presents a discussion on the results achieved, analysing significant findings. All results and findings made through the project are concluded at the end of the report in the final chapter.

Chapter 2

Background Information and Related Studies

This chapter of the report presents all the background information relating to the project. The chapter includes four key sections, autonomous vehicles, machine learning, mobile and web applications within the field of autonomous vehicles, and related studies. Each section discusses literature focused on the corresponding area. The related studies section overviews a collection of studies, where a review and discussion is produced on the findings concluded. A summary of all findings is presented at the end of the chapter.

2.1 Autonomous Vehicles

The field of autonomous vehicles is commonly perceived as a recently emerging area of technological growth in the automotive industry. However, research and development dates back to the mid-1980s. The first significant development was produced by Ernst Dickmanns [6] who developed a self-driving vision-based Mercedes-Benz van. However, this creation was never adopted by automobile manufacturers or institutes due to research and technology costs.

Advances in technology and an increase in demand for safety resulted in the research and development of semi-autonomous systems, known as Advanced Driver Assistance Systems (ADAS) [7]. ADAS feature on many modern cars with systems including lane-keeping assist, adaptive cruise control, stop assist and park assist [8]. Due to advances in technologies such as ADAS and Artificial Intelligence (AI) fully autonomous vehicles have become a reality of the not too distant future.

2.1.1 Levels of Automation

Due to the increase in technological enhancements in the motor vehicle industry, the Society of Automotive Engineers (SAE) determined classification levels of automation in vehicles [9]. There are six levels of automation with the classifications being adopted by the National Highway and Traffic Safety Administration (NHTSA) [10]. The following are the levels determined by the SAE:

- Level 0 (No Automation): Driver performs all the driving without the aid of any ADAS.
- Level 1 (Driver Assistance): Form of driver assistance such as power steering and brake assist. Majority of vehicles currently on the road possess level 1.
- Level 2 (Partial Automation): Vehicle can control steering and acceleration in particular instances. Examples include vehicles with adaptive cruise control and lane-keep assist technologies.
- Level 3 (Conditional Automation): Allow the driver to be unaware of the environment or surroundings. However, the driver must be ready to intervene if required [11].
- Level 4 (High Automation): Vehicles that are classified as level 4 or above are considered as autonomous, with the vehicle performing the majority of driving tasks [12]. However, level 4 classified vehicles can not be driven in specified environmental conditions [13].
- Level 5 (Full Automation): Vehicle is fully autonomous and are unrestricted to environmental conditions.

2.1.2 What is a Fully Autonomous Vehicle?

Nikowitz [14] defines an autonomous vehicle as “A vehicle that is designed to travel between destinations without a human operator.”. Fundamentally, this defines that a fully autonomous vehicle requires no driver with no steering wheel or operating pedals. Fully autonomous vehicles are classified as level 5 by the SAE definitions of automation.

A fully autonomous vehicle operates via multiple software algorithms that utilize data captured from various sensors arranged around the vehicle. Most algorithms are based around various machine learning implementations, allowing the software to adapt to changing environments [15, 16]. Sensors are arranged around the vehicle to create a 360° field of view, which is a necessity for a fully autonomous vehicle to function safely and effectively (*figure 2.1*).

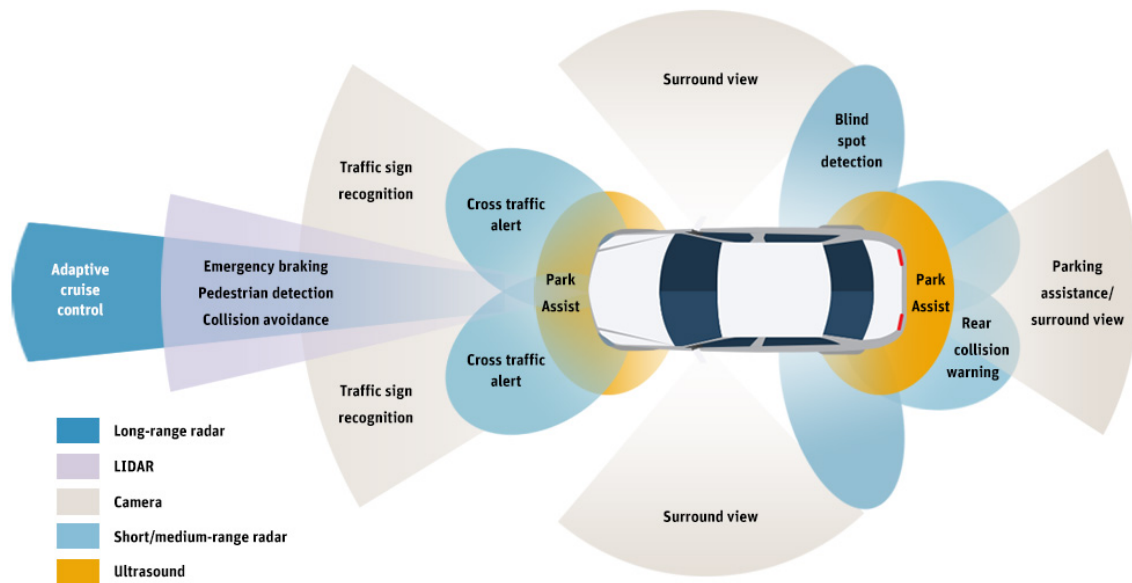


Figure 2.1: Autonomous vehicle sensor layout [17]

Sensor technologies implemented include RADAR, LiDAR, ultrasonic, camera and GPS for locational data. Internal sensors connected through the vehicles CAN bus are also used to monitor internal operations [18]. The data collected through the sensors is processed via multiple embedded systems, which use various algorithms to select the most appropriate manoeuvre or response [18, 19].

2.2 Machine Learning

Driving is a task which is fraught with changes in surroundings and the environment, requiring us as humans to adapt and perform intelligent instinctive decisions. Autonomous vehicles provide a machine with the full responsibility of driving a vehicle. This poses the question, how can a machine perform human-like decisions? The answer is Artificial Intelligence (AI). AI is defined by Boden as "... [a way to make] computers do the sort of things minds do" [20]. The need for AI has grown rapidly in the past decade. This is justified to the problems which computer systems require to solve growing in complexity, requiring a system to become more intelligent [21].

Although AI may seem initially as an emerging field, the conceptualization was around the 1950s when Alan Turing first questioned whether computers and machines could be given intelligence, by proposing the question "Can machines think?". Turnings article presented a proposal on how AI could be developed in a computer system [22]. Shortly following, the development of the first neural network by Minsky and Edmonds was made. The neural network architecture is one of the fundamental concepts used to implement AI and allowed further research to be made in the field

[23].

One of the breakthroughs in AI occurred during the late 90s when IBM developed a computer named Deep Blue. This computer was developed to beat world chess champion Garry Kasparov. The system was developed using various search algorithms which looked at every possible move and calculated the most rewarding. Deep Blue lost the first game to Kasparov. However, due to the success IBM invested in developing iterative versions which ultimately went on to beat Kasparov [24].

Up until the 2000s, all developed systems and research of AI was based on expert systems, which have one purpose only. This is the problem with standard software development where the decision-making logic is programmed explicitly for a specific purpose. The limitation to this is modern complex applications require systems to generalize, adapting the decision-making process based on changes [25, 26]. Machine learning is the technology utilized to eliminate this limitation.

2.2.1 What is Machine Learning?

Machine learning provides a computer system with the ability to learn from past data and adapt to any changes without having to be explicitly programmed [27]. Additionally, it is a sub-field in the continuously growing field of AI, in which it studies statistical and logical models which computer systems adopt to analyse and highlight patterns from data. The importance of machine learning is growing rapidly in current and future systems. This growth is due to factors such as increased complexity, requirement for systems to be adaptable and big data [21, 28].

Although machine learning has a fairly short-lived history, the mathematics and theoretical principles were cemented well before the 1900s. Mathematical principles such as Bayes Theorem, Least Squares and Markov Chains are all adopted in current machine learning models. However, the term machine learning was first conceptualized by Arthur Samuel [29] in 1959, shortly after the introduction of AI and neural networks. Research within the field was slow with the main concepts produced mainly theoretical [30].

Only within the last 15 years has research into machine learning grown, with the introduction of Support Vector Machines (SVMs), Recurrent Neural Networks (RNNs) and Deep Learning catapulting the field. Currently, machine learning is used in an ever-growing variety of applications. One of the most popular currently is for advertisements or product recommendation engines. These algorithms are used to display relevant advertisements and results, based on a users local session data [31]. Without machine learning, the logic for these systems would be confined to specific outcomes providing the same results every time. By applying machine learning it allows the logic to predict outcomes, changing the logic for any changes which may occur. This is an overwhelming

benefit to autonomous vehicles as it allows for the prediction and ultimately appropriate reaction to environmental changes [32, 33].

2.2.1.1 Machine Learning Process

There are five primary classifications of machine learning these being supervised, unsupervised, semi-supervised, reinforcement and deep learning. Although the machine learning process varies it can be generalized to the following stages:

1. Data Collection: This is the start of implementing any machine learning model. This stage is where all data is collected to supply to the model.
2. Data Preparation: This stage is where data optimization and selection occurs. All features are extracted from the dataset and it is formatted ready for the algorithm to consume. This stage and data collection stages are regarded as the most time-consuming [34, 35]. Data is usually split into training and test data with a 20% to 30% split commonly used.
3. Build: This stage is where the machine learning algorithm is developed.
4. Training: This is the stage where the algorithm learns from prepared and formatted data. This is performed using the supplied training data set.
5. Evaluation and tuning: This stage is where the performance of the algorithm is evaluated, with tuning requirements listed to maximize performance. This is achieved by performing various tests from the supplied test data set [36].

It is important to note this is an iterative process. During the evaluation stage, if the model adheres to the desired specification, the algorithm can be deployed. Alternatively, the cycle is repeated until the built model conforms to the desired specification. The diagram below depicts this iterative cycle (*figure 2.2*).

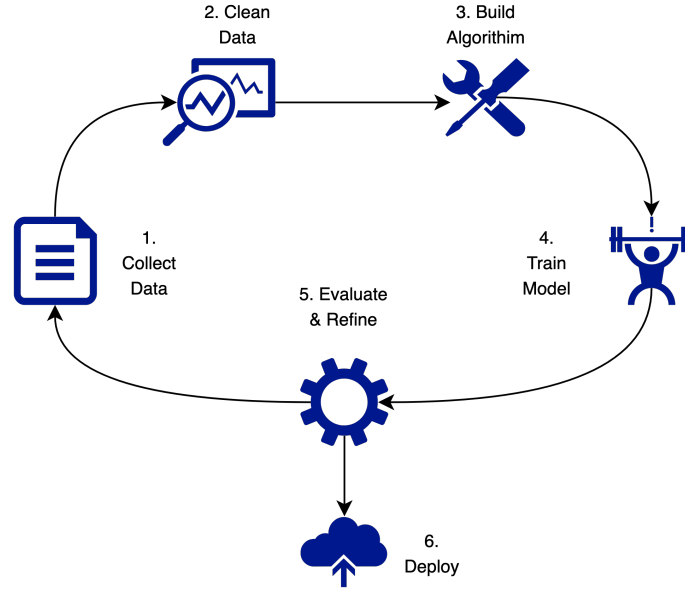


Figure 2.2: Machine learning process [36]

2.2.2 Supervised Learning

Supervised learning is a machine learning model which maps pairs of set inputs and outputs [37]. In this type of model training data is supplied with labelled data. This is data where the answer is already known and is made up of a feature/predictor variable (input) and a target variable (output). Labelled data is usually retrieved from either historical data, experiment data or crowd-sourcing.

The underlying aim of supervised learning is to develop a model which can predict the target variable from the predictor variable. This is achieved using previous inputs and known outputs to predict unknown results [32, 37]. There are two primary types of algorithms used to implement supervised learning, classification and regression.

2.2.2.1 Classification

Classification algorithms are used to determine the category of a response variable based on the predictor variable [38]. Additionally, it is the process of predicting a class from supplied data points. A class is usually referred to as a target variable or labelled data. Classification algorithms are used in autonomous vehicles mainly for pattern recognition and object detection, allowing the detection of road marking/boundaries, other vehicles and pedestrians [39, 40, 41].

Types of classification algorithm include Naive Bayes Classifier, K-Nearest Neighbour (KNN), Support Vector Machines (SVM), Decision Trees, Boosted Trees, Random Forest and Neural Networks. Many of the algorithms are used in a variety of alternative applications to autonomous vehicles. These include speech recognition, handwriting recognition and biometric identification.

2.2.2.2 Regression

Regression algorithms are used to determine the quantity or value of a response variable based on a predictor variable. Response variables in regression problems are quantifiable data which holds some value. It is frequently utilized for predicting and forecasting [32], where it is commonly employed for predictions on pricing scenarios such as the housing market.

In autonomous vehicles, regression algorithms are used for predicting events such as steering angles, which is achieved by generating statistical models of repeatable events in the environment. By adopting these models and comparing variable relations, predictions on appropriate manoeuvres can be made [42].

There are many types of algorithms used to solve regression problems and are commonly driven by three metrics, which represent the number of independent variables, shape of regression line and type of dependent variables [43]. Commonly used algorithms include SVM, linear regression, multivariate regression, logistic regression, polynomial regression and regression trees.

2.2.3 Unsupervised Learning

Unsupervised learning is a model which interprets a data set consisting of just a predictor variable. Unlike supervised learning, the data is regarded as unlabelled as the output/response variable is unknown [44]. This means that there is no historical data set or training data supplied, meaning unsupervised algorithms learn on their own accord.

The goal of unsupervised learning is to model the distribution of data into structured groups based around distinct features. Algorithms use this grouping to achieve interesting structures which can be manipulated to draw out decisions [32, 45]. Unsupervised models are utilized in many applications including facial recognition and online chatbots. In autonomous vehicles, unsupervised learning is used for predictions through real-time video data and to process patterns from raw data [46, 47]. There are two primary types of unsupervised learning algorithms used, clustering and association.

2.2.3.1 Clustering

Clustering algorithms classify/group data based on similarity or dissimilarity [32, 48]. Classified groups are identified as clusters, which can be thought of as a collection of objects whose attributes are similar. Optimal clusters are determined through various measures with the two most commonly used measures being intra-cluster cohesion and inter-cluster separation.

Both measures fundamentally determine the ideal distance between a given set of data points. Intra-cluster cohesion measures how close data points within a cluster are to the centroid, whereas inter-cluster separation is the measure of how far dissimilar cluster centroids are apart from one another [49]. Although calculations can be performed to determine the optimal measures, judgements based on application specification are key [32, 49].

Applications of clustering algorithms include spam filtering and identifying fraud activity, where labelling data may be difficult or costly. In autonomous vehicles clustering is primarily used for image processing in scenarios where low-quality image data is recorded. Clustering takes the image and views the structure of data points so it can ultimately be classified [50, 51].

2.2.3.2 Association

Association is a type of machine learning algorithm which highlights the probability of relationships between variables in a data set [32]. Commonly known as association rules, it functions by two parts, an antecedent and a consequent. An antecedent (if) is defined as an object found within a data set and a consequent (then) is an object with a resulting relationship to the antecedent. Rules are then created from frequently found if-then patterns [49].

A significant factor in association is the strength of the relationship between an antecedent and consequent. To measure relationship strength there are three commonly used metrics, these being support, confidence and lift [49]. Support is the measure of how frequently an item set occurs in the data set and is used to determine how significant a relationship is [52].

Confidence is the measure which defines how frequently a rule is found to be true in a dataset [52]. The limitation with this metric is data items which do not possess a high strength may still compute a significant occurrence, even though the strength of the relationship may be considered weak. To overcome this problem is to use the lift metric [53, 54].

Lift measures the frequency of a consequent concerning the probability of the occurrence of that consequent from a given antecedent [52, 53]. Association algorithms are commonly used in marketing tools such as shopping recommendation engines and for data mining [55].

2.2.4 Semi-Supervised Learning

Semi-supervised learning is a model which employs a combination of both supervised and unsupervised learning. This type of machine learning is typically used in scenarios where a large data set is present [56, 57]. In this type of scenario, it may be unfeasible and too costly to label the whole dataset. Therefore, a portion of the data is labelled leaving the other portion unlabelled.

Supervised learning is performed on the labelled data to achieve the desired response. After this unsupervised learning is performed on the unlabelled portion, using the predictions generated previously to label the remaining data. As the full dataset is now labelled, the model can be trained on this full dataset [45].

2.2.5 Reinforcement Learning

Reinforcement learning is a machine learning model which reviews an input action through a reward [58, 59]. The fundamental concept includes an agent, which has a set state and an environment. The agent interacts with the environment by performing an action, where the taken action is defined through a policy. A policy defines the action which the agent is to take for a given state. This action is reviewed either positively or negatively. The reward is returned as feedback to the agent, as well as the state of the environment [60] (*figure 2.3*). This is how the algorithm learns, as over time it will look to maximize the reward and minimize penalties. This type of learning is commonly compared with teaching behaviour to a child [61].

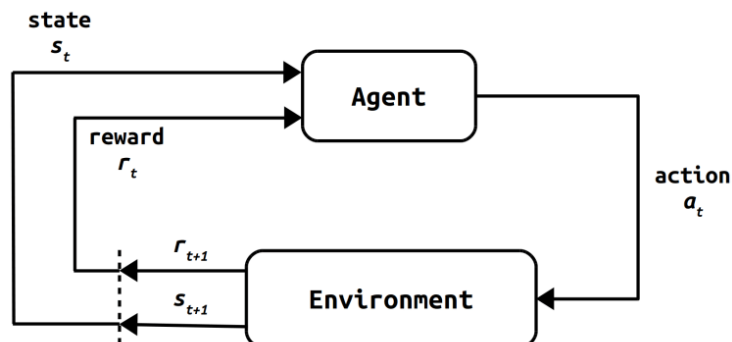


Figure 2.3: Reinforcement learning state diagram [62, 63]

The majority of reinforcement learning models are based off Markov Decision Process (MDP). The goal of MDP is to find the optimal policy for a given state. It does this through a set of environment states, a set of actions for each state, a reward scheme and a transition model [58].

An optimal policy is based off two main goals, these being to gain new knowledge, known as exploration and to maximize the reward, known as exploitation. This trade-off is a big dilemma in the field of reinforcement learning and is highly researched [64, 65]. Balancing the two is key, with ideally the agent collecting a satisfactory amount of information to make the overall best decision for a given environment and state.

The applications of reinforcement learning are increasing with it being used in a diverse number of fields. It is used in robotics, recommendation engines and video games [66, 67, 68]. In

autonomous vehicles, it is mainly used for refinement of steering and learning driving styles, commonly implemented in a deep neural network [69].

2.2.6 Deep Learning

Deep learning is a machine learning technique which encompasses algorithms which are modelled by the structure and function of the human brain. It learns features directly from data and concentrates on extracting high level or complex features from large unstructured data [70].

There is an increasing number of applications which utilise deep learning including, medical screenings, image processing and robotics. Tasks related to autonomous driving are usually solved via multiple machine learning algorithms. However, the recent re-emergence of deep learning looks at simplifying the autonomous driving process, by featuring models such as end to end learning [71, 5]. A question which arises is why is deep learning surpassing traditional machine learning in many applications?

Vargas [72] believes that deep learning provides the advantage of making systems adaptable to new problems. This is due to the use of deep neural networks which are versatile and outperforms traditional machine learning in applications such as image processing and speech recognition [72, 73]. However, this is only the case in situations where large data sets are present [74].

Deep learning algorithms are built from Artificial Neural Networks (ANN). Although neural network architectures are used in other machine learning approaches, deep learning emphasis large neural network architectures which have more than one hidden layer.

2.2.6.1 What is an Artificial Neural Network (ANN)?

An Artificial Neural Network (ANN) is an architecture modelled off the human brain. The human brain consists of billions of small cells known as neurons. A neuron in an ANN commonly referred to as a perceptron is modelled with an input, hidden (processing) and output layer. A single perceptron is the simplest neural network algorithm. It works by dividing an input into binary data (true/false or 1/0). Each input signal is assigned a weight which signifies its priority and importance when processing occurs. The process takes the input and multiplies the value with its corresponding weight [75]. Before this can be transformed into an output a feedback mechanism known as the bias is accounted for. The bias in a perceptron feeds back the resulting output for a given input and weight. It then modifies the weight values until the desired output is reached [75].

The computed value is applied to a function known as an activation function which calculates the output, where it is assigned a 1 or 0 [75, 76]. Fundamentally, the activation function is a

mathematical function which acts as a gate between the input and output [76]. It determines whether the corresponding neuron should be fired. These can be simple functions which turn the input on and off or non-linear functions which are used on complex data such as images. There are many activation functions, including step, sigmoid, tangent and ReLU. These functions are suited to different architectures. The diagram below shows the simple structure of a single perceptron network, where $xn.wn$ are the input signals fed into the network with y representing the output (*figure 2.4*).

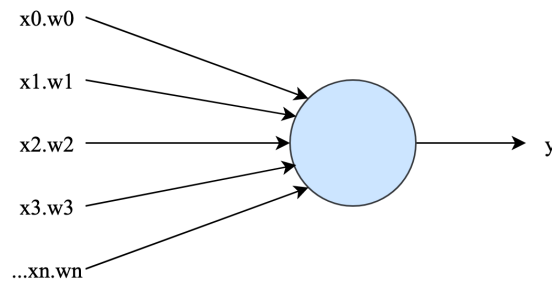


Figure 2.4: Perception Neural Network

The question which arises is how can a simple binary output be used to solve complex problems which may have multiple possible outputs? The answer to this is through implementing multiple perceptron layers. Implementing neural networks with more than one layer is the fundamental building block of deep learning. There are three main classifications of neural networks implemented in deep learning, Feed-Forward, Convolutional Neural Network (CNNs) and Recurrent Neural Networks (RNNs).

2.2.6.2 Feed-forward Neural Networks

Feed-forward neural networks are neural network models where the flow of information through nodes is in a single direction, usually from the input to hidden (processing) layers through to the output. Furthermore, there are no cycles present in feed-forward neural networks, with the output having no effect on the next input [75, 77]. Feed-forward neural networks are commonly used in image and pattern recognition tasks, as the network is suited to model the effect a set of predictor variables has on a response variable.

2.2.6.3 Convolution Neural Networks (CNN)

A Convolution Neural Network (CNN) is a feed-forward neural network which is implemented with at least one layer of convolution in place of basic matrix multiplication [71]. CNN are typically composed of three-layer types, convolution, pooling, and fully connected layers. The first two, convolution and pooling layers, perform feature extraction, whereas the fully connected layer, extracts features and maps it into an output. CNN's are used primarily with image recognition problems and typically use a three-dimensional input (height, width, colour channel). Within in the convolutional layers is where pattern detection is performed [78]. This performed via a feature extractor known as a kernel, which is a small grid of hyper-parameters and is applied to each image position.

A kernel is also referred to as filter when there is a concatenation of multiple kernels. Filters are specified matrix with a set number of rows and columns with it initially holding random weight values. The filter will convolve each part of the input image by a set stride, until all dimensional spaces have been filtered. With this filtering the dot product of each convolve is computed and stored in a matrix. This matrix is passed to the next layer as an input. There may be many convolutional layers which are passed through to other deep fully connected layers where ultimately classification occurs [75]. The more convolutional layers the more complex the patterns recognised [79].

One of the advantages of CNN is that they use a reduced number of parameters compared to standard MLP networks. This is due to CNN's having more sparsely connected layers rather than fully connected layers in an MLP, which results in redundancy, inefficiency and reduced memory requirements [71, 78]. Additionally, the weights of a filter applied at various locations are shared, meaning rather than learning separate parameters for every part of an image, one set is required to learn on. Thus further reducing the total number of parameters that the network needs to learn [71].

Pooling

Pooling layers have the purpose of progressively down-sampling the image, reducing the amount of parameters. There are many types of pooling layers to be considered. In practice, the most common is max pooling (*figure 2.5*). This functions by using a filter and a stride of the same length, which it applies to the input volume. It outputs the maximum number in every subregion that the filter convolves around, discarding all the other values. Pooling allows a CNN to overcome variances in feature positioning within an image, without requiring precise pixel translations and sizes [75].

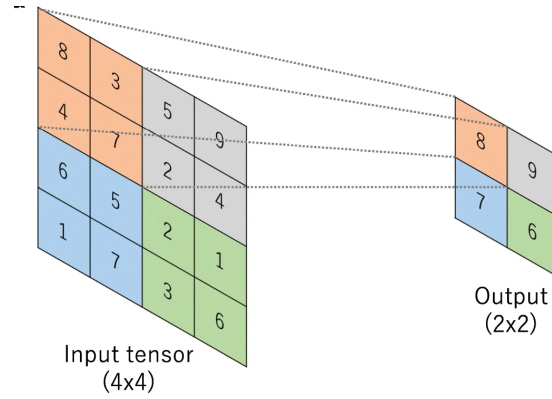


Figure 2.5: Max pooling depiction [80]

2.2.6.4 Recurrent Neural Networks (RNN)

Recurrent neural networks (RNN) also referred to as feedback or interactive neural networks, are neural network models where information flows in both directions through loops [81]. Fundamentally, RNNs perform the same computations for each input with the output, however, the output is dependent on previous results. RNNs can be thought of as having its own memory, with the previous results fed into the input.

