Autonomous Racing Car Model

Thesis Project

Saeed Alqahtani

Master of Engineering (Electronics)

Student ID

2190770

Dr Nasser Asgari

Academic supervisor

October 2019

Submitted to the College of Science and Engineering in partial fulfilment of the requirements for the degree of Master of Engineering (Electronics) at Flinders University-Adelaide Australia

Certificate of Originality

I certify that this work does not incorporate without acknowledgment any material previously submitted for a degree or diploma in any university; and that to the best of my knowledge and belief it does not contain any material previously published or written by another person except where due reference is made in the text.

Saeed Saleh Alqahtani 02/26/2020

Acknowledgements

This dissertation would not have been possible without the guidance and help of several individuals who, in one way or another, contributed and extended their valuable assistance in the preparation and completion of this study.

First and foremost, my utmost gratitude to Dr Nasser Asgari, my Topic Coordinator and supervisor. I will never forget him. He has been an inspiration as I hurdle all the obstacles in the completion of this thesis work. My thanks and appreciation also to my family and friends for their continued support.

In addition, I would like to thank Elite Editing for proofreading and editing my thesis. Where their editorial intervention was restricted to Standards D and E of the *Australian Standards for Editing Practice*

Abstract

This project focused on building a self-driving, one-tenth scale radio-controlled (RC) rally car. Machine learning is heavily involved in the field of self-driving cars. This project took the Nvidia end-to-end convolutional neural network (CNN) as a model to create a trained CNN. The Nvidia model was developed using Python and other Python machine learning libraries, such as Keras and OpenCV (Open-Source Computer Vision Library). Image processing was central to this project; lane detection was used to set a track for the car. The image dataset was processed before it was provided to the CNN. The simulation test had a positive result—the simulated car was able to clone the operator’s driving behaviour.

A practical, one-tenth scale RC rally car platform was then built and equipped with the CNN control system, a power system and sensors, including a camera and an inertial measurement unit (IMU). The practical result shows that it is not possible to run or to train the CNN model on a mini computer, such as the Odroid XU4Q. This issue is discussed in this thesis, and an approach for future development is suggested—using model predicted control (MPC). The complexity of MPC prevented it from being added to this project. For example, it requires an accurate mathematical model of the actual, scaled car.

This thesis comprises five chapters. The first two chapters discuss the background of self-driving cars, the proposed method and the recently developed work in the field. Chapter 3 explains how Udacity’s self-driving car simulator was used to run tests and train the CNN model. Chapter 4 details the practical steps for building the platform and the test results from each section. It also discusses the reasons that the CNN was not able to run on the practical platform and suggests different approaches for future studies.

Contents

List of Figures

List of Tables

List of Abbreviations

BARC	Berkeley Autonomous Race Car
CNN	Convolutional Neural Network
DARPA	Defense Advanced Research Projects Agency
FIA	Fédération Internationale de l'Automobile
GM	General Motors
GPS	Global Positioning System
HD	High-Definition
LiDAR	Light Detection and Radar
MIT	Massachusetts Institute of Technology
MPC	Model Predictive Control
NHTSA	National Highway Traffic Safety Administration
Tech	Technology
UGV	Unmanned ground vehicle
API	Application Programming Interface

Chapter 1: Introduction

The technical challenges of creating a self-driving vehicle present a problem, despite significant advancements from universities, car manufacturers and technology companies [1]. Autonomous racing cars have recently been attracting the attention of researchers, who are applying new methods, which are expected to take the field to another level, where speed and safety are the key elements of the systems [7]. Entertainment inventors are also advancing this technology by holding a formula event involving a race car and a human [2].

This section will give a brief background of developments in this sector. It will provide an in-depth explanation of the ongoing research on different methodologies. These methods are categorised into two main approaches. The first approach is developing a novel model predictive control (MPC). The second is the method of deep learning. This project was influenced by the latter, which formed the basis for the development of an autonomous racing car.

1.1 Background and Motivation

'Full autonomy' is defined as full-time performance, where all aspects of the dynamic driving task, under all roadway and environmental conditions, can be managed by an automated driving system [1]. Level zero is no automation and level five is full automation. Figure 1 shows the levels of autonomy according to the United States National Highway Transport Safety Administration [3].

Image removed due to copyright restriction.

Figure 1: Car autonomy level according to the NHTSA [3]

Self-driving cars are considered important by government and non-government organisations, for both military and commercial purposes. The Defense Advanced Research Projects Agency (DARPA) was the first government agency to show a significant interest in the field of self-driving cars. Their first autonomous vehicle event, the DARPA Grand Challenge 2004, invited universities and technology companies from all the over world to compete. Unfortunately, none of the robot vehicles finished the 150-mile route [4]. In the subsequent DARPA Grand Challenge 2005, contestants were able to use fully autonomous cars with unique methods and techniques for self-driving. The Stanley robot car, Figure 2, from Stanford University took first place—owing to the team's use of the deep learning technique [5].

Image removed due to copyright restriction.

Figure 2: Stanley Stanford self-driving car in action, self-driving in The DARPA Grand Challenge 2005 [12]

In 2017, humans and machines competed against one another, driving electrically powered vehicles at Roborace, an FIA (Fédération Internationale de l'Automobile) Formula E Championship event [2], Figure 3. According to the Roborace team, at that time, their machine had not successfully beaten the human recorded lap time for a single lap. However, the machine's time was less than 19 seconds behind the time recorded by the human driver.

Image removed due to copyright restriction.

Figure 3: The 2017 electrical Robocar [2]

There have also been a number of competitions at the level of start-up companies and individuals. For example, both full-size cars and DIY robocars compete in San Francisco's Self Racing Cars competition [6]. These contests, and many others, are supported by a community of inspired engineers and researchers who share the same aspiration of building an autonomous car. According to the Stanford Dynamic Design Lab, pushing the self-driving car to its limits will improve safety [7]. They justify this approach to development with the assumption that if a driver has no experience on which to draw in a potential crash situation, an autonomous system could provide a solution and avoid the scenario [7].

1.2 Thesis Outline and Contributions

The RC car constructed for this project consists of a mechanical part, electronic hardware and a controller (algorithms). The mechanical part is a VR46 Ford Fiesta ST Rally edition one-tenth scale model (Figure 4). The primary focus of this project was the algorithms, such as MPC, AMPC (adaptive predictive control), the extended Kalman filter and stochastic optimal control methods. Using this existing one-tenth scale model as a platform allowed more time for implementing, creating and testing the algorithms.

Image removed due to copyright restriction.

Figure 4: VR46 Ford Fiesta ST Rally one-tenth scale

For the electronic hardware, a complete system was implemented within the platform. The main controller is the Odroid XU4Q, which is equipped with a Nano Arduino board (which controls the servos), an inertial measurement unit (IMU) and a camera. Additional sensors, such as light detection and radar (LiDAR), global positioning system (GPS) and a structural camera (Kinect), will be installed in future iterations of this project. The controller development was primarily based on a trained CNN to control the steering angle of the car. Building this project required familiarity with various software, which is summarised in Table 1.

Table 1: Essential software with which to be familiar for this project

Application	Software
Deep learning	PyTorch Google Colab
Coding and algorithms	Robotic operating system (ROS), Python, OpenCV (Open-Source Computer Vision Library), C++, Arduino IDE (integrated development environment)
Operating system	Raspbian, Ubuntu 16.04
Simulator	Gazebo, Udacity self-driving car

Project goals

1. Build a one-tenth scale model autonomous (level five) racing car.
2. Have the car self-drive on a track based on a trained CNN model.
3. Allow the car to autonomously learn how to tackle sharp turns while maintaining a maximum speed and without losing control.

