End-to-End Learning for Autonomous Driving Robots

Anthony Ryan

21713293

Supervisor: Professor Thomas Bräunl

GENG5511/GENG5512 Engineering Research Project

Submitted: 25th May 2020

Word Count: 8078

Faculty of Engineering and Mathematical Sciences

School of Electrical, Electronic and Computer Engineering

Abstract

This thesis presents the development of a high-speed, low cost, end-to-end deep learning based autonomous robot driving system called ModelCar-2. This project builds on a previous project undertaken at the University of Western Australia called ModelCar, where a robotics driving system was first developed with a single LIDAR scanner as its only sensory input as a baseline for future research into autonomous vehicle capabilities with lower cost sensors. ModelCar-2 is comprised of a Traxxas Stampede RC Car, Wide-Angle Camera, Raspberry Pi 4 running UWA's RoBIOS software and a Hokuyo URG-04LX LIDAR Scanner.

ModelCar-2 aims to demonstrate how the cost of producing autonomous driving robots can be reduced by replacing expensive sensors such as LIDAR with digital cameras and combining them with end-to-end deep learning methods and Convolutional Neural Networks to achieve the same level of autonomy and performance.

ModelCar-2 is a small-scale application of PilotNet, developed by NVIDIA and used in the DAVE-2 system. ModelCar-2 uses TensorFlow and Keras to recreate the same neural network architecture as PilotNet which features 9 layers, 27,000,000 connections and 252,230 parameters. The Convolutional Neural Network was trained to map the raw pixels from images taken with a single inexpensive front-facing camera to predict the speed and steering angle commands to control the servo and drive the robot.

This end-to-end approach means that with minimal training data, the network learns to drive the robot without any human input or distance sensors. This eliminates the need for an expensive LIDAR scanner as an inexpensive camera is the only sensor required. This will also eliminate the need for any lane marking detection or path planning algorithms and improve the speed and performance of the embedded platform.

Declaration

I, Anthony Ryan, declare that:

This thesis is my own work and suitable references have been made to any sources that have been used in preparing it.

This thesis does not infringe on any copyright, trademark, patent or any other rights for any material belonging to another person.

This thesis does not contain any material which has been submitted under any other degree in my name at any other tertiary institution.

Date: 25th May 2020

Acknowledgements

I would first like to thank my supervisor, Professor Thomas Bräunl, for his advice and guidance throughout the duration of this project as well as giving me the chance to carry out my research on this topic.

I would also like to thank Omar Anwar for donating so much of his time and expertise to this project, Marcus Pham and Felix Wege for their support with the EyeBot software and Pierre-Louis Constant for his help with CNC milling components for this project.

Contents

List of Figures

List of Tables

Acronyms

ALVINN Autonomous Land Vehicle in a Neural Network

API Advanced Programming Interface

AUD Australian Dollar

CNC Computer Numerical Control

CNN Convolution Neural Network

CPU Central Processing Unit

DARPA Defence Advanced Research Projects Agency

DAVE DARPA Autonomous Vehicle

EECE Electrical, Electronic and Computer Engineering

ESC Electronic Speed Controller

ELU Exponential Linear Unit

GPIO General Purpose Input/Output

GPU Graphics Processing Unit

GUI Graphical User Interface

LIDAR Light Detection and Ranging

LSTM Long Short-Term Memory

PWM Pulse Width Modulation

RC Radio Control

RGB Red-Green-Blue

RoBIOS Robotics Basic Input/Output System

ROS Robot Operating System

SAE Society of Automotive Engineers

SLAM Simultaneous Localisation and Mapping

UWA University of Western Australia

VNC Virtual Network Computing

1 Introduction

1.1 Background

Autonomous vehicles have been increasing in popularity over the recent years as many companies are actively researching and developing new technologies that are leading to fully autonomous driving systems that require no human intervention. However, there is much debate in the industry over which of these technologies will be able to achieve full automation and make self-driving cars commercially feasible. The result of this debate could shape the path of autonomous vehicles forever.

Currently, most autonomous systems rely heavily on forms of remote sensing such as LIDAR which uses pulses of electromagnetic radiation to calculate distances to nearby objects to generate a detailed map of the environment. While driving systems using LIDAR can be accurate and have large distance and angular ranges, they are incredibly expensive and require complex algorithms with finely tuned parameters. As a result, no LIDAR based autonomous vehicles are available on the market and the testing of these vehicles is limited to certain regions.

A novel approach to producing autonomous vehicles has started to gain popularity within the industry that involves using cameras and machine learning. Neural networks are able to control vehicles by learning from data provided by human driving. Images provided from the cameras are recorded and labelled with their respective vehicle control values to serve as training and validation data for the neural network. If this neural network approach can match or improve on the autonomous driving performance of LIDAR based systems then the cost of self-driving cars is greatly reduced as inexpensive cameras are the only sensors required and expensive LIDAR sensors can be made redundant. The end-to-end control also eliminates the need for building any conditional driving algorithms and setting any pre-defined parameters.

The NVIDIA Corporation has created their own end-to-end driving neural network architecture called PilotNet, which is a Convolution Neural Network that learns to control a vehicle by using pixels from images taken with a front-facing camera and map them directly to steering commands [1]. The network learns from data recorded by human drivers.

Providing a fair comparison for both of these approaches could be very useful in supporting whether an end-to-end neural network such as NVIDIA's PilotNet is a viable method for the future of autonomous vehicle design as reducing their production time and costs could bring them closer to commercial availability.

In 2017, an autonomous robot called ModelCar was developed as part of a research project at the University of Western Australia and used a relatively expensive LIDAR scanner as input for a complex driving algorithm to navigate around corridors with impressive autonomy. The project aimed to provide a baseline for future research into autonomous driving with more cost-effective sensors [3]. The ModelCar also features a manual driving mode and possesses the necessary requirements for conducting a comparison between these two autonomous driving systems.

1.2 Problem Statement and Scope

The aim of this thesis is to continue from the ModelCar project and develop a lower-cost autonomous robot driving system called ModelCar-2 that uses an end-to-end deep learning approach. The new system will use an inexpensive front-facing digital camera and use the NVIDIA PilotNet neural network architecture. With the robot running both driving systems on the same machine, a comparison between the two approaches will be balanced and hopefully provide insight into the validity of end-to-end neural network driving.

Training data will be in the form of recorded images taken from the camera and labelled with the steering and speed servo values at the time of capture, these will be collected from both the manual and LIDAR driving modes. When sufficient data has been collected, the model will then be trained and be able to produce steering angle and speed commands in real-time based on the image that is being supplied by the camera to drive the robot autonomously.

When an optimal model has been developed, the new end-to-end ModelCar-2 system will compete against the original LIDAR ModelCar driving system in the same environment that both of them used for development and training to limit any bias. The systems will be tested for their autonomy, processing speed and cost.

All hardware and software used in the development of the ModelCar-2 robot will utilise common components and software used at the University of Western Australia, as to make further work on this project easier to extend and reproduce as well as support any educational uses of this project.

1.3 Document Structure

This thesis starts with a short review of autonomous vehicles and the uses of LIDAR and neural networks in their development, which is followed by a description of the hardware and software which will be used to assemble the ModelCar-2 robot. The final sections focus on how the end-to-end learning method operates and contains an analysis into the results of a comparison between the LIDAR and the neural network driving systems.

2 Literature Review

2.1 Levels of Autonomy

The Society of Automotive Engineers has developed a "Levels of Driving Automation" standard that describes six different levels of autonomy that rank from no automation to full automation [6].

Table 1: Description of SAE Levels of Autonomy

SAE Level of Autonomy	Description
Level 0 - No Automation	Human in full control of all aspects of driving.
Level 1 - Driver Assistance	Vehicle assists driver with only one aspect of control at a time (e.g. Cruise Control, Lane Keeping, Brake Assistance).
Level 2 – Partial Automation	The control of speed and steering is controlled by the vehicle, but driver still performs all remaining tasks (e.g. Tesla Autopilot).
Level 3 – Conditional Automation	The driving system takes complete control over driving. Human driver is expected to intervene only when required.
Level 4 – High Automation	The driving system takes complete control over driving. Human driver is not required to intervene as vehicle is able to stop and park safety (e.g. Waymo Chrysler Pacifica).
Level 5 – Full Automation	The driving system takes complete control over Driving. Human driver does not need to be inside the vehicle.

The aim for all autonomous driving research is to develop a vehicle that is classified as a Level 5, fully autonomous vehicle that is able drive without the need for any human input [18].

As of yet, the highest level of automation that has been officially awarded is a Level 4, which belongs to the Waymo Chrysler Pacifica. The Waymo fleet of vehicles make extensive use of LIDAR technology [22].

2.2 LIDAR

LIDAR is a remote sensing method that involves using light in the form of pulsing lasers to measure the distances to elements in the surrounding environment. LIDAR data allows for a HD map of the surroundings to be generated in real-time which makes it perfect for applications in self-driving cars.

LIDAR sensors feature heavily in early attempts at autonomous vehicle design and continue to be one of the most important components of the current wave as well. LIDAR sensors today are extremely accurate, reliable and can have large distance and angular ranges. However, LIDAR sensors are expensive and involve using complex algorithms and conditional logic structures to process the data for use in driving.

The current industry leader in LIDAR based autonomous vehicles is Waymo, a subsidiary of Google. The main fleet of Waymo vehicles has been able to achieve Level 3 autonomy but their newest vehicle, the Chrysler Pacifica Hybrid has been classified as having Level 4 autonomy. The Pacifica is fitted with 3 LIDAR sensors, each costing AUD $25,000 which contributes to almost half of the estimated total vehicle production cost of AUD $154,905 which is incredibly expensive compared to other vehicles on the market [7]. As of today, No Waymo vehicles are currently available to the general public and there are no plans for this to change. They are also geo-fenced to California and can't be driven autonomously outside the state.

Tesla CEO Elon Musk has repeatedly suggested that "Anyone who relies on LIDAR is doomed" and that "Expensive sensors are unnecessary" [8]. Instead of LIDAR, Tesla vehicles use much less expensive cameras together with Convolutional Neural Networks. Contrast to any LIDAR based systems, Tesla vehicles are currently available to the public, have relatively low production costs and their autonomy performance is growing faster than any of their competitors [22].

2.3 Convolutional Neural Networks

Convolutional Neural Networks are a branch of deep neural networks that are primarily used in image classification. CNNs typically involve convolution layers which are designed to perform feature extraction on images to allow them to recognise patterns. This is usually performed in a process called supervised machine learning, where models learn to map features to labels based on training data examples [17].

Companies such as Tesla are beginning to implement CNNs into autonomous driving systems.

2.3.1 End-to-End Driving

The standard approach to solving the challenge of autonomous driving has been to decompose the problem into several steps such as object, pedestrian and lane marking detection, path planning and motor control [4]. This approach is the reason for remote sensing technologies such as LIDAR being popular in the early stages of autonomous vehicle development.

With end-to-end control, this pipeline is simplified by applying neural networks to directly produce control outputs from sensory inputs without having to process data through a series of algorithms [4].