Unlike feedforward neural networks RNNs can be supplied with a sequence of predictor variables and output a sequence of response variables. By combining this with the fact RNNs provide direct feedback from the output it makes them extremely useful in applications which require forecasting [75, 82]. Such applications include predictive text and image recognition.

2.2.6.5 Training Neural Networks

Training neural networks are considered a difficult task compared to other machine learning algorithm architectures [83, 84, 85]. This is mainly due to the number of parameters and functions which are required to be fine-tuned, with this increase in complexity with every added dimension [86]. Neural networks are trained by learning the values of the weights and biases. The initial phase is forward propagation, which occurs when the network is supplied with training data which it passes through each neuron. Once each neuron has applied its activation function, an output is produced with the prediction for the data supplied.

The prediction made is then compared to the actual label, which is measured in terms of how close the prediction is. This is done by a loss function which estimates how close the prediction made was from the actual label. The goal of these iterations is to minimise the loss of each prediction with it optimally being as close to zero. To minimise the loss an optimisation technique is used. There are two main types of optimisation algorithms, first and second-order algorithms.

The most popular are first-order algorithms which are gradient-based. One of the most commonly used algorithms is gradient descent [75].

Gradient descent is a technique which changes the weight value of neurons in increments by calculating the gradient (derivative) of the loss function. By tuning the parameters of the model in a direction the loss function can be minimised. To propagate the loss to all parameters of a model a technique known as back-propagation is used. Back-propagation feeds the loss signal into each neuron proportional to the contribution on the output [76]. Thus changing the parameter values of each neuron in the network.

2.2.6.6 Optimization

Optimization is a key stage of implementing a neural network. This is done to maximise the performance of network which is done by fine-tuning a models architecture and hyper-parameters. There are many hyper-parameters which are required to be tuned when optimizing a network. These include the following:

- **Batch size:** The batch size relates to how much of the training data is supplied in each training cycle. This is done to split large data sets which may be inefficient to train. It is important to note that overfitting can occur if the batch size is too large.
- **Number of layers:** The main goal is to keep the architecture as simple as possible whilst maximising accuracy. This is usually optimised manually through trial and error.
- **Learning rate:** This is the number of weights which are updated during training. Usually, it is a small value from 0.0 to 1.0.
- **Epoch:** An epoch is the number of times that the neural network trains on the full dataset. This is optimised by measuring the loss and accuracy of the model with an increasing number of training cycles [87].

An optimisation method which has not been included above is Regularisation. Regularisation is a technique for modifying the network to increase generalisation. It is one of the key methods to reduce overfitting. Regularisation works by penalizing the weight matrices of nodes and is done through various methods. These methods included image argumentation, L1 and L2 regularization, early stopping and Dropout. Dropout is one of the most commonly used methods. Dropout works by selecting random nodes and removes them along with all of their incoming and outgoing connections [87, 71]. After each training iteration the network features different active nodes which results in different outputs. Thus allowing the network to improve generalisation.

2.2.7 Evaluating Machine Learning Algorithms

Evaluation is performed on a test data set which is split from the data set used to train the model. The main purpose of this is to gauge how well the model generalises unseen data [45]. There are two main methods for evaluating a models performance, these are holdout and cross-validation.

Holdout is an evaluation approach where the full data set is split randomly into usually two subsets, training and test data. Commonly the data set is split with training data accounting for 80% and test accounting for 20%. Once the model is trained, the test data is used to make predictions [45]. This approach is efficient and simple to implement. However, as the metrics gained are dependent on the training-test split it may cause variance in results [88].

Cross-validation reduces this variance by providing multiple training-test splits. The most common cross-validation approach is k-fold cross-validation. This approach splits the original dataset into k number of equally divided subsets known as folds. Each k split is used as the validation/test set where the other sets make up the training data [45, 89]. After each k set has been used as a test set the average of the metrics achieved is taken to gauge the performance of the algorithm. Although this method requires higher computational power it is preferred over holdout as it significantly reduces the variance of results achieved [90].

2.2.7.1 Metrics

The metrics which are used to quantify the performance of a tested algorithm are based on the task. General parameters include reliability, robustness, precision and recall. Both robustness and reliability are similar metrics and can be measured using techniques such as Kullback-Leibler divergence probability distribution. This evaluates reliability by using two sets of results from two identical data sets, checking for any differences [91].

Precision and recall are the other two metrics which are closely related. Precision aims to find what proportion of positive identifications were true. It is evaluated as a percentage and can be calculated by dividing the total positive predictions by the total size of the data set. The recall is the measure of what actual positive identifications were correct. This is evaluated by dividing the true positives by the total of true positives and false negatives [92]. Although generalised evaluation metrics exist, model-specific metrics are preferred to evaluate the true performance.

Classification

There are many metrics and techniques used to evaluate classification algorithms, common metrics being accuracy, confusion matrix, logarithmic loss, area under curve and f-measure. One of the

most popular is classification accuracy. Classification accuracy is defined as the number of correct predictions made against all predictions expressed as a ratio. Applying cross-validation to classification accuracy can highlight over-fitting and parameter tuning as various model parameters can be tested on the same explicit dataset.

An additional classification metric is the ROC (Receiver Operating Characteristic) curve which is a graph highlighting the performance of a classification model at all classification thresholds. The curve plots true positive rate against false positive rate [90]. To compute this for various classification thresholds the AUC is used. AUC provides an aggregate measure of performance across all possible classification thresholds. One way of interpreting AUC is as the probability that the model ranks a random positive example more highly than a random negative example. AUC ranges in value from 0 to 1. A model whose predictions are 100% incorrect has an AUC of 0.0, with 100% correct predictions holding an AUC of 1.0 [90].

Regression

There are two main metrics used for evaluating regression models, these being Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE). Both of these metrics measure the average magnitude of errors in a set of predictions. It is the average over the test sample of the absolute differences between prediction and actual observation where all individual differences have equal weight. Root Mean Squared Error (RMSE) measures the average magnitude of the error by taking the square root of the average of squared differences between prediction and actual observation. The main differences between the two metrics are that RSME gives a high weight to large errors due to the squaring factor. This makes it useful in situations where large errors are undesirable.

2.2.7.2 Evaluating Autonomous Vehicles

For a vehicle to drive autonomously there are two main factors to be controlled these are the speed and steering angle. The majority of literature available discuss the measurement of these factors in real-world applications, where they use a variety of stages which includes software testing (testing at code level), simulation testing and real-world testing [93]. As this project will be situated in lab conditions these methods are not suitable for the project.

There are two main tests which would allow the effectiveness of the vehicle to be tested, these are how automated is the vehicle and how well does the vehicle drive. The automation level achieved by the vehicle is defined as the percentage of automation without human intervention the vehicle can withstand over time. There are two similar studies which have tested the automation levels

of their respected systems. Ungreaun [71] and Bojarski [94] use the same automation metric. This metric calculates the percentage of time the car can drive without human intervention. The formula below is used to calculate the automation (*equation 2.1*).

$$autonomy = (1 - \frac{(numberofinterventions)x5}{totaltime(seconds)})x100 \quad (2.1)$$

The most relevant testing procedures performed are in similar studies. However, there is a lack of testing found in the literature for evaluating the speed and steering in regards to how well the vehicle drives. To overcome this a there are several considered methods to evaluate both steering and speed of the vehicle. The speed of the vehicle could be measured by setting the vehicle off around the track with expected speed markers. The predicted speed could be then compared with the actual expected speed.

The steering can be measured in various ways. One of these ways is to use image processing to compute the optimal angle to drive between the tracks and overlay the image with both predicted and optimal angle. This was done by Ungreaun [71] and led to good results which could be analysed efficiently. However, this is believed to take up a lot of development time. Instead, a method proposed is to measure how far the vehicle deviates from the centre of the track, with thresholds set to constitute good driving, average driving and poor driving. Methods to measure this are discussed in *chapter 3*.

2.2.8 Technologies

The starting point to implementing any machine learning algorithm is to clearly define the problem which is trying to be solved. Problem definition can be achieved by following Mitchells [95] machine learning formula where a task (T), a experience (E) and a performance (P) are defined.

Due to the growth of applied machine learning many programming languages have been adopted for development. Python is one of the most popular languages used as it provides the freedom of combining powerful general-purpose functionality with domain-specific scripting languages, such as MATLAB [96]. However, there are many other languages used including, C++, C#, JavaScript, Java and R [97].

Implementing machine learning from scratch is a complex discipline. To aid developers there are many libraries and frameworks which can be used across multiple languages. Currently, common libraries and frameworks include Scikit-Learn, PyTorch and TensorFlow.

2.2.8.1 Scikit-Learn

The Scikit-Learn library is an open-source machine learning library, developed for Python. It has gained recognition and popularity in both industry and academia. This being due to its continuously improving features and extensive documentation. Furthermore, the library features many pre optimised algorithms ready to be implemented in a project. Thus, speeding up the development process. However, due to the extensive selection of algorithms and lack of discrete functions the library lacks customisation options, reducing the scalability and ability to implement deep learning algorithms [98].

2.2.8.2 TensorFlow

TensorFlow is an open-source machine learning library developed by Google. It was initially released around November 2015 with continuous releases made since. It is consumed via an API developed in python and with functions processed in C++. TensorFlow is aimed for both small and large scale machine learning projects with an evince on deep learning and neural networks [98]. Additionally, it provides functions which add a layer of abstraction over the mathematical details of an algorithm's implementation. This is one of the major benefits to TensorFlow as it allows developers to build up complex neural networks and algorithms customising the logic for an applications needs [99].

2.2.8.3 PyTorch

PyTorch is a Python-based library targeted for two main purposes which are, as an alternative for NumPy to take advantage of a systems GPU and for implementing deep learning models [100]. Developed by a team of researchers at Facebook, who initially released the library during October 2016, making it more recent than the likes of Scikit-Learn, TensorFlow and Apache Spark. The library has similarities to TensorFlow in that users can abstract a series of functions to build custom models, rather than importing a pre-built model. However, PyTorch features dynamic computation graphs, whereas TensorFlow implementations rely on static graphing. This is one of the main reasons why PyTorch potentially could be preferred, as it provides the advantage of identifying CPU and GPU usage at run-time [101, 102].

2.2.8.4 Summary

The machine learning library which is considered best suited to the project is TensorFlow. One of the main reasons for selecting TensorFlow is due to it being extensively documented with a large amount of support. Other libraries such as Spark and PyTorch although are equally as powerful they lack the community and support TensorFlow possess. TensorFlow is also selected over Scikit-learn due to its flexibility and greater power. Although Scikit-learn provides out of the box algorithms to use it does not provide flexibility with design and training techniques due to developers being dictated by pre-built algorithms.

2.2.9 Challenges in Autonomous Vehicles

There are various challenges which are highlighted by many publications on the development of machine learning in autonomous vehicles. One of the main challenges is that of accuracy and reliability. Since autonomous vehicles must be 100% reliable the algorithms need a large amount of training to be able to operate safely. Even current autonomous vehicle projects are still being road trained [4].

This issue of safety relates to the ethical issues which surround autonomous vehicles. Goodhall [103] believes that ethical decision making cannot be taught or programmed into algorithms. He highlights that in situations where the death or injury of someone is unavoidable, what ultimate decision will the vehicle make, will it protect the driver or others involved. Howard and Sparrow [104] agree with Goodhall [103] and believe that autonomous vehicles should not be marketed until problems such as these are 100% implemented within the technology.

Another problem which is highlighted in literature is that of completeness in the training data. As autonomous vehicles require to learn from a huge data set, it is extremely difficult to know if the training data is sufficient to train the algorithm in every scenario [105, 106]. The only way this can be achieved is to increase training time. However, who decides on when the algorithm is complete, this again ties into the ethical issues.

2.3 Web and Mobile Applications in Autonomous Vehicles

Mobile and web applications are changing the way people interact with their surroundings, from smart home devices to modern-day cars. The release of autonomous vehicles is expected to expand this interactivity on vehicles even further [107], aiming to enhance consumer convenience and comfort. There are many applications available for current vehicles which are being developed,

both by car manufacturers and third-party companies.

The most common application is connectivity with the vehicles built-in entertainment system, with this being achieved commonly through Bluetooth and occasionally through the internet. It allows users to stream music, hands-free calling and voice-activated controls all through the built-in media system. Additionally, there are highly sophisticated applications currently on the market for vehicles with higher automation [107]. These applications include the ability to provide commands to the vehicle in various ways. These systems will be presented in the following section.

2.3.1 Existing Systems

There are many applications on the market which give the ability to control semi-autonomous vehicles (Level 3+). Tesla is one the largest companies who are heavily investing in autonomous vehicles and other related technologies [108]. Currently, Tesla have developed an application which provides features allowing users to unlock, lock and start a compatible model from the application. This is done through NFC and Bluetooth which authorises an unlock when the phone is within proximity to the vehicle. This feature is being proposed by other large manufacturers including Hyundai [109].

However, perhaps the biggest feature of Tesla's latest released application is the ability to park and summon a vehicle remotely. Summon is one of the most recent features of Tesla's application, it allows control of the vehicle from a set distance away. The application uses Wifi or mobile networking to connect to the system of the car, allowing the vehicle to perform operations such as follow and move to a given location [108]. These current systems do not necessarily provide users with full control of a vehicle, instead allow pre-set commands to be given.

A question which arises is what if full control of a vehicle could be given to a user? Currently, there are systems which provide this feature, however, these systems currently are used for training autonomous vehicles. Kang et al [110] believe that remote control systems act as a safe and economic back up as one user can monitor and control multiple vehicles being trained. What if an application could be developed to allow a user to train a vehicle and provide a consumer with the ability to remotely control a vehicle? The challenges and technologies which could be used to implement this are discussed in the following sections.

2.3.2 Challenges

Current applications developed such as Tesla's summon and park present advantages to user experience regarding both current semi-autonomous vehicles and for future fully autonomous vehicles.

However, some authors present challenges and limitations. Dickman and Burns [?] who produce a user perception study on the summon feature, believe that some users show distrust with the features, as they do not understand the technology behind autonomous vehicles. They believe that people need to be made more aware of the technology.

Another major challenge is that of hacking. With the increase in technology and connectivity, which current and proposed applications include, it can make them targets for hackers. Autonomous vehicles connecting over an application may have personal data which can be retrieved by hackers [111]. Security is a huge problem in the field of autonomous vehicles.

Kang et al [110] present the challenge of network infrastructure believing that high response times and low latency networks are required with many response centres. They also believe that 5G networks need to be designed to provide low latency and high bandwidth for such systems to be successful. As this is an infrastructure challenge it is not considered for this project as the implementation will be used as a simulation rather than a real-life system.

2.3.3 Proposed Application

The proposed system will be a web application which will allow a user to register and ultimately control a selected vehicle remotely, providing the ability to adjust steering angles and speed. The desired system will be web-based. This is considered best suited as the project is limited in time and it is believed a native mobile application would take longer to develop. The system will require a database to store users data, a back-end server and a front-end client. Technologies considered to implement the proposed system are discussed in the following sections.

2.3.3.1 Database

The role of the database in the system is to store user information. There are two main types of database technologies to consider, these being SQL and NoSQL. There are many differences between the two technologies. SQL databases are relational and represent data in the form of tables, whereas NoSQL databases represent data in forms such as key-value pairs, document, graph, and more.

NoSQL databases are considered highly flexible as they are dynamic in operation and do not require a pre-set schema. This becomes advantageous in situations where the data stored changes over time. SQL databases are more structured and require a pre-defined schema which is followed. NoSQL databases are also horizontally scalable, meaning increased scalability is achieved through additional servers, rather than additional processing power which increases scalability in

SQL databases [112]. Since the schema will not change as the database will follow a set design an SQL database is considered most suitable. The relational database management system (RDMS) which will be used is PostgreSQL. This is selected due to familiarity making database development time more efficient.

2.3.3.2 Server

The server will be implemented in the form of a RESTful API which will be used to expose the business logic and database to the web client. There are many different frameworks and technologies which can be used to implement a RESTful API. Popular frameworks include .NET, Flask, Django, Spring and many more. Considered as the most appropriate framework to follow for this project is Spring.

Spring is a Java-based framework which is orientated towards dependency injection, allowing developers to build decoupled components. It allows developers to select various abstract transaction management, data access and object-relational mapping frameworks. Thus making Spring non-invasive, meaning that technology changes will not break the business logic, decoupling key layers within a service [113]. Spring was also considered best suited for the project due to familiarity. This was considered an important factor as the project is emphasised towards machine learning, meaning that time to learn new technologies for application development is limited.

2.3.3.3 Client

The applications client will be implemented using a JavaScript framework. There are many front-end frameworks available. Currently, the three most popular frameworks are Vue, Angular and React. Regarded as most suited for this project is React. The main aim of React is to allow developers to build dynamic UI by developing reusable components and layouts. There are two key features which React possess, these being JSX and the virtual DOM, which allows dynamic rendering. Thus meaning page sections can be re-rendered during set actions, meaning the whole page does not require rendering in specified instances. By taking advantage of these features the performance of React surpasses other frameworks [114].

Although performance factors are advantageous the main reason for selecting React is for component reusability. This potentially means complex UI can be created more efficiently than using vanilla JavaScript [114, 115]. Additionally, React was also selected due to its flexibility as both Angular and Vue are considered stricter frameworks. React is considered by many as a library, providing the flexibility of adding third-party libraries and other frameworks rather than fixating

developers to use set libraries.

2.4 Related Studies

This section presents the related studies found in machine learning and mobile/web applications in the realms of autonomous vehicles.

2.4.1 Machine Learning

There are many related studies which divulge into the field of machine learning within autonomous vehicles. One of the studies found is by Nauman [116] which is titled DrivingMatter: An Autonomous RC Car using Raspberry Pi and was produced in 2018. Nauman [116] looks into the use of reinforcement and supervised learning, applying a combination of both to control the steering of an RC car fitted with a Raspberry Pi. A CNN network was used for steering control with three cameras providing input. Obstacle detection was implemented using an ultrasonic sensor. Reinforcement learning was implemented on top of the CNN architecture which reconsigned signs and traffic lights. The study concludes that results achieved show 73% test and 89% train accuracy. They state that these results are successful, however, results are achieved through testing in stable environments. This means that real environmental factors such as light are not accounted for when testing the effectiveness of the algorithm. However, this would require a large amount of training data and possibly additional image processing.

Another study found is by Thorat et al [117] titled Self Driving Car using Raspberry-Pi and Machine Learning published in International Research Journal of Engineering and Technology in 2019. The study uses a Raspberry Pi combined with a supervised machine learning algorithm. The study presents results which discuss the hardware implications and refer to the camera resolution being the major limitation. Additionally, the study concludes that the speed of the vehicle does not gradually slow down, instead, it just stops when sensing a collision. This can be implemented by equipping the device with distance measuring sensors such as ultrasonic sensors or using Pulse Width Modulation to control the motors.

The next study which will be discussed is by Ungurean [71] which is titled DeepRCar: An Autonomous Car Model and was produced as a Thesis at the Faculty of Information Technology CTU Prague in 2018. The study uses Deep End-to-End learning through a Raspberry Pi. They also developed a web application which implemented various endpoints to control the vehicle in various ways. The study concludes that noise factors affect the quality of the dataset and shows

that the algorithm makes inaccurate choices in certain instances. The study does not control the speed of the vehicle and is always fixed which is another improvement which can be made. The largest criticism of the study is not testing the system in external and real-time lighting conditions.

From the related studies researched it is found that many systems focus on steering control using regression-based models. This project aims to simulate the workings of an autonomous car using machine learning, ultimately controlling the speed and steering of the vehicle. To simulate this a classification problem can be presented labelling images with both speed and steering. The majority of studies found used the Raspberry Pi as the main source of hardware, primarily due to the computational power to stream live video at higher resolutions. Additionally the majority of use CNN architecture, this is mainly used due to its greater performance over other models in image processing. Many systems do not combine distance control with the algorithms applied, which is a reason for the lack of speed control. These areas could be improved upon in this project.

2.4.2 Web/Mobile Applications

The field of mobile and web applications within autonomous vehicles is fairly small compared with the field of machine learning. One of the studies found which is also discussed in the machine learning section is by Ungurean [71]. The study develops a web application to control and train a deep learning neural network following a CNN architecture. The web application includes a display of a live video stream, live data and control over the vehicle. One of the major criticisms is that it does not use any credential login or security, meaning any user who hits the endpoint can control the vehicle. To simulate the proposed system to control a vehicle, credential authorization is a must. This could be done using a JWT authentication token.

Another study found is by George et al [118] which is titled Android controlled Smart Car which was published in the International Journal of Computer Engineering In Research Trends 2015. The study uses an android developed application which is connected to the vehicle through Bluetooth. The results show that the developed system is successful and controls the vehicle in various ways. However, the main criticism is the use of Bluetooth as this limits the range for the system to be controlled.

From the two related studies found there are differences in the communication made between the application and vehicle. George et al [118] use Bluetooth and Ungurean [71] uses HTTP endpoints over the internet. The most viable connection is over the internet as it will not limit the range on controlling the vehicle. The application developed by Ungurean [71] has many considered improvements with authorisation/authentication being the main focus.

2.5 Chapter Summary

This section of the report presented all related background information to the project. From all the research collected a system which fits the objectives of the project has been idealised. The system will be developed using a CNN neural network to process images which will be fitted to a small chassis to simulate the workings of a self-drive car. A CNN is considered best suited for the project due to its high performance in image classification and feature recognition. The network will use labelled training data, with speed and steering labelled as a category making this a classification problem. This was decided as it would simplify the hardware requirements as the control angle and speed of the motors would be fixed at specific classes.

The algorithm will be implemented using TensorFlow. TensorFlow is selected due to its high performance, extensive documentation and ability to build deep neural networks. The network will be trained using a web application which will connect to the hardware on the system. The application will feature authentication and control of the vehicle over HTTP. This is decided as it will allow for remote training of the vehicle.

From related studies, it was found that there are many similar studies. Many of which focus on supervised learning using deep neural networks. Many of the systems focus on solely controlling the steering angle using a regression-based model. Although this is the method which would be used in the real world, this project will look at controlling both the speed of the vehicle and steering using a classification-based approach.

The system will be evaluated with regards to how well the vehicle drives and the automation levels achieved. This will be done by monitoring the vehicle on various tracks. Both the speed and steering will be measured to evaluate the vehicles performance with methods on how this will be performed discussed in the next chapter.

Chapter 3

Methodology

This chapter of the report outlines the overall methodology undertaken. The chapter is separated into two key sections, research methodology and development methodology. Discussed within the research methodology section are the considered philosophies, approaches, strategies and data collection methods. Additionally, methods of measuring the automation levels and vehicles performance is discussed. Following is a section discussing the adopted development methodology, which is based around both traditional and agile approaches. A summary is provided at the end of the chapter.

3.1 Research Methodology

The adoption of a research methodology is vital to any research project. Hilary [119] believes it provides a fundamental platform to initialize any type of research. There are many components which a research methodology encapsulates, including a philosophy, approach, strategies, collection methods and time horizon. To determine an appropriate methodology, the research onion was used (*figure 3.1*).

Developed by Saunders et al. [120] as a visual approach to selecting a methodology, selection is achieved by starting at the outer layer and moving towards the centre. Each layer passed represents the various components to establish a methodology.

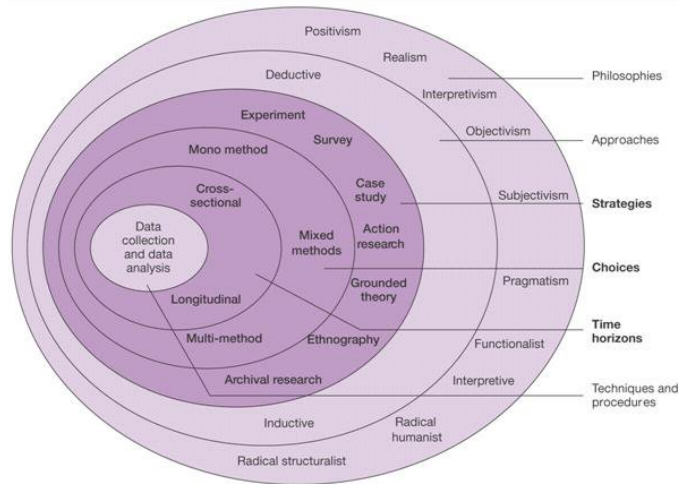


Figure 3.1: The research onion [120]

3.1.1 Philosophy

A research philosophy is a set of assumptions and beliefs which aid in the collection and analysis of data about a phenomenon [120]. Five main philosophies are commonly considered, Phenomenology, Pragmatism, Positivism, Realism and Interpretivism [121]. The philosophy adopted for this project was Positivism.

Positivism is described by Schutt [122] as “... there is an objective reality that exists apart from the perceptions of those who observe it, and that the goal of science is to better understand... [this objective]”. Fundamentally, this means that a researcher views the world relying on statistics and scientific study, revealing the truth on the operation of a field. The philosophy is regarded as most suited as it is highly structured, works effectively with quantitative data sets and is experimentation orientated [123].