Future development

1. Add model predictive controller to the system to control the speed.

Note that Figure 5 shows in blue and red the project goals and future developments

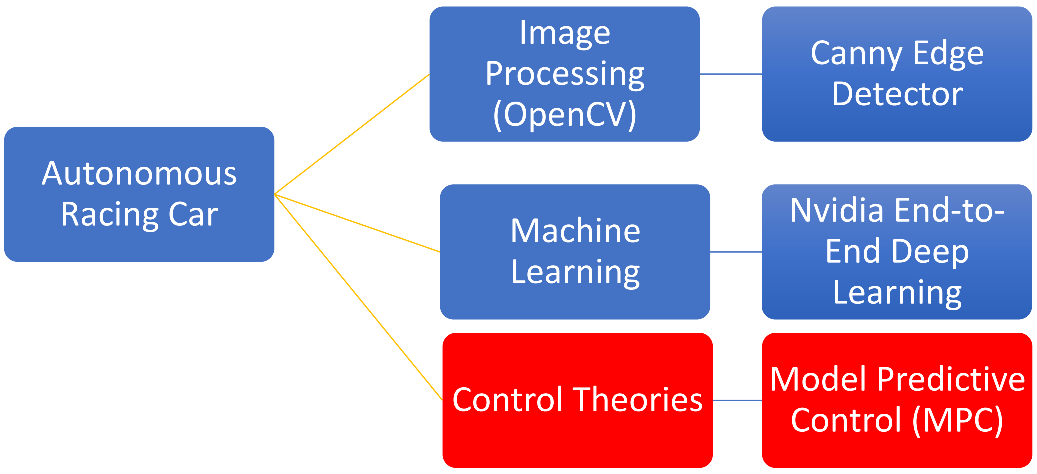

Autonomous Racing Car
Image Processing (OpenCV)
Canny Edge Detector
Machine Learning
Nvidia End-to-End Deep Learning
Control Theories
Model Predictive Control (MPC)

Figure 5: Blue present goals, Red future developments

Chapter 2: Literature Review

2.1 Racing Car Platforms

A full-scale car provides the best conditions for testing and experimenting with a self-driving system. However, this is expensive for students and researchers at universities. The only viable alternative is to use a scaled RC car, which has similar characteristics to a full-scale car [17]. Almost all high-ranked universities (such as the Massachusetts Institute of Technology [MIT], the University of California, Berkeley [Berkeley] and the Georgia Institute of Technology [Georgia Tech]) and research labs have built their own platforms. Table 1 provides a brief overview of prominent platforms by different universities and start-up companies. The most expensive system shown in Table 1 comes from Georgia Tech, with a budget of more than US$19,000. Georgia Tech equipped their car with a fully built-in control system, which has a powerful computer and accurate sensors, including GPS, LiDAR and a high-definition (HD) stereo camera. According to the MPC Laboratory at Georgia Tech [10], their system is suitable for both indoor and outdoor experiments, because of its novel, adaptive MPC system [11].

This literature review found that the most suitable platforms from Table 2 (for a researcher or student) are the 'f1/10th' from the University of Pennsylvania, the 'BARC' (Berkeley Autonomous Race Car) from Berkeley and the 'Donkey Truck', Figure 11. The reasons for choosing these projects are that they are cost effective, use a variety of sensors, have excellent documentation and are all open-source projects.

Table 2: Prominent scaled robot car platforms

Platform	**owner**	**Scale**	**Software**	**Sensors**	**Main Controller**	**Est. Cost**
RACECAR	MIT	Traxxas 1/10th scale car	Open-source ROS Python	LiDAR, IMU, depth camera Stereo camera	Nvidia Jetson TX1	$2500
f1/10th	University of Pennsylvania	Traxxas NOS (Nitrous Oxide Systems) Deegan 38 Rally 1/10th scale car	Open-source ROS Python	IMU, Scanning laser (LiDAR), Stereo camera	Nvidia Jetson TX1	$2500
AutoRally	Georgia Tech	HPI Baja 5SC 1/5th scale car	Open-source	Flea3 Camera, a Microstrain IMU NavTech GPS	ASUS Z170I motherboard with an Intel LGA 1151 processor	$19000
Donkey Truck	Open-source	1/16th Scale truck	Open-source Raspbian Python	Raspberry Pi camera (depends)	Raspberry Pi 3	$250
Hamster Micro UVG	Cogniteam	1/10th scale ground vehicle	Non-open-source	LiDAR, HD Camera, 3D Compass, GPS wheel encoders	Raspberry Pi 3	$3000
BARC	University of California at Berkeley	Traxxas Rally 1/10 scale car	Open-source ROS Python	IMU Stereo camera Wheel encoders Sonar	ODROID-XU4	$650

Images removed due to copyright restriction.

Figure 6: RACECAR by MIT, one-tenth scale platform [13]

Figure 7: f1/10th by the University of Pennsylvania, one-tenth scale platform [14]

Images removed due to copyright restriction.

Figure 8: AutoRally by Georgia Institute one-fifth scale platform [10]

Figure 9: Donkey Truck (open-source), one-fifth scale platform [15]

Images removed due to copyright restriction.

Figure 10: Hamster Micro UVG by Cogniteam, one-tenth scale commercial autonomous ground vehicle [16]

Figure 5:BARC by Berkeley, one-tenth scale platform [17]

2.2 Approach to Designing the Model Predictive Controller

An MPC is a feedback controller that uses a model to make a prediction about the future outputs of a process. As the MPC is driving a car, it uses the car characteristics to make a prediction about the future trajectory, based on previously observed actions. Then it chooses the optimal trajectory and drives to that trajectory [18]. An MPC has advantages over other controllers. For example, it can handle multi-input and multi-output systems, handle constraints (safety, speed and physical constraints) and incorporate information for future reference, to improve controller performance [18]. However, because the MPC solves an online optimisation problem at each step, it requires a powerful processor with a large memory and a fast processing speed. Many research papers, such as [24], [25] and [26], have demonstrated the importance of the MPC.

Studies of autonomous vehicles have focused on developing novel and enhanced MPCs. Rosolia et al. [19] developed a learning MPC for iterative tasks. They stated that the controller was able to improve its performance by learning from previous iterations [19]. The controller was tested on a one-tenth scale BARC platform [17], which shows (on a recorded video) that the car was able to increase its speed after each iteration, until it reached the maximum speed (Figure 12).

Image removed due to copyright restriction.

Figure 6: Evolution of the velocity profile over the iterations

Another unique study at Georgia Tech developed a tube-based MPC [11]. The algorithm combines the sampling-based MPC with the linearisation-based trajectory optimisation method. The resulting performance was justified in a simulation and on an outdoor track, using a one-fifth scale autonomous car (Figures 13 and 14).

Images removed due to copyright restriction.

Figure 14: Test track for the one-fifth scale vehicle

Figure 13: Trajectory traces of test run with tube-MPPI (model predictive path integral) controller

An earlier study by Kritayakirana [23] also argued for pushing autonomous race cars to their limit. Inspired by race car drivers, the controller is separated into steering and longitudinal modules that consist of feedforward and feedback controller submodules. Kritayakirana [23] stated that this would make it easier to analyse the controller (Figure 15).

Image removed due to copyright restriction.