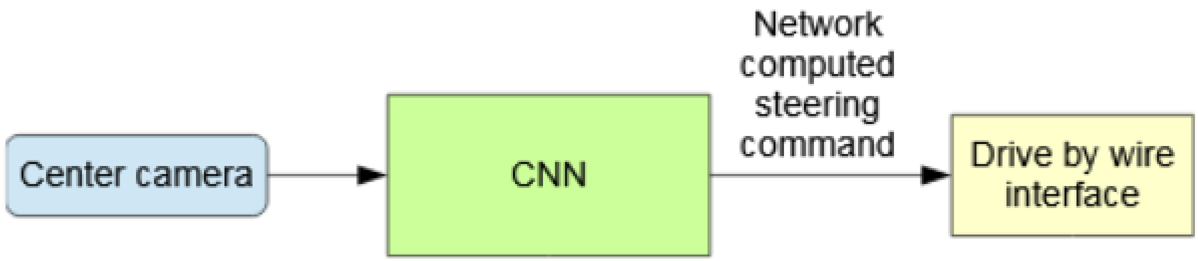

Figure 1: NVIDIA DAVE-2 End-to-End Driving Pipeline [1]

The use of neural networks for end-to-end driving and autonomous systems was first demonstrated in 1989, where the ALVINN system showed how an end-to-end trained neural network could steer a vehicle on public roads. ALVINN featured a 3-layer neural network that took images from a camera with a resolution of 30x32 and used only the blue channel, as it provided higher contrast between road and non-road pixels [10]. These images combined with distances from a laser scanner produced an output for vehicle direction. The system is considered primitive by today's standards as it only consisted of fully connected layers, but more data and computational power over time has allowed for more complex networks that make use of convolution [4].

In 2004, a DARPA project called DAVE created a 1/10th scale autonomous car that was able to drive through an obstacle course featuring different terrains. DAVE featured a 6-layer Convolution Neural Network that was trained on hours of data supplied by manual driving. The training data included images from two cameras mapped to left and right steering commands [1]. DAVE achieved an average distance of 20 metres between collisions in a complex environment [1].

2.3.2 NVIDIA PilotNet

The DAVE project served as the groundwork for NVIDIA's autonomous driving system called DAVE-2, where they implemented their state-of-the-art neural network architecture called PilotNet. In the paper *End-to-End Learning for Self-Driving Cars* [1], NVIDIA demonstrated how PilotNet was able to successfully drive a full-size vehicle on public roads by being trained on over 70 hours of human driving [1]. The DAVE-2 vehicle achieved an autonomy of 98% on flat roads with minimal traffic [2].

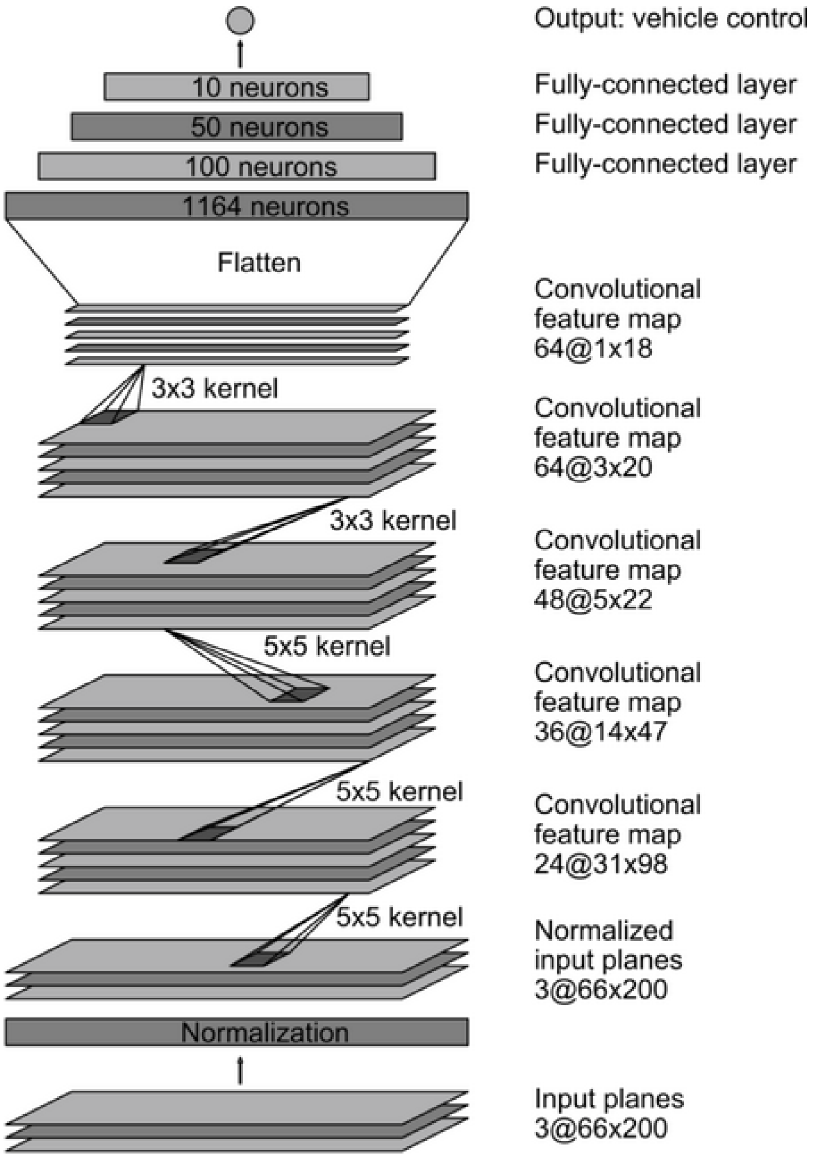

Figure 2: NVIDIA PilotNet Architecture [1]

The network consists of 9 layers; a normalisation layer, 5 convolutional layers and 3 fully connected layers. The normalisation layer operation is constant and not adjusted by training. Normalisation is performed as part of the network to benefit from GPU acceleration and is used to avoid model saturation and improve the quality of the gradients [1].

The first three convolution layers use filters of size (5x5) and stride (2x2) which are used to detect large shape features within the image such as road lines and edges. The last two layers use filters of size (3x3) and stride (1x1) which extract more nuance features in the image [3].

The input into the network is a 3 channel YUV image of size 200x66 that produces a single output of steering angle [1]. NVIDIA has stated that these parameters are a result of extensive experimentation with multiple layering configurations [1].

The neural network is trained through a process called backpropagation which calculates the weights of the model that minimise the loss or mean squared error between the steering commands from the manual driving and the model predictions [17].

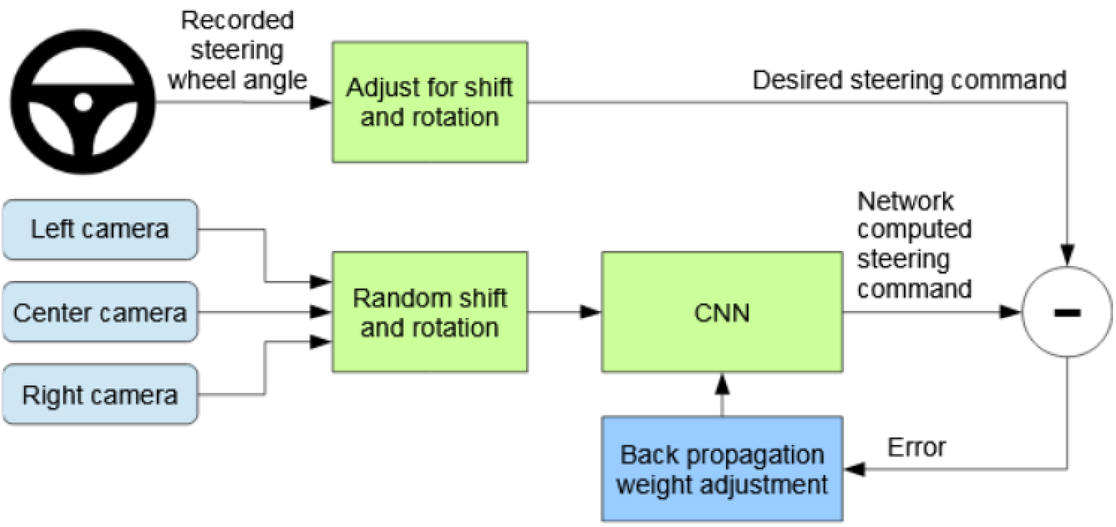

Figure 3: Training the NVIDIA DAVE-2 Network [1]

This continues until the loss is reasonably low and the model has learned to control the vehicle. This process is also called regression and is different from image classification as the output is a numerical value instead of a class type.

3 Design

This section provides an overview of the hardware and software components used in assembling the ModelCar-2 robot as well as the driving program. One the aims of this project is cost reduction and therefore most of the components used are relatively inexpensive and can be easily sourced from electronics stores and hobby shops. This project also aims to use open-source software to reduce cost but to also make reproducing and understanding the driving program easier. A full cost breakdown can be found in the results section.

3.1 Hardware

3.1.1 Robot Chassis

The Traxxas Stampede XL-5 is a commercially available, 1/10th scale, Rear-Wheel Drive, RC Car that serves as the base for the ModelCar-2 robot. The Traxxas Stampede was made available to the UWA faculty as a result of the previous ModelCar project.

Figure 4: Traxxas Stampede XL-5

The Stampede is perfect for use in this project as the ESC is separate from its signal receiver. For complete end-to-end control, the steering servo and motor needs to be connected directly to the embedded controller in order to receive commands from the neural network model.

The steering needs to have an Ackermann setup that has a servo with very fine resolutions of steering angle. There are some inexpensive hobby vehicles that only have three discrete angle values such as left, right and straight which although could still achieve autonomous driving, greater control over the steering will reduce the margins of error and smooth many aspects of the driving [4].

3.1.2 Embedded Controller

The Raspberry Pi is the most popular single board computer and although it may not be as powerful and as optimised for machine learning as others on the market such as the NVIDIA Jetson Nano or DRIVE PX (used in DAVE-2), the Raspberry Pi is cost-effective. The Pi also includes built-in Wi-Fi, Bluetooth and USB 3.0 for easy connectivity to peripherals.

Figure 5: Raspberry Pi 4 Model B

The Raspberry Pi was an obvious choice as the RoBIOS API was developed to be used with Raspberry Pi and is used in the ‘EyeBot’, the primary teaching robot at UWA. This makes further work involving the ModelCar-2 easier to implement..

3.1.3 Camera

The resolution is not the most important detail when considering what camera to use, as the images will get resized down before they are inputted through the network. Having a larger field of view is more beneficial, as more information can be captured from the surroundings in a single image and as such a wide-angle camera lens is preferred.

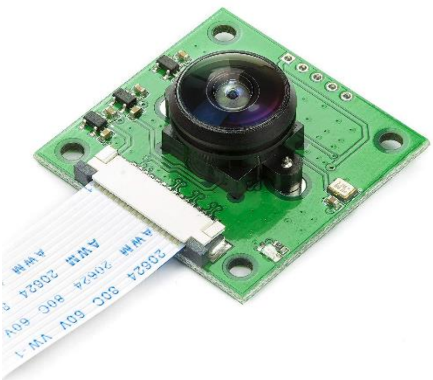

Figure 6: 5MP OV5647 Digital Camera

The 5MP OV5647 Camera Board with a LS-40180 Fisheye Lens was chosen as it is compatible with the Raspberry Pi and is cost-effective, especially compared to the LIDAR sensor that it aims to compete with. The fisheye lens also provides a 180˚ field of view as well as having an infrared filter.

3.1.4 LIDAR Unit

The Hokuyo URG-04LX-UG01 was the 2D LIDAR unit used for developing the original ModelCar project and therefore remains for the new ModelCar-2 project to allow the original LIDAR driving system to still operate. Although relatively expensive compared to the rest of the components listed, this sensor provides easy interfacing with the Raspberry Pi through the use of USB connections and open-source software.