3.1.2 Approach

A research approach is the guide and procedures which outline methods of collection, analysis and interpretation of data [124]. There are two primary types of research approaches, these are inductive and deductive approaches. An inductive approach is where a new hypothesis is formulated as a result of the research conducted [125]. This approach allows for flexibility, as the researcher retains no requirement to follow pre-determined information.

Alternately, a deductive approach concentrates on the analysis of a hypothesis using conclusions made from research [126]. A deductive approach does not possess the flexibility of an inductive approach. However, it is structured and focused. As the project required to deduct hypotheses for

test criteria from related studies, a deductive approach was considered most suitable.

3.1.3 Strategies

Research strategies provide a researcher with a plan on the methods used to collect and analyse data [127]. Two strategies were selected, these being document analysis and controlled experimentation.

3.1.3.1 Document Analysis

Document analysis was primarily used during the research stages. Sources utilized consisted of journal articles, books, conference proceedings, online articles and approved websites. The sources were utilized to find related studies and background information to generate system requirements and metrics for experimentation. This strategy works effectively with the deductive approach, as literature could be reviewed to find comparable studies where results achieved could be compared.

3.1.3.2 Controlled Experimentation

Controlled experimentation was used to evaluate the effectiveness of the developed system. This strategy was used for primary data collection. Controlled experiments are experiments where the researcher is in control of independent variable(s) and observes dependent variables(s) [128]. For this project, independent variables are the track the vehicle is tested on, where the speed and steering represent dependent variables monitored.

To perform a successful experiment, there must be some form of planning and/or design [128, 129]. Experiments were developed with informal experimental design performed before each experiment was carried out. The primary consideration for the experiments was to keep the data collection valid and reliable. This was achieved by keeping independent variables equal on each track run, repeating each experiment and ensuring measurements taken were accurate by using the same metrics. A summary of the experiment plan can be found in *Appendix A*.

3.1.4 Time Horizon

The time-horizon highlights the time plan which a project follows [120]. There are two primary types considered, longitudinal and cross-sectional. Longitudinal studies concentrate on investigating a situation over an extended period. This is achieved by continuously investigating and

collecting data over-determined time intervals. Studies are performed for years to highlight any changes in dynamics or perceptions impacted by time [130, 131].

Cross-sectional studies concentrate on investigating a situation within a specified point in time. Ordinarily, the study is used in research where there are multiple variables involved [132]. Rindfleisch et al. [133] believe that cross-sectional studies are best suited to studies which rely on a diverse array of measurement variables. As this project was limited to time and included multiple parameters, a cross-sectional study was adopted.

3.1.5 Data Collection

Data was collected from both primary and secondary sources. Secondary data was collected from all sources which were found during the research stages of the project. The purpose of collecting secondary data was for providing a deep background to the subject, metrics to measure the system and to compare results achieved.

Primary data was collected from experiments performed on the developed system. The key tests performed were to test the automation level of the vehicle and how well the vehicle drives. Additionally, tests were performed to evaluate the effectiveness of the implemented machine learning algorithm. The effectiveness was assessed by measuring common machine learning parameters found through research, see *chapter 2, section 2.9*.

All tests were performed on trained tracks and two unseen tracks to measure how well the model generalizes. The data collected was solely quantitative data which aided in determining how well the vehicle drives. Data collected was displayed in the form of graphs and charts, where analysis was performed.

3.1.5.1 Automation Levels

Automation level tests considered were based on metrics discussed in *chapter 2*. The method employed was to test the automation by measuring how much human intervention was required. To perform this the implemented vehicle was set to drive around the track autonomously for 20 minutes.

To keep the experiment fair and valid it was repeated three times on each track. An average was taken to present the results. The time was measured precisely, with the time paused once the vehicle had completed the track. The vehicle was placed at the starting line of the track. To keep the starting position valid it was marked. If the vehicle drove off the track, it was repositioned centrally on the track.

3.1.5.2 Vehicle Performance

The vehicles performance was measured through testing the speed and steering on trained tracks and two unseen tracks. The speed was measured by setting boundaries up at intervals around the track with the expected speed. These expected speeds would be compared with the predicted speed. The vehicle was positioned at the start where it was set to drive autonomously. If the vehicle drove off the track it would be placed in the optimal position near the incident. To keep this test fair it was repeated three times with averages recorded. A consideration made was that of boundary positioning. These could not be placed near the track as it would disrupt the predictions for the model, producing invalid results.

The steering testing was performed by measuring how far the vehicle diverted from the centre line of the track. One of the considered methods was to add a centre line to the track for testing. However, this line would have affected predictions making results invalid. Another method considered was through image processing. This could be achieved by recording each predicted image and overlaying it with the computed optimal angle. This was considered costly in terms of time. The method used to evaluate steering was using a pause process where the system predictions were paused allowing time to take a reading.

To keep this valid the tests were performed three times on each track. Measurements were taken in cm, smaller units were considered to be insignificant with regards to the size of the track and vehicle. Another consideration was that of track width and layout. The track was insured to be the same width all around, so the optimal centre line was easier to compute and more reliable. The track layout was additionally considered, with bends created with sharper edges to allow an accurate reading, as natural curvatures are more difficult to measure from. This was done to ensure maximum accuracy.

3.2 Development Methodology

A development methodology is defined as the process of dividing a project into stages to solve a problem [134]. There are two primary classifications of methodology in software development, traditional and agile approaches. Traditional methodologies are incremental processes where each stage is distinct. The most commonly used is the waterfall model. It divides the development process into five stages which are requirements analysis, design, implementation, testing and maintenance.

Traditional methodologies provide a structured and rigid approach. These properties make it beneficial to projects which have numerous artefacts to be developed and are large scale. However,

this structured nature means they lack flexibility [135]. Many modern-day software projects have continuously changing requirements, which makes traditional methodologies disadvantageous in these projects.

Agile methodologies are being adopted in many modern software projects as it resolves the flexibility issues which come with traditional methodologies. Agile approaches combine iterative and incremental cycles through various stages. There is a focus on continuous requirement gathering, adaptability, team collaboration and quick delivery of functional software [136].

As this project is constricted in time, adopting a full methodology would be disadvantageous. Therefore, a methodology is created by combining both traditional and agile principles. The methodology was decided to consist of four distinct stages, these being requirements analysis, design, implementation and testing.

3.2.1 Requirements

Requirements gathering and analysis is an integral part of software development, providing purpose and clarity for features and functionality. Many studies regard requirements gathering highly, with indications of 40% to 60% of all defects discovered are due to errors made during this stage [137].

Agile methodologies predominantly rely on the generation of requirements through user stories. User stories are a descriptive way of capturing a software feature, usually written with collaboration and communication with a customer [137]. Typically formatted in a few sentences such as “As a ... I want to ... so that ...” [138]. This is an iterative process where requirements are collected through various development cycles.

Traditional requirement gathering processes require less customer interaction and focus on formalized documentation. Typically described through use cases combined with a use case diagram, which is part of the Unified Modelling Language (UML) [139]. A use case diagram describes the interaction between an actor (a type of user) and a part of the system.

For this project, a traditional approach will be employed with use cases generated from an initial list of system requirements. This approach is selected as requirements are generated from the aim and objectives, as there is no customer base to generate the requirements. This is one of the primary reasons why an agile approach was not adopted.

3.2.2 Design

The design stage is where the full systems architecture is blueprinted. In an agile approach, design is continuous with changes occurring as new requirements are gathered. Formal design

documentation although exists, is limited and commonly achieved through scribbles and small prototypes. Traditional approaches generally follow UML to draw out the systems architecture [139].

The projects requirements were fixed towards the objectives with no significant requirement changes. Therefore, a structured design approach was undertaken with a focus on UML. Additionally, as the project was limited in time, it was beneficial for a full system blueprint to be followed. Both static and behavioural UML documentation were employed.

A package diagram was produced for depicting the systems architecture. The system features many components which are separated across a web application and a vehicle system. Therefore, it was regarded that alternative static diagrams such as class diagrams, would be overly complex [140]. Additionally, it was selected to highlight all key components and package dependencies [139].

Both activity diagrams and sequence diagrams were created to represent the behaviour of the system. Many of the interactions are through HTTP requests, with a sequence diagram best suited to represent these interactions. This allowed each requests subsequent events to be represented in order concerning time [140]. Activity diagrams were selected to represent key component logic. It allowed important logic to be represented, making the designed logic easy to replicate during implementation.

3.2.3 Implementation

The implementation stage is where the designed system is developed. Within an agile approach implementation is iterative, where design and requirement changes reflect what is implemented [136]. Traditional approaches are incremental with formal design documentation followed strictly.

Due to the project featuring two core components, it was decided to split the development stages into manageable cycles. This was achieved by splitting up use cases into tasks. Although not followed strictly, Kanban was adopted to manage tasks allocated in each cycle. Each cycle was allocated two weeks with a total of six cycles allocated in the plan. The final two cycles were devoted to training the vehicle. An initialization cycle was also performed which was devoted to ensuring all resources were available before development commenced. By adopting this process, it allowed use cases to be prioritized and allowed specific tasks to be focused on during each cycle.

3.2.4 Testing

The testing stage is where the implemented software is tested concerning performance and functionality. In agile approaches, testing is performed conventionally using a testing methodology

such as Test-Driven Development (TDD), where tests are written and performed in line with development [141]. Traditional methodologies rely on the complete system to be developed before extensive testing is performed.

The majority of performance testing was performed during the experimentation stages, where the implemented system was tested. However, where required a TDD approach was undertaken to test the functionality. This allowed unit tests to be written against developed logic. Additionally, manual testing was performed on each feature, allowing bugs to be resolved before the next cycle.

3.3 Chapter Summary

The research methodology undertaken for the project consisted of a positivist philosophy with a deductive approach following two strategies, document analysis and controlled experimentation. The project relied on both secondary and primary research. Secondary research was collected from books and journals, whereas primary research was collected from experiments performed on the developed system. Experiments were performed to test the automation level of the vehicle and how well it drives through custom-built tracks. Additionally, the machine learning algorithm is evaluated with standard metrics. Data collected was ensured to be valid and reliable by repeating and controlling factors affecting the experiment.

The development methodology adopted combined both traditional and agile approaches. The adopted approach consisted of four stages, requirements, design, implementation and testing. Requirements were generated through use cases, depicted through a use case diagram. The design stage comprised of formal design documentation through UML diagrams. The implementation stage was split into two week-long cycles, where the design was split allowing development to be manageable. Use cases were split into tasks allocated to each cycle. The majority of testing was applied once the system was developed. With forms of manual and unit tests written following a TDD approach. This allowed bugs to be minimized.

Chapter 4

Design

This chapter of the report overviews the design process. There are three key sections in this chapter, requirements, hardware design and software design. All related design choices are discussed within each relevant section, with key diagrams presented. A summary of the final designed system is presented at the end of the chapter.

4.1 System Requirements

To be able to measure the automation levels and gauge how well the vehicle drives, the system must be autonomous. The proposed system was a small-scale simulation of an autonomous vehicle which utilized a machine learning model. After collating all research it was decided to focus on a system which could navigate through various marked routes, adjusting the speed and steering angle via a Convolutional Neural Network (CNN). Supervised learning will be performed, where the speed and the steering angle will be labelled from images captured from a front-facing camera. Additionally, the system was proposed to be controlled and trained remotely through a web application.

The systems requirements were generated around the projects objectives and what was required to be measured. A design specification was produced to outline high-level requirements. This specification is presented as the following:

1. Vehicle system:
 - 1.1. Must be able to drive autonomously through set tracks so the automation levels and vehicles performance can be measured.
 - 1.2. Must feature a chassis with both a DC motor for speed and a servo motor for steering control.
 - 1.3. Must be fitted with an on-board computer system to manipulate the fitted motors.

- 1.4. Must be equipped with a camera to capture images for training data and for feeding real-time input to the CNN.
 - 1.5. Must be able to control the system and train the model remotely from the web application.
 - 1.6. Must be able to collect training data labelling the images with the set speed and steering angles.
 - 1.7. Must be able to store and load training data locally on the on-board computer.
2. Web application:
- 2.1. Must allow an unregistered user to create a user account.
 - 2.2. Must allow all registered users to log in and logout.
 - 2.3. Must allow all registered users to register a vehicle.
 - 2.4. Must allow all registered users to set the vehicle to auto control mode.
 - 2.5. Must allow all registered users to manually control the vehicle by setting the status, speed and steering angle.
 - 2.6. Must allow an administrative user to collect training data for the vehicle remotely by setting the speed and steering angle.
 - 2.7. Must allow an administrative user to train and save a model remotely.
 - 2.8. Could allow an administrative user to set the training parameters.
 - 2.9. Could allow all registered users to update their account information.

The system will be designed around three user types, an unregistered, registered and administrative users. An unregistered user will be able to create a user account to utilize the resources. Registered users will be provided with the ability to register a vehicle, control a vehicle manually and set the vehicle to drive autonomously. This functionality will be provided to administrative users with additional functionality, including the ability to train the CNN and to manage user accounts. Presented below is a use case diagram which depicts the requirements and interactions of each user type (*figure 4.1*). Use case descriptions are presented within *Appendix B*.

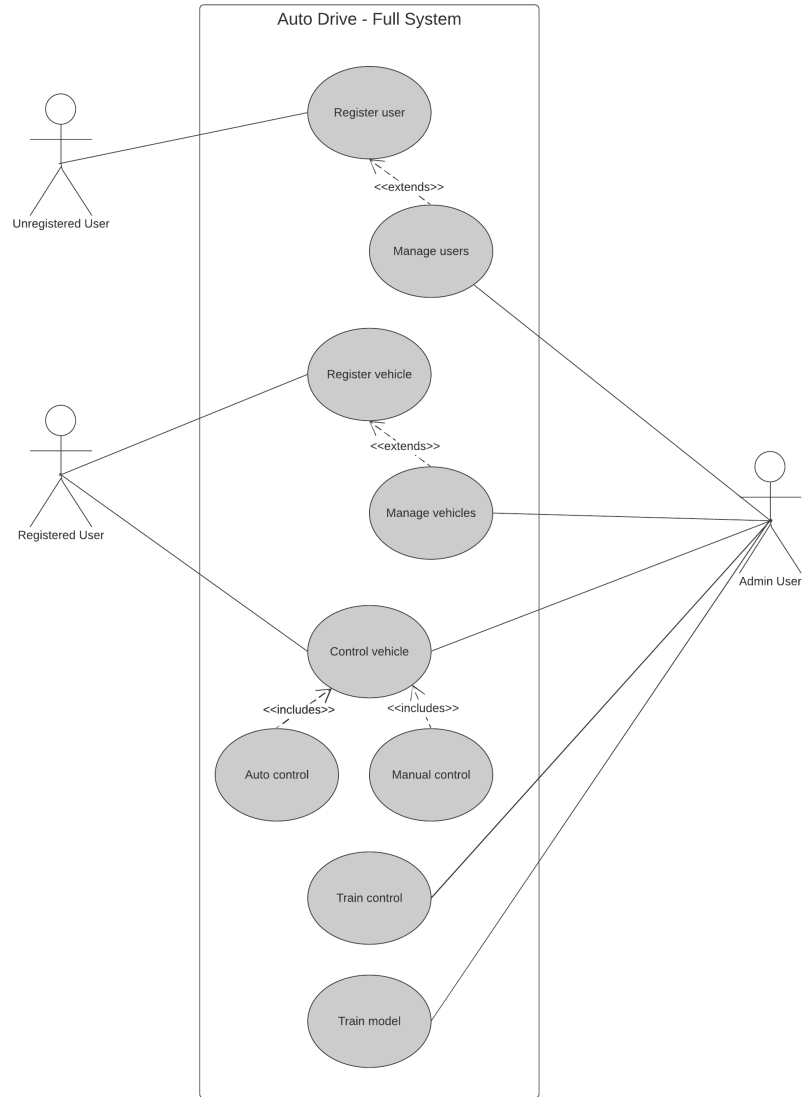


Figure 4.1: Use case diagram based around the full system

4.2 System Overview

The system consists of two main components a vehicle system and a web application. The web application consists of a database, an API and a client. The role of the database is to primarily hold user account information, fulfilling the user registration and login requirements. Data from the database is manipulated through the API, which is consumed by the client. The client provides a UI exposing functionality to the user.

The vehicle system consists of a camera, machine learning scripts, API and motors. Both the machine learning scripts and API are implemented on an onboard computer system. The camera provides input to the CNN for predictions to be made and allows for the collection of training data. The purpose of the API is to expose the various CNN functionality and to provide remote

control of the vehicle, manipulating the motors over HTTP calls. The diagram below shows a block diagram of the system (*figure 4.2*).

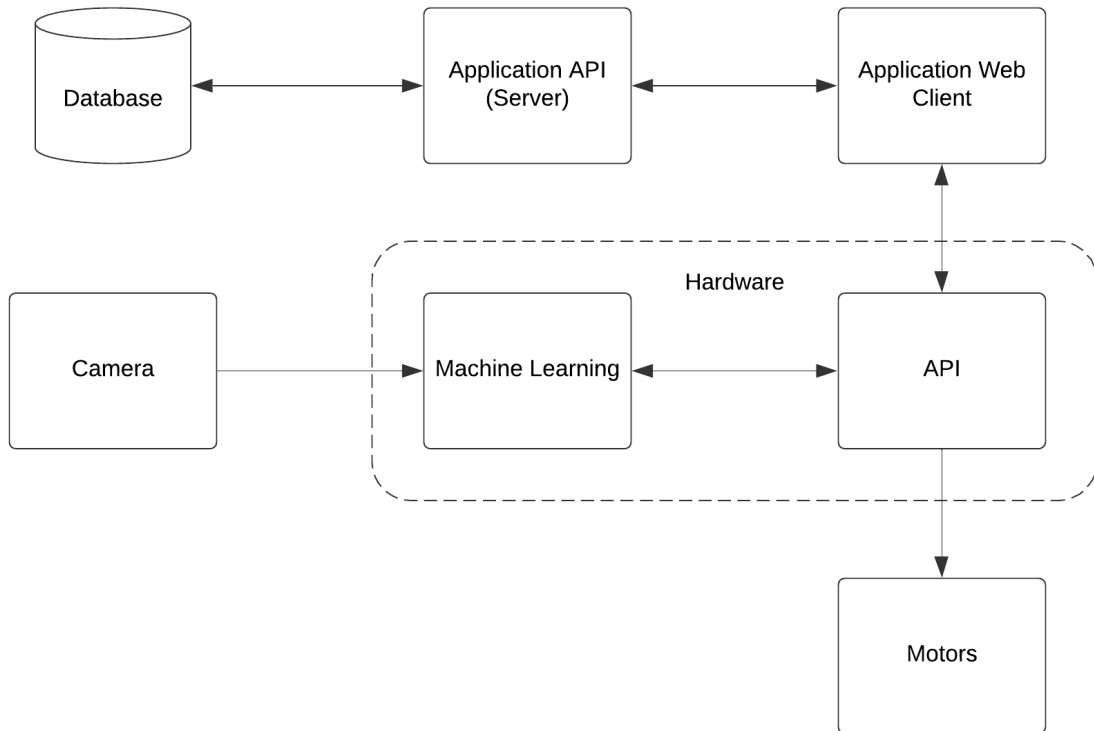


Figure 4.2: Initial system block diagram based around the full system

4.3 Hardware

The system required four main hardware components, these being a chassis, onboard computer, camera and motor driver controller. All required hardware components were selected considering the requirements. The selection process consisted of researching suitable components and comparing their specification.

4.3.1 Onboard Computer

The onboard computer was required to run the machine learning TensorFlow scripts, as well as the API to expose the control functionality. Two main considerations for the computer were computational power to process the cameras input frames and low in cost. Additional requirements include General Purpose Input Output (GPIO) pins and the ability to generate Pulse Width Modulation (PWM) waveforms.

There were two main computer systems considered, the Raspberry Pi (RPi) and Arduino. The key difference between both systems is the Arduino is fundamentally a microcontroller, whereas the RPi is a computer. Due to the Arduinos architecture being based around a single microcontroller, it has no operating system overhead making written software easier to execute. However, it lacks important features such as an absence of built-in network connectivity, which was considered significant as the vehicle is manipulated through an API.

The RPi was regarded most suitable and compared favourably in processing power, memory and connectivity. The additional power was significant as it could allow the processing of the image feed more efficiently. The Raspberry Pi 3 B+ was used in this project (*figure 4.3*).

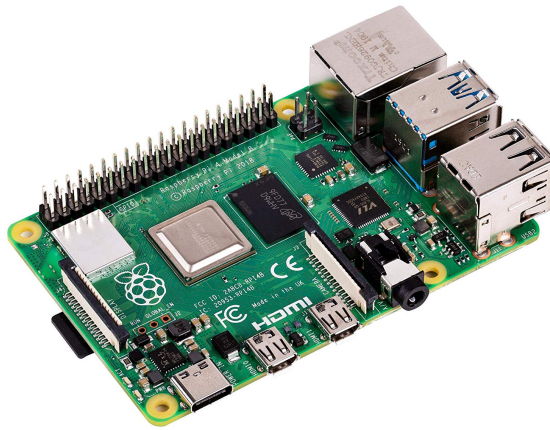


Figure 4.3: Raspberry Pi 3 B+ [142]

4.3.2 Chassis

There were three key considerations for selecting a chassis. These were size, DC motor specification and feature of a servo motor. The servo motor was one of the most important factors as it would allow full control of the steering angle using PWM.

The chassis selected was the Seeed Studio robot chassis (*figure 4.4*). This chassis included a 4.8V to 7.8V servo motor and a 6V to 12V DC motor with an operating current of 500mA. Additionally, the chassis dimensions allowed the onboard computer to be fitted. Although other chassis boasted similar specifications this model was a cost-effective alternative.

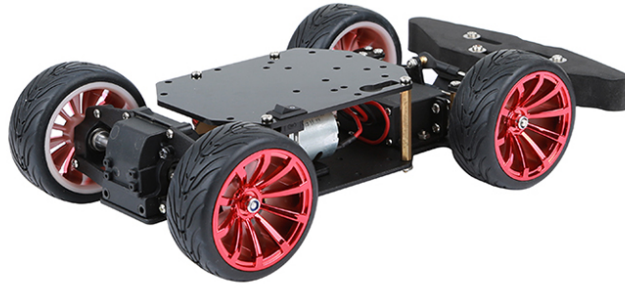


Figure 4.4: Served Studio robot chassis [143]

4.3.3 Camera

The camera was required to provide the CNN with input data for predictions and collecting training data. Key considerations for the camera were, size and Field of View (FOV). The camera required to be a suitable size to fit the selected chassis and feature a large FOV, allowing the system to collect a wider range of data. The frame rate for the camera was considered unimportant as the image feed would be processed and scaled.

The camera selected was the Fish Eye camera module (*figure 4.5*). This camera was selected due to its low cost, size and wide viewing angle of 160° . Additionally, the camera was compatible with the RPi via a dedicated 15 pin MIPI Camera Serial Interface (CSI).



Figure 4.5: Fish Eye camera module [144]

4.3.3.1 Motor Driver Controller

The motor driver controller was required to control the speed and direction of the DC motor. There were two main methods considered for implementing motor control, these being to implement a custom H-Bridge circuit or to use an Integrated Circuit (IC). The method selected was to use a motor driver IC as this project was limited in time, with custom circuitry taking up development

time.

Before a suitable motor driver IC could be selected the motors peak stall current needed to be measured. The stall current for the DC motor was 1.5A with the operating range sitting at 2.5V to 6V. The motor driver IC selected was the TB6612FNG (*figure 4.6*). This IC was selected as it can output up to 3.2A, features a motor operating voltage of up to 15V and was cost-effective.

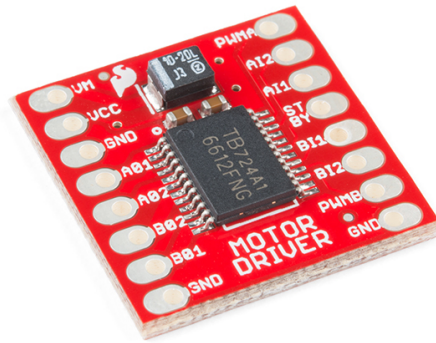


Figure 4.6: TB6612FNG motor driver IC [145]

4.3.4 Hardware Schematic

Presented below is the final hardware schematic (*figure 4.7*). The hardware features two power supplies, a 5V USB battery to power the RPi and four AA 1.5V batteries connected in series to power the motors. Both the motor driver IC and servo control pins are connected to alternative PWM channels on the RPi. The motor driver IC logic pins are connected to digital control pins, allowing the polarity of the motor to be changed. Additionally, the camera is connected to the RPi's CSI.