Figure 15: Driving at the limits

2.3 Neural Network Method

2.3.1 Convolutional Neural Network

Deep learning is a subset of machine learning that provides machines with the ability to perform human-like tasks without human involvement [20]. Deep learning is implemented through artificial neural networks, which are now considered the most promising approach to many complex designs, including image and speech recognition, robots, virtual assistants and self-driving cars' [21]. According to Raschka [21], convolutional neural networks (CNN) have led to an extraordinary improvement in machine learning and computer vision applications. CNN resembles how the human brain works. It consists of an input layer, an output layer, and one or more hidden layers (Figure 16).

Image removed due to copyright restriction.

Figure 7: Example of a CNN model for classifying a handwriting digit [22]

2.3.2 Nvidia End-to-End Deep Learning for Self-Driving Cars

Recent research from Nvidia states that it is possible to create a self-driving car system based on CNN [22]. They achieved this using a data set comprising only the driver steering angle with 10 fps (frames per second) images from a front camera [22] (Figure 17).

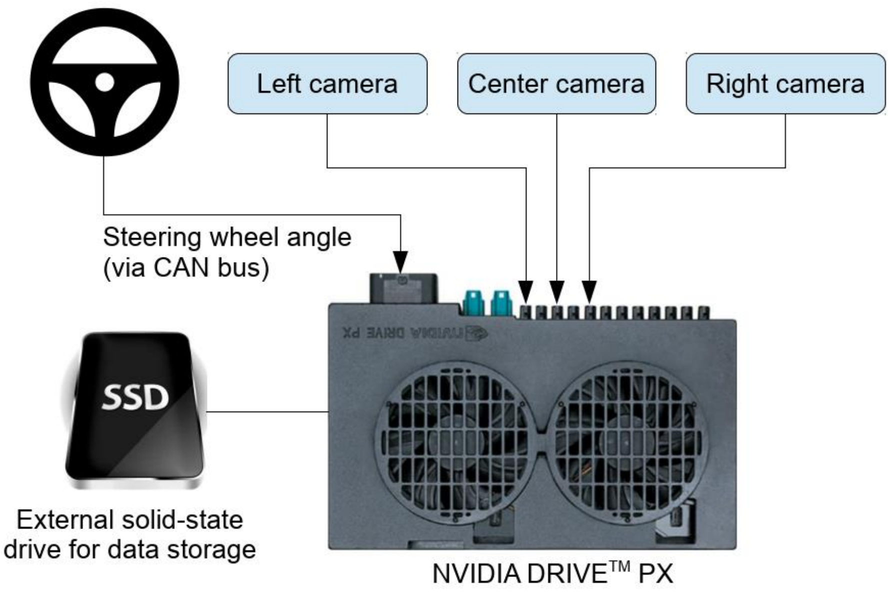

Left camera
Center camera
Right camera
Steering wheel angle (via CAN bus)
SSD
External solid-state drive for data storage
NVIDIA DRIVE™ PX

Figure 8: High-level view of the data collection system

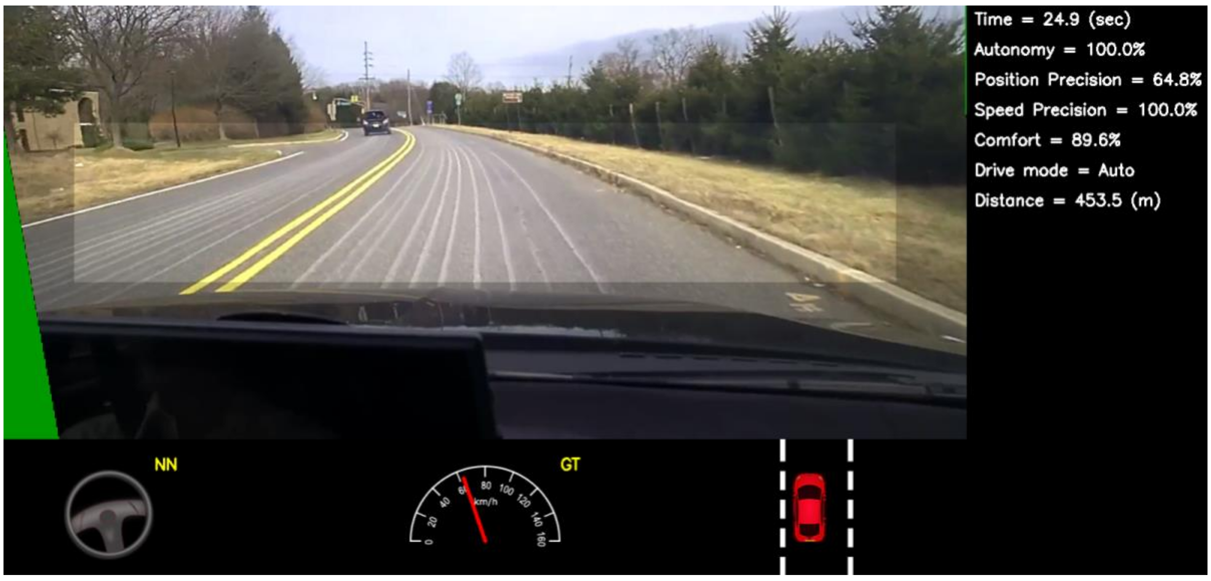


Figure 9: Screen shot of the simulator in interactive mode. The green area on the left is unknown because of the viewpoint transformation. The highlighted, wide rectangle below the horizon is the area which is sent to the CNN.

Nvidia states that, as a result of their experiment Figure 18, their system learned how to drive in traffic on local roads, with and without lane markings and on highway areas or unpaved roads, all with minimum training data [22]. However, the CNN model could only control the steering angle of the car; the speed was controlled by an adaptive cruise control.

Chapter 3: Simulation Steps and Result

Before implementing the Nvidia CNN model in the actual car, the system was first tested on a simulator [9]. This helped to run an enormous number of tests, collect and process data. The author of this thesis used the Udacity Self-driving car course [9] as a learning resource on how to build and implement the Nvidia CNN model. The original contribution of this work, apart from the Udacity course, was developing a CNN model suitable for Autonomous racing cars. Fast driving behavior of the car was recorded and gathered as a learning data inputs to train the developed CNN model. At the end of chapter three, the simulation result shows that the trained CNN model was able to drive the car autonomously on a high speed.

3.1 Simulator Software

The study of self-driving cars has encouraged an interest in developing simulation software to test them. Carsim software is one of the highest ranked simulators and used by car manufacturers including GM (General Motors), Toyota and Land Rover [8]. It creates a realistic environment and returns accurate measurements. However, this accuracy comes with a high price and individuals cannot afford to buy this system. Fortunately, there are several open-source software programs supported by leading companies and organisations such as Udacity and Nvidia.

Udacity developed an open-source car simulator for testing self-driving cars to use in their nanodegree course [9]. Their aim was to enable their students to test their code before applying it to a real car scenario, reducing time and risk. When the simulator is started, it first displays a training mode and an autonomous mode with two different maps (Figure 19). The idea behind this app is to manually drive a car, using either a controller or a keyboard, with the recording feature activated to capture images of the track, driving angle and speed values. After that, the data are processed to create a trained CNN model. The model (created in Python) is then tested by establishing a connection with the car simulator software. Running the autonomous mode prompts the CNN model to drive the simulated car autonomously.

Figure 10: Udacity's self-driving car simulator, main window

3.2 Developing the Neural Network

As disclosed in the second chapter, the CNN was developed based on the Nvidia end-to-end model (Figure 20) [20]. This means that, in the first stage, the input images are 66x200 pixels and consist of three channel colours. Then, in the second stage, the images are normalised (e.g. a 255 pixel image will be equal to 1 after normalising), which reduces the computational process without affecting the image appearance.