Figure 7: Hokuyo URG-04LX-UG01 LIDAR

The sensor has a 10 Hz scan rate, 240˚ field of vision, 0.352˚ angular resolution and a range of 20-5600 mm [3].

3.1.5 Handheld Controller

To be able to control the robot manually and navigate the program menu, a wireless handheld controller is required.

Figure 8: Logitech F710 Gamepad

The Logitech F710 Gamepad uses a wireless USB receiver which can be connected to the Raspberry Pi to allow inputs from the buttons and joysticks to be recognised by the driving program.

3.1.6 Mounting Plate

As the robot can reach speeds of +35 km/h in all driving modes, a plastic mounting plate was made using CNC milling to replace the original ModelCar setup and to make sure all components are securely fitted to the robot.

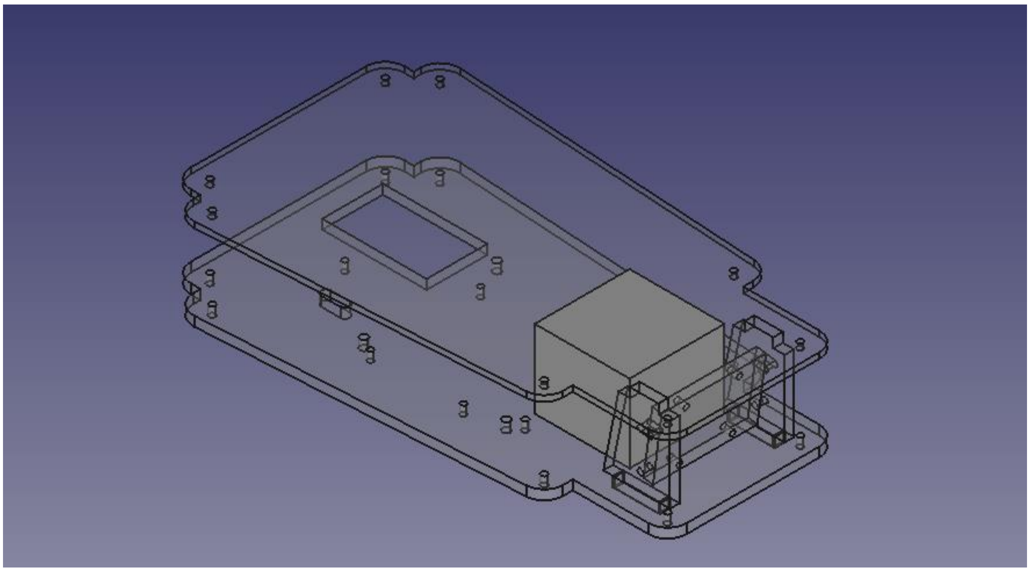

Figure 9: CAD Mounting Plate Design

The plate has several supports which allow the Raspberry Pi, LIDAR and Battery Pack to be mounted. There are two access holes, the larger is to be able to reach the ESC power switch and the smaller is to allow the servo cable to connect to the ESC module. With only removing 3 screws, the entire mounting plate can be taken off to allow easier maintenance on the robot. Along with the plate, a camera mount was also designed which can be adjusted to different angles.

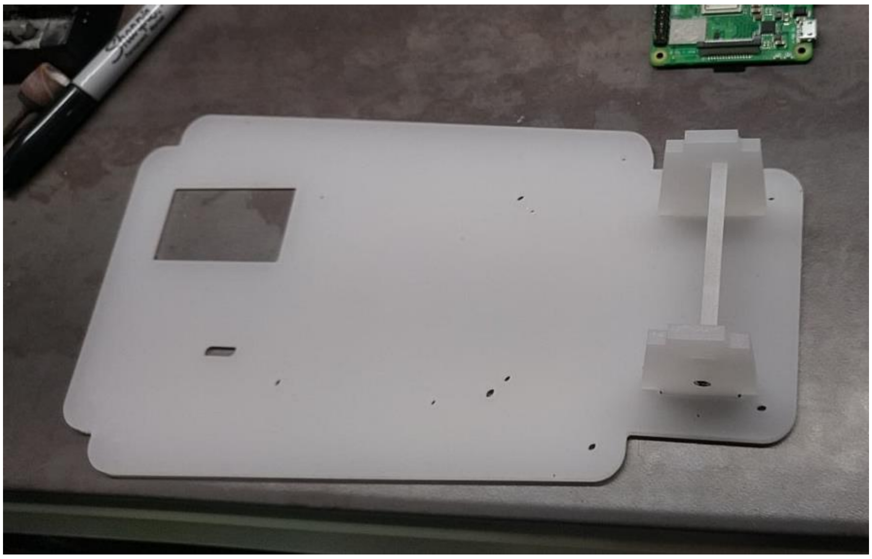

Figure 10: CNC Produced Mounting Plate

3.2 Software

3.2.1 RoBIOS

The RoBIOS-7 library is an API that is built on the latest distribution of Linux for Raspberry Pi called Raspbian Buster. The software was developed at the University of Western Australia for use with the 'EyeBot' family of mobile robots as well as the simulation package known as 'EyeSim'. RoBIOS provides an extensive library that allows simple design of robotics programs that can be written in the C, C++ or Python languages [11]. The library provides a simple Graphical User Interface that can be displayed on either an external LCD touchscreen or on virtual desktop display software such as Microsoft Remote Desktop or VNC Viewer. Information on the RoBIOS library functions can be found at:

https://robotics.ee.uwa.edu.au/eyebot/Robios7.html

3.2.2 TensorFlow/Keras

Keras is an open-source neural network library written for Python and acts as a backend for TensorFlow. Its API is simple, user-friendly and most importantly has versions that are compatible and optimised for Raspberry Pi, which should provide impressive processing speeds. Although Torch 7 was used to create the network used in NVIDIA's DAVE-2 system, Keras includes all the features required to replicate the PilotNet neural network architecture on a small-scale [1]. The repository can be found at:

https://github.com/keras-team/keras

3.2.3 OpenCV

OpenCV is an open-source computer vision and image processing library that is available for multiple processing languages including Python. Images need to be processed in real-time to be available as inputs for the neural network. These processes include resizing the image to a resolution of 200x66 as well as converting them to the YUV colour space. Therefore, to achieve successful end-to-end driving, OpenCV is a necessity for inclusion in this project. The repository can be found at:

https://github.com/opencv/opencv

3.2.4 Pygame

Pygame is an open-source Python API used for developing games and features a module for interacting with joysticks and gamepads such as the Logitech F710 Gamepad. This makes it perfect for allowing manual control of the robot through the joysticks. The repository can be found at:

https://github.com/pygame/pygame

3.2.5 ServoBlaster

The Traxxas Stampede originally used a 2.4GHz radio transmitter and receiver to send PWM signals to the ESC to control both the steering and speed of the vehicle. However, as the values for steering angle and speed will come from the driving program running on the Raspberry Pi, a new method of sending PWM signals to the ESC is required.

ServoBlaster is an open-source software package for the Raspberry Pi that provides an interface to control servos via the GPIO pins. PWM signals can be generated on any of pins according to the timing value and pin specified by the user. This approach proved to be perfect as signals can be send through Python commands, allowing for the outputs for the vehicle control to come from the neural network. The repository can be found at:

https://github.com/richardghirst/PiBits/tree/master/ServoBlaster

3.2.6 Breezy LIDAR/SLAM

Breezy is a collection of open-source libraries that supports many different LIDAR units including the Hokuyo URG-04LX [12]. BreezyLIDAR is used for receiving LIDAR scan data from the sensor and BreezySLAM is used for producing SLAM images. The API works with Python on Linux machines and is efficient on processing power, which makes it perfect for use on Raspberry Pi compared with other interfaces such as ROS. The original ModelCar used the URG library for interfacing with the LIDAR unit but this library is only available for the C language and therefore BreezyLIDAR was used instead. The repositories can be found at:

https://github.com/simondlevy/BreezyLidar

https://github.com/simondlevy/BreezySLAM

3.3 ModelCar-2

The complete robot features all the components that have been discussed as well as a USB Battery Pack to power both the Raspberry Pi and the LIDAR unit as well as a 3.5 Inch WaveShare LCD Touchscreen for displaying the user interface.

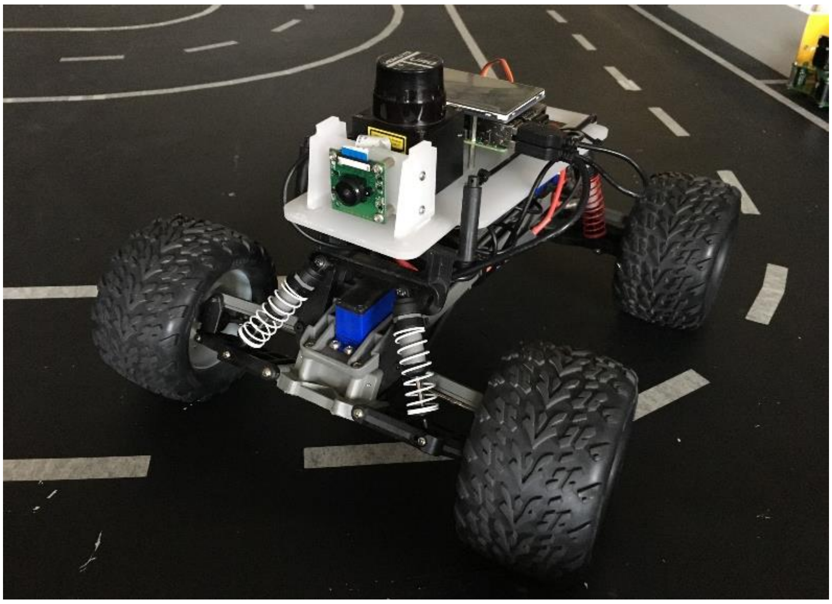

Figure 11: ModelCar-2 Autonomous Driving Robot

The original ModelCar driving program was written in the C programming language, this proved problematic as Python is the main environment for machine learning applications. The entire LIDAR driving code was then converted to Python and compiled together with the new ModelCar-2 neural network driving code to form one driving program.

The ModelCar-2 source code features two main Python programs, the driving program and the model training program. The driving program is called *DriveModelCar-2.py* and makes extensive use of the RoBIOS GUI.

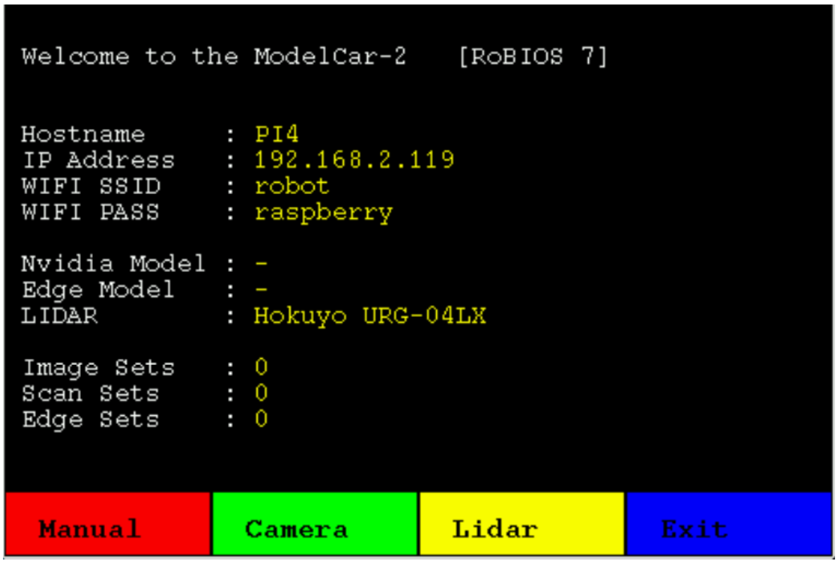

Figure 12: ModelCar-2 GUI Main Menu

The main menu screen can be navigated by using the touchscreen or the Logitech F710 Gamepad. The screen also provides live information on the robot including the IP address and network connectivity, the number of training images that have been recorded, the name of the currently loaded neural network model and if there is a LIDAR scanner connected.