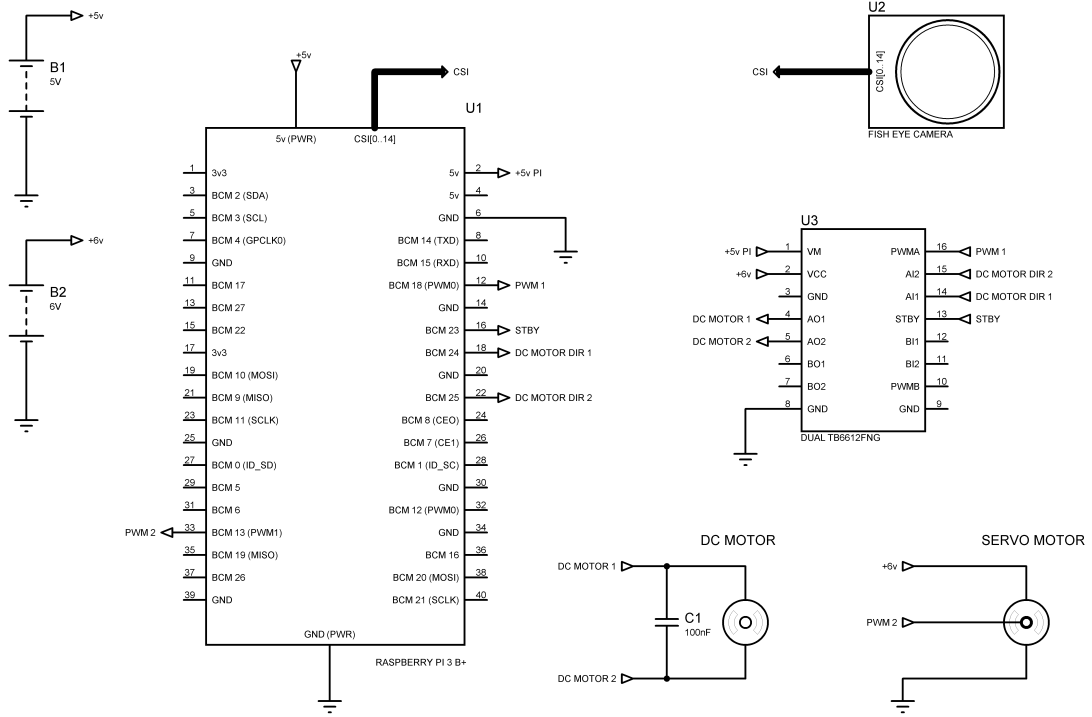


Figure 4.7: Final hardware schematic

4.4 Software

The systems software incorporates two components, the web application and the vehicle system. The web application includes a database, an API and a client, whereas the vehicle system incorporates an API and machine learning scripts, which run on the discussed hardware.

4.4.1 Web Application

The web application was designed by adopting a three-tier architecture. One of the primary reasons this was selected is it allows each component of the system to be worked on independently. This was considered significant as it reduced dependencies between tasks, leading to a more efficient implementation process. Furthermore, it allows the development of loosely coupled systems with high cohesion, making a system easier to test and maintain [146, 147]. The individual layers present are a PostgreSQL database (data layer), Spring API (application layer) and a React client (presentation layer) (*figure 4.8*).

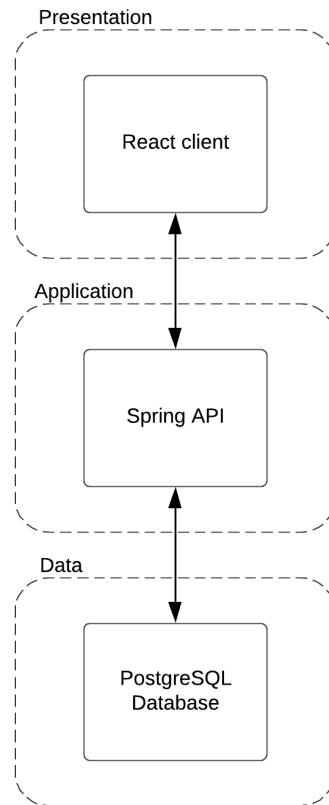


Figure 4.8: Web application three tier layered architecture overview

4.4.1.1 Database

The purpose of the database in the system is to hold user account and vehicle information. A key consideration for the database design was normalization. The database was decided to be normalized to third normal form (3NF). This was decided to improve efficiency, data integrity and consistency [148, 149]. 3NF was regarded as optimal for this project as over-normalising the database can reduce efficiencies in terms of insert and update speeds, as there is an increased number of tables [150]. An Entity Relationship Diagram (ERD) was created to show all database entities and relationships (*figure 4.9*).

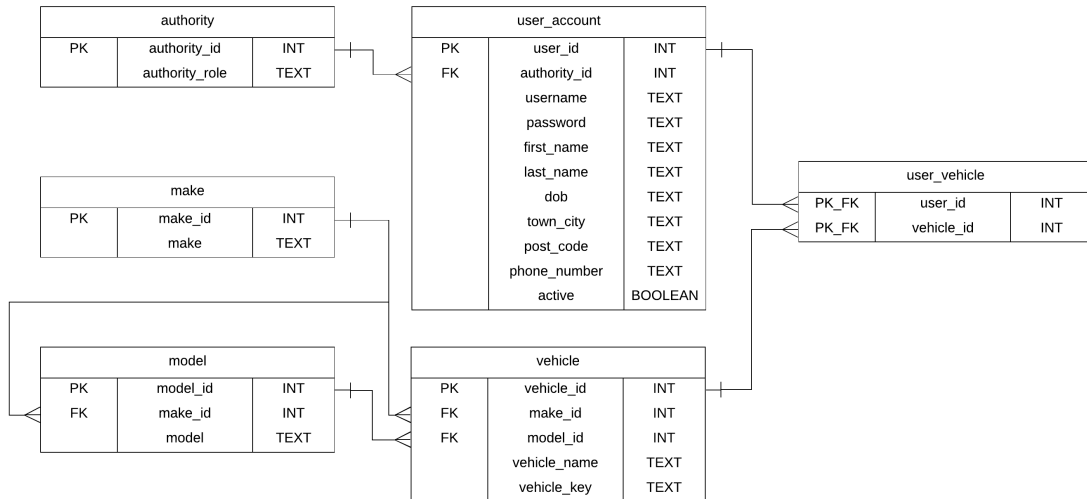


Figure 4.9: Entity relationship diagram of the web applications database

4.4.2 API and Client

The purpose of the API is to manipulate and expose data from the database. The client is to provide users with a UI to manage which data is manipulated and exposed. Additionally, the client will expose the functionality of the vehicle system. The following sections are key areas of design undertaken for both API and client.

4.4.2.1 Authentication

One of the main design considerations required for the web application was user authentication and session management. The problem initially was the system relying on a REST API as the server, meaning no session could be held. This was to comply with the statelessness property of a REST API. To overcome this, it was decided to implement a JSON Web Token (JWT) as the authentication and session management mechanism.

The sequence diagram below depicts the sequence of actions for a user attempting to log in (*figure 4.10*). Authentication functions by the client requesting a JWT. Once a request is received, the API generates a new JWT, providing an existing user name with a matching password is received. This token is sent back to the client where the client stores the JWT and uses it in requests.

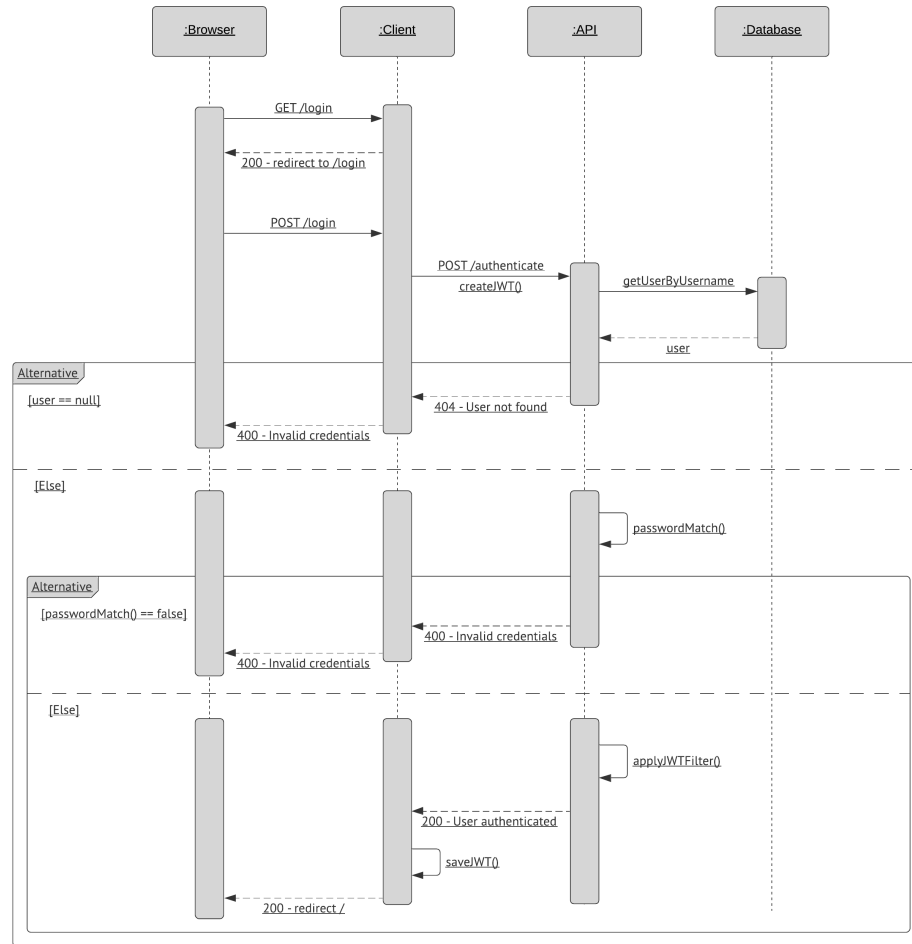


Figure 4.10: Sequence diagram for web application depicting JWT authentication

4.4.2.2 Validation

The majority of validation logic will be handled by the API. The client will handle validation by checking the HTTP status code of the response. It will subsequently either perform a redirect to the appropriate error page or will render the page with error messages.

The sequence diagram below overviews the validation for registering a vehicle (*figure 4.11*). Initially, the client performs a POST request with the vehicle details in the body. Thereafter, the API firstly validates the JWT token sent in the request header. A 401 is returned from the client redirecting to the login page if the JWT is invalid. Alternatively, if the JWT is valid each vehicle field is validated, with a 400 returned if a field fails validation. The client handles this by rendering error messages. A 200 is returned if the vehicle has successfully been created with the client performing a redirect.

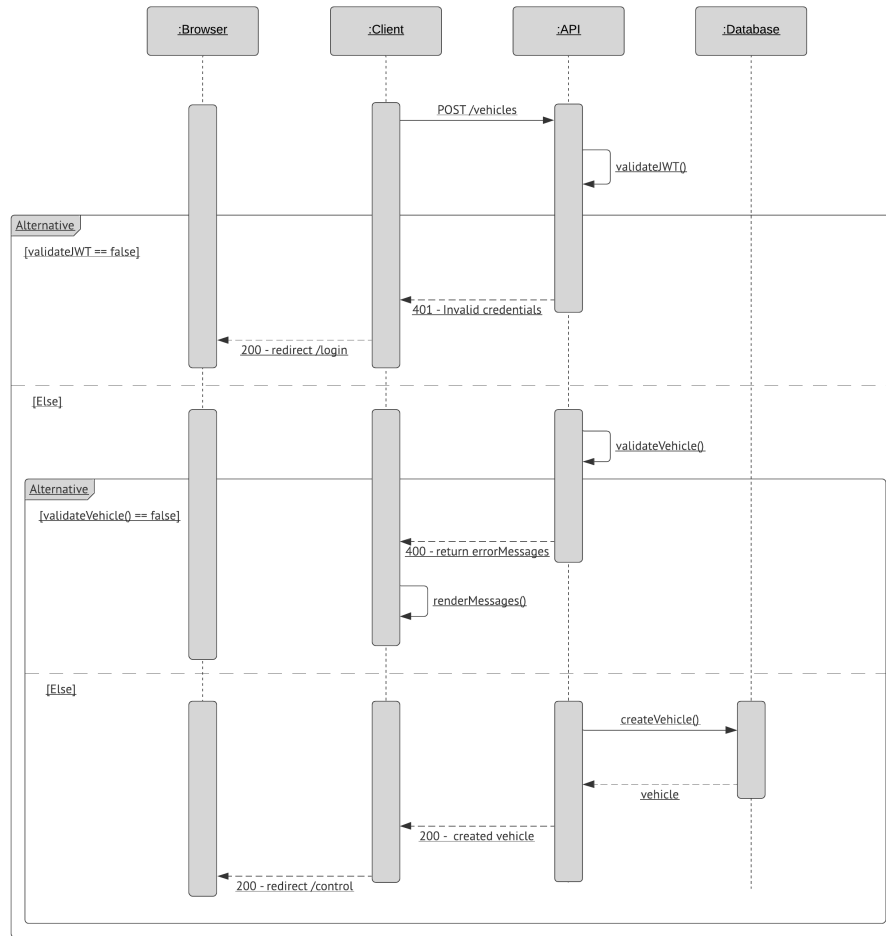


Figure 4.11: Sequence diagram for highlighting validation for registering a vehicle

4.4.3 Vehicle System

The vehicle system incorporates three main software components, these being configuration scripts, machine learning scripts and an API to expose the functionality to the client. Logic for these components will be separated in individual modules, allowing the machine learning scripts to be tested without being reliant on the API. The following sections discuss the key design considerations for the vehicle system.

4.4.3.1 Manual Control

One of the requirements of the vehicle system is to allow both administration and registered users to remotely control the vehicle. This is achieved by remotely controlling the vehicle system from the client via a control dashboard. The sequence diagram below shows the workings of the manual control functionality (*figure 4.12*).

Initially, the client performs a POST request with the status, speed and steering angle to the vehicle API. Each field is validated with a 400 returned if validation fails. If all fields are valid then the PWM duty cycle is set for the speed and steering. Once successfully set a 200 is returned.

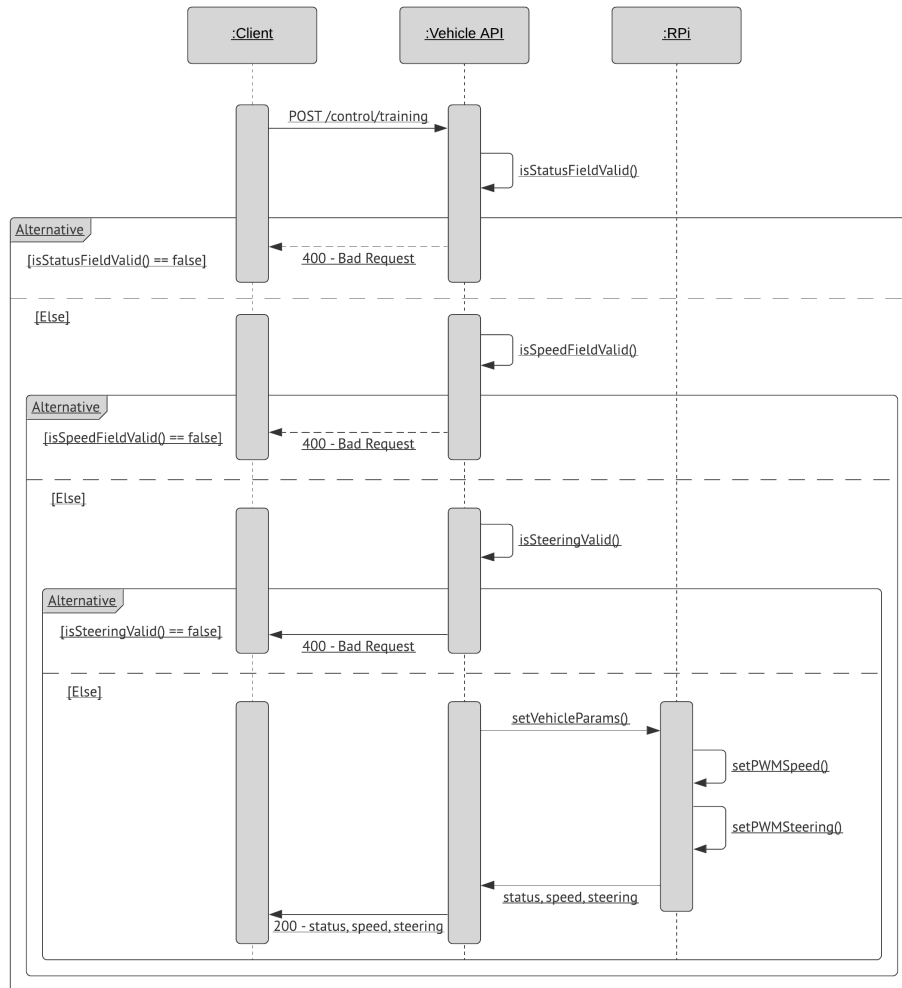


Figure 4.12: Sequence diagram for vehicle system depicting manual vehicle control

4.4.3.2 Training Control

The training control requirement is to allow an administrative user to collect training data to train the CNN. For this, a sequence of events with similarities to the manual control function was proposed. However, an image will be captured and labelled with the set speed and steering.

The activity diagram below shows the training control operation (*figure 4.13*). Initially, each parameter is validated with an exception being raised if any parameter is invalid. Alternatively, if each field is valid an image is taken by the camera and stored on the Raspberry Pi. The speed and steering are set to allow the vehicle to manoeuvre with a short delay before the duty cycles

are reset.

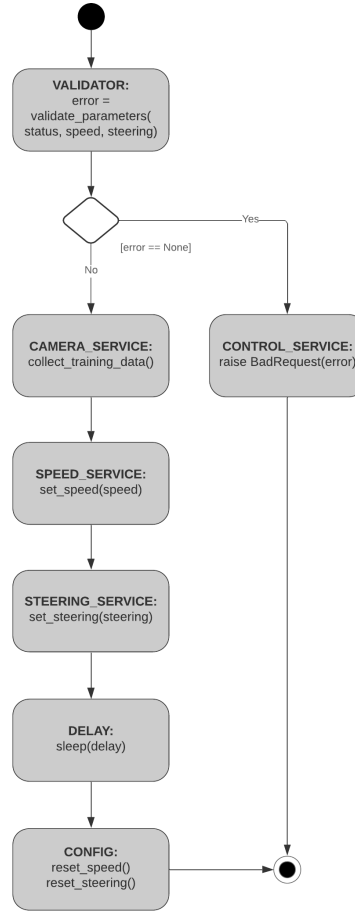


Figure 4.13: Activity diagram for vehicle system depicting training vehicle control

4.4.3.3 CNN Architecture

Many CNN architectures have been reviewed and been well-received across various image recognition applications. Notable architectures include VGG net, ResNet, and Dense net [151, 152, 153]. One of the most notable architectures used in the field of autonomous vehicles is Pilot net designed by NVIDIA presented in their End to End Learning for Self-Driving Cars paper [94]. Shown below is the Pilot net architecture (*figure 4.14*).

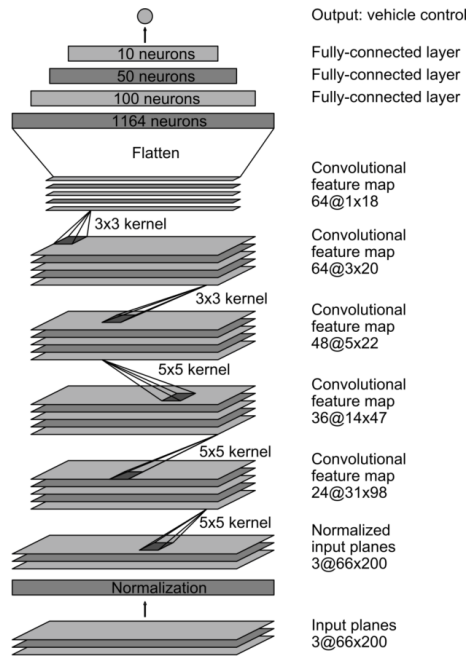


Figure 4.14: NVIDIA Pilot net CNN architecture [94]

Many CNN architectures have been reviewed and been well-received across various image recognition applications. Notable architectures include VGG net, ResNet, and Dense net [151, 152, 153]. One of the most notable architectures used in the field of autonomous vehicles is Pilot net designed by NVIDIA presented in their End to End Learning for Self-Driving Cars paper [94]. Shown below is the Pilot net architecture (*figure 4.14*).

Although Pilot net functioned strongly in real-world applications [94], this system was tested on custom-made tracks. Therefore, following a full architecture was considered too complex, as feature extraction was solely based on line-markings. Alternatively, a custom-built architecture was implemented.

Supervised learning was performed by labelling image data with the steering and speed. The steering is regarded as a regression problem as the steering angle was a continuous value between a range of angles. Alternatively, speed was considered a classification problem as the speed would take only six levels. This was decided as the speed should be a set value for particular points on the track e.g. speed one for sharp bends and five for straights. The custom architecture was proposed to be used for both classification and regression models to detect the same features. The only changes required were to the various hyper-parameters which will be highlighted during the optimization process (*chapter 5*).

4.4.3.4 Autonomous Control

The autonomous control was designed to employ the CNN classifier to predict the speed, with the regression model predicting steering. The activity diagram below shows the process of the autonomous control functionality (*figure 4.15*). The process is started once the endpoint is hit from the client. This triggers the function to load labels for the classifier and load both trained models. A loop is then entered which obtains the image from the camera feed, processes the image and passes it into each model for a prediction to be made. These predictions are used to set the speed and steering angle.

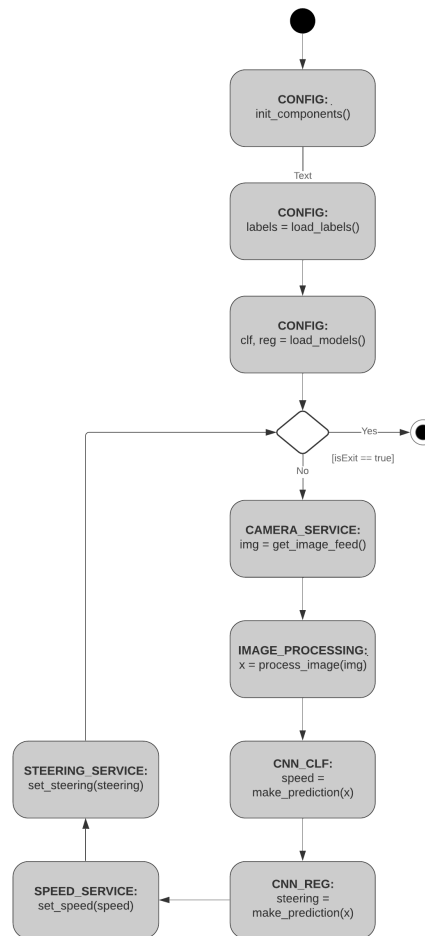


Figure 4.15: Activity diagram for vehicle system depicting the main auto-drive loop

4.4.4 Full System

Presented in this section is a representation of the complete system, incorporating software from both vehicle system and web application.

4.4.4.1 Package Diagram

The package diagram highlights the main packages in each of the components (*figure 4.16*). The main web application is highlighted in three distinct layers, presentation, application and data layers. The vehicle system includes three packages, an API, a machine learning and a configuration package.

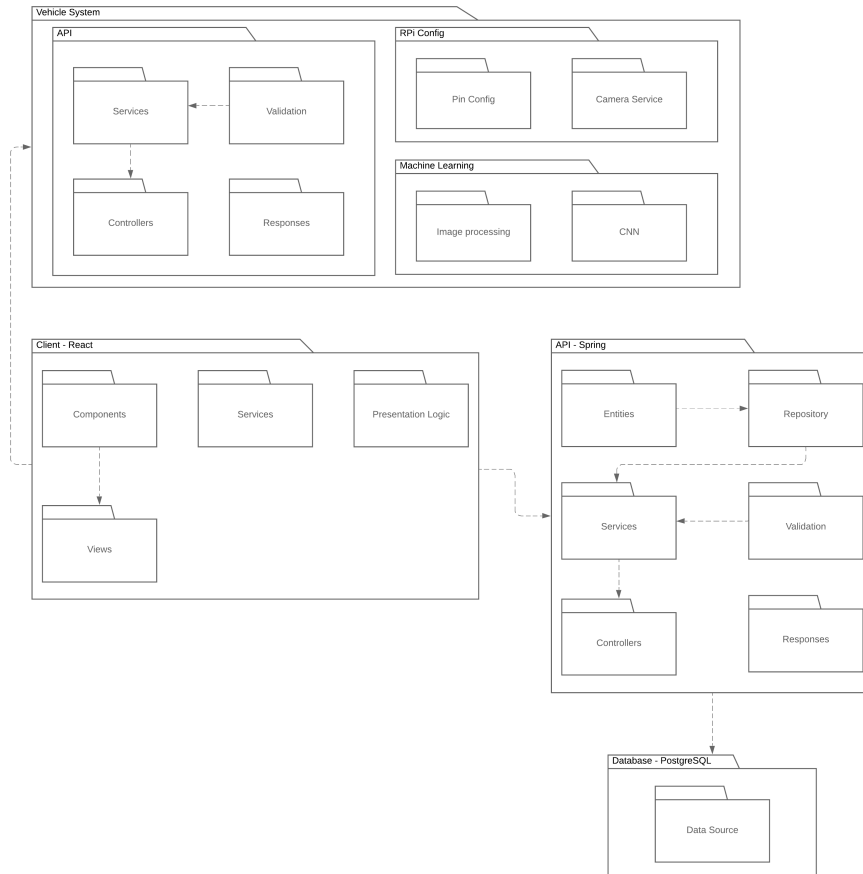


Figure 4.16: Package diagram showing the full systems package structure

4.4.4.2 Deployment Diagram

The deployment diagram highlights all environments, illustrating all hardware and software components (*figure 4.17*). The RPi comprises of a Flask API which exposes Python machine learning and configuration scripts. The client communicates with the vehicle system and the API through HTTP. The web application will be run locally and will not be deployed as a cloud service as the project is limited in time.