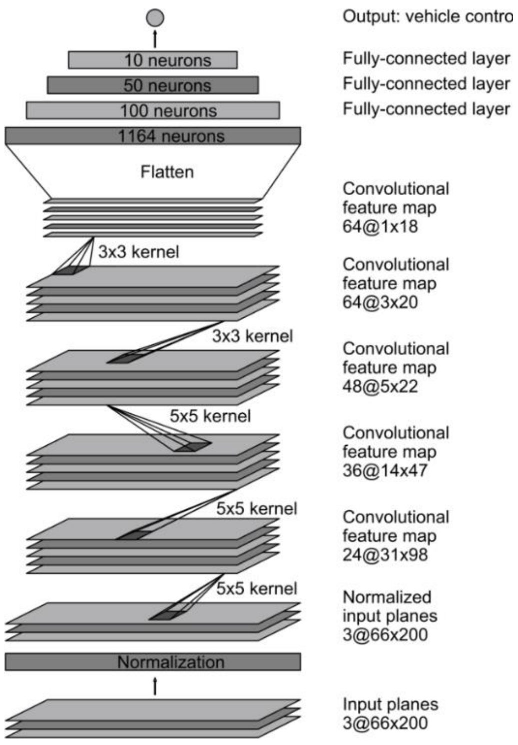


Figure 20: Nvidia end-to-end model

From the third stage to the sixth stage, the convolutional neural layer is applied, with different kernel and image sizes. The image size is reduced from one layer to the next, which enables the CNN to extract detail features of an image. The number of filters also increases (e.g. from 24 to 36, 48 and 64).

In the last stage of this model, the data are flattened. This converts the array of input images into a one-dimensional array (1x18x64) so it can be fed to the fully-connected layer. Nvidia's paper provides more detail about the workings of the model [20].

Writing the code for this model using Python alone would not be an easy task. One solution to this is to use Keras, an open-source library. Keras define their program as 'a high-level neural networks API [application programming interface], written in Python and capable of running on top of TensorFlow, CNTK [the Microsoft Cognitive Toolkit], or Theano')[27], Figure 21. Appendix 2 contains the Python code developed for the Nvidia model, using Keras.

Image removed due to copyright restriction.

Figure 21: Keras was founded by François Chollet, an MIT researcher

Six steps must be followed before creating the Nvidia CNN model. The first step is collecting the data, followed by balancing the data, processing the data and data augmentation. Processing the data requires splitting the data into training and validation groups. The following section explains these steps in detail.

3.2.1 Collecting the Data

Building a trained CNN model based on the behavioural cloning technique requires data of a human operator actually driving [20]. The quality of the recorded data affects how well the trained CNN model makes it decisions when it is driving the car autonomously. A poor dataset will result in a poorly trained CNN model. This means that the CNN drives are determined by how well the operator drives the car—their skills are reflected in the neural network. In addition, a huge dataset is required to familiarise the CNN with different driving scenarios and train it to extract more features from the track. These qualities are essential for a well-trained CNN.

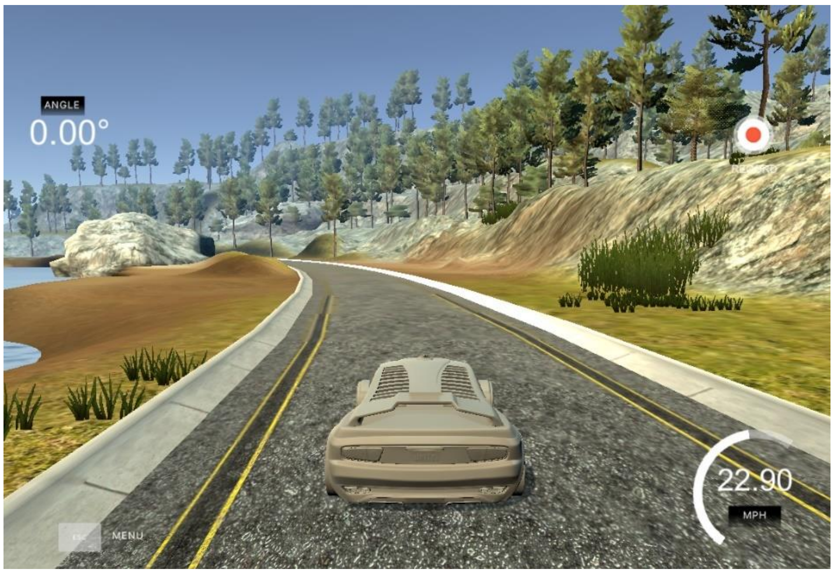


Figure 11: Udacity car simulator, running the car in training mode

Collecting the data begins with running the simulated car in training mode (Figure 22). The selected track has a flat terrain, which is designed to challenge the neural network to overcome sharp turns. Different parts of the track have different textures, edges and borders. For example, the border on the bridge and the border of sand. There are also different degrees of curvature in the layout, with different levels of brightness and shadow. All of these features are extracted by the CNN.

The car laps the track three times in one direction (counter clock wise), then in the reverse direction for another three laps. The first laps around the track produce a left turn bias, which is caused by predominantly steering left. Driving the car in only one direction around the track would skew the data to one side. This would create a problem for the neural network by predisposing the model to predict left turns. In short, driving in both directions helps to ensure that the car is not biased to drive to one side.

The simulated car is equipped with three cameras, located right, left and centre. When recording, the simulator captures images and stores them in a file called ‘Data’ (Figure 23).

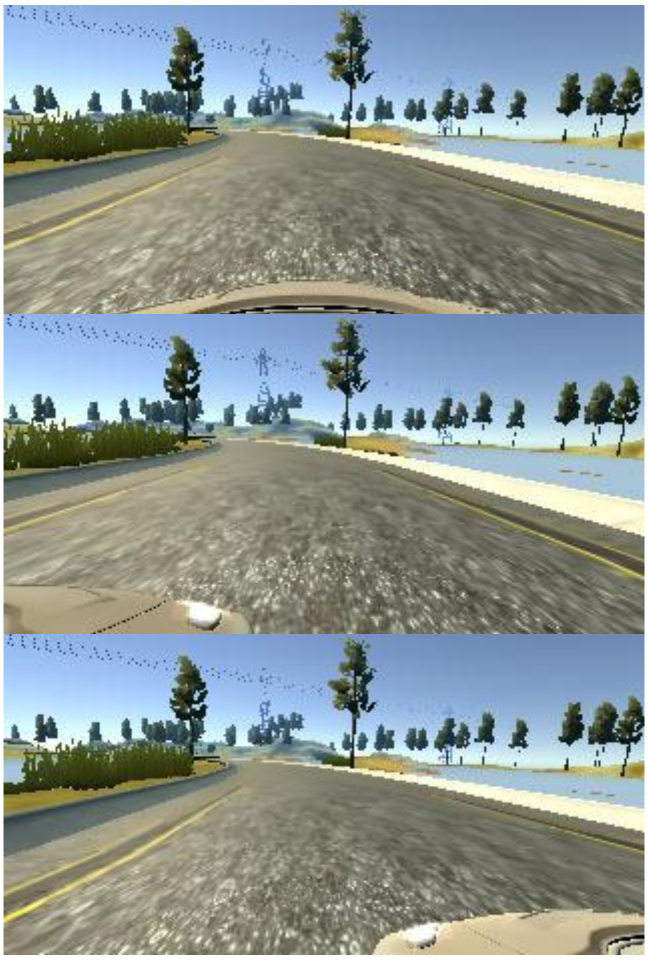
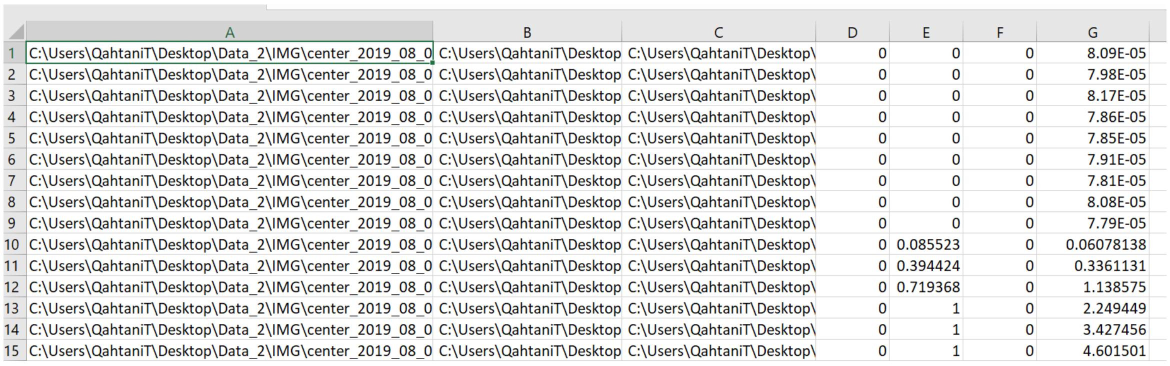