The program features 3 different robot driving modes: Manual Drive, Camera Drive and Lidar Drive. Each of these modes can be accessed by pressing the buttons at the bottom of the screen or using the corresponding coloured buttons on the handheld controller.

3.3.1 Manual Drive

The manual driving mode provides complete manual operation of the robot using the Logitech F710 controller with the speed and steering of the robot controlled by the right trigger and left thumb stick respectively.

The mode includes a recording setting that when activated will begin saving images taken from the front-facing camera and stores them with the current steering angle and speed as labels. These images will be used later to train the neural network. There is also a cruise control feature that allows the user to set a certain speed that will be maintained by the robot until the setting is disabled, the brakes are manually applied, or the program is exited.

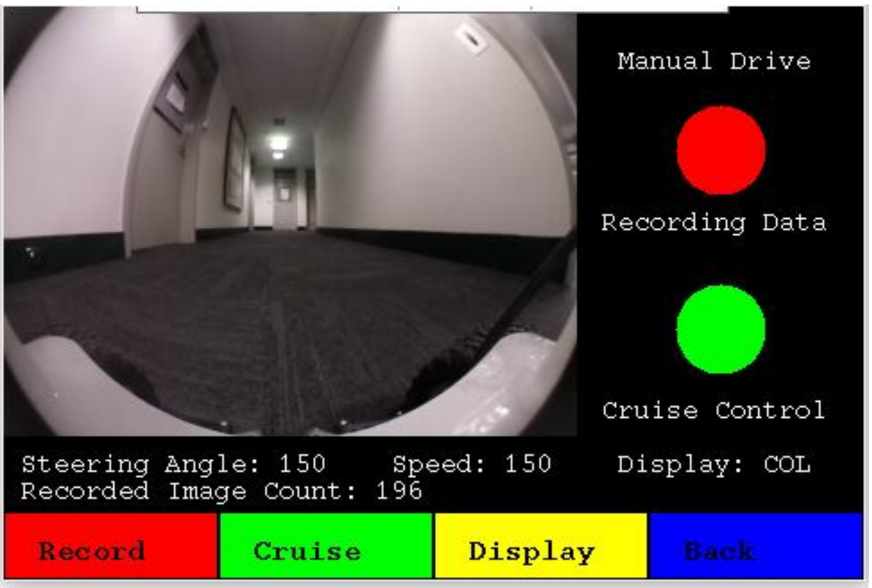

Figure 13: Manual Drive Mode Display

The display in this mode shows the feed from the camera, the current steering angle and speed values, the current number of recorded images as well as the recording and cruise control indicators. The values for speed and steering are displayed as a Pulse Width Duration and are in units of microseconds.

3.3.2 Camera Drive

The camera driving mode uses the NVIDIA PilotNet neural network end-to-end driving approach that has been created for this project. When a neural network model has been transferred to the Raspberry Pi and placed in the correct directory, the driving program allows the user to load it and begin autonomous driving of the robot.

For every control loop of the camera driving mode, an image is captured from the camera which then goes through the image processing function that is defined in Table 3. The image is then forward passed to the neural network model which based on its prior training will predict the optimal speed and steering angle values for that image. These values are then sent to the steering servo and motor to control the robot using the ServoBlaster library. This operation is continuously looped until the user exits the program or the power is disconnected.

Table 2: Simplified End-to-End Driving Control Loop

```
while(True):
    image = RoBIOS.CAMGet()
    #Take Image from Camera

    X = Image_Processing(image)
    #Pass to Image Processing Function

    Speed, Steering_Angle = int(model.predict(X))
    #Model Predicts Speed & Steering Angle using the Image

    ServoBlaster.write(Steering_Angle)
    ServoBlaster.write(Speed)
    #Send Control Commands to Servo
```

As discussed earlier, each image has to be processed before being accepted by the neural network. For this, a function that utilised the OpenCV library was created.

Table 3: OpenCV Image Processing Function

```
def Image_Processing(Image):
    Image = Image[60:180, :, :]
    #Crop Top and Bottom 25% of Image

    Image = cv2.cvtColor(Image, cv2.COLOR_RGB2YUV)
    #Convert RGB Image to YUV

    Image = cv2.resize(Image, (200, 66))
    #Resize Image Resolution to 200x66

    return Image
```

The ModelCar-2 program captures images with a resolution of 340x240 in the RGB colour space. The image is first cropped to remove 60 pixels from the top and bottom which corresponds to 50% of the image. These pixels do not contain any useful information for the network and could introduce overfitting, the focus for driving should be placed on the middle section of the image [4]. The RGB image is converted to the YUV colour space and then resized to a resolution of 200x66. The normalisation of the images is taken care of inside the network.

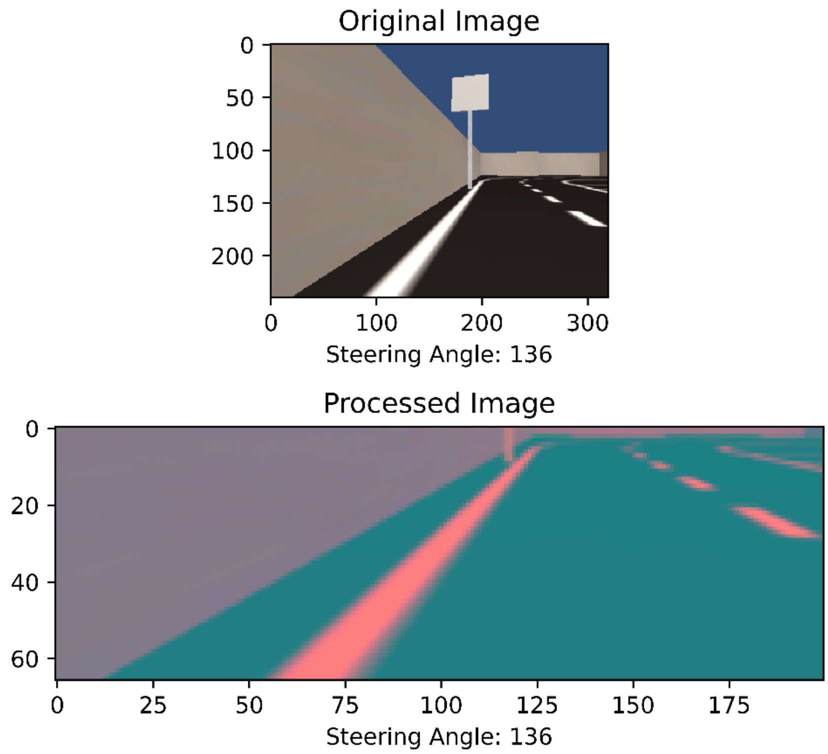

Figure 14: Image Processing Transformation

3.3.3 LIDAR Drive

The LIDAR driving mode uses an algorithm developed in the original ModelCar project which as well as being converted to Python, has been largely adjusted throughout this project to achieve better performance.

This mode allows the user to set a direction priority: Left, Right or Balanced, which can be selected using the D-pad of the Logitech controller that will tell the robot to prioritise its driving in the set direction [3]. The current priority setting is shown in the current mode display. For every control loop, a new LIDAR scan is captured from the sensor which along with the current direction priority and current speed is used as input for the algorithm [3].

.

The driving algorithm works by separating the scan data into sectors of 5° and the minimum and maximum horizontal and forward distances for each sector are calculated [3]. The algorithm then checks whether any of the minimum distances require the robot to avoid any obstacles and calculates the optimal speed and steering angle if this is the case. However, if the robot is in a straight corridor with walls running along both sides, the algorithm calculates the steering angle that allows the robot to align to the centre of the corridor and increments the speed to maximum. The algorithm also checks if there are any corners for which a major turn is required. This is achieved by using the horizontal and forward distances to detect gaps wider than a user-defined parameter and if one is found, the required steering angle and speed to make the turn are calculated [3]. The speed and steering angle values are then sent to the servo via ServoBlaster to control the robot.

Although relatively simple to explain, this algorithm is quite complex in its implementation which is a result of having many pre-defined parameters that require experimentation to optimise and multiple calculations that are performed for each loop. This is why the idea of end-to-end driving with neural networks has the potential to simplify autonomous robot design.

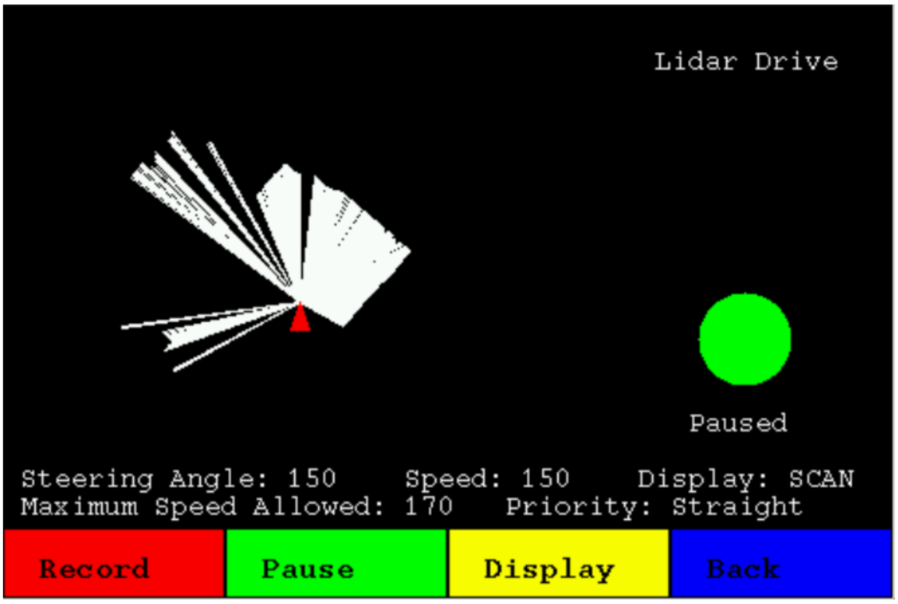

Figure 15: LIDAR Drive Scan Display Mode

The display is similar to the other driving modes, except there are additional display options as a result of the LIDAR. The first new display is the raw scan data that is plotted in real-time.

The second is the point map that shows the locations of the obstacles from the scan data relative to the position of the robot. The white points represent detected surfaces and objects and the robot is displayed as a red triangle.