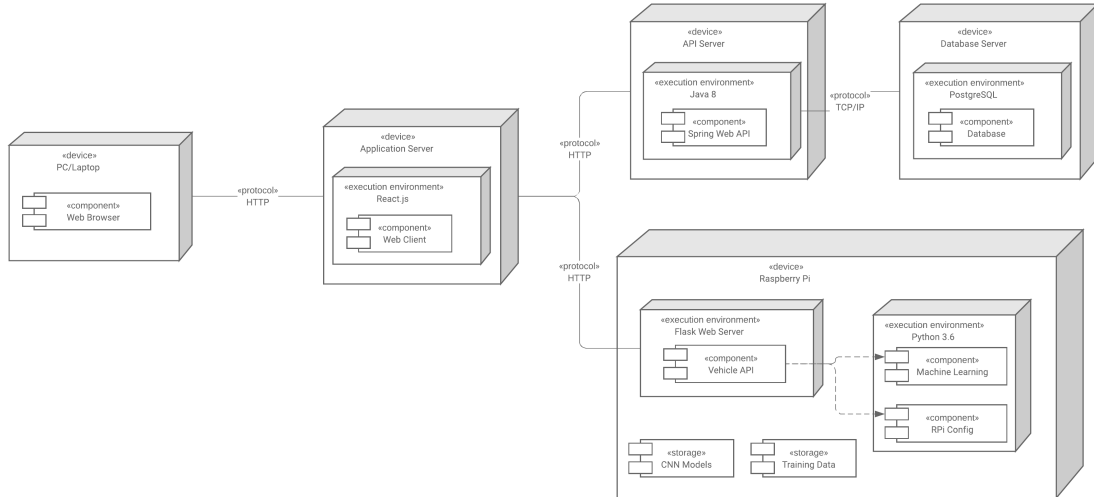


Figure 4.17: Deployment digram of the full system

4.5 Chapter Summary

The primary requirement was to create a simulation of an autonomous vehicle which can predict the speed and steering angle by road markings. Additionally, the vehicle was required to be remotely controlled and trained through a web application, including user authentication. The designed system can be summarized as featuring two components, a web application and a vehicle system. The web application consists of a PostgreSQL database, Spring API and a React web client. A three-tier architecture is adopted. This was selected to create cohesive and loosely coupled components.

The vehicle system consists of both hardware and software components. The main hardware component is the Raspberry Pi which runs the developed software. Additional hardware components include a camera for capturing training data and input frames, a chassis, a DC motor, a servo motor and motor driver IC. The software components include machine learning scripts, configuration scripts and a Flask API which exposes remote control functionality. The remote control schemes include manual control, training control and autonomous control. Autonomous control uses a Convolutional Neural Network (CNN) to predict the speed and steering angles from the supplied input frame.

The CNN was broken down into two models. The steering was regarded as a regression problem as the steering angle will be a continuous value. Alternatively, speed was considered a classification problem as the speed will take only six levels. This was decided as the speed should be a set value for particular points on the track. It was decided to adopt a custom CNN architecture for both classification and regression models. The custom architecture was developed through optimization

during the development process.

Chapter 5

Implementation

This chapter of the report outlines the implementation of the designed system. Discussed within this chapter are the key components and functionalities implemented in both the web application and vehicle system. Additionally, included are any difficulties encountered during the implementation process and how these were overcome. A summary is found at the end of the chapter.

5.1 Implementation Process

Implementation followed an incremental process consisting of two-week-long cycles. A cycle consisted of tasks generated from the user requirements. Tasks were generated in an initial cycle, which additionally was allocated to highlight dependencies and prioritize tasks. For each cycle, tasks were selected based on priority and dependencies. Kanban was used to keep track of completed tasks.

Version control was adopted with Git employed for conducting local changes and Bitbucket used as the host for remote repositories. Branches were established for each task, allowing changes to be made without breaking previously working code. Furthermore, testing was performed on each task. Unit tests were written following a Test Driven Development (TDD) methodology where required. Manual testing was additionally performed on each feature.

5.2 System Overview

This section overviews the architecture and structure of both the web application and vehicle system.

5.2.1 Web Application

The web application was developed following a layered architecture. By adopting this it allowed each component of the system to be developed in isolation. Three layers were implemented, a data, application and presentation layer. The data layer consisted of a database developed using PostgreSQL.

Situated within the application layer of the system is the REST API. The API was implemented using Spring and followed the classical three-tier Spring design pattern. The pattern follows the Separation of Concerns (SoC) principle [154, 155]. This pattern was adopted as it allowed the creation of reusable functionality and improved testability, as each layer could be tested in isolation. This was implemented by splitting the applications logic into three layers:

- **Controller:** Responsible for serialization of HTTP responses and requests. This layer was equally responsible for calling related functionality from the service layer.
- **Service:** Responsible for holding all the logic and process validation. This layer also followed Springs service-repository pattern, meaning that each service wired the CRUD operations held within the implemented repository interfaces.
- **Data Access:** Responsible for serialization of objects to and from the PostgreSQL database. To map database objects to Java objects an Object Relational Mapping (ORM) library was required. Hibernates Java Persistence API (JPA) implementation was adopted as the main ORM and database connectivity tool.

The presentation layer consisted of a React client. Using React allowed for common UI components to be split up into separate classes, improving code reusability. The client was structured by separating components, views and services. The components package consisted of all reusable components, allowing complex views to be created from simple components, improving maintainability and testability.

The views package contained all the main pages. These were developed as React component classes which acted as a wrapper, calling required common components. The services package was created for holding all logic related to making API calls. To make calls to the APIs an HTTP client library was required. The library selected was Axios, which is a promise-based HTTP library. Axios was selected over other libraries due to its ease of use and advanced features.

5.2.2 Vehicle System

The vehicle system consists of hardware components controlled through software run from a Raspberry Pi. The hardware includes a DC motor (speed control), servo motor (steering control), two power supplies and a camera required to feed input data to the CNN models. All of these components are fitted and mounted to the chassis of the vehicle (*figure 5.1*).

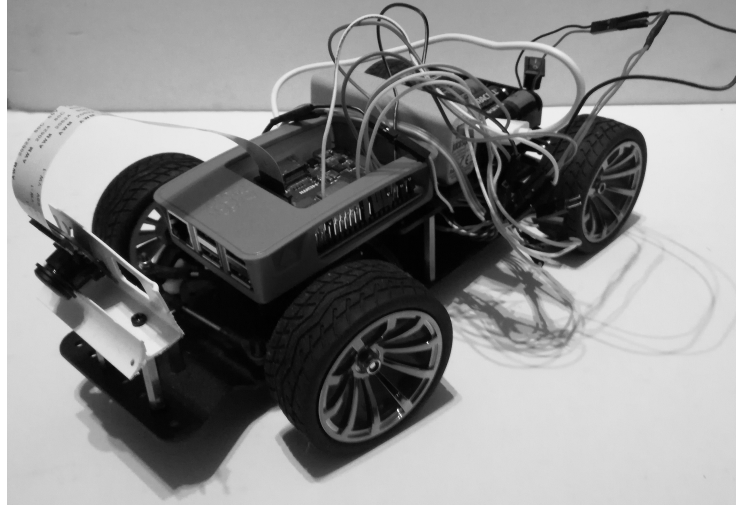


Figure 5.1: Final hardware implementation

One of the key considerations was the mounting angle of the camera. This was considered a significant factor as variations in the camera positioning would negatively affect results, reducing the validity and reliability of data collected. To control this the camera was configured with the angle calibrated manually. The camera position was marked to ensure that if the camera moved it could be repositioned and recalibrated. The optimal camera angle was considered to include maximum edge and boundary features of the track.

The developed software consisted of machine learning and pin configuration scripts developed in Python. The functionality of these scripts is exposed through a Flask API consumed by the client. Each software component was separated into individual Python packages. This was implemented to follow the Separation of Concerns (SoC) principle, allowing the development of highly cohesive modules.

5.3 Authentication

One of the key user requirements for the web application was user authentication. The authentication mechanism was implemented using JSON Web Token (JWT). This was decided to comply

with the staleness property of a REST API as it would mean no session would be held server-side. Two key considerations were taken during the implementation of the authentication mechanism. These were performing authentication on every request and the locking of authenticated URLs on the client.

To overcome the problem of performing authentication on every request, a security filter was added to the API (*listing 5.1*). This was implemented by creating a custom security class, overriding the Spring security configuration. This filter functioned by intercepting all resource requests, validating the JWT.

Listing 5.1: JWT authentication filter function

```
@Override
protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response,
                                FilterChain filterChain) throws ServletException, IOException {
    try {
        String jwt = getJwt(request);
        if (jwt != null && tokenProvider.validateJwtToken(jwt)) {
            String username = tokenProvider.getUserNameFromJwtToken(jwt);
            UserDetails userDetails = userService.loadUserByUsername(username);

            UsernamePasswordAuthenticationToken authentication =
                new UsernamePasswordAuthenticationToken(
                    userDetails,
                    null,
                    userDetails.getAuthorities());
            authentication.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));

            SecurityContextHolder.getContext().setAuthentication(authentication);
        }
    } catch (Exception e) {
        logging.error("Can NOT set user authentication -> Message: {0}", e);
    }
    //perform next element in the filter chain
    filterChain.doFilter(request, response);
}
```

The final consideration was the locking of authenticated URLs on the client. The React application was configured with React router. This is a standard routing library which allows URLs to be mapped to React components. The router library additionally allows the creation of custom route logic, allowing logic to be performed before the component is rendered. This allowed different routing logic to be created for both registered and administrative users.

5.4 Vehicle Controller

One of the key requirements for the vehicle system was to provide users with various forms of control over the vehicle remotely. Both registered and administrative users required different permissions. Registered users were provided with manual control over the vehicle. Administrative users were provided with this control and with additional training control.

For the control modes to be operated remotely, a UI was required. This was achieved by creating a controller dashboard component on the client (*figure 5.2*). The control dashboard was developed by providing a user with the ability to set the status, speed and steering angle. This was achieved by providing different control schemes for each control mode.

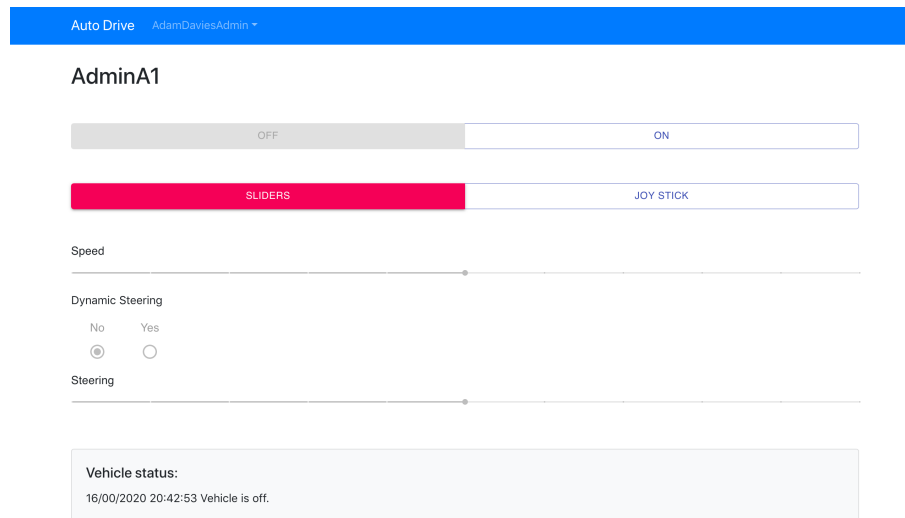


Figure 5.2: Manual control dashboard layout

5.4.1 Manual Control

There were two control options implemented for manual control, these were slider and joystick control schemes which set speed and steering. Joystick control was implemented using *react-joystick* library. This library included a joystick in the form of a React component which could be manipulated by passing in property variables. One of the problems encountered was the output from the joystick required to be scaled for the manual control endpoint. This was overcome by adding custom logic to transform the output into a suitable value.

An endpoint was developed on the vehicle system API which expected a POST request with a JSON body consisting of status, speed and steering values. Both slider and joystick control schemes were attached to an on-change handler function, performing an Axios POST every time the values changed.

The endpoint on the Flask API firstly validates each parameter, throwing an exception if validation fails. Once validated the API sets the speed through the speed service which processes the value provided. The possible values were zero to five, with the service logic converting the value into a predefined PWM duty cycle. The steering value differs as it is a continuous value from 40° (left) to 140° (right). This requested value is processed through the steering service which converts the steering angle passed into a PWM duty cycle.

The implementation of the manual controller achieved the remote control requirement of the system. The majority of the implemented code was reused in the training controller. This was one of the significant advantages of splitting the API into service modules as it allowed functionality to be reused.

5.4.2 Training Control

Before the level of automation and how well the vehicle drives could be measured, the CNN models required to be implemented and trained. The training control required to provide an administrative user the ability to remotely train the vehicle. One of the considerable differences between the training and manual controllers was the usage of the camera to collect imagery data, saving and labelling the images on the Raspberry Pi.

One of the difficulties encountered implementing the training controller was the use of the on-change handler on the UI. This was a problem due to the vehicle being too responsive and difficult to control. This, therefore, led to poor quality images which would have negatively affected the performance of the final model. To overcome this a submit button was added to the training control dashboard, allowing the speed and steering value to be set before a POST was sent. This allowed higher-quality images to be captured.

An additional problem encountered was the data collection speed. Initially, the training controller was developed to capture one image before the vehicle speed and steering was set. This was considered too slow with the training projected to overrun the allocated time plan. To overcome this, a loop was added with a minimal delay. This meaning every time the speed and steering were submitted, the system would capture multiple images of each position within the manoeuvre. This increased the speed of data collection and led to improved data, with the training data encapsulated varying features of the tracks. The listing below shows the training controller endpoint which highlights the developed data collection loop (*Listing 5.2*).

Listing 5.2: Training control endpoint function which collects training data

```
@VEHICLE_CONTROLLER.route("/training", methods=['GET', 'POST'])
@cross_origin()
def training_control():
    """Endpoint function which allows user to pass speed, steering and status to
    control the connected vehicle in training mode"""

    content = request.json
    required_request_body_check(content)
    vehicle = Vehicle

    for _ in range(10):
        status, speed, steering = training_control_service . training_control_service (content)
        sleep(0.05)
        pin_config.reset_pwm_speed()
        vehicle = Vehicle(status, speed, steering, 'training control')

    pin_config.reset_pwm_steering()

    return jsonify ( vehicle=vehicle. __dict__ , status=200, mimetype='application/json')
```

5.5 Custom CNN

Two CNN models were developed using TensorFlow and Keras. The two models were a classification model and a regression model. The speed of the vehicle was predicted from the classification model. Alternatively, steering predictions were achieved through the regression model. Both models were developed to follow similar architectures, as both models were interested in the markings on the road.

5.5.1 Training

The vehicle was trained through the training control functionality. The training process consisted of data collection, image preparation, fitting the data and testing the model. Data collection was time-boxed, which was decided as this is regarded by many as the most time-consuming process of implementing machine learning [38, 35]. The data collection process was achieved through driving the vehicle manually through set training tracks. There were four tracks set up for the vehicle to be trained on (*figure 5.3 — start line green/stop line red*).

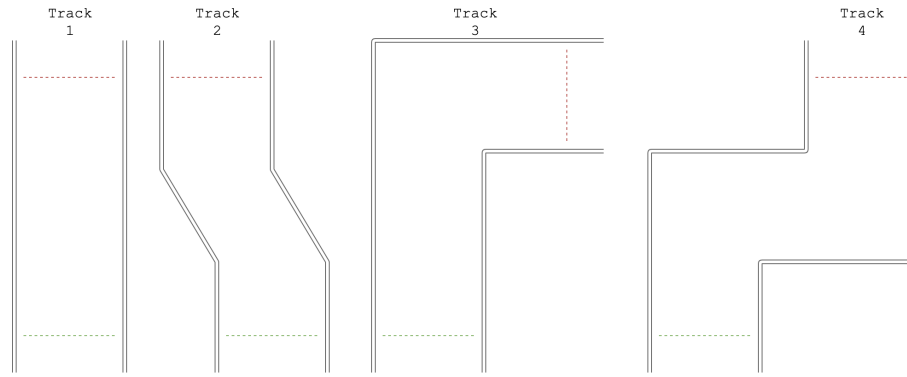


Figure 5.3: Training track layouts

Once data was collected on a trained track, the images required preparation before they could be supplied into the CNN (*figure 5.4*). The image processing was developed using OpenCV, a Python library developed for computer vision. OpenCV features many pre-optimized image processing functions and features extensive documentation. The image processing function developed performed the following steps:

1. Crop: The vehicle is solely interested in the lane markings on the track. Therefore, to prevent the models from over-fitting to features within the background the images were cropped. Cropping was performed horizontally cutting out the background features, leaving the features of the track.
2. Greyscale: The vehicle is not interested in colour as the track markings used were light on a dark floor. To reduce the complexity of the model, every image was transformed into greyscale. This is justified to greyscale images having only one colour channel, whereas coloured images contain three channels (RGB).
3. Resize: Images were initially captured with a resolution of 640x480. This was excessively large to feed into the CNN and would reduce training and prediction performance. The image size was experimented with during the optimization process. The optimal image size was 64x64 as this provided increased performance without diminishing the accuracy of each model.

After all images were processed they were supplied into each CNN for training. The full dataset was used to train the model through each training cycle. This meaning that data collected from training the vehicle on one track was used for training on subsequent tracks. The full dataset consisted of over 8,000 images.

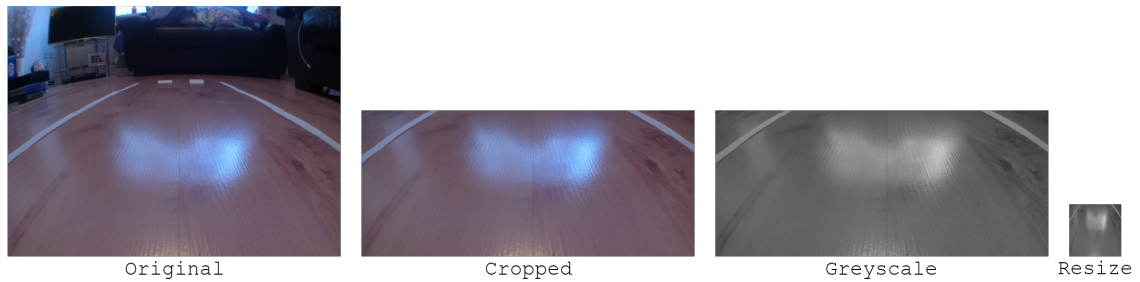


Figure 5.4: Image processing sequence

5.5.2 Optimization

The optimization process consists of finding the most suitable hyperparameters for a given dataset. There are two techniques used for hyper-parameter optimization, manual and automatic. Manual techniques are based on previous experience and experimentation. Automatic techniques involve algorithms to set given hyper-parameters. In the real world both offer viable solutions, with the trade-off being time and complexity considerations which come with manual optimization, against the significant computational requirement which comes with automatic optimization [156, 157].

There are many automatic optimization techniques, including Bayesian, Grid Search, Random Search and Gradient Descent approaches. For this project Grid Search was employed. This utilises a grid of hyper-parameters which is created and searched, with the most suitable combination of hyper-parameters highlighted. The problem with this technique is inefficiency in situations where a high dimensional hyper-parameter space is defined [156, 59]. To overcome this, many reduce the dimension of hyper-parameters by utilizing a combination of manual and grid search optimization, with results showing it can outperform other optimization techniques [156, 158]. It was decided to adopt this approach with manual optimization used for determining the number of convolutional layers, fully connected layers and pooling configurations. The Grid Search was used for finding the optimal number of neurons, filter sizes, optimizer, learning rate, epochs, etc.

There are many libraries which feature pre-optimised Grid Search algorithms. The library adopted was Scikit-learn. This was implemented by wrapping the Keras model in a Scikit-learn model, which can be passed into the grid search function with desired hyper-parameters to be searched (*Listing 5.3*).

Listing 5.3: Grid search functionality implementation

```

# wrap model in keras classifier to be used for GridSearchCV
model = KerasClassifier(build_fn=build_model, verbose=0)

param_grid = dict(optimizer=optimizer,
                    batch_size=batch_size,
                    epochs=epochs,
                    dropout_rate=dropout_rate,
                    weight_constraint=weight_constraint,
                    neurons=neurons)

grid = GridSearchCV(estimator=model, param_grid=param_grid, cv=3)

# train model
grid_result = grid.fit(X, y)

```

5.5.2.1 Number of Convolutional Layers

The fundamental basis of a CNN architecture is the number of convolutional layers. These layers are used for feature detection. To find the optimal number of convolutional layers, the performance of the model was evaluated with varying amounts of convolutional layers. These layers were tested in conjunction with equally stepped filter sizes and the same number of fully connected layers. Shown below are the results achieved (*figure 5.5*).

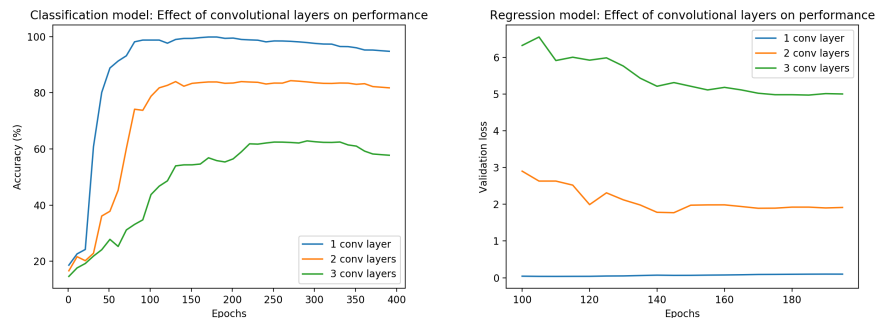


Figure 5.5: Results from convolutional layers optimization

The results show, that architectures with a lower number of convolutional layers produce increased performance. In theory, more convolutional layers can increase the number of features detected [71, 78]. However, for this project, there is only interest in edge detection. Therefore, the extra convolutional layers are likely overfitting to the data set, which is the reason for the decrease in accuracy for both models.

5.5.2.2 Training Optimizer

The training optimizer algorithm was one of the hyper-parameters tuned using the Grid Search functionality. There were three main optimizers tested, these included ADAM, Stochastic Gradient Descent (SGD) and Adamax. The results from the optimization process showed that the ADAM optimizer was the most suited for both CNN models, significantly outperforming the alternative algorithms. Shown below are the results plotted showing validation accuracy for the classification model and validation loss for the regression model (*figure 5.6*).

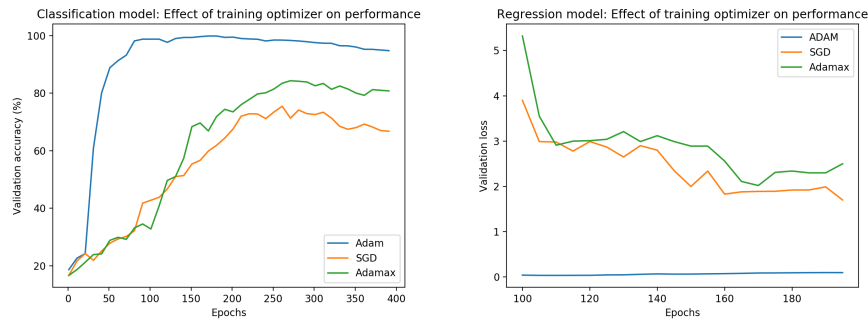


Figure 5.6: Results from training optimizer optimization

5.5.3 Autonomous Driving

Before the level of automation and how well the vehicle drives could be measured, autonomous driving control was required. To implement autonomous control, a button was implemented on the UI of the control dashboard. This would perform a POST on click to the auto control endpoint, activating the autonomous driving functionality. To test the system a pause button was added to pause the functionality, allowing readings to be taken.

The autonomous control required to adopt the CNN models to generate predictions on the speed and steering from the input frame. The logic for this was to first load the models. Thereafter, each model is loaded and the current input frame is retrieved where image processing is applied. Ultimately, this image is supplied to each model where both speed and steering predictions are made. Shown below is the main prediction loop (*listing 5.4*).

Listing 5.4: Predication loop implemented for autonomous driving

```
# capture frames from the camera
for frame in CAMERA.capture_continuous(raw_capture, format="bgr", use_video_port=True):
    try:
        # grab the raw NumPy array representing the image, then initialize the timestamp
        # and occupied/unoccupied text
        img = frame.array
        X = image_processing.process_images_cnn(img)
        X = np.array(X)
        X = np.reshape(X, [-1, 64, 64])
        speed = cnn.make_speed_prediction(clf, X, label_encoder)
        steering = cnn.make_steering_prediction(reg, X)

        speed_service.speed_control(speed)
        steering_service.steering_control(steering, True)
        raw_capture.truncate(0)

        sleep(0.1)
        pin_config.reset_pwm_speed()
        pin_config.reset_pwm_steering()
        sleep(0.5)

    except KeyboardInterrupt:
        break
```

5.5.4 Final Architecture

The final architecture for the CNNs was distinguished after the vehicle was trained on all four tracks. The primary basis of the final architecture was used for both models. It includes two convolutional layers with dimensions of 8x3x3 and 16x3x3. Both convolutional layers feature a max-pooling layer with a 2x2 pool size. The activation functions used on each convolutional layer were ReLU as it was found most suited to this architecture. The final output from the convolutional layers is flattened before being fed into three fully connected layers featuring neurons of 512, 256 and 32.