Figure 12: Recorded images of different views (middle, right and left)



	A	B	C	D	E	F	G
1	C:\Users\QahtaniT\Desktop\Data_2\IMG\center_2019_08_0	C:\Users\QahtaniT\Desktop	C:\Users\QahtaniT\Desktop\	0	0	0	8.09E-05
2	C:\Users\QahtaniT\Desktop\Data_2\IMG\center_2019_08_0	C:\Users\QahtaniT\Desktop	C:\Users\QahtaniT\Desktop\	0	0	0	7.98E-05
3	C:\Users\QahtaniT\Desktop\Data_2\IMG\center_2019_08_0	C:\Users\QahtaniT\Desktop	C:\Users\QahtaniT\Desktop\	0	0	0	8.17E-05
4	C:\Users\QahtaniT\Desktop\Data_2\IMG\center_2019_08_0	C:\Users\QahtaniT\Desktop	C:\Users\QahtaniT\Desktop\	0	0	0	7.86E-05
5	C:\Users\QahtaniT\Desktop\Data_2\IMG\center_2019_08_0	C:\Users\QahtaniT\Desktop	C:\Users\QahtaniT\Desktop\	0	0	0	7.85E-05
6	C:\Users\QahtaniT\Desktop\Data_2\IMG\center_2019_08_0	C:\Users\QahtaniT\Desktop	C:\Users\QahtaniT\Desktop\	0	0	0	7.91E-05
7	C:\Users\QahtaniT\Desktop\Data_2\IMG\center_2019_08_0	C:\Users\QahtaniT\Desktop	C:\Users\QahtaniT\Desktop\	0	0	0	7.81E-05
8	C:\Users\QahtaniT\Desktop\Data_2\IMG\center_2019_08_0	C:\Users\QahtaniT\Desktop	C:\Users\QahtaniT\Desktop\	0	0	0	8.08E-05
9	C:\Users\QahtaniT\Desktop\Data_2\IMG\center_2019_08_0	C:\Users\QahtaniT\Desktop	C:\Users\QahtaniT\Desktop\	0	0	0	7.79E-05
10	C:\Users\QahtaniT\Desktop\Data_2\IMG\center_2019_08_0	C:\Users\QahtaniT\Desktop	C:\Users\QahtaniT\Desktop\	0	0.085523	0	0.06078138
11	C:\Users\QahtaniT\Desktop\Data_2\IMG\center_2019_08_0	C:\Users\QahtaniT\Desktop	C:\Users\QahtaniT\Desktop\	0	0.394424	0	0.3361131
12	C:\Users\QahtaniT\Desktop\Data_2\IMG\center_2019_08_0	C:\Users\QahtaniT\Desktop	C:\Users\QahtaniT\Desktop\	0	0.719368	0	1.138575
13	C:\Users\QahtaniT\Desktop\Data_2\IMG\center_2019_08_0	C:\Users\QahtaniT\Desktop	C:\Users\QahtaniT\Desktop\	0	1	0	2.249449
14	C:\Users\QahtaniT\Desktop\Data_2\IMG\center_2019_08_0	C:\Users\QahtaniT\Desktop	C:\Users\QahtaniT\Desktop\	0	1	0	3.427456
15	C:\Users\QahtaniT\Desktop\Data_2\IMG\center_2019_08_0	C:\Users\QahtaniT\Desktop	C:\Users\QahtaniT\Desktop\	0	1	0	4.601501

Figure 13: Comma separated values (CSV) containing a list of images with corresponding throttle, steering angle, speed and break data

Another .CSV file is also created, which includes a record (index) of the angle, throttle, break and speed values assigned to their exact image (Figure 24).

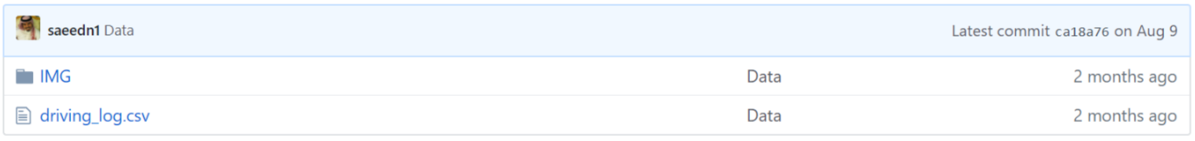
The most important data category in this project is the steering angle. The steering angle is the only factor that must be predicted to ensure that the car knows how to steer based on the part of the track it is navigating. Helpful data can be added by rerecording specific turns, so that the model has a more complete knowledge of the sharp turns. Steering values are used to analyse the drive. A value of zero means that the car drives straight, positive (+) values mean the car drives right and negative values (−) mean the car drives left.

Google Colab and GitHub

Google Colab is a free ‘Jupyter notebook environment that requires no setup and runs entirely in the cloud’ [28]. It includes all Python libraries, including OpenCV, and machine learning libraries (TensorFlow, Keras).

The dataset needs to be uploaded to the cloud to work in Google Colab. This can be done using GitHub, which provides a hosting space for data and software development, Figure 25. Uploading the dataset to GitHub involves the following steps.

1. Cd directory of the data using cmd
2. git init
3. git add
4. git commit -m “Data”
5. git remote add origin https://GitHub.com/saeedn1/ARCAR.git
6. git push -u origin master



saeedn1 Data		Latest commit ca18a76 on Aug 9
IMG	Data	2 months ago
driving_log.csv	Data	2 months ago

Figure 14: Dataset uploaded to GitHub

3.2.2 Balancing the Data

Google Colab commands are used to import the dataset from GitHub to the Jupyter notebook. After that, the driving angle data are extracted and plotted as a histogram distribution (Figure 26).