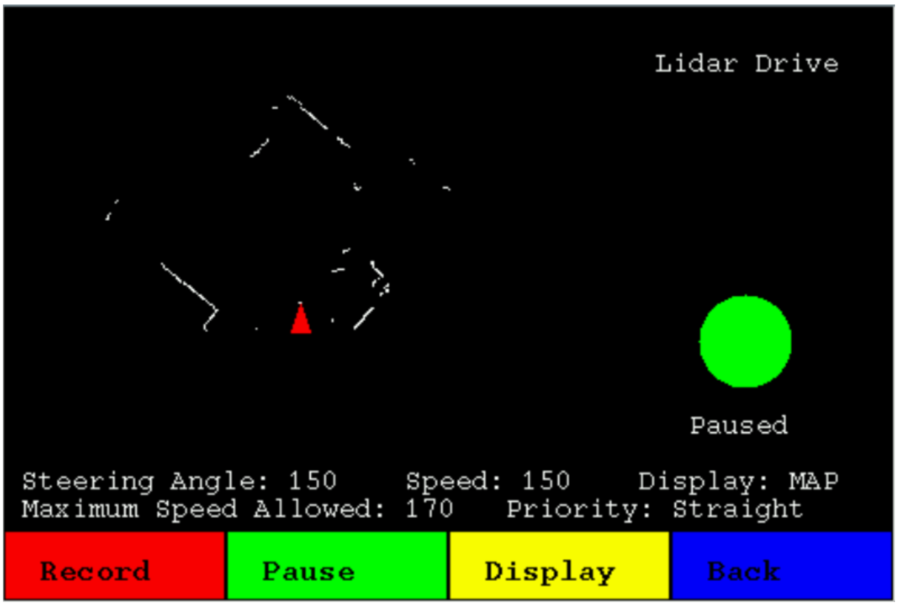

Figure 16: LIDAR Drive Point Map Display Mode

3.3.4 Control Flow Diagram

The control flow diagram in Fig. 17 shows the direction of sensor or control data in each driving mode.

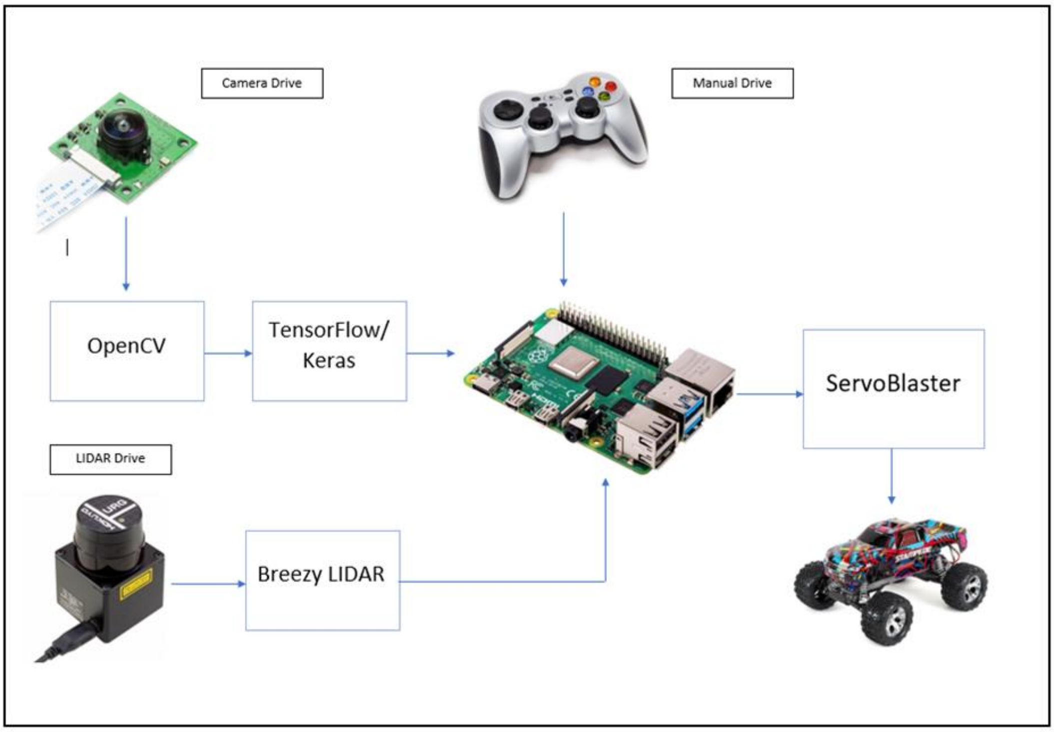

Figure 17: Simplified Control Flow for ModelCar-2

4 Method

4.1 Simulation

As with most robotics projects, a simulation was developed as a proof of concept using the EyeSim robotics simulation software which is developed and used for teaching at UWA. One of the main reasons for choosing the NVIDIA PilotNet model for this project was that it is also used for assignments in neural networks at UWA, where the same simulation software is used. A robot was trained in much the same way that the real robot will be, to drive around a small track and avoid obstacles along the way.

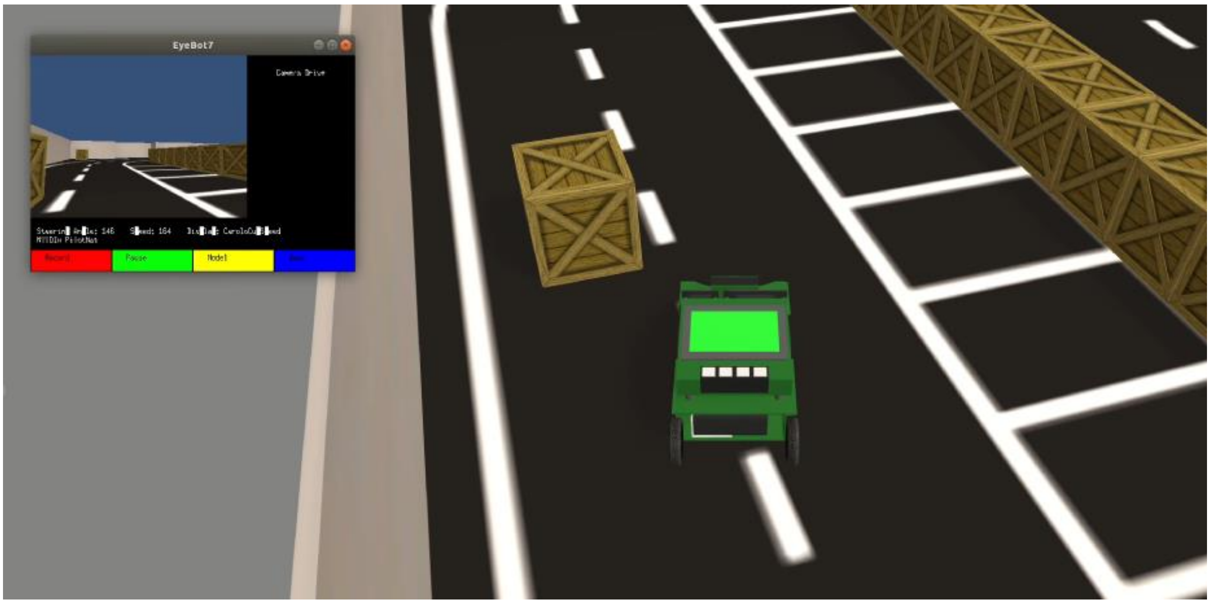

Figure 18: Simulation of Robot Avoiding Crate

The simulated robot successfully learned to stay on the left side of the track as well as recognising the obstacles along the track, avoiding them and then re-joining the track afterwards. The robot is also able to drive through small crossroad sections where there are limited lane markings. This is the behaviour that is desirable in the real ModelCar-2. The EyeSim software can be found at:

https://robotics.ee.uwa.edu.au/eyesim/

4.2 Data Collection

To collect training data for the real ModelCar-2, the robot will drive around the testing environment while constantly recording images and labelling them with the steering angle and speed values at the time of capture. The data is collected using a mix of both the manual and LIDAR driving modes. The reason for this is to increase the robustness of the model, manual driving allows training of very specific behaviour that is desired such as the obstacle avoidance and path following seen in the simulation. However, manual driving can be inconsistent as a result of human error, LIDAR driving is more consistent as it is following a structured algorithm. This means that the ModelCar-2 is hoping to use LIDAR driving to train the neural network which will be used to replace LIDAR driving.

The data also contains cases where the robot has made mistakes and driven off the course which are important as they allow the neural network to recover from its own mistakes, the robot needs to be able to adapt to new situations.

56,100 images have been collected in total. Validation data is a subset of the data that is used as an unbiased comparison to the training data during the model fitting process.

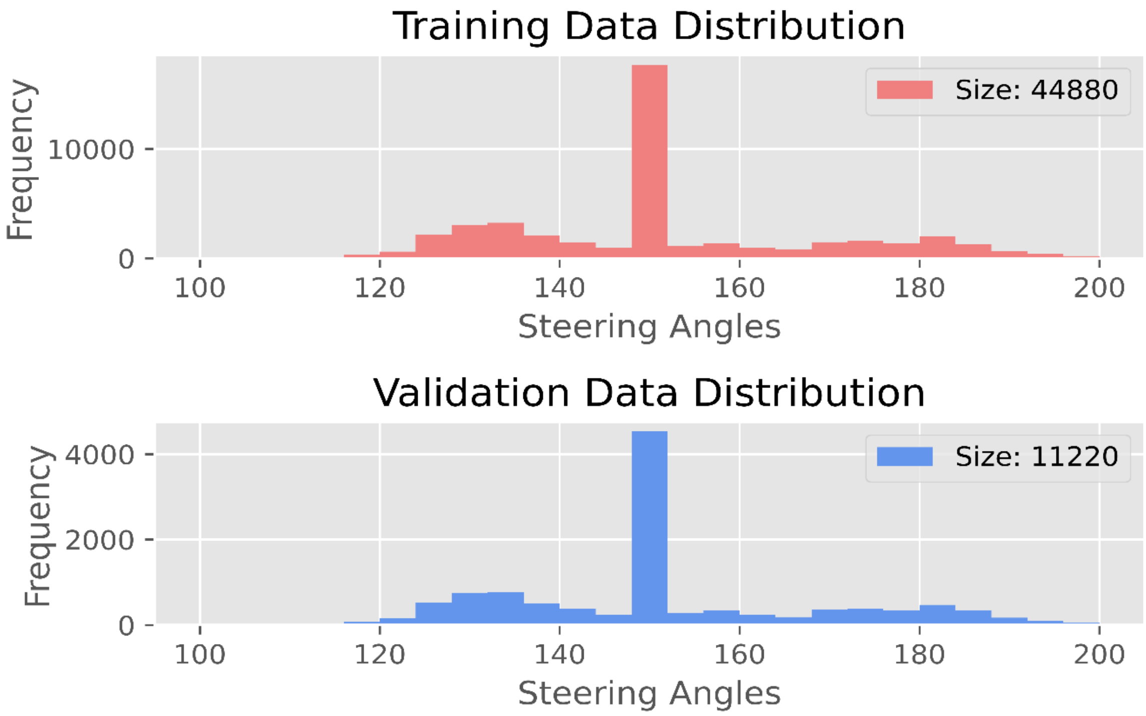

Figure 19: Steering Angle Data Distribution

It is clear that steering angle values around 150 are prominent in the data, 150 is the value corresponding to a straight turn while 100 and 200 are full right and full left turns respectively. Bias towards producing straight steering outputs is a real concern with ModelCar-2 as the testing environment that was used for collecting data involved many straight corridors. Ideally, the plot would also be symmetrical about the 150 value to avoid bias towards turning in any given direction. These issues can all be addressed using data augmentation.

4.3 Data Augmentation

Data augmentation is the process of deliberately manipulating the training data sets to enrich its potential, not just to increase its size but to force the network to adapt to unusual situations [25]. Data augmentation helps increase the model's robustness and achieve greater generalisation [4]. Augmentations are applied before the image processing function and only to the training data. For ModelCar-2, two data augmentation procedures were added to the training data.

The first augmentation involves a simple horizontal image flip, which is classified as a light augmentation. As discussed earlier, the distribution of steering angles preferably needs to be symmetrical to avoid any bias to one direction. The proportion of flipped augmentations can be changed to balance this bias in the data without collecting new examples. The steering angle of the flipped image needs to be adjusted accordingly.