The key differences in the regression model and classification model are the output. A final output layer outputting one is utilized on the regression model and an output size of six is used on the classification model, representing the binary encoded labels. The regression function features an activation function to represent a linear output, in this case linear regression. The classification model features a non-linear activation in the form of softmax which outputs decimal probabilities of each class. Both models have over 170,000 trainable parameters. Shown below is the final network architecture of both models (*figure 5.7*).

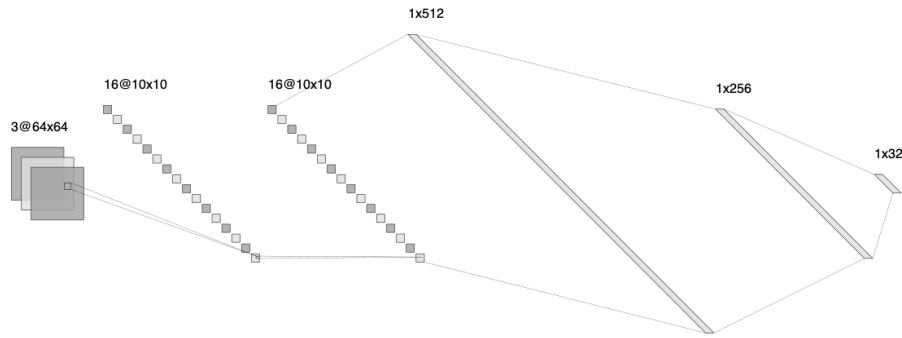


Figure 5.7: Final CNN architecture

5.6 Chapter Summary

Implementation of the web application followed a three-tier architecture. The key considerations taken were for authentication, where considerations were taken for the locking of URLs on the client, which was overcome by using custom routing logic and authenticating every request, which was overcome by adding a security filter on the API.

The vehicle system consists of both hardware and software. One of the main considerations when implementing the hardware was the angle of the camera. Any changes in position would have a negative effect on the CNN models performance. To overcome this, the camera was calibrated and positioned to view as much of the track as possible with the position marked.

Autonomous control was implemented using the CNN models for predicting the speed and steering. The CNN was implemented using TensorFlow and Keras. Both classification and regression models were trained on four tracks where images were collected, labelled and applied with image processing, before being fed into the CNN models. Both models were optimized using an informal Grid Search approach with hyper-parameters such as learning rate, dropout, epochs and batch size optimized.

The final architecture includes two convolutional layers with dimensions of 8x3x3 and 16x3x3. Both convolutional layers feature a max-pooling layer with a 2x2 pool size. The activation functions used on each convolutional were ReLU as it was found most suited to this architecture. The output from the convolutional layers is flattened, before being fed into three fully connected layers featuring neurons of 512, 256 and 32. Overall the implemented system allows the automation level and how well the vehicle drives to be measured.

Chapter 6

Results and Analysis

This chapter of the report discusses the results achieved from the tests performed on the implemented system. The chapter is broken down into two key sections, these being the Convolutional Neural Networks (CNNs) performance and vehicle performance. Discussed within the CNN performance section are the results achieved from the final implemented models. The vehicle performance section overviews the automation levels the vehicle achieved and gauges how well the vehicle drives. All findings and conclusions are summarized at the end of the chapter.

6.1 Convolutional Neural Network (CNN) Performance

This section presents the results achieved from the final implemented Convolutional Neural Network (CNN) models. Results from the various metrics used on both classification and regression models are provided in individual sections.

6.1.1 Classification Model

The classification model in the system was solely used for predicting speed values. The model was evaluated using two main metrics, these being classification accuracy and AUC (Area Under The Curve) ROC (Receiver Operating Characteristics). Both these tests were performed on the final architecture with the full final dataset. The following sections overview the results from these tests.

6.1.1.1 Classification Accuracy

The classification accuracy of the model is the number of correct predictions made out of the total dataset. Since the CNN model was built using TensorFlow and Keras, these metrics are outputted as default metrics. The classification accuracy is shown in the graph below, where the accuracy is plotted over training time (*figure 6.1*).

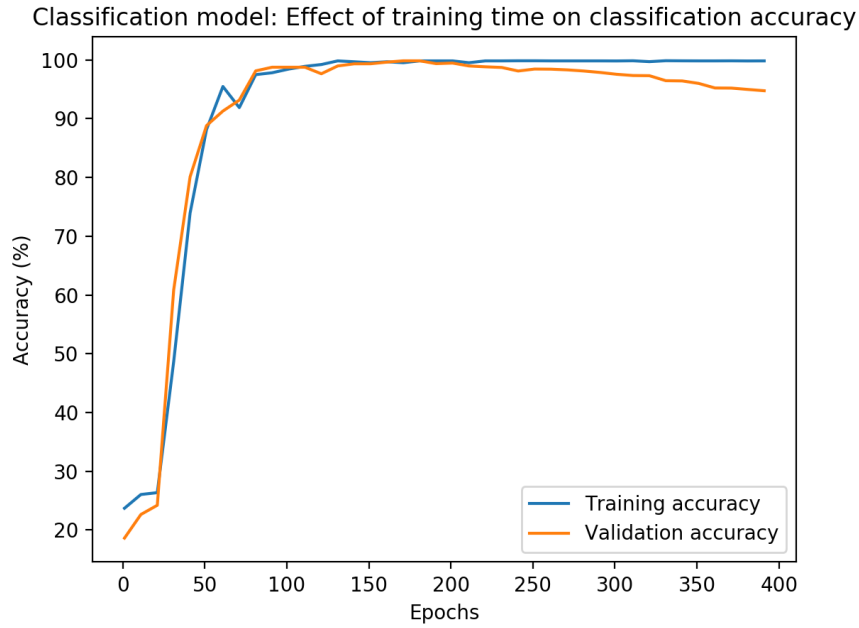


Figure 6.1: Effect of classification accuracy over training time

As shown from the results achieved the final classification model has a peak training accuracy of 99.84% and a peak validation accuracy of 99.85%. The training accuracy of the model stabilizes at around 180 to 200 epochs. At around this training time, it can be perceived that the validation accuracy gradually declines. This is an indication that the model is over-fitting to the dataset and starts failing to generalize to the validation set. The performance shows that the classification model has an optimal accuracy of 99.81% occurring at around 200 epochs where the validation accuracy is at 99.79%.

6.1.1.2 AUC - ROC

AUC (Area Under The Curve) ROC (Receiver Operating Characteristics) curve is one of the most commonly used metrics for evaluating classification algorithms. ROC is a probability curve with AUC representing the amount of separability between classes. This metric presents an overview of how well the model distinguishes and predicts classes. Generally, the higher the AUC, the more

accurate the model is at predicting class A as class A.

AUC — ROC is primarily used in binary classification. Since the speed of the vehicle is a multi-class problem, to plot the averages of each class and the performance of the model the method of micro and macro averaging is used. Micro averaging takes the individual true positives, true negatives, false positives, and false negatives of the model and averages the value. Macro averaging is where the average performances of each independent class are computed. To plot this, Scikit-learn ROC metric function was used. The graph below shows the AUC — ROC plot for all speed classes (*figure 6.2*).

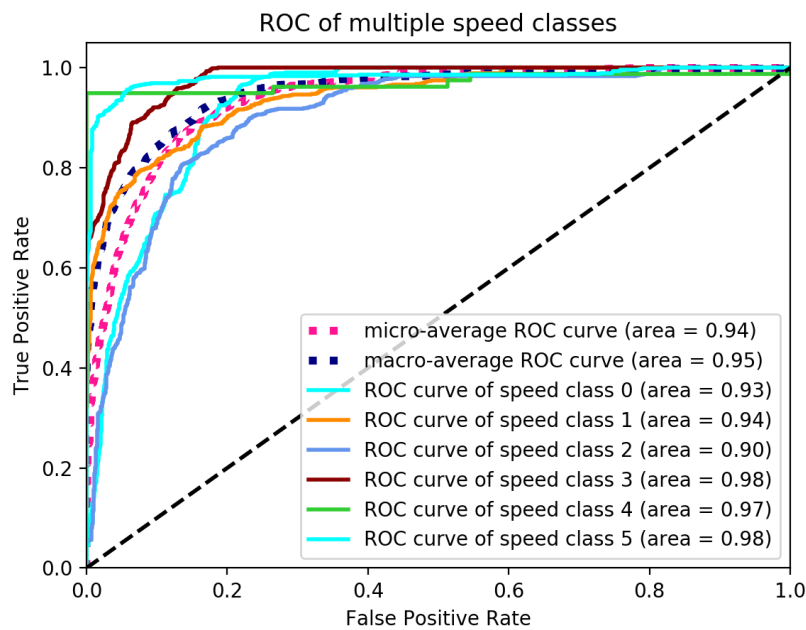


Figure 6.2: Effect of classification accuracy over training time

The results achieved show that generally the model distinguishes between classes strongly. This is apparent to how far each class is spaced from the diagonal line, which represents the model distinguishing classes with no skill. As shown the model can distinguish most strongly between speed class five and three, predicting the class 90% of the time. The class which has the lowest AUC value is speed two, with the model predicting the class correctly 90% of the time. The reason for this is the class may not be distinct enough, with the model being unable to accurately highlight differences between the image sets. This is to be explored further in the on-track testing section.

6.1.2 Regression Model

The regression model was responsible for predicting the steering angles of the vehicle. The model was evaluated solely through the loss values achieved. This is due to the performance being evaluated through how well the vehicle drives and is covered through vehicle performance testing. The loss function used to evaluate the regression model was the Mean Squared Loss (MSL) error. This loss function is the mean overseen data of the squared differences between true and predicted value. The metric was achieved through Keras metrics library which outputs the metric on fitting the model with data. Shown below is a graph which plots the loss value of the model over training time (*figure 6.3*).

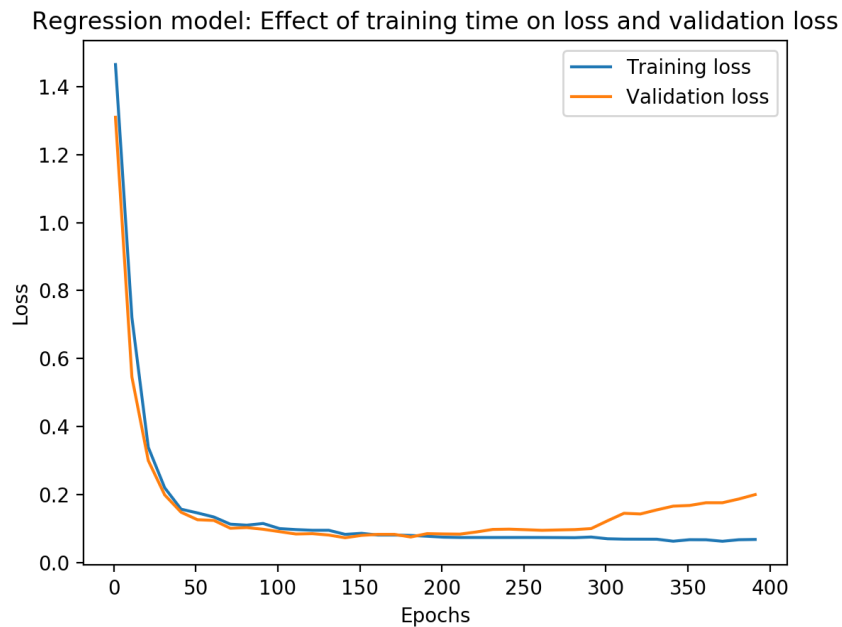


Figure 6.3: Effect of regression loss over training time

The results achieved highlight that the model achieved a peak loss of 0.0753 which occurred after 310 training cycles. This, however, is suspect to false accuracy as the validation loss starts increasing which is a sign that the model is overfitting to the data. This is apparent at around 250 epochs. The implemented models peak accuracy is believed to be at 250 epochs which highlights a validation loss of 0.0782 and a training loss of 0.0784.

Ungurean [71] who presents results on developing a custom CNN for a similar project (see section 3.3.1), achieves a peak validation loss of less than 0.01 after just 100 epochs. The loss achieved by Ungureans model, is impressive taking almost under half the training time to maximize the loss. Additionally, the model features a lower loss throughout training and

validation. However, this can be explained due to Ungurean developing a more complex CNN architecture with over double the size of the dataset. To evaluate the true effectiveness of the full system on-track testing is performed. This is discussed in section 6.2.

6.2 Vehicle Performance

Tests were performed to measure the automation level the vehicle has achieved and to answer how well the vehicle drives. The tests were performed on both the trained tracks (see *figure 5.4*) and two unseen tracks (*figure 6.4*). This was performed to gauge how well the vehicle generalizes to unseen track layouts. The following sections outlined the performed tests presenting and discussing the results.

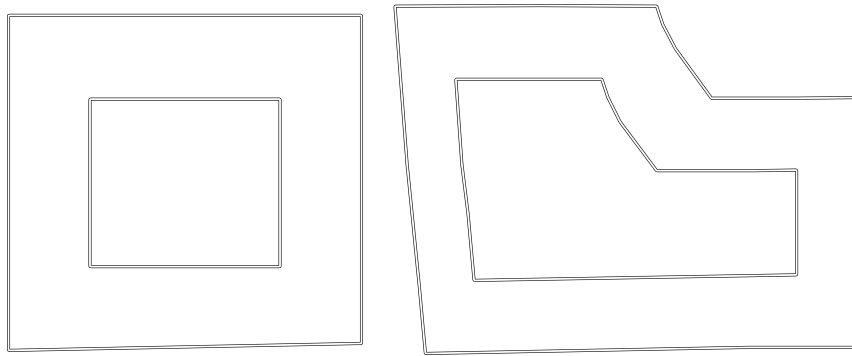


Figure 6.4: Unseen test track layouts

6.2.1 Automation Level

The automation level achieved from the vehicle was tested using the metric described in section 3.2.8 of the report. The test is performed to assess the amount of human intervention required, whilst the vehicle is driving autonomously over a specified time. This test was performed on both the trained and unseen tracks where the vehicles driving time was recorded until the track was complete. The vehicle would then be placed at the start where the timer is restarted, with the test finishing after a total of 20 minutes driving time was achieved on each track. The number of times human intervention was required was recorded and arranged into *equation 2.1*. To execute the tests fairly this was performed a total of three times on each track, where an average of each was taken. The results of the tests are presented in the graph below (*figure 6.5*).

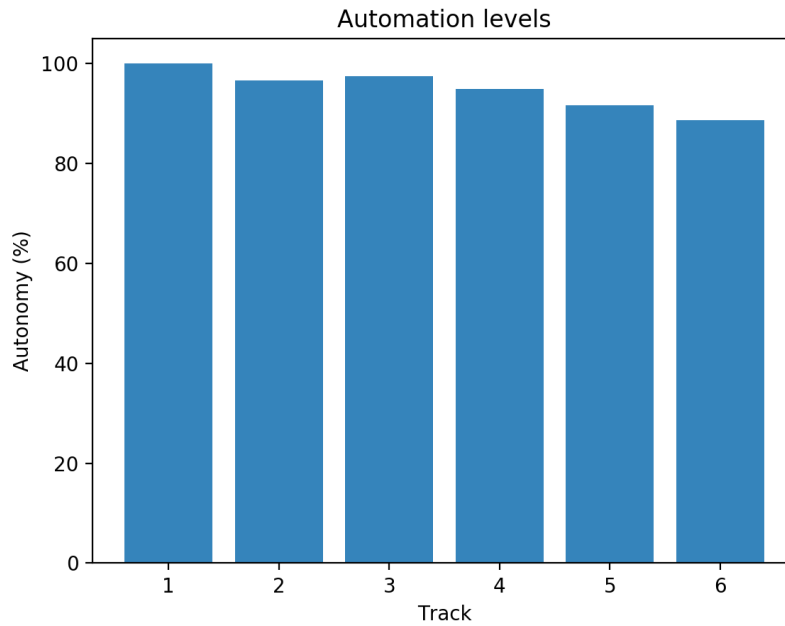


Figure 6.5: Unseen test track layouts

The results show that on average the vehicle possesses a high level of automation with an average of 99.4% across all tracks. Track one shows that no human intervention was required over the 20 minutes of testing, achieving 100% automation. This is to be expected as this track is the least complicated out of all test tracks. Additionally, it can be regarded the more complex the track, the worse the vehicles performance. However, the vehicle achieved 88.75% on track six which is considered the most complex track. This is, nevertheless, a reasonable level of automation. The vehicle performs well on the unseen tracks and can be observed that the CNN models generalize strongly.

The automation metric adopted is utilised by both Bojarsk et al. [94] and Ungurean [71]. Results achieved compare favourably. Ungurean tested an implemented CNN on two tracks for 10 minutes and concluded the achievement of 88% autonomy. Although Ungureans tracks are longer and more complex, with the complexity comparable to track six, the results achieved are regarded as favourable. This due to with the CNN implemented in this project being less complicated and utilises less training data.

6.2.2 Driving Performance Results

The vehicles driving performance was examined to answer how well the vehicle could drive. The performance was measured by evaluating the speed and steering of the vehicle relative to a position on the track. The tests were performed on all trained tracks and the two unseen tracks. Both speed

and steering tests were conducted separately, with the following sections discussing the results.

6.2.2.1 Speed

To measure the speed, the track was placed with conceptual boundaries, where the desired speed for a specific part of the track was labelled. The effective predicted speed was then compared with the speed of the boundary, noting any classification errors. To ensure these tests were valid, the width of the track was measured and was ensured to be the same width throughout. Additionally, these tests were performed multiple times with an average taken to ensure results were valid. Shown below are the results of the speed tests on track one (*figure 6.6*).

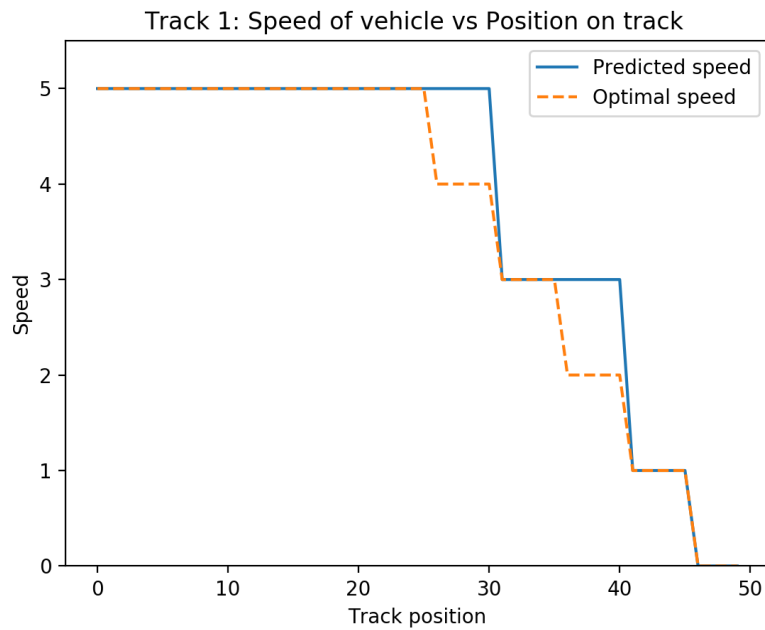


Figure 6.6: Speed results achieved on track one

The results achieved from testing on track one show the vehicle miss classifies the speed from positions 25 to 30. The speed around this positioning is expected to be four where the vehicle is performing a speed of five. This is a similar occurrence at positions 35 to 40 where the speed of the vehicle is expected to be two. However, the effective speed of the vehicle is set to three. This meaning that the vehicle miss classifies speed for 20% of the track. Track one is considered the simplest track with it being straight, the results from the most complex track is shown below (*figure 6.7*).

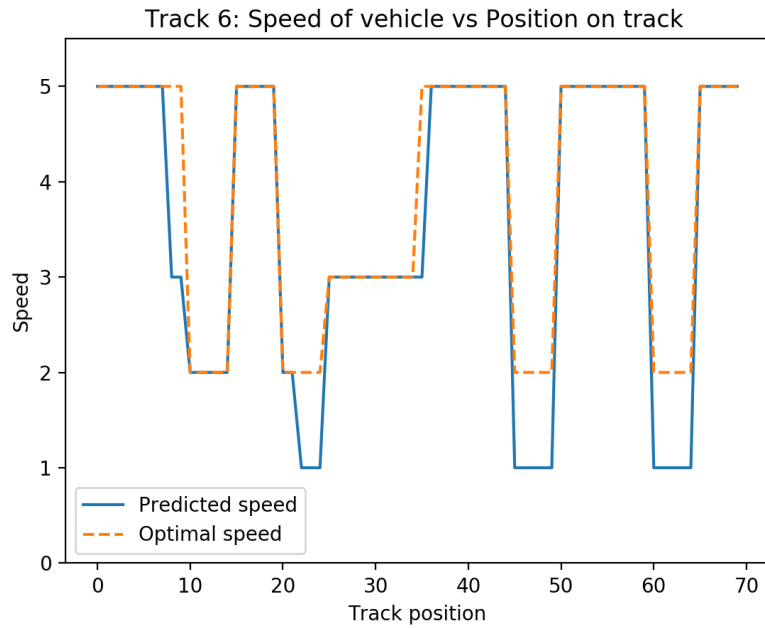


Figure 6.7: Speed results achieved on track six

The results of the speed test performed on the final track show the vehicle miss classifying mainly on sharp corners. An explanation for this is the vehicle miss judges the steering angle, therefore due to it deviating from the ideal path it slows down to readjust its steering angle. The vehicle miss classifies 27.1% of the track.

Overall the vehicle on average miss classifies speed 32.65% of the time over all tracks. The vehicle performs the best on straights where it classifies the speed accurately. However, there are inaccuracies as the vehicle approaches the stop sign where it is expected to slow down. The vehicle continues with a speed of five and slows down to a three, and then a one, with it ultimately stopping. The vehicle misses out speeds four and two, preventing a gradual decrease in speed.

The explanation for this leads to a comparison of the results achieved from the ROC metric. Class two was the most miss classified class. Results achieved from track testing confirm the model is predicting a speed of three where it should be a two. Upon investigating the dataset, it was discovered that the two classes are difficult to distinguish, which is a reason for the inaccurate prediction. To overcome this, the track layouts could be modified with markers indicating the speed at each section of the track. This would allow the model to distinguish from the two classes.

6.2.2.2 Steering

To measure the steering, the vehicle was tested by measuring how far it deviated from the centre line on the track. This was achieved by using the autonomous test mode implemented. The vehicle was placed on the starting position of each track, where once the prediction button was pressed, would perform the predicted steering manoeuvre. Before the button was pressed again, the distance the vehicle was from the centre was recorded. To ensure these tests were valid, the width of the track was measured and was ensured to be the same width throughout. Additionally, these tests were performed multiple times with an average taken to ensure results were valid. Thresholds were conceptualised with results recorded in between -2.5 to 2.5 cm was considered good driving. Average driving was considered lying in between 2.5 and 5.0 cm for left and -2.5 to -5.0 for left. Poor driving is concerned over this with anything out of these thresholds. Shown below are the results of testing on track one (*figure 6.8*).

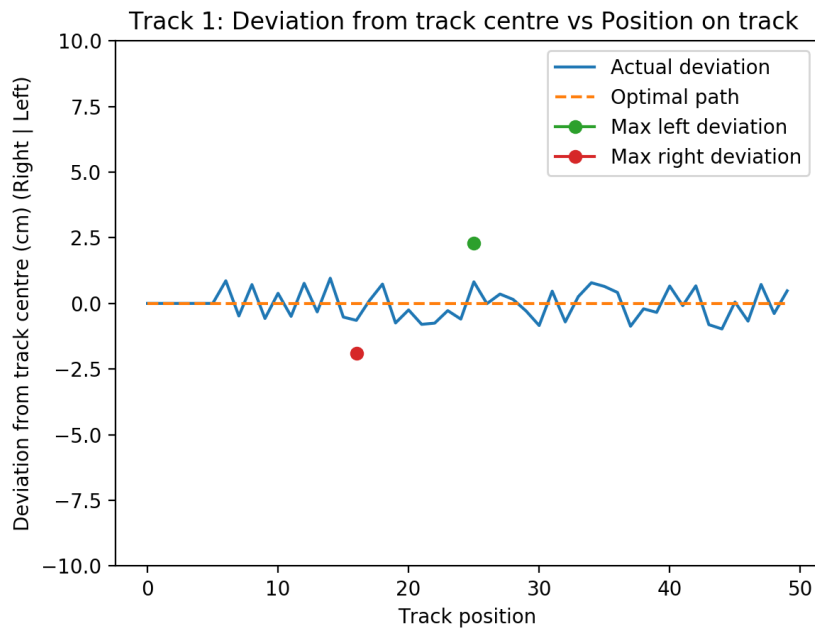


Figure 6.8: Steering results achieved on track one

The results from the first track show that the vehicle maintains proximity to the centre line of the track throughout, adjusting the steering angle accordingly. The maximum deviation to the left of the centreline was 2.3cm and to 1.9cm to the right. This shows even in worst cases the vehicle still remains within the good driving threshold. There are points where the vehicle oversteers and requires to rectify the angle more sharply than other points. This is more apparent when the vehicle is turning right. This may be justified to a servo motor limitation, where it may react more responsively due to the larger steering angle swing. Alternatively, it may be due to bias in the

dataset.