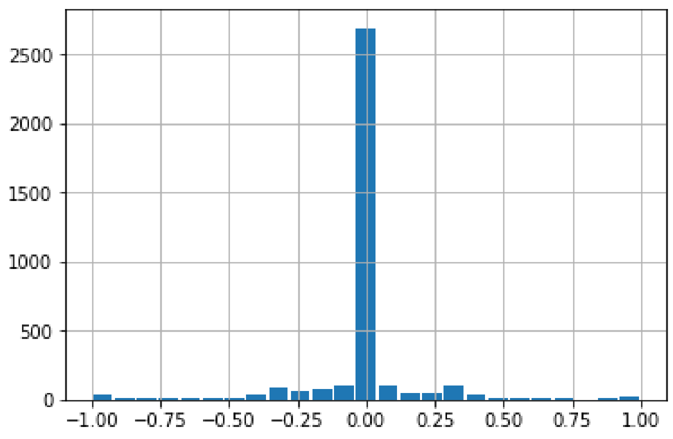


Figure 15: Dataset driving angle histogram distribution

The histogram shows that a large number of the driving angles are at zero, which indicates that the car is driving forward most of the time. If the model was trained based on this data, it would create a problem for the neural network, which could bias the model towards predicting a zero angle. This would make the car biased towards driving straight all the time.

To fix this, all the samples above a certain threshold are rejected, to ensure that the histogram is more uniform and not biased towards a specific steering angle. Every bin can have up to 350 samples (Figures 27 and 28).

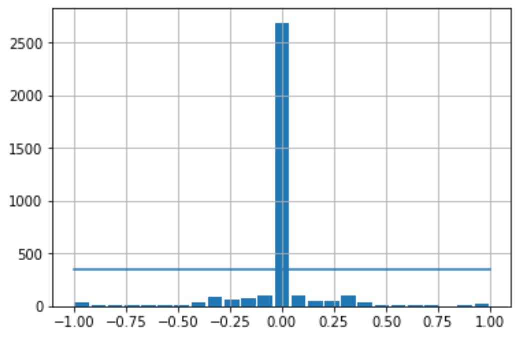


Figure 16: Applying a 350 limit

```
total data: 3546
removed: 2389
remaining: 1157
[<matplotlib.lines.Line2D at 0x7f66a97c3940>]
```

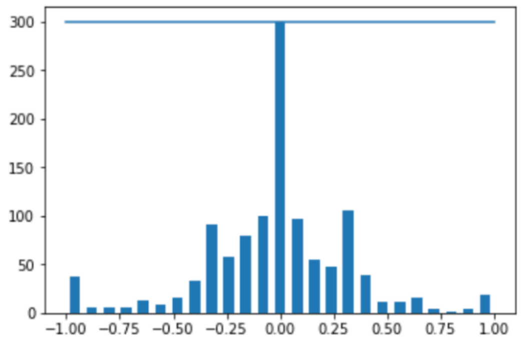


Figure 17: Removing data above the limit

Before removing the excess, the data need to be shuffled. This is because driving forward (zero angle) data are collected from various parts of the track. For that reason, angle data from across the dataset is removed, rather than data from only one part of the track. In some cases, recovery laps are added to the dataset to make the car drive back to the middle of the track. The data collected in this study are well balanced, so it was not necessary to add a recovery lap.

3.2.3 Processing Images

Processing the images is an important step [22]. It reduces the computational time, increases the neural network's accuracy and speeds up the learning time. Steps for processing the images are as follows.

1. Remove unnecessary features in images. The top section of the image is almost entirely scenery, consisting of trees and mountains. The very bottom of the image is just the hood of the car. These features are not relevant for determining the steering angle, as there is no relationship between the scenery surrounding the road and the steering angle of the car.
2. Change the colour space to 'yuv' (as recommended by Nvidia) [20].
3. Add a Gaussian blur to smooth the image and reduce noise.
4. Decrease the image size for faster computations as a smaller image is easier to work with. Match the image size used by the Nvidia model architecture [20].
5. Normalise the image by dividing by 255. This does not have any visual effect. The resulting image can be seen below (Figure 29).

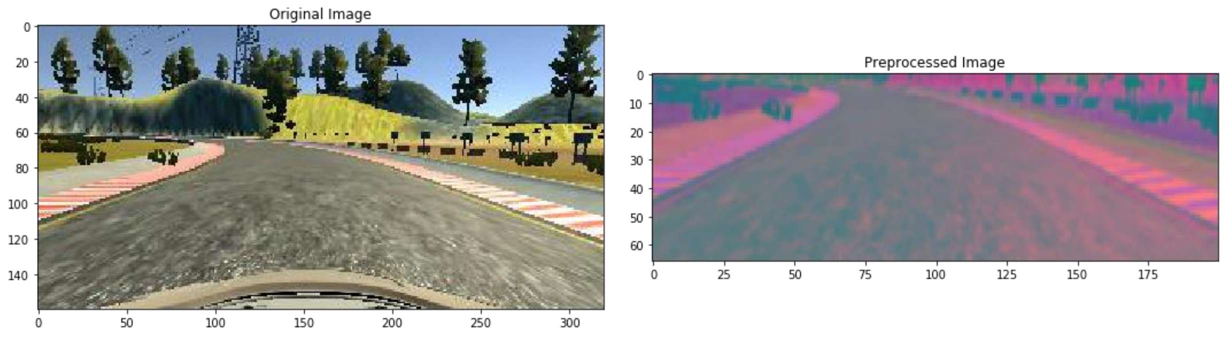


Figure 18: Original image and the pre-processed image on the right

3.2.4 Data Augmentation

Data augmentation is the process of using the existing dataset to create new images. This is done by applying transformations to the existing images, which results in a brand-new set of images that can be added to the dataset. This method helps to add variety to the dataset and helps the model learn more efficiently (because it acquires more data to work with). The actual dataset includes around 1000 images with corresponding steering angles. Then, 50% of the data are augmented, which increases the dataset to around 1500 images with corresponding steering angles. The imgaug Python library is used to augment the images [30]. This library has a long list of different types of augmentation, but only four types were used in this project. These techniques are zooming, panning, adjusting brightness and flipping the steering angle. Figures 30 to 33 show the different augmentation methods and their effects on the original image.

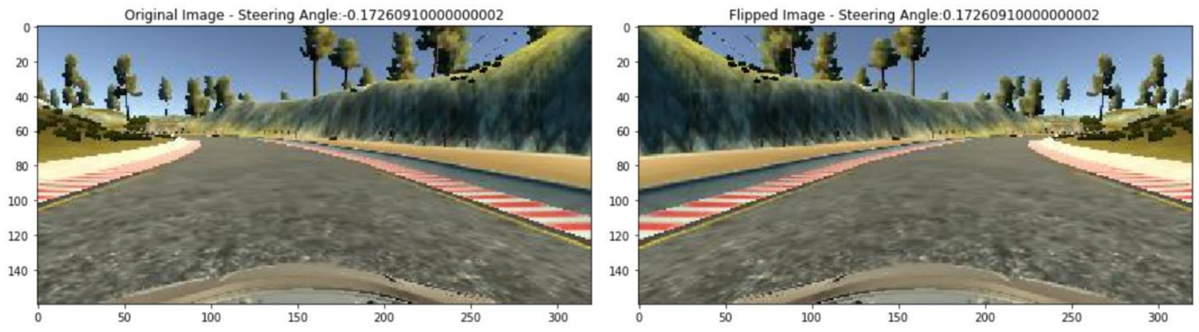

Original Image - Steering Angle:-0.17260910000000002
Flipped Image - Steering Angle:0.17260910000000002