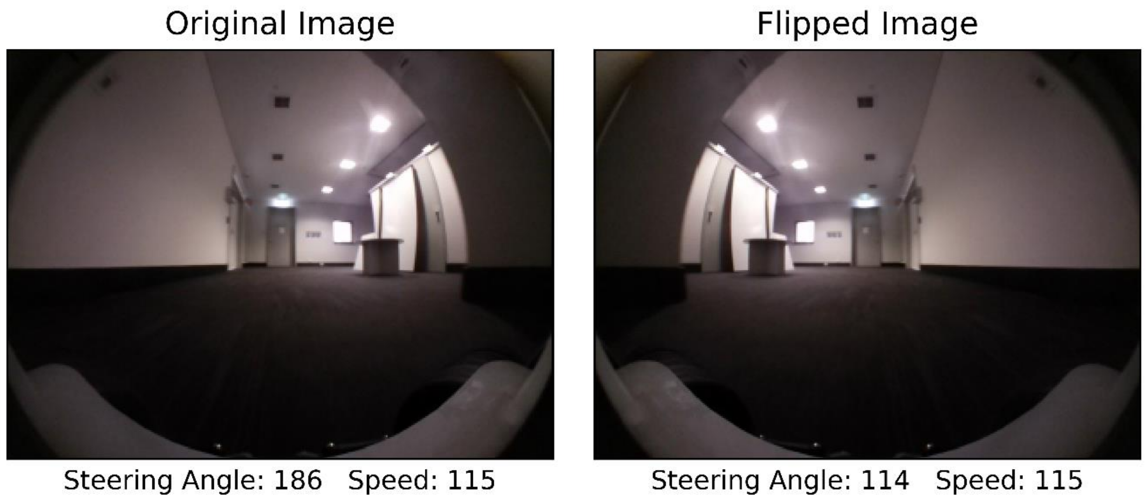

Figure 20: Image Flip Augmentation

The second augmentation involves blurring the image, which is classified as a moderate augmentation. This is added to help the model make sense of the images even when the camera vision quality is low or being obstructed. A Gaussian blur with a random standard deviation ranging from 3 to 5 is applied to the image.

Figure 21: Image Blur Augmentation

For every batch in the training data, the augmentations are randomly assigned with a probability of 5% for each. No heavy augmentations were performed as they can alter the image too drastically and the model may have difficulty picking up on these changes [4].

4.4 Model Training

As stated earlier, the ModelCar-2 source code features two main Python programs, the driving program and the model training program. The model training program is called *TrainModelCar-2.py* and it recreates PilotNet as faithfully as possible. The training process involves performing back-propagation operations to update the weights of the model to reduce the mean squared error [17]. Table 4 shows a simplification of how the recreation of NVIDIA PilotNet is achieved using the Keras API.

Table 4: NVIDIA PilotNet recreation with Keras

```
def NVIDIA_PilotNet():
    input_shape=(66, 200, 3)
    model = Sequential()
    model.add(Lambda(lambda x: (x/127.5 - 1.0), input_shape)
    model.add(Conv2D(24, (5, 5), activation='elu'))
    model.add(Conv2D(36, (5, 5), activation='elu'))
    model.add(Conv2D(48, (5, 5), activation='elu'))
    model.add(Conv2D(64, (3, 3), activation='elu'))
    model.add(Conv2D(64, (3, 3), activation='elu'))
    model.add(Flatten())
    model.add(Dropout(0.2))
    model.add(Dense(100, activation='elu'))
    model.add(Dense(50, activation='elu'))
    model.add(Dense(10, activation='elu'))
    model.add(Dense(1, name='Model_Speed'))
    model.add(Dense(1, name='Model_Steering'))
    model.compile(loss='mse', learning_rate=1e-3))
    return model
```

There are only two changes from the original PilotNet: the addition of speed as an extra model output and the inclusion of dropout.

Overfitting is when a model is too closely fitted to a set of data and can result in the model finding trends amongst the noise. Overfitting can be mitigated by increasing the size of the training data or by introducing dropout. Dropout is the process of randomly dropping units from the neural network to prevent them from co-adapting too much [9]. After a series of experiments, a dropout probability of 0.2 was determined to achieve the best results. Another method for preventing overfitting is early stopping, if the validation loss of the model doesn't change with 10 epochs then model training is stopped.

Once a satisfactory amount of training data has been collected, the model training is performed on a Linux machine running Ubuntu 19.04 using two NVIDIA 1080Ti GPUs. The Raspberry Pi's CPU can't be used for training as it is too small, a machine with greater graphics capabilities is required. When the model has been trained, it is then copied back to the Raspberry Pi to be tested against the LIDAR driving mode.

5 Results

5.1 Losses and Accuracy

Loss in a neural network refers to the residual sum of the squares for the errors made for each image within the training and validation data. Therefore, a model with a lower loss at the end of the training process will most likely be better at predicting outputs.

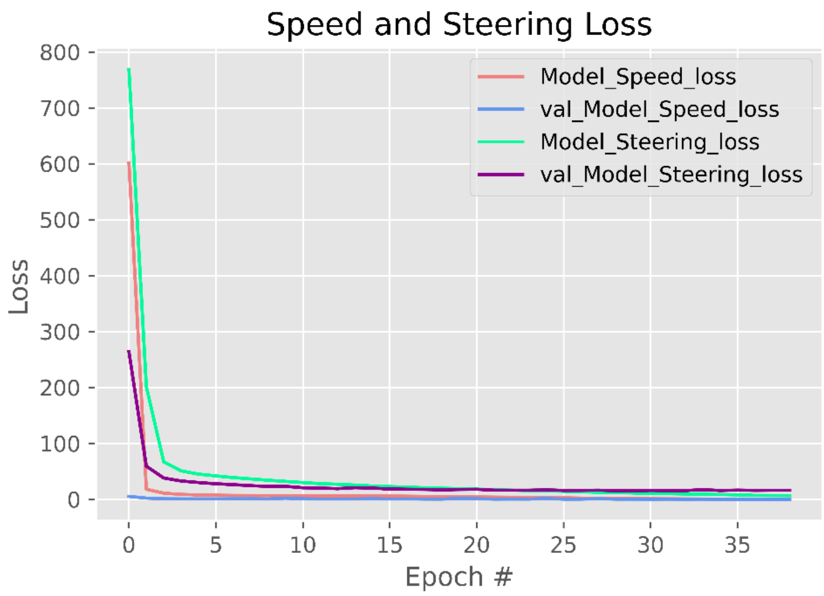

Figure 22: Training and Validation Loss

Both the speed and steering losses quickly converge towards a very small loss value early in the training process and remain almost unchanged until the early stopping is engaged. This could be a sign of model overfitting but no evidence of this was observed in the on-track tests.

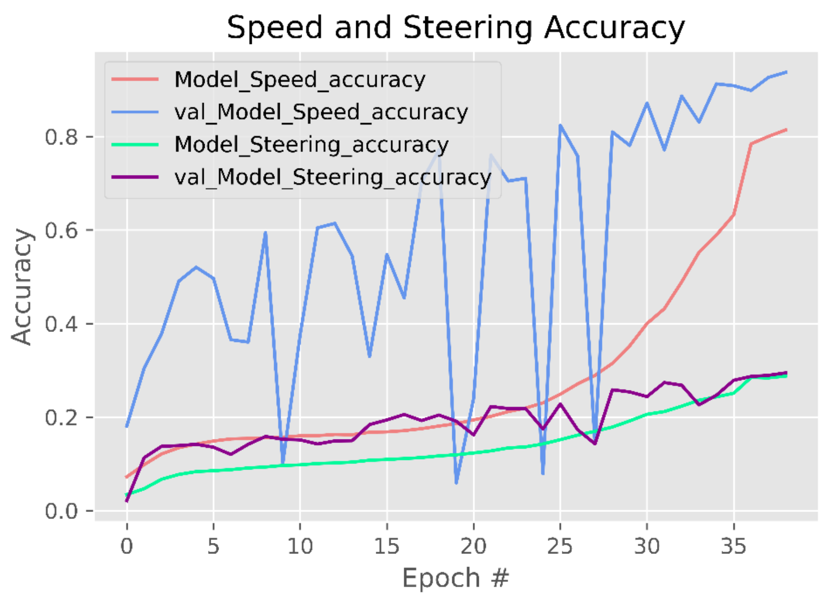

Figure 23: Training and Validation Accuracy

Accuracy is a percentage value that represents how many times the model has predicted the correct speed value. The model has guessed more than 80% of the speeds correctly.

The steering accuracy is not as impressive with a final accuracy of 30%, again due to a larger variation of classes. This is not a large issue; a model needs to be 100% accurate to register as a correct prediction and as the values for steering as decimal values with a fine resolution, total accuracy is going to more difficult.

The important aspect in this case is the severity of the error, which is why loss is a more reliable indicator of driving performance. Both loss and accuracy are functions of training steps which are called epochs

5.2 Predictions

In addition to the training and validation data sets, a separate testing data set was collected that hasn't been used in any part of the model training and therefore the network hasn't seen these examples. If these images are passed to the model, the outputs can be compared against the actual recorded outputs to determine if the model is learning correctly before any on-track tests are performed.

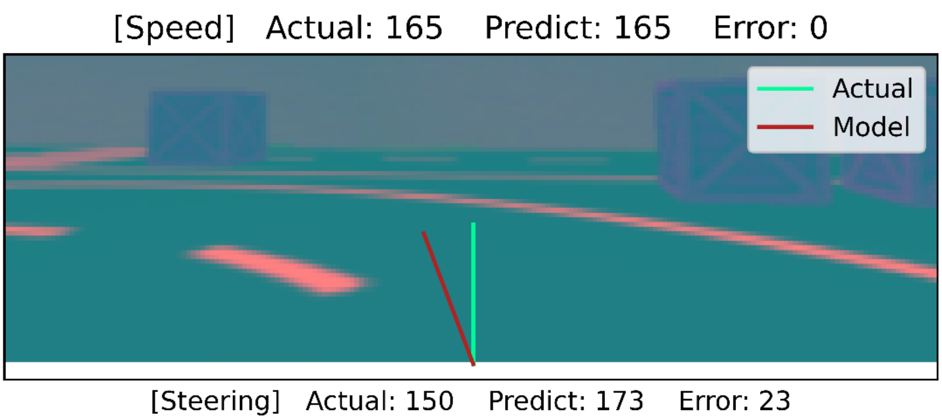

Figure 24: Simulation Model Prediction Test

For this simulation example, the speed was predicted correctly, and the actual recorded steering angle was 150 which represents a straight direction, but the image clearly shows that a left turn is required. This could be a result of human error when collecting data, but the model still predicts a left turning value of 173 that correctly follows the road pattern in the image.

This demonstrates that at least for the simulation, the model is correctly learning.

5.3 Autonomy

To measure the autonomy of both the LIDAR and camera drive systems, the method from NVIDIA was used [1]. This method calculates the percentage of time the robot can drive without any interventions. An intervention is defined in this case as any human involvement that was required to prevent or rectify any mistakes made by the robot that would result in a collision with an object, barrier or person. The autonomy can then be calculated using the following equation:

$$Autonomy = \left(1 - \frac{(Number\ of\ Interventions) \times 5}{Total\ Time\ [seconds]}\right) \times 100$$

Both driving systems were tested in 5 sessions each lasting 10 minutes on different days. The testing environment included both the 3rd and 4th floor of the Electrical Engineering Building at UWA. The autonomy scores for each of the runs were recorded and an average was calculated.