Generally, the steering is well biased towards the centre on this track with the average distance from the centre being 0.45 cm. This is regarded as good driving from the vehicle. Shown below are the results of track three (*figure 6.9*).

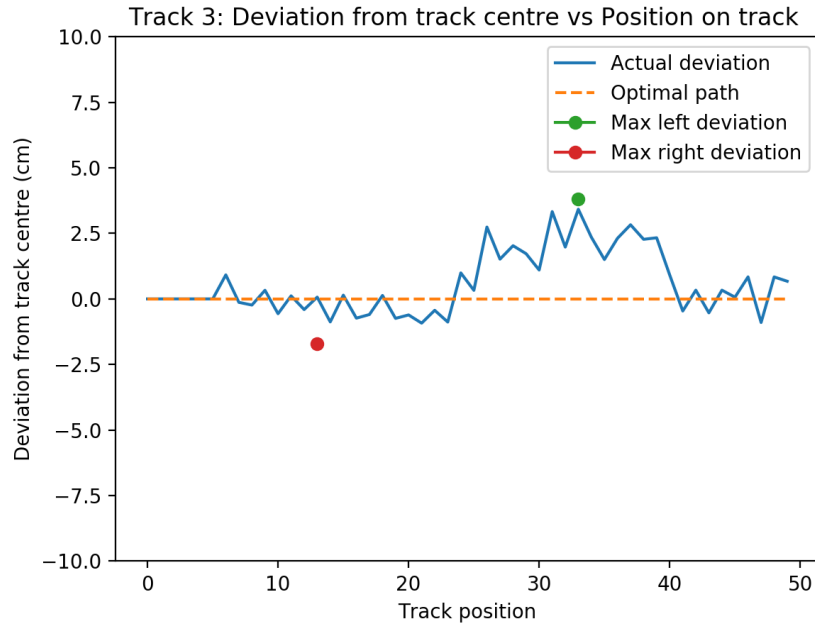


Figure 6.9: Steering results achieved on track three

The results for the third track are of interest as this was the first track tested with a sharp bend. The results show that the vehicle still performs well on the straight portions of the track. However, the vehicle understeers on approaching the bend causing it to deviate up to 3.1 cm from the centre. Although this deviation the vehicle satisfactorily completes the track and straightens up successfully after the bend. Shown below are the results collected from the final test track (*figure 6.10*).

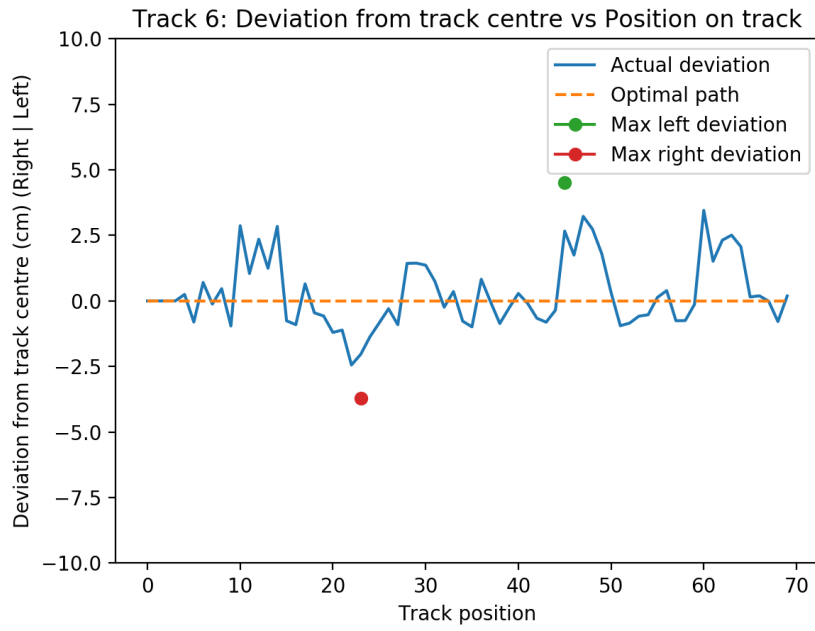


Figure 6.10: Steering results achieved on track six

The final track the vehicle was tested on was the most complicated. Predominantly the vehicle performs well with a maximum deviation of 4.6 cm from the centre line. The results also highlight the vehicle struggling on sharp bends where the majority of the deviation occurs. One of the considerable differences on this track is the vehicle does not perform as well on straights. This is due to the number of turns on the course as the vehicle does not have time to straighten up fully before the next bend.

Overall it is believed the vehicle drives well on each track. The vehicle encounters most difficulties with sharp bends where it does seem to understeer. A reason for this is that the dataset could be biased towards slighter turns. To investigate this further the dataset bias was evaluated.

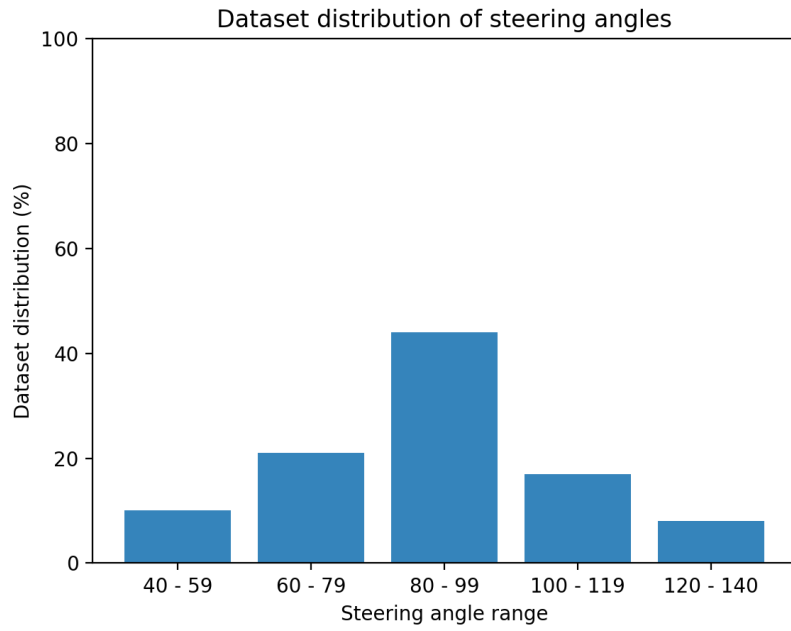


Figure 6.11: Dataset distribution of steering labels

Shown in the graph above (*figure 6.11*) is the bias of the full dataset concerning steering angle labels. It can be noted that the majority of the labels in the dataset are biased for driving straight, this is correct as the vehicles first choice should be to drive straight. However, as it is highlighted in the graph sharper turns account for around 10% for left turns and 11% for right. The vehicle is biased towards slighter turns with it taking up 21% left and 17% right. Further investigation into the dataset, found it is difficult to distinguish between sharper turns and slight turns in certain instances. This means the model is biased to predicting less sharp turns as it may be unable to distinguish distinctive values due to the slight variations.

6.3 Chapter Summary

The main tests performed on the system were the automation level testing and vehicle performance testing with regards to steering and speed. Results achieved from the automation level testing were positive with the level achieved averaging 94% across all tracks. It achieved 100% automation on track one with 88.75% on track six. This was expected as track size is the most complex track and with it being unseen. The levels achieved compared favourably to similar studies such as Ungrean who posted 88% automation with his test performed for a shorter amount of time.

The results achieved regarding speed tests were positive with the vehicle stopping at designated stop markers. Results did highlight that across all track the vehicle has difficulties distinguishing between speed class two and three, and five and four. It did not predict a speed four or two which

prevented gradual stops when approaching the stop markers. This was further investigated with the problem occurring with images being too similar in the dataset. This was additionally apparent from the ROC - AUC where these classes were misclassified the most.

Steering results showed that overall the system drives smoothly with good driving considered on the majority of track positions. The vehicle performed best on straight portions of the track which is expected as these are the most simple manoeuvre with less rectifying involved. The vehicle does struggle in instances with sharper bends where it understeers. Although initially thought of as a hardware problem, further investigating led to that the data set was biased towards slighter turns. This may be causing the model to predict the lighter turns due to it being difficult to distinguish between the image sets. Overall results show the vehicle performs well and generalises well to new data, highlighted in results achieved on unseen tracks.

Chapter 7

Conclusion and Future Work

Machine learning provides a computer system to make predictions without the requirement to be explicitly programmed. The growth in this field has led to autonomous driving becoming a reality of a not too distant future. Techniques such as Deep Neural Networks have catapulted the use of machine learning in fields such as image recognition. The main goal of this project was to develop a system simulating the workings of an autonomous vehicle utilising a machine learning algorithm. This was proposed by utilising a camera to predict speed and steering on several test tracks, with the vehicle being controlled and trained by a web application. The aim to be achieved was measuring the level of automation and how well the vehicle drives.

Firstly the research literature in the field of machine learning concerning autonomous vehicles was reviewed. The emergence of Deep learning in the field of autonomous vehicles is apparent due to the sheer volume of data autonomous vehicles require. Furthermore, additional research was performed in this area which found that one of the networks which have been applied to image recognition are Convolutional Neural Networks (CNNs). CNN's have come to fruition in various applications including autonomous vehicles, which have been used to mainly predict steering angles using regression-based models. As the goal for this project was to control both speed and steering it was decided to adopt two CNN models. A classification model for predicting speed and a regression model for steering.

The full system was developed and designed adopting a custom methodology which applied various approaches from both agile and traditional. The methodology was divided into the following stages, requirements, design, implementation and testing. The web application was developed to control and train the vehicle remotely and consisted of a PostgreSQL database, Spring API and React client. Additionally, user authentication was achieved, with registered users provided manual control and administrative users being provided with the addition of training control. The vehicle system consisted of a hardware system which was controlled through a Raspberry Pi fitted to a

vehicle chassis with a DC motor for speed control and a servo for steering control. This system utilised the two CNNs implemented which were developed using TensorFlow and Keras. This functionality was exposed through a Flask API.

Tests were performed on the implemented systems performance, on training tracks and two unseen tracks. There were two main tests performed these being tests to measure the automation levels achieved and how well the vehicle drives. The method adopted for testing the automation was to measure the level of human intervention undertaken over 20 minutes. The vehicle showed a high level of automation with results showing it can drive autonomously on the most complex track, over 88% of the time without human intervention. This is regarded as high automation and compared favourably to similar studies [71].

The vehicles performance was tested through two variables, speed and steering. The steering was measured through the distance the vehicle deviates from the centerline of the track. This was performed by pausing the prediction process at intervals where the distance from the centreline could be manually measured. This was performed for a total of three times on all tracks. Results show that the vehicle drives well on the majority of the tracks tested. The best performance occurred on straight tracks and tracks which featured light bends. This is with the exception of tracks which feature sharper bends, where the vehicle tends to understeer and deviates a greater distance from the centre of the track. However, this is not considered bad driving and the vehicle remains a fair distance from the edge of the track.

Speed was measured by the use of boundary markings, where the expected speed was marked at intervals on the track. The vehicles predicted speed was compared with the expected speed. This was performed a total of three times on each track. Results show that the system predicts speed accurately, with it stopping and slowing down for the positioned stop markers on all tracks. However, the vehicle does tend to miss-classify particular speed classes. This being due to the image dataset having classes which are difficult to distinguish between.

From the results achieved it is believed that CNN models excel at image processing and can be applied to autonomous vehicles in lane marking detection. However, a lot of training data and time is required to make the systems robust and accurate. The final system is regarded to drive considerably well on the majority of tracks, with it predicting the speed and steering effectively. However, there are improvements which can further enhance the project. Although improvements can be made, it is believed this research can be expanded and aid as a resource for similar projects in the field of autonomous vehicles.

7.1 Future Work

The following are a list of recommendations and future work:

- Training data: One of the main ways to improve the system is to collect additional training data. This could be performed on the current tracks or new track layouts. By collecting additional data it would make the system more robust and accurate. One of the main inaccuracies that results showed was the vehicle understeering on sharper bends. This was believed to stem from the data bias, with not enough training data captured on sharper bends. To overcome additional data on sharper bends could be collected.
- Obstacle detection: Obstacle detection could be an improvement which would allow the vehicle to respond to obstacles and avoid them. This could be done by implementing object detection through a classification network, allowing the vehicle to classify an object and take the appropriate manoeuvre to avoid a collision. Alternatively, this could be implemented with extra hardware including a RADAR and ultrasonic sensor fitted to the vehicle, allowing extra information such as distance to be easily read.
- Sign detection: One of the problems the vehicle encountered was the misclassification of speed in certain instances. This was believed to come from speed classes being too similar. To overcome this the vehicles classification network could be trained to classify signs. This would allow the vehicle to set the appropriate speed for a particular speed zone and to stop for stop signs and traffic lights.
- Implementing alternative networks: A recommendation for this project is to implement the same system using other networks. Networks which could be implemented are feed-forward neural networks, Recurrent Neural Networks or to apply a technique such as reinforcement learning. The results achieved could be compared with this project.
- Improving steering: The accuracy of the steering can be improved in terms of how well it drives. A way to overcome this could be to use a form of reinforcement learning which has been used successfully for similar studies such as refining driving styles [69]. This could be implemented on top of the implemented network or separately with a comparison made on each system.

Reflection

During the initial phases of the project, there were many positive thoughts and developments. I was excited to start the project as the field of AI and machine learning is growing and current. This mainly due to the variety of applications which umbrella under AI. Applying this field to autonomous vehicles was also exciting as I already had a strong interest in this field performing an undergraduate project in this area. Although I was very enthusiastic in the earlier stages of the research I did encounter several difficulties.

One of these difficulties faced upon at the beginning was the sheer amount of information found within this rapidly growing subject area. This was initially perceived as a good thing as I could gather a lot of reference material. However, as I continued my research I found myself being led down various routes which were not entirely applicable, finding myself distracted with other related research areas. I found this disheartening and felt I wasted too much research time reading up on areas which were not entirely that applicable to my project. This unpredicted circumstance meant that I was already behind schedule, as originally planned from my Gantt chart. This shows that I did not plan my time well as I did not rely enough on the Gantt chart. To overcome this I decided to update and use my Gantt chart effectively. Additionally, I utilized Google calendar to set myself goals each day via notifications. This pushed me to go in the right direction by focusing on relevant studies. This has helped me personally manage my time more effectively in all circumstances.

Another part of the research I found overwhelming was the mathematics being discussed. Due to me being completely new to machine learning and AI, I was initially overwhelmed by the amount and complexity of mathematics involved in explaining the theory behind the fields. This affected me negatively and I found it at times demotivating. It was until I concentrated on the basic concepts rather than the complex maths behind each system, my morale and positivity rose and I started to enjoy the research. Moreover, I realized that the concepts were crucial to me as an engineer. Understanding how a machine learning algorithm worked with key principles helped me with implementation and optimization, something which deep mathematics may not have added.

After the initial research stages, I was informed that my current supervisor at the time would be leaving and I would be handed over to a new supervisor. This initially was concerning and

was taken as a negative. This was due to me building a working relationship with the supervisor, where they had given me good direction and advice. However, even with the initial concerns, this transition was smoother than anticipated. I would like to credit my supervisor for a seamless translation, providing me with good constructive feedback and advice, which contributed to the completion of the project. This has taught me not to think negatively in given circumstances.

During the development stages I encountered a few problems. For this stage, I had to implement a system with unfamiliar new technologies including React, Keras, TensorFlow and Python. The majority of the problems occurred when implementing the CNN models through TensorFlow and Keras. I had to rectify faults for problems, such as the shaping of tensors and NumPy arrays, incorrect number of outputs on the CNN, etc. I had to research to solve these problems during the implementation stages which wasted a lot of development time. This affected me negatively as I fell further behind on my project plan. Looking back, these were simplistic problems which could have easily been resolved and avoided if I allocated myself time to learn these new technologies before starting development. I believe this would have helped greatly and would have allowed me to get around some of these problems much more efficiently.

Concerning the previous problem was the technology selection of the web application. I chose to use React which I had to learn whilst developing the system. However, the system could have been implemented more efficiently using a familiar technology such as vanilla JavaScript for the client. Although this was time-consuming and demotivating, it shows that I can quickly learn new technologies and adapt. In the end, this was a positive factor which strived me to complete the project. Additionally, learning these technologies can be implemented in future endeavours.

Overall I found that this project has challenged me to the best of my abilities and has led me to gain a big interest in machine learning and AI, with the hope to continually grow my knowledge in this area. Additionally, this project has shown me to always be confident in my abilities, as I have shown the capability to self learn and apply myself to achieve desired results. I am extremely proud of what I have achieved in this project.

References

- [1] L. Fridman, D. E. Brown, M. Glazer, W. Angell, S. Dodd, B. Jenik, J. Terwilliger, A. Patsekin, J. Kindelsberger, L. Ding, S. Seaman, A. Mehler, A. Sipperley, A. Pettinato, B. D. Seppelt, L. Angell, B. Mehler, and B. Reimer, “Mit advanced vehicle technology study: Large-scale naturalistic driving study of driver behavior and interaction with automation,” IEEE Access, vol. 7, pp. 102 021–102 038, 2019.
- [2] T. Vanderbilt, Traffic: Why we drive the way we do (and what it says about us). Penguin Books Limited, 2009.
- [3] S. Singh, “Critical reasons for crashes investigated in the national motor vehicle crash causation survey,” Tech. Rep., 2015.
- [4] B. Dickson. The predictions were wrong: Self-driving cars have a long way to go. [Online; accessed 2019-04-01]. [Online]. Available: www.pcmag.com/commentary/366394/the-predictions-were-wrong-self-driving-cars-have-a-long-wa
- [5] A. Paladini, “End-to-end models for lane centering in autonomous driving,” 2018.
- [6] E. D. Dickmanns and V. Graefe, “Applications of dynamic monocular machine vision,” Machine vision and Applications, vol. 1, no. 4, pp. 241–261, 1988.
- [7] K. Bimraw, “Autonomous cars: Past, present and future - a review of the developments in the last century, the present scenario and the expected future of autonomous vehicle technology,” ICINCO 2015 - 12th International Conference on Informatics in Control, Automation and Robotics, Proceedings, vol. 1, pp. 191–198, 01 2015.
- [8] A. Ziebinski, R. Cupek, D. Grzechca, and L. Chruszczyk, “Review of advanced driver assistance systems (adas),” in AIP Conference Proceedings, vol. 1906, no. 1. AIP Publishing, 2017.
- [9] P. Warrendale. Sae international releases updated visual chart for its ,“levels of driving automation,” standard for self-driving vehicles. [Online; accessed 2018-03-05]. [Online].

- Available: www.sae.org/news/press-room/2018/12/sae-international-releases-updated-visual-chart-for-its-%E2%80%9Clevels-of-driving-automation%E2%80%9D-standard-for-self-driving-vehicles
- [10] U. S. D. of Transportation. Automated driving systems: A vision for safety. [Online; accessed 2018-03-01]. [Online]. Available: www.nhtsa.gov/sites/nhtsa.dot.gov/files/documents/13069a-ads2.0_090617_v9a_tag.pdf
- [11] R. A. Daziano, M. Sarrias, and B. Leard, “Are consumers willing to pay to let cars drive for them? analyzing response to autonomous vehicles,” Transportation Research Part C: Emerging Technologies, vol. 78, pp. 150–164, 2017.
- [12] T. Litman, Autonomous Vehicle Implementation Predictions. Victoria Transport Policy Institute Victoria, Canada, 2017.
- [13] C. Rödel, S. Stadler, A. Meschtscherjakov, and M. Tscheligi, “Towards autonomous cars: the effect of autonomy levels on acceptance and user experience,” in Proceedings of the 6th International Conference on Automotive User Interfaces and Interactive Vehicular Applications. ACM, 2014, pp. 1–8.
- [14] M. Nikowitz, Fully Autonomous Vehicles: Visions of the Future Or Still Reality? epubli GmbH, 2015.
- [15] J. Stilgoe, “Machine learning, social learning and the governance of self-driving cars,” Social studies of science, vol. 48, no. 1, pp. 25–56, 2018.
- [16] M. Daily, S. Medasani, R. Behringer, and M. Trivedi, “Self-driving cars,” Computer, vol. 50, no. 12, pp. 18–23, 2017.
- [17] S. Carpenter, “Autonomous vehicle radar:improving radar performance with simulation,” ANSYS Advantage, vol. 12, no. 1, pp. 32–37, 2018.
- [18] M. Gerla, E.-K. Lee, G. Pau, and U. Lee, “Internet of vehicles: From intelligent grid to autonomous cars and vehicular clouds,” in 2014 IEEE world forum on internet of things (WF-IoT). IEEE, 2014, pp. 241–246.
- [19] K. Jo, J. Kim, D. Kim, C. Jang, and M. Sunwoo, “Development of autonomous car,Äipart ii: A case study on the implementation of an autonomous driving system based on distributed architecture,” IEEE Transactions on Industrial Electronics, vol. 62, no. 8, pp. 5119–5132, 2015.
- [20] M. Boden, AI: Its Nature and Future. Oxford University Press, 2016.

- [21] S. J. Russell and P. Norvig, Artificial intelligence: a modern approach. Malaysia; Pearson Education Limited,, 2016.
- [22] B. G. Buchanan, “A (very) brief history of artificial intelligence,” Ai Magazine, vol. 26, no. 4, pp. 53–53, 2005.
- [23] S. S. Haykin et al., Neural networks and learning machines/Simon Haykin. New York: Prentice Hall,, 2009.
- [24] M. Campbell, A. J. Hoane Jr, and F.-h. Hsu, “Deep blue,” Artificial intelligence, vol. 134, no. 1-2, pp. 57–83, 2002.
- [25] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski et al., “Human-level control through deep reinforcement learning,” Nature, vol. 518, no. 7540, p. 529, 2015.
- [26] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton et al., “Mastering the game of go without human knowledge,” Nature, vol. 550, no. 7676, p. 354, 2017.
- [27] M. I. Jordan and T. M. Mitchell, “Machine learning: Trends, perspectives, and prospects,” Science, vol. 349, no. 6245, pp. 255–260, 2015.
- [28] D. E. O’Leary, “Artificial intelligence and big data,” IEEE Intelligent Systems, vol. 28, no. 2, pp. 96–99, 2013.
- [29] A. L. Samuel, “Some studies in machine learning using the game of checkers,” IBM Journal of Research and Development, vol. 3, pp. 210–229, 1959.
- [30] R. Michalski, J. Carbonell, and T. Mitchell, Machine Learning: An Artificial Intelligence Approach. Elsevier Science, 2014, no. v. 1.
- [31] H. Brink, J. Richards, and M. Fetherolf, Real-world machine learning. Manning Publications Co., 2016.
- [32] E. Alpaydin, Introduction to Machine Learning, ser. Adaptive Computation and Machine Learning series. MIT Press, 2009.
- [33] E. Brynjolfsson and T. Mitchell, “What can machine learning do? workforce implications,” Science, vol. 358, no. 6370, pp. 1530–1534, 2017.
- [34] S. Marsland, Machine learning: an algorithmic perspective. Chapman and Hall/CRC, 2014.

- [35] A. Singh, N. Thakur, and A. Sharma, “A review of supervised machine learning algorithms,” in 2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom). IEEE, 2016, pp. 1310–1315.
- [36] K.-H. Jung, “A study on machine learning for steganalysis,” 05 2019.
- [37] J. E. T. Akinsola, “Supervised machine learning algorithms: Classification and comparison,” International Journal of Computer Trends and Technology (IJCTT), vol. 48, pp. 128 – 138, 06 2017.
- [38] S. Suthaharan, Machine Learning Models and Algorithms for Big Data Classification: Thinking with Examples for Effective Learning, ser. Integrated Series in Information Systems. Springer US, 2015.
- [39] A. Giusti, J. Guzzi, D. C. Cireşan, F.-L. He, J. P. Rodríguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. Di Caro et al., “A machine learning approach to visual perception of forest trails for mobile robots,” IEEE Robotics and Automation Letters, vol. 1, no. 2, pp. 661–667, 2015.
- [40] P. Navarro, C. Fernandez, R. Borraz, and D. Alonso, “A machine learning approach to pedestrian detection for autonomous vehicles using high-definition 3d range data,” Sensors, vol. 17, no. 1, p. 18, 2017.
- [41] L. Kurnianggoro, D.-W. Seo, K.-H. Jo et al., “Visual perception of traffic sign for autonomous vehicle using k-nearest cluster neighbor classifier,” in 2014 11th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI). IEEE, 2014, pp. 672–675.
- [42] Ü. Dogan, J. Edelbrunner, and I. Iossifidis, “Autonomous driving: A comparison of machine learning techniques by means of the prediction of lane change behavior,” in 2011 IEEE International Conference on Robotics and Biomimetics. IEEE, 2011, pp. 1837–1843.
- [43] R. Sunil. 7 regression techniques you should know! [Online; accessed 2019-09-30]. [Online]. Available: www.analyticsvidhya.com/blog/2015/08/comprehensive-guide-regression/
- [44] W. Wachenfeld and H. Winner, “Do autonomous vehicles learn?” in Autonomous Driving. Springer, 2016, pp. 451–471.
- [45] P. Flach, Machine Learning: The Art and Science of Algorithms that Make Sense of Data. Cambridge University Press, 2012.
- [46] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, “Unsupervised learning of hierarchical representations with convolutional deep belief networks,” Communications of the ACM, vol. 54, no. 10, pp. 95–103, 2011.