Figure 30: (Left) original image, (right) flipped version

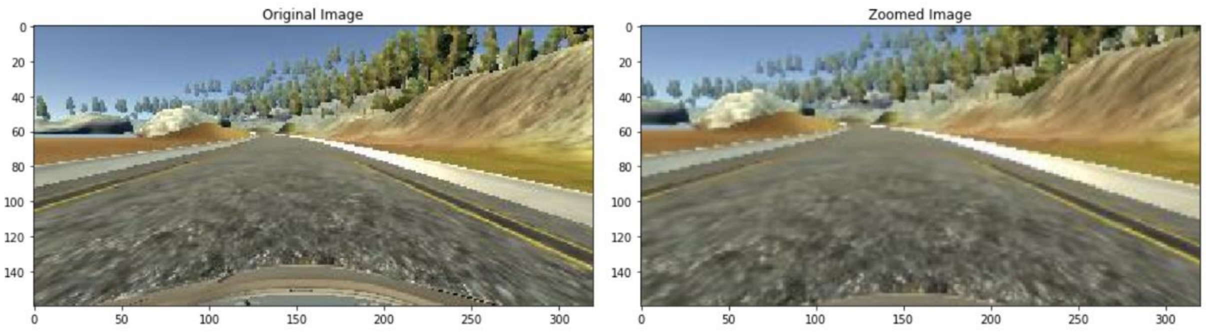


Figure 19: (Left) original image, (right) zoomed version

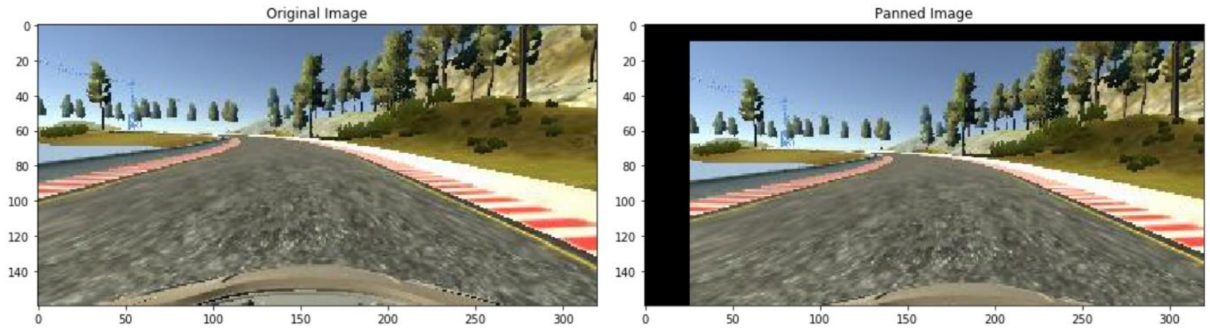


Figure 20: (Left) original image, (right) panned version

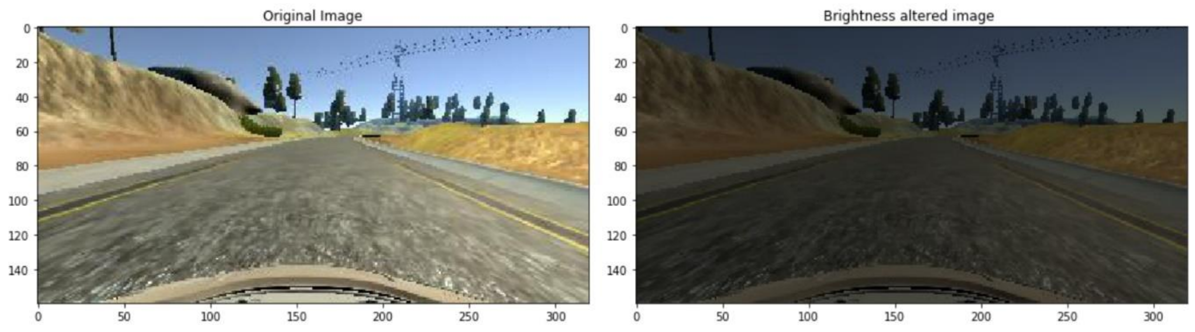


Figure 21: (Left) original image, (right) changed brightness version

3.2.5 Training and Validation Split

Data are split into two categories, training and validation. The first category (training) is used to teach the neural network the appropriate steering angles for each part of the track by extracting the necessary features. The second category (validation) is used to test the trained neural network to determine whether it can produce valid steering angle values. In other words, the neural network should not overestimate or underestimate (underfit or overfit) the steering angle, compared with the actual steering angle.

The data are split using the Python library, sklearn [29]. One-quarter of the data are split to use as the validation dataset and the rest are used to train the neural network. Sklearn has the capability to split arrays or matrices into random train and test subsets of uniform distribution. This ensures that the steering angles in both datasets (training and validation) are balanced and that the model is slightly more biased towards predicting a zero angle. From Figure 34, it is clear that the data are not exactly similar, but both follow a general trend, so that both the right and left steering angles are somewhat balanced.

```
Text(0.5, 1.0, 'Validation set')
```

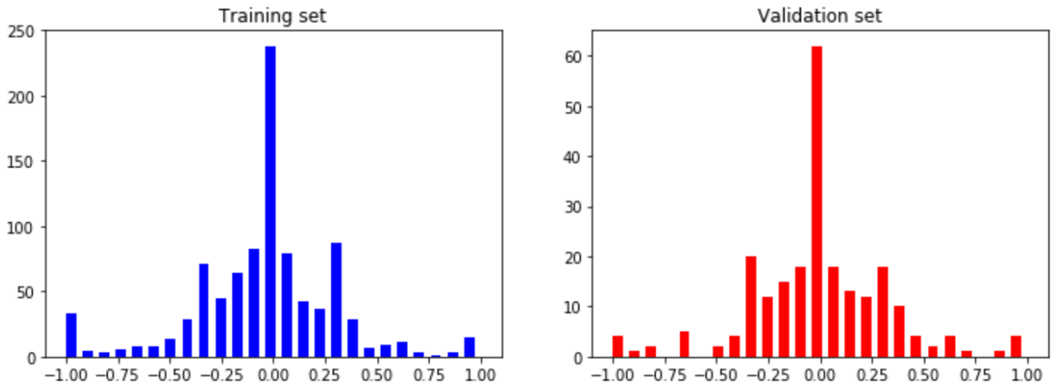


Figure 22: Validation dataset histogram shows that the model produces an acceptable distribution compared with the training dataset, which indicates that it follows the same trend

3.2.6 Batch and Fit Generator

The final step for creating the generator to train the model is to use a batch generator. The main benefit of the generator is that it can create augmented images while operating, rather than augmenting all of the images at one time and storing them, which would use valuable memory space. This is much more memory efficient, as the data are only used when they are required, rather than being stored in memory even when they are not being used. In this study, the number of epochs was set to ten and the number of steps (images) per epoch was 350. In addition, 200 validation steps (images) were used. It is important to validate the model with actual, collected data, not with the augmented data. Keras documentation provides further details about the parameters of the fit generator [33]. Figure 35 shows that the model has been trained well, as it does not overestimate or underestimate the steering angle during validation.

```
Text(0.5, 0, 'Epoch')
```

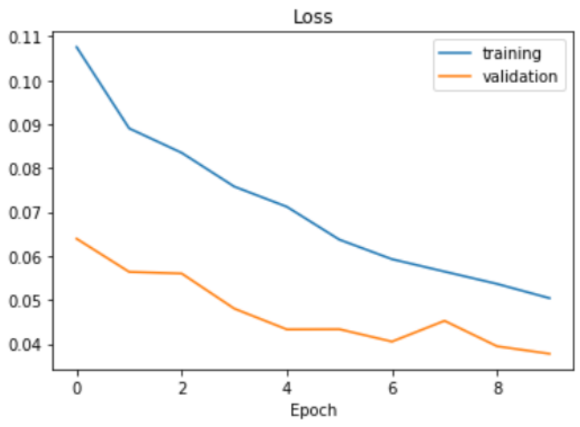


Figure 23: Training and validation of the developed model

3.3 Running the Simulator with Flask

The Nvidia end-to-end CNN model was created in Python. To run this model in the Udacity simulator a web application must be used. This project uses Flask, a lightweight WSGI (web server gateway interface) web application framework. It is designed to make getting started quick and easy, with the ability to scale up to complex applications. It began as a simple wrapper for Werkzeug and Jinja and has become one of the most popular Python web application frameworks [31]. Flask was created in Python, from the Python script (drive.py) [32]. It is used to connect the Nvidia model with the Udacity simulator. The block diagram below gives an overview of how Flask connects the server with the client.