Table 5: Driving Autonomy Scores

Mode	**Trial 1**	**Trial 2**	**Trial 3**	**Trial 4**	**Trial 5**	**Mean**
LIDAR	96.51%	94.91%	98.68%	97.09%	96.42%	**96.722%**
Camera	93.70%	90.53%	95.38%	95.26%	96.54%	**94.282%**

The LIDAR driving system is consistently scoring higher than the neural network across every trial resulting in a higher average autonomy. The difference here is not drastic and there is a trend throughout the camera trials that seems to suggest that with each trial when more data was being collected and modifications to the model were made, the neural network was able to perform slightly better. This could indicate that as data continues to be collected, the neural network could be able to match the autonomy of the LIDAR system.

Referring to the SAE levels of autonomy, both the LIDAR and camera driving systems can be classified as having Level 3 autonomy. In both systems, the vehicle has complete control over all aspects of driving including steering, accelerating, braking and reversing but manual interventions are still required for stopping completely and errors do still occur.

5.4 Saliency

A saliency map is a form of image classification that is used to visualise the attention that the Convolutional Neural Network is primarily focussed on for determining the output class [15]. In the context of the ModelCar-2 project, saliency can be used for highlighting the pixels of the camera images that the model feels are most important in its decision for steering angle.

As discussed earlier, the first convolution layer is responsible for detecting the larger shapes and patterns of the image and therefore performing saliency on this layer will give the greatest insight into what the model is learning.

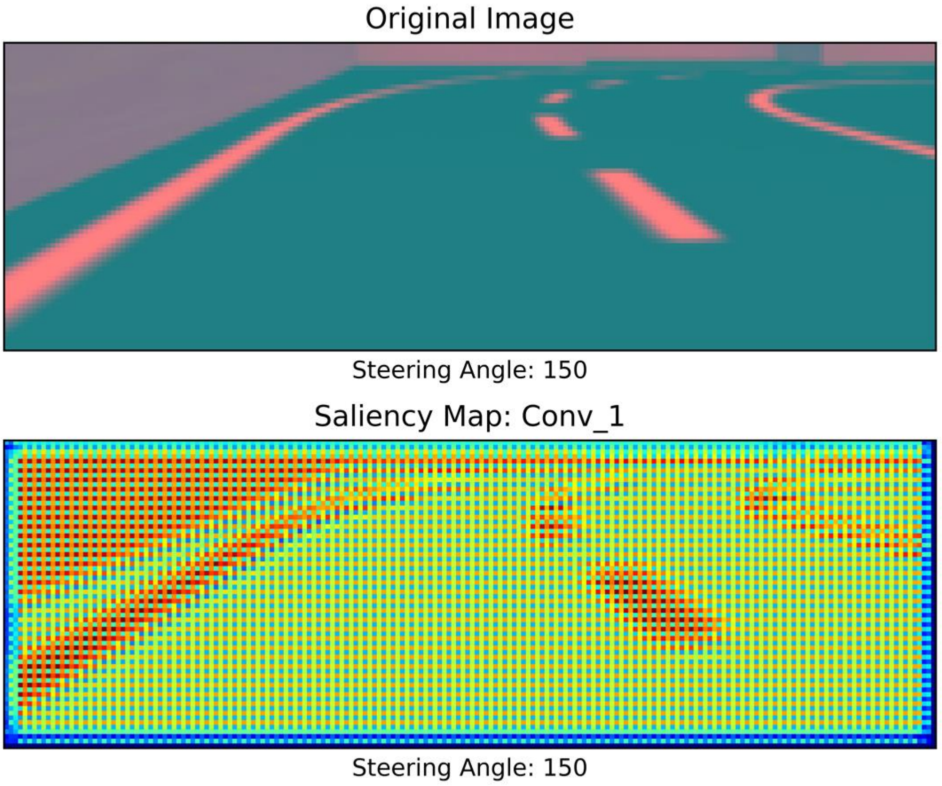

Figure 25: Simulation Saliency Map

The pixels with the highest priority are shown in red and rank down to the lowest priority which are represented in blue. As you can see for the simulation, the model is correctly identifying the lanes and edges as being the focus of its analysis.

5.5 Confusion Matrix

A confusion matrix shows whether the model is correctly predicting values and can identify the involvement of Type I and II errors in the data. A confusion matrix for the steering angles from the real ModelCar-2 can be seen in Fig. 26, the actual values are shown along the top and the model predicted values are run down the y-axis.

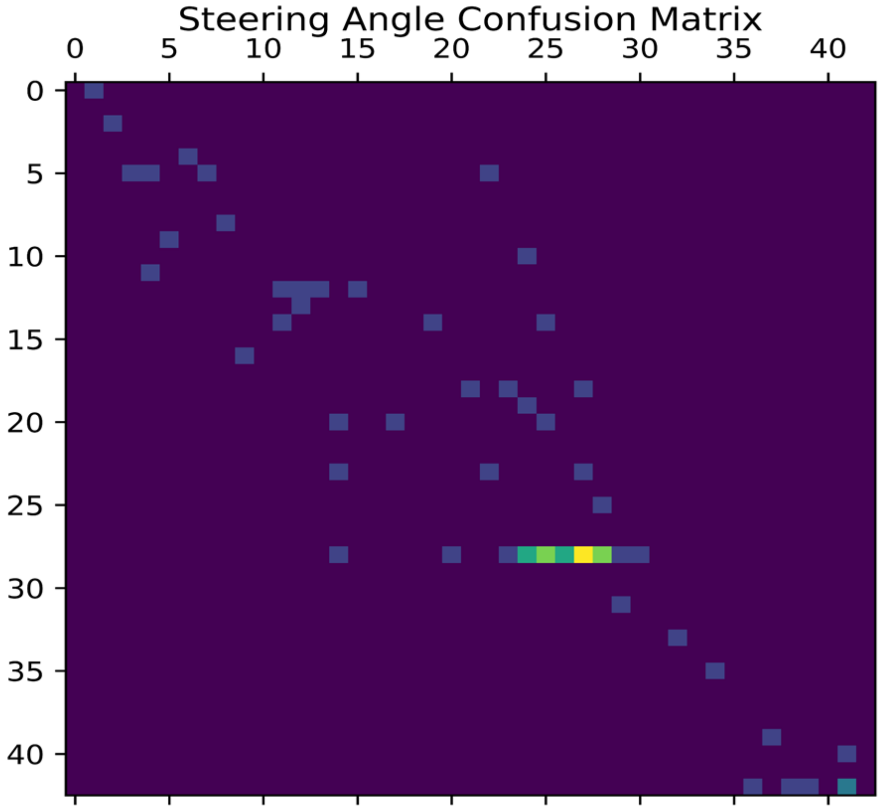

Figure 26: Steering Angle Confusion Matrix

The ideal situation would be to have only values along the diagonal, this would mean that all the model predictions would match the actual values and the model is correctly learning across all classes.

The trend of the matrix does follow the diagonal but there are areas where the model is "confused". There is a cluster around the value of 25 which represents gentle left turns, the model is predicting the same value for all classes in this region. This suggests that this is an area that requires further training.

There is also quite significant error towards the middle of the trend where the steering values are close to straight. This could be a result of the straight bias discussed earlier, for further testing more gentle left and right curves should be introduced to balance out this discrepancy.

5.6 Processing Speeds

For real-time control of any vehicle or robot, the control loop processing time must be sufficiently low enough so the vehicle can react quickly enough to new obstacles and changing environments [5]. In general, control performance of the vehicle improves when processing time decreases which makes it an important criterion when comparing the LIDAR and Camera drive systems.

Fig. 27 shows the times taken for the processing of each control loop for all three driving modes. The Manual Drive processing times have been added in to serve as a baseline, showing a consistent pattern with a mean time of 0.033 s corresponding to a frame rate of 30.3 Hz.

The Camera Drive processing time appears to be as consistent as Manual Drive, with a slightly slower mean time of 0.042 s corresponding to a frame rate of 23.81 Hz. However, the LIDAR drive system is producing wildly inconsistent results that average to a 0.105 s processing time that corresponds to a frame rate of only 9.52 Hz.

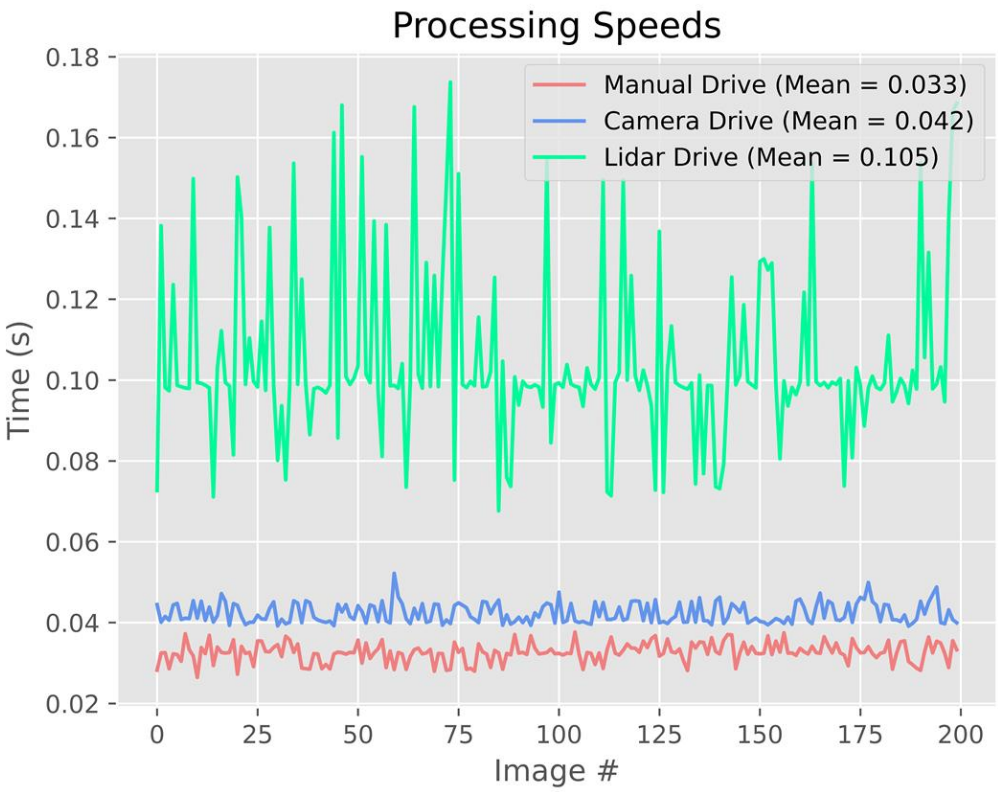

Figure 27: Processing Times for All Driving Modes

In the context of autonomous driving performance, this means that the camera drive system is able to react over twice as quickly in real-time to new changes in the environment as the LIDAR drive system which provides it a clear advantage.

This large inconsistency is a result of how the LIDAR data is processed by the algorithm and how different sections are activated at different times. If the path is clear ahead, the LIDAR doesn't do much processing and quickly produces a straight steering with an incrementing speed. However, if the vehicle comes across any obstructions or turns, the program has to calculate more information to decide its new path and therefore processing time for these loops will increase.

This problem is not limited to this algorithm, all LIDAR and remote sensing driving implementations will involve forms of conditional logic that will reduce the processing speed capabilities of their systems. This is avoided in neural network driving approaches as the end-to-end nature of the control makes any conditional logic structures redundant. As all control loops follow the same processing pipeline, the processing times are more consistent.