- [47] C. Finn, I. Goodfellow, and S. Levine, “Unsupervised learning for physical interaction through video prediction,” in Advances in neural information processing systems, 2016, pp. 64–72.
- [48] S. Mishra. Unsupervised learning and data clustering. [Online; accessed 2019-09-27]. [Online]. Available: towardsdatascience.com/unsupervised-learning-and-data-clustering-eeecb78b422a
- [49] M. Celebi and K. Aydin, Unsupervised Learning Algorithms. Springer International Publishing, 2016. [Online]. Available: <https://books.google.co.uk/books?id=1M8YDAAAQBAJ>
- [50] W. Maddern, A. Stewart, C. McManus, B. Upcroft, W. Churchill, and P. Newman, “Illumination invariant imaging: Applications in robust vision-based localisation, mapping and classification for autonomous vehicles,” in Proceedings of the Visual Place Recognition in Changing Environments Workshop, IEEE International Conference on Robotics and Automation (ICRA), Hong Kong, China, vol. 2, 2014, p. 3.
- [51] J. Yang, D. Parikh, and D. Batra, “Joint unsupervised learning of deep representations and image clusters,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 5147–5156.
- [52] K. J. Cios, R. W. Swiniarski, W. Pedrycz, and L. A. Kurgan, Unsupervised Learning: Association Rules. Boston, MA: Springer US, 2007.
- [53] A. Garg. Complete guide to association rules. [Online; accessed 2019-10-02]. [Online]. Available: towardsdatascience.com/association-rules-2-aa9a77241654
- [54] T. Hastie, R. Tibshirani, and J. Friedman, Unsupervised Learning. New York, NY: Springer New York, 2009, pp. 485–585.
- [55] L. Ellis, N. Pugeault, K. Öfjäll, J. Hedborg, R. Bowden, and M. Felsberg, “Autonomous navigation and sign detector learning,” in 2013 IEEE Workshop on Robot Vision (WORV). IEEE, 2013, pp. 144–151.
- [56] X. Zhu, “Semi-supervised learning,” Encyclopedia of Machine Learning and Data Mining, pp. 1142–1147, 2017.
- [57] X. J. Zhu, “Semi-supervised learning literature survey,” University of Wisconsin-Madison Department of Computer Sciences, Tech. Rep., 2005.
- [58] S. Shalev-Shwartz, S. Shammah, and A. Shashua, “Safe, multi-agent, reinforcement learning for autonomous driving,” arXiv preprint arXiv:1610.03295, 2016.

- [59] A. E. Sallab, M. Abdou, E. Perot, and S. Yogamani, “Deep reinforcement learning framework for autonomous driving,” Electronic Imaging, vol. 2017, no. 19, pp. 70–76, 2017.
- [60] M. Lapan, Deep Reinforcement Learning Hands-On: Apply modern RL methods, with deep Q-networks, value iteration, policy gradients, TRPO, AlphaGo Zero and more. Packt Publishing, 2018.
- [61] S. Marsland, Machine Learning: An Algorithmic Perspective, Second Edition, ser. Chapman & Hall. CRC Press, 2014. [Online]. Available: <https://books.google.co.uk/books?id=6GvSBQAAQBAJ>
- [62] D. A. Dugaev, I. Matveev, E. Siemens, and V. Shuvalov, “Adaptive reinforcement learning-based routing protocol for wireless multihop networks,” 10 2018, pp. 209–218.
- [63] R. S. Sutton and A. G. Barto, “Reinforcement learning: An introduction,” 2011.
- [64] M. Yogeswaran and S. G. Ponnambalam, “Reinforcement learning: exploration–exploitation dilemma in multi-agent foraging task,” OPSEARCH, vol. 49, no. 3, pp. 223–236, Sep 2012.
- [65] V. Derhami, V. J. Majd, and M. N. Ahmadabadi, “Exploration and exploitation balance management in fuzzy reinforcement learning,” Fuzzy sets and systems, vol. 161, no. 4, pp. 578–595, 2010.
- [66] J. Kober, J. A. Bagnell, and J. Peters, “Reinforcement learning in robotics: A survey,” The International Journal of Robotics Research, vol. 32, no. 11, pp. 1238–1274, 2013.
- [67] G. Zheng, F. Zhang, Z. Zheng, Y. Xiang, N. J. Yuan, X. Xie, and Z. Li, “Drn: A deep reinforcement learning framework for news recommendation,” in Proceedings of the 2018 World Wide Web Conference. International World Wide Web Conferences Steering Committee, 2018, pp. 167–176.
- [68] M. S. Emigh, E. G. Kriminger, A. J. Brockmeier, J. C. Príncipe, and P. M. Pardalos, “Reinforcement learning in video games using nearest neighbor interpolation and metric learning,” IEEE Transactions on Computational Intelligence and AI in Games, vol. 8, no. 1, pp. 56–66, 2014.
- [69] M. Kuderer, S. Gulati, and W. Burgard, “Learning driving styles for autonomous vehicles from demonstration,” in 2015 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2015, pp. 2641–2646.
- [70] I. Goodfellow, Y. Bengio, and A. Courville, Deep learning. MIT press, 2016.

- [71] U. David, “Deeprcar: An autonomous car model,” Faculty of Information Technology CTU in Prague, Tech. Rep., 2018.
- [72] R. Vargas, A. Mosavi, and R. Ruiz, “Deep learning: A review,” Advances in Intelligent Systems and Computing, vol. 5, 2017.
- [73] R. K. Srivastava, K. Greff, and J. Schmidhuber, “Training very deep networks,” in Advances in neural information processing systems, 2015, pp. 2377–2385.
- [74] M. Ganji, L. Rashidi, B. Fung, and C. Wang, Trends and Applications in Knowledge Discovery and Data Mining: PAKDD 2018 Workshops, BDASC, BDM, ML4Cyber, PAISI, DaMEMO, Melbourne, VIC, Australia, June 3, 2018, Revised Selected Papers, ser. Lecture Notes in Computer Science. Springer International Publishing, 2018.
- [75] J. Rosa, Artificial Neural Networks: Models and Applications. IntechOpen, 2016.
- [76] K. Priddy and P. Keller, Artificial Neural Networks: An Introduction, ser. SPIE tutorial texts. SPIE Press, 2005.
- [77] D. Svozil, V. Kvasnicka, and J. Pospichal, “Introduction to multi-layer feed-forward neural networks,” Chemometrics and intelligent laboratory systems, vol. 39, no. 1, pp. 43–62, 1997.
- [78] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in Advances in neural information processing systems, 2012, pp. 1097–1105.
- [79] Y. Zhang, M. Pezeshki, P. Brakel, S. Zhang, C. L. Y. Bengio, and A. Courville, “Towards end-to-end speech recognition with deep convolutional neural networks,” arXiv preprint arXiv:1701.02720, 2017.
- [80] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, “Convolutional neural networks: an overview and application in radiology,” Insights into imaging, vol. 9, no. 4, pp. 611–629, 2018.
- [81] A. Graves, A.-r. Mohamed, and G. Hinton, “Speech recognition with deep recurrent neural networks,” in 2013 IEEE international conference on acoustics, speech and signal processing. IEEE, 2013, pp. 6645–6649.
- [82] D. P. Mandic and J. Chambers, Recurrent neural networks for prediction: learning algorithms, architectures and stability. John Wiley & Sons, Inc., 2001.
- [83] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” in Proceedings of the thirteenth international conference on artificial intelligence and statistics, 2010, pp. 249–256.

- [84] R. Pascanu, T. Mikolov, and Y. Bengio, “On the difficulty of training recurrent neural networks,” in International conference on machine learning, 2013, pp. 1310–1318.
- [85] B. Curry and P. H. Morgan, “Model selection in neural networks: some difficulties,” European Journal of Operational Research, vol. 170, no. 2, pp. 567–577, 2006.
- [86] K. Chakraborty, S. Bhattacharyya, R. Bag, and A. E. Hassanien, “Comparative sentiment analysis on a set of movie reviews using deep learning approach,” in International Conference on Advanced Machine Learning Technologies and Applications. Springer, 2018, pp. 311–318.
- [87] J. Schmidhuber, “Deep learning in neural networks: An overview,” Neural networks, vol. 61, pp. 85–117, 2015.
- [88] S. M. Basha and D. S. Rajput, “Chapter 9 - survey on evaluating the performance of machine learning algorithms: Past contributions and future roadmap,” in Deep Learning and Parallel Computing Environment for Bioengineering Systems, A. K. Sangaiah, Ed. Academic Press, 2019, pp. 153 – 164.
- [89] P. Flach, “Performance evaluation in machine learning: The good, the bad, the ugly and the way forward,” in 33rd AAAI Conference on Artificial Intelligence, 2019.
- [90] N. Japkowicz and M. Shah, Evaluating Learning Algorithms: A Classification Perspective. Cambridge University Press, 2011.
- [91] D. Vora and K. Iyer, “Evaluating the effectiveness of machine learning algorithms in predictive modelling,” International Journal of Engineering & Technology, vol. 7, p. 197, 06 2018.
- [92] J. Davis and M. Goadrich, “The relationship between precision-recall and roc curves,” in Proceedings of the 23rd international conference on Machine learning, 2006, pp. 233–240.
- [93] L. Li, W.-L. Huang, Y. Liu, N.-N. Zheng, and F.-Y. Wang, “Intelligence testing for autonomous vehicles: A new approach,” IEEE Transactions on Intelligent Vehicles, vol. 1, no. 2, pp. 158–166, 2016.
- [94] M. Bojarski, P. Yeres, A. Choromanska, K. Choromanski, B. Firner, L. Jackel, and U. Muller, “Explaining how a deep neural network trained with end-to-end learning steers a car,” arXiv preprint arXiv:1704.07911, 2017.
- [95] T. M. Mitchell, Machine Learning, 1st ed. New York, NY, USA: McGraw-Hill, Inc., 1997.
- [96] A. Müller, M. C, and S. Guido, Introduction to Machine Learning with Python: A Guide for Data Scientists. O’Reilly Media, 2016.

- [97] N. Heath. The top 10 programming languages for machine learning. [Online; accessed 2019-09-09]. [Online]. Available: www.techrepublic.com/article/github-the-top-10-programming-languages-for-machine-learning
- [98] A. Géron, Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. O'Reilly Media, 2017.
- [99] M. Schrimpf, "Should i use tensorflow," 11 2016.
- [100] N. Ketkar, "Introduction to pytorch," in Deep learning with python. Springer, 2017, pp. 195–208.
- [101] V. Subramanian, Deep Learning with PyTorch: A practical approach to building neural network models using PyTorch. Packt Publishing Ltd, 2018.
- [102] A. Katte. 9 reasons why pytorch will become your favourite deep learning tool. [Online; accessed 2019-09-20]. [Online]. Available: analyticsindiamag.com/9-reasons-why-pytorch-will-become-your-favourite-deep-learning-tool
- [103] N. J. Goodall, "Can you program ethics into a self-driving car?" IEEE Spectrum, vol. 53, no. 6, pp. 28–58, 2016.
- [104] R. Sparrow and M. Howard, "When human beings are like drunk robots: Driverless vehicles, ethics, and the future of transport," Transportation Research Part C: Emerging Technologies, vol. 80, pp. 206–215, 2017.
- [105] P. Koopman and M. Wagner, "Challenges in autonomous vehicle testing and validation," SAE International Journal of Transportation Safety, vol. 4, no. 1, pp. 15–24, 2016.
- [106] N. Kalra and S. M. Paddock, "Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability?" Transportation Research Part A: Policy and Practice, vol. 94, pp. 182–193, 2016.
- [107] R. Elezaj. How apps are changing the driving experience. [Online; accessed 2019-09-20]. [Online]. Available: <https://www.automotive-iq.com/autonomous-drive/articles/how-apps-are-changing-the-driving-experience>
- [108] M. Dikmen and C. M. Burns, "Autonomous driving in the real world: Experiences with tesla autopilot and summon," in Proceedings of the 8th international conference on automotive user interfaces and interactive vehicular applications. ACM, 2016, pp. 225–228.
- [109] P. Shah. Hyundai 'digital key': Now use smartphone to start and lock/ unlock car! [Online; accessed 2019-04-03]. [Online].

- Available: www.msn.com/en-in/autos/practicalmotoring/hyundai-digital-key-now-use-smartphone-to-start-and-lock-unlock-car/ar-BBUnUCv
- [110] L. Kang, W. Zhao, B. Qi, and S. Banerjee, "Augmenting self-driving with remote control: Challenges and directions," in Proceedings of the 19th International Workshop on Mobile Computing Systems & Applications. ACM, 2018, pp. 19–24.
- [111] J. Liu, C. Yan, and W. Xu, "Can you trust autonomous vehicles: Contactless attacks against sensors of self-driving vehicles," DEFCON24 (Aug. 2016), 2016.
- [112] B. G. Tudorica and C. Bucur, "A comparison between several nosql databases with comments and notes," 2011 RoEduNet International Conference 10th Edition: Networking in Education and Research, pp. 1–5, 2011.
- [113] F. Gutierrez, Introducing Spring Framework: A Primer, ser. Expert's voice in Spring. Apress, 2014.
- [114] A. Fedosejev, React.js Essentials. Packt Publishing, 2015.
- [115] A. Accomazzo, N. Murray, and A. Lerner, Fullstack React: The Complete Guide to ReactJS and Friends. Fullstack.io, 2017. [Online]. Available: <https://books.google.co.uk/books?id=ppjUtAEACAAJ>
- [116] M. Nauman, "Drivingmatter: An autonomous rc car using raspberry pi," 2018.
- [117] Z. Prof. Thorat, S. Mahadik, S. Mane, S. Mohite, and U. Aniket, "Self driving car using raspberry-pi and machine learning," International Research Journal of Engineering and Technology (IRJET), vol. 06, no. 03, pp. 969–972, 2019.
- [118] J. George, A. Sreyaj, D. Joseph, and P. Sajeer, "Android controlled smart car," International Journal of Computer Engineering In Research Trends, vol. 02, no. 12, pp. 1087–1090, 01 2015.
- [119] H. Collins, Creative Research: The Theory and Practice of Research for the Creative Industries, ser. Required Reading Range. Bloomsbury Publishing, 2017.
- [120] M. Saunders, P. Lewis, and A. Thornhill, Research Methods for Business Students. Financial Times/Prentice Hall, 2007.
- [121] M. Mora, Research Methodologies, Innovations and Philosophies in Software Systems Engineering and Information Systems, ser. Premier Reference Source. Information Science Reference, 2012.
- [122] R. Schutt, Investigating the Social World with SPSS Student Version 14.0: The Process and Practice of Research. SAGE Publications, 2006.

- [123] J. Michell, "Pragmatism, positivism and the quantitative imperative," Theory & Psychology, vol. 13, no. 1, pp. 45–52, 2003.
- [124] J. Creswell, Research Design: Qualitative, Quantitative, and Mixed Methods Approaches. SAGE Publications, 2014.
- [125] C. Williams, "Research methods," Journal of business & economic research, vol. 5, no. 3, pp. 65–72, 2007.
- [126] R. Babbie, The Practice of Social Research. Cengage Learning, 2012.
- [127] R. Richey and J. Klein, Design and Development Research: Methods, Strategies, and Issues. Taylor & Francis, 2014.
- [128] C. Wohlin, M. Höst, and K. Henningsson, Empirical Research Methods in Software Engineering, R. Conradi and A. I. Wang, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003.
- [129] M. Lázaro and E. Marcos, "Research in software engineering: Paradigms and methods." 2005.
- [130] W. Goddard and S. Melville, Research Methodology: An Introduction. Juta, 2004.
- [131] E. Ruspini, An Introduction to Longitudinal Research, ser. Social Research Today. Taylor & Francis, 2003.
- [132] A. Bryman and E. Bell, Business Research Methods. Oxford University Press, 2015.
- [133] A. Rindfleisch, A. J. Malter, S. Ganesan, and C. Moorman, "Cross-sectional versus longitudinal survey research: Concepts, findings, and guidelines," Journal of marketing research, vol. 45, no. 3, pp. 261–279, 2008.
- [134] C. Jones, Software Methodologies: A Quantitative Guide. CRC Press, 2017.
- [135] S. Balaji and M. S. Murugaiyan, "Waterfall vs. v-model vs. agile: A comparative study on sdlc," International Journal of Information Technology and Business Management, vol. 2, no. 1, pp. 26–30, 2012.
- [136] P. Abrahamsson, O. Salo, J. Ronkainen, and J. Warsta, "Agile software development methods: Review and analysis," arXiv preprint arXiv:1709.08439, 2017.
- [137] R. Kaur and J. Sengupta, "Software process models and analysis on failure," in of Software Development Projects, International Journal of Scientific & Engineering Research. Citeseer, 2011.

- [138] J. Patton and P. Economy, User Story Mapping: Discover the Whole Story, Build the Right Product. O'Reilly Media, 2014.
- [139] F. Tsui, O. Karam, and B. Bernal, Essentials of Software Engineering. Jones & Bartlett Learning, 2013.
- [140] J. Schmuller, Sams Teach Yourself UML in 24 Hours, ser. Sams Teach Yourself Series. Sams, 2004.
- [141] H. Munir, M. Moayyed, and K. Petersen, "Considering rigor and relevance when evaluating test driven development: A systematic review," Information and Software Technology, vol. 56, no. 4, pp. 375–394, 2014.
- [142] raspberrypi.org. Raspberry pi 3 model b+. [Online; accessed 2019-11-04]. [Online]. Available: <https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/>
- [143] seeedstudio.com. Robot car kit- rc smart car chassis kit. [Online; accessed 2019-11-04]. [Online]. Available: <https://www.seeedstudio.com/Robot-car-Kit-RC-Smart-Car-Chassis-p-4226.html>
- [144] robotshop.com. Raspberry pi camera module (g) w/ fisheye lens. [Online; accessed 2019-11-04]. [Online]. Available: <https://www.robotshop.com/uk/raspberry-pi-camera-module-g-fisheye-lens.html>
- [145] S. Fun. Sparkfun motor driver - dual tb6612fng (1a). [Online; accessed 2019-11-04]. [Online]. Available: <https://www.sparkfun.com/products/14451>
- [146] F. Akmel, E. Birhanu, B. Siraj, and S. Shifaw, "A comparative analysis on software architecture styles," International Journal in Foundations of Computer Science & Technology, vol. 7, pp. 11–22, 11 2017.
- [147] M. Richards, Software Architecture Patterns: Understanding Common Architecture Patterns and when to Use Them. O'Reilly Media, 2015.
- [148] G. Blokdyk, Rapid Application Development a Complete Guide - 2019 Edition. Emereo Pty Limited, 2019.
- [149] S. Bagui and R. Earp, Database Design Using Entity-Relationship Diagrams, Second Edition, ser. Foundations of Database Design. Taylor & Francis, 2011.
- [150] G. Blokdyk, Database Normalization A Complete Guide - 2019 Edition. 5STARCooks, 2018.

- [151] A. Khan, A. Sohail, U. Zahoor, and A. Saeed, “A survey of the recent architectures of deep convolutional neural networks,” 01 2019.
- [152] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” arXiv preprint arXiv:1409.1556, 2014.
- [153] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.
- [154] D. Rajput, Spring 5 Design Patterns: Master efficient application development with patterns such as proxy, singleton, the template method, and more. Packt Publishing, 2017.
- [155] P. Kainulainen. Understanding spring web application architecture: The classic way. [Online; accessed 2019-10-28]. [Online]. Available: <https://www.petrikainulainen.net/software-development/design/understanding-spring-web-application-architecture-the-classic-way/>
- [156] S. R. Young, D. C. Rose, T. P. Karnowski, S.-H. Lim, and R. M. Patton, “Optimizing deep learning hyper-parameters through an evolutionary algorithm,” in Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments. ACM, 2015, p. 4.
- [157] D. Marinov, “Automatic hyperparameter optimisation and neural architecture search,” 2019.
- [158] J. Bergstra and Y. Bengio, “Random search for hyper-parameter optimization,” Journal of Machine Learning Research, vol. 13, no. Feb, pp. 281–305, 2012.

Appendix A - Data Collection

Summary

Test	How to measure?	Process	How to keep the test valid?
Automation	On how much human intervention is required over a 20-minute period. Will be tested on training tracks and unseen tracks, to test how well the vehicle generalizes to new data.	- The vehicle will be set at the starting position of the track. - Once positioned the vehicle will be set in autonomous mode. - The timer will start once the vehicle makes its first move. - If the vehicle drives off the track the timer will be stopped with the vehicle positioned in the optimal place where it drove off. - Once the vehicle crosses the finish line the timer is stopped with the vehicle positioned back at the starting position. Ready to take on the track again. - Any human interventions are recorded.	The same starting position will be used for each test with it being marked before starting each run. The timer will be stopped once the any of the vehicle's wheels have crossed the line. All tracks widths will be measured to ensure they are the same distance.
Steering	Measure the distance the vehicle is deviates from the centre line of the track. Will be tested on training tracks and unseen tracks.	- The vehicle will be set at the starting position. - Once positioned vehicle will be set in autonomous mode. - The vehicle will be measured from the centre of the track by pausing the automation. - The vehicle will be measured from the centre line of the track, recording the deviation. - If the vehicle crosses the line it will be positioned in the optimal position from where it failed - The vehicle will be set at the starting position. - Once positioned vehicle will be set in autonomous mode. - Markers will be placed with the expected speed ensuring they are not interfering with the track. - The speed set will be compared with the expected speed value. - If the vehicle crosses the line it will be positioned in the optimal position from where it failed.	The same start position will be used with it being marked before each start. The tracks width will be considered with every track featuring the same width. Measurements will be taken in cm. The track layouts will also be squared, this will make measuring the centre of the track more accurate. This test will be repeated three times for each track.
Speed	Record the predicted speed against the expected speed for given portions of the track. Will be tested on training and unseen tracks.	- The vehicle will be set at the starting position. - Once positioned vehicle will be set in autonomous mode. - Markers will be placed with the expected speed ensuring they are not interfering with the track. - The speed set will be compared with the expected speed value. - If the vehicle crosses the line it will be positioned in the optimal position from where it failed.	The same starting position will be used with it being marked before each start. Markers will be positioned at regular intervals to improve accuracy and resolution of results. This test will be repeated three times for each track.

Appendix B - Use Case Descriptions

Use Case Name	Register user	Control vehicle
Actors	Unregistered user	Admin user and registered user
Process	Unregistered user accesses sign up page. Fills in details and submits the details. The new user details are validated.	User logs in access control dashboard. Selects a vehicle to control.
Tasks	- Access sign up page - Fill out registration form - Submit details to be created	- Manual control: user selects manual control. And controls vehicle manually setting speed and steering - Auto control: user selects auto control. And clicks auto mode start button
Pre-Conditions	A user with no session	- Login - Access control dashboard - Select vehicle to control
Post-Conditions	A new user is created	A vehicle control layout with options for speed and steering

Use Case Name	Manage user	Train control
Actors	Admin user	Admin user
Process	Admin user access admin menu and selects manage users. A user account is selected where updates are entered and submitted for validation.	User access admin menu selecting training control option. The admin selects a vehicle to train. Speed and steering are set and sent to the vehicle
Tasks	- Access admin menu - Select user to update - Update required filed - Submit details to be updated	- Login - Access admin menu - Select training control - Select vehicle - Set speed and steering for vehicle - Submit to vehicle
Pre-Conditions	A user with admin privileges	An admin user with credentials.
Post-Conditions	A selected user account is updated	A vehicle controlled with training mode, collecting training data

Use Case Name	Register vehicle	Train model
Actors	Registered user	Admin user
Process	Registered user logs in. Access register vehicle option. User fills in vehicle details and submits form for validation.	User access admin menu selecting training model option. The admin clicks train button. The model metrics are displayed.
Tasks	- Login - Access register vehicle option - Fill in vehicle details - Submit details to be created	- Login - Access admin menu - Select train model - Metrics are displayed from vehicle
Pre-Conditions	A registered user with login credentials	An admin user with credentials.
Post-Conditions	A new vehicle is created to the user account	A trained model

Use Case Name	Manage vehicles	
Actors	Admin user	
Process	Admin user logs in. Access manage register vehicle option. Selects vehicle to be updated. Updates fields in vehicle details and submits form for validation.	
Tasks	- Login - Access manage vehicles - Select vehicle to be updated - Submit details to be updated	
Pre-Conditions	An admin user with login credentials	
Post-Conditions	A vehicle is updated	

Appendix C - Project Plan