Table 6: Frame Rates for All Driving Modes

Mode	**Frame Rate (Hz)**
Manual	**30.303**
Camera	**23.810**
LIDAR	**9.524**

As for the discrepancy between average frame rates, the Hokuyo LIDAR scanner used in the ModelCar has a quoted maximum scan frequency of 10 Hz. For the LIDAR driving system, this scan rate acts as a limiting agent, meaning the control loop frequency can't exceed 10 Hz which is why the average frame rate was measured to be 9.52 Hz. The camera drive system doesn't use the LIDAR scanner and so doesn't experience this limiting agent on its processing speed, instead the performance of the Raspberry Pi 4 embedded system is what limits the control frequency.

For additional reference, NVIDIA have reported that their DAVE-2 system which uses the same PilotNet neural network architecture as ModelCar-2 has an average frame rate of 30 Hz [5]. GPU acceleration for TensorFlow is not available for the Raspberry Pi and instead only uses the CPU cores for processing which may explain how NVIDIA was able to achieve a slightly higher frame rate by using the DRIVE PX system. However, a frame rate of 23.81 Hz on the Raspberry Pi is still surprisingly impressive considering the complexity of the deep neural network.

5.7 Cost

As cost reduction is a large focus of the ModelCar-2 project, the overall cost of both driving systems is important in their comparison.

Table 7: ModelCar-2 Bill of Materials

Item	Cost ($AUD)
Traxxas Stampede R/C Car	$ 304.43
Raspberry Pi 4 Model B	$ 94.60
5MP OV5647 Camera	$ 33.79
Hokuyo URG-04LX LIDAR	$ 1770
Logitech F710 Gamepad	$ 53.86
Mounting Plate Plastic	$ 12.99
USB Battery Pack	$ 31.95
3.5 LCD Touchscreen	$ 46.51
Total	**$ 2348.13**

However, the total budget considers all the components for both driving systems, the LIDAR is not needed for neural network driving and the same for the camera with LIDAR driving. Table 8 shows the total cost for each individual driving system.

Table 8: Driving System Cost Comparison

Mode	Cost ($AUD)
LIDAR	**$ 2314.34**
Camera	**$ 578.13**

The camera driving system is 4x more cost-efficient and can be considered well within the budget of a typical small-scale autonomous robotics project. The LIDAR sensor unit makes up over 76% of the budget for the LIDAR driving system, whereas the camera only makes up 6% of the camera driving system. Clearly being able to replace sensors such as LIDAR with cameras is more cost-effective, especially for small-scale projects.

This cost analysis does not include expenses that relate to model training on GPU accelerated machines such as the purchase and ongoing electricity costs which can also be quite significant for small-scale applications. However, the research and labour costs involved in developing the LIDAR driving algorithm have also been excluded which due to the large amount of time required for the task can be notable as well.

6 Conclusion

This thesis has demonstrated that autonomous driving implementations using cameras and neural networks can be used as a successful alternative to classic approaches that use expensive LIDAR sensors. A Convolutional Neural Network was trained to control the steering and speed of a small-scale driving robot.

While the autonomy of the neural network driving system was shown to be lower than that of a LIDAR driving system, this project aimed to find whether the end-to-end neural network approach could match the autonomy level of a LIDAR based system with a lower cost structure, which has been supported. Both driving systems are classified as having Level 3 SAE autonomy.

The processing speed of the end-to-end driving system is much more consistent and more than twice as quick as the LIDAR system on average which demonstrates how camera systems are not only more cost-effective but more efficient on processing power.

The ModelCar-2 robot designed and constructed in this thesis can serve as a new platform for autonomous robotics research and teaching at the University of Western Australia. All tools and resources used in this project have been chosen to make any following work easy to append and recreate.

6.1 Future Work

There are countless ways to continue this project and plenty of room for improvement when it comes to model training.

LIDAR driving does have a small advantage over end-to-end driving when it comes to knowledge of previous states, the LIDAR algorithm remembers the previous speeds and steering angles to help calculate the next iteration. The end-to-end driving method demonstrated in this thesis, only makes decisions on the current image and does not account for the previous and future images. As driving is a continuous task, a recurrent neural network approach such as LSTM to record the relationship between adjacent images could produce better results. In situations where the neural network is uncertain of the current image, information from past images can better estimate decisions for the future.

A feature that was to be implemented but was restricted because of time, was LIDAR SLAM which could map out the environment that the robot drives through. The current ModelCar-2 driving program does have BreezySLAM working but the results are disappointing, we were hoping to replicate the results of the map shown in Fig. 28.

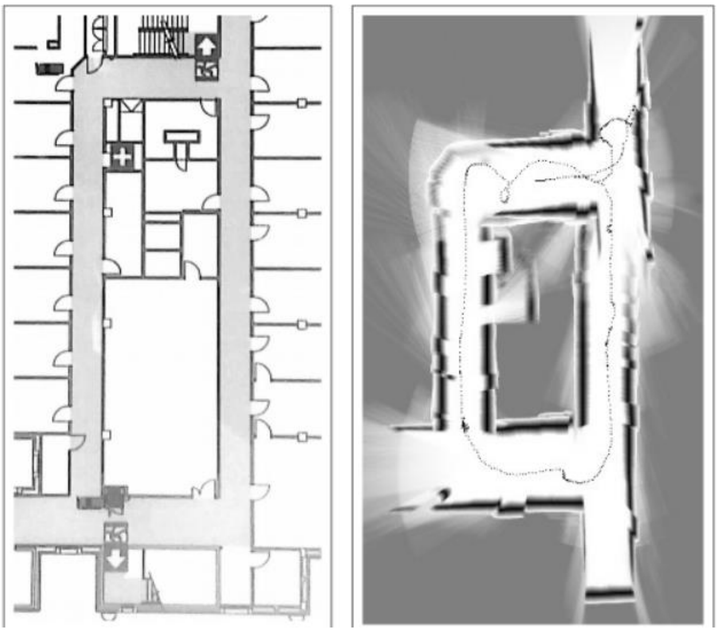

Figure 28: LIDAR SLAM Map of EECE 4th Floor UWA [3]

Another proposal is a camera SLAM feature that would be based on another UWA project called ORB-SLAM [13]. As this thesis compared the autonomy of LIDAR and camera driving systems, an interesting addition could explore the results of both LIDAR and camera SLAM implementations.

There is also F1TENTH, which is a racing competition involving 1/10th scale autonomous vehicles that are built by students [16]. The ModelCar-2 robot is eligible to compete in this competition and future work on this project could involve adapting the robot for such as racing setting.

References

[1] M Bojarski, DD Testa, D Dworakowski, B Firner, B Flepp, P Goyal, LD Jackel, M Monfort, U Muller, J Zhang, X Zhang, J Zhao, K Zieba. NVIDIA Corporation. **End to End Learning for Self-Driving Cars.** (2016).

[2] M Bojarski, P Yeres, A Choromanaska, K Choromanski, B Firner, LD Jackel, U Muller, J Zhang, X Zhang, J Zhao, K Zieba. NVIDIA Corporation. **Explaining How a Deep Neural Network Trained with End Learning Steers a Car.** (2017).

[3] M Mollison. University of Western Australia, School of Electrical, Electronic and Computer Engineering. **High Speed Autonomous Vehicle for Computer Vision Research and Teaching**. (2017).

[4] D Ungurean. Czech Technical University in Prague, Faculty of Information Technology. **DeepRCar: An Autonomous Car Model**. (2018).

[5] MG Bechtel, E McEllhiney, M Kim, H Yun. University of Kansas, Indiana University. **DeepPiCar: A Low-Cost Deep Neural Network-based Autonomous Car**. (2018).

[6] Society of Automotive Engineers. **SAE International Releases Updated Visual Chart for Its "Levels of Driving Automation" Standard for Self-Driving Vehicles** https://www.sae.org/standards/content/j3016_201806/. (2018).

[7] A Kiulian. Medium. **Let's Talk About Self-Driving Cars**. https://medium.com/swlh/lets-talk-about-self-driving-cars-387cd5adb834. (2017).

[8] TopSpeed. **Tesla Autonomy Day 2019 - Full Self-Driving Autopilot - Complete Investor Conference Event.** https://www.youtube.com/watch?v=-b041NXGPZ8. (2019).

[9] N Srivastara, G Hinton, A Krizhevsky, I Sutskever, R Salakhutdinov. University of Toronto, Department of Computer Science. **Dropout: A Simple Way to Prevent Neural Networks from Overfitting**. (2014).

[10] D Pomerleau. In Advances in Neural Information Processing Systems (NIPS), **ALVINN: An Autonomous Land Vehicle in a Neural Network**. (1989).

[11] T Bräunl, M Pham, F Hidalgo, R Keat, H Wahyu. University of Western Australia, School of Electrical, Electronic and Computer Engineering. **EyeBot-7 User Guide**. (2018).

[12] Aerospace Robotics. **Unravelling BreezySLAM**. https://aerospacerobotics.com/blogs/learn/14881837-unraveling-breezyslam. (2014).

[13] C Kahlefendt. University of Western Australia, Hamburg University of Technology. **Implementation and Evaluation of Monocular SLAM for an Underwater Robot**. (2018).

[14] N Burleigh, J King, T Bräunl. Conference on Digital Image Computing: Techniques and Applications (DICTA). **Deep Learning for Autonomous Driving**. (2019).

[15] K Simonyan, A Vedaldi, A Zisserman. University of Oxford. **Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps**. (2014).

[16] M O'Kelly, V Sukhil, H Abbas, J Harkins, C Kao, Y Vardhan-Pant, R Mangharam, D Agarwal, M Behl, P Burgio, M Bertogna. University of Pennsylvania, Oregon State University, University of Virginia, University of Modena & Reggio Emilia. **F1/10: An Open-Source Autonomous Cyber-Physical Platform**. (2019).

[17] D Tian. Medium. DeepPiCar: **How to Build a Deep Learning Driving Robotic Car on a Shoestring Budget.** https://towardsdatascience.com/deeppicar-part-1-102e03c83f2c. (2019).

[18] N Burleigh. University of Western Australia, School of Electrical, Electronic and Computer Engineering. **Autonomous Driving on a Model Vehicle: Lane Detection and Control.** (2019).

[19] S Karaman, Ariel Anders, Michael Boulet, J Connor, K Gregson, W Guerra, O Guldner, M Mohamoud, B Plancher, R Shin, J Vivilecchia. Massachusetts Institute of Technology. **Project-based, Collaborative, Algorithmic Robotics for High School Students: Programming Self-Driving Race Vars at MIT**. (2017).

[20] C Manliguez. University of the Philippines. **Generalised Confusion Matrix for Multiple Classes**. (2016).

[21] D Clevert, T Unterthiner, S Hochreiter. Institute of Bioinformatics, Johannes Kepler University. **Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)**. (2016).

[22] A Braylon. Wheels Joint. **Tesla Could Beat Waymo in the Race to Self-Driving Future**. https://www.wheelsjoint.com/tesla-could-beat-waymo-in-the-race-to-self-driving-future/. (2019).

[23] RR Selvaraju, A Das, R Vedantam, M Cogswell, D Parikh, D Batra. Virgina Technology. **Grad-CAM: Why did you say that? Visual Explanations from Deep Networks via Gradient-based Localisation.** (2016).

[24] M Zeiler, R Fergus. Department of Computer Science, New York University. **Visualising and Understanding Convolutional Networks**. (2013).

[25] S Wong, A Gatt, V Stamatescu. Defence Science and Technology, Edinburgh. **Understanding Data Augmentation for Classification: When to Warp?** (2015).