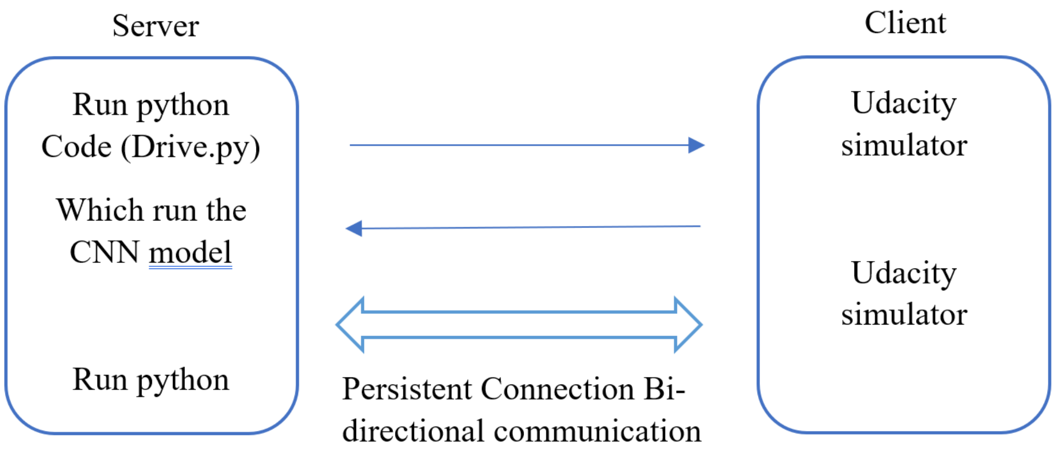


Figure 36: Flask run python as Server to connect to the Udacity simulatore

3.4 Simulation Result

The CNN model was tested in the Udacity simulator, on the same track from which the data were collected. A server connection was established between the CNN model and the Udacity simulator, using Flask, to run the car in autonomous mode. The CNN model was able to drive the car autonomously at maximum speed of 22 MPH. The car drove the entire track without driving off the road. It was driving in the middle for most of the track, which indicates that the CNN model was well trained. The model was then tested on a different track, which the CNN model had not seen before. The car was not able to drive on the second track because the model was trained with a limited dataset and it was not trained to drive up or down hills. Because the actual track (in the robotic lab) has a flat surface (no hills) the CNN model was not trained with this additional dataset and it was assumed that it would be capable of driving in the lab environment with the limited dataset.

Chapter 4: Platform Building Methodology

Many scaled versions of self-driving cars have been developed for advanced research with an affordable budget. This method has been used by many high-ranked universities, specialist laboratories and commercial companies (Table 2). This chapter will present the RC car used in this project, the electronic components (with their mechanical modifications, e.g. 3D prints) and the software developed for this study. Table 3 lists the components and the total project cost (AUD) at the time of building.

Table 3: Project materials bill

Num.	Description	Qty.	Distributor	Cost (AUD)
1	ODROID-XU4Q	1	Hardkernel Co., Ltd	72.61
2	RTC backup battery	1	Hardkernel Co., Ltd	3.71
3	eMMC module reader board for OS upgrade	1	Hardkernel Co., Ltd	2.22
4	ODROID-XU4 case clear	1	Hardkernel Co., Ltd	8.00
5	myAHRS+	1	Hardkernel Co., Ltd	111.18
6	WiFi module 0	1	Hardkernel Co., Ltd	7.12
7	USB3.0 eMMC module writer	1	Hardkernel Co., Ltd	14.68
8	WiFi module 5 A	1	Hardkernel Co., Ltd	11.71
9	128 GB eMMC module XU4 Linux	1	Hardkernel Co., Ltd	97.69
10	5 V/4 A power supply Australia plug	1	Hardkernel Co., Ltd	8.15
11	Kuman 180 pieces M3 nylon male female hex utility	1	Amazon	16.59
12	12 volt regulator, DROK	1	Amazon	10.97
13	TRX male to xt60 connector plug female	1	Amazon	7.78

Num.	Description	Qty.	Distributor	Cost (AUD)
14	ELP USB with camera 2.1 mm lens 1080 p	1	Amazon	79.99
15	WINGONEER expansion prototype shield i/o extension board module for Arduino Nano v3.0	1	Amazon	9.65
16	Python machine learning: machine learning and deep learning with Python, scikit-learn, and TensorFlow, 2nd Edition	1	Amazon-Kindel	33.87
17	Male XT60 to female TRX Traxxas connector	1	AliExpress	17.98
18	3PCS XT60 parallel battery connector cable	1	AliExpress	11.36
19	4PCS T / XT60 male plug to 5.5 mm DC plug	1	AliExpress	11.36
20	1 Set of 120Pcs male female m3 metric brass standoff spacer	1	AliExpress	14.52
21	Traxxas 2200 mAh 7.4 V 2-cell 25 C LiPo battery	1	metrohobbies	69.99
22	Traxxas charger dual EZ-peak 100 W	1	metrohobbies	164.99
23	Traxxas Ford Fiesta ST Rally Valentino Rossi edition 1/10 4WD rally car	1	metrohobbies	539.99
24	Electronic component deck	1	Flinders University at Tonsley	0
25	IMU 3D printed case	1	Flinders University at Tonsley	0
26	Camera 3D printed case	1	Flinders University at Tonsley	0
		Shipping		66.71
		Total		1,392.82

4.1 RC Car Platform

As stated in the thesis outline (Section 1), the Traxxas VR46 Ford Fiesta ST Rally one-tenth scale RC car was chosen to be the project chassis (Figure 5). This RC car is suitable for on-road or off-road driving tests. The rally RC car comes as a ready-to-use unit, equipped with a water proof 2.4 GHz (Gigahertz) top qualifier (TQ) radio transmission system for remote-control, a XL-5 electronic speed controller (ESC) and a powerful Titan 550 motor. It has trick, oil-filled shocks suspension and four-wheel drive. The transmission shaft transmits torque from the rear axle to the front axle [34].

Image removed due to copyright restriction.

Figure 37: VR46 Ford Fiesta ST Rally dimensions [34]

Aside from these features, the main reason for choosing this car was its larger size, which can accommodate the electronics system designed for this project (Figure 37). Finally, the rally car is appropriate for rugged terrain, which is required for several tests where the car may drive off the track and hit objects.

4.2 Electrical Components

The electronic system was designed with the goal of a platform where control theories, algorithms and deep learning methods could be implemented. The main computer for the system is Odroid XU4Q by Hardkernel. The Arduino Nano works as a sub controller, which controls the servo and the DC (direct current) motor output, in addition to reading the remote-control RC signals. The system has only two types of sensors, a camera and an IMU device. Figure 38 is an overview of how the system is connected and the following sections provide expiations of each component.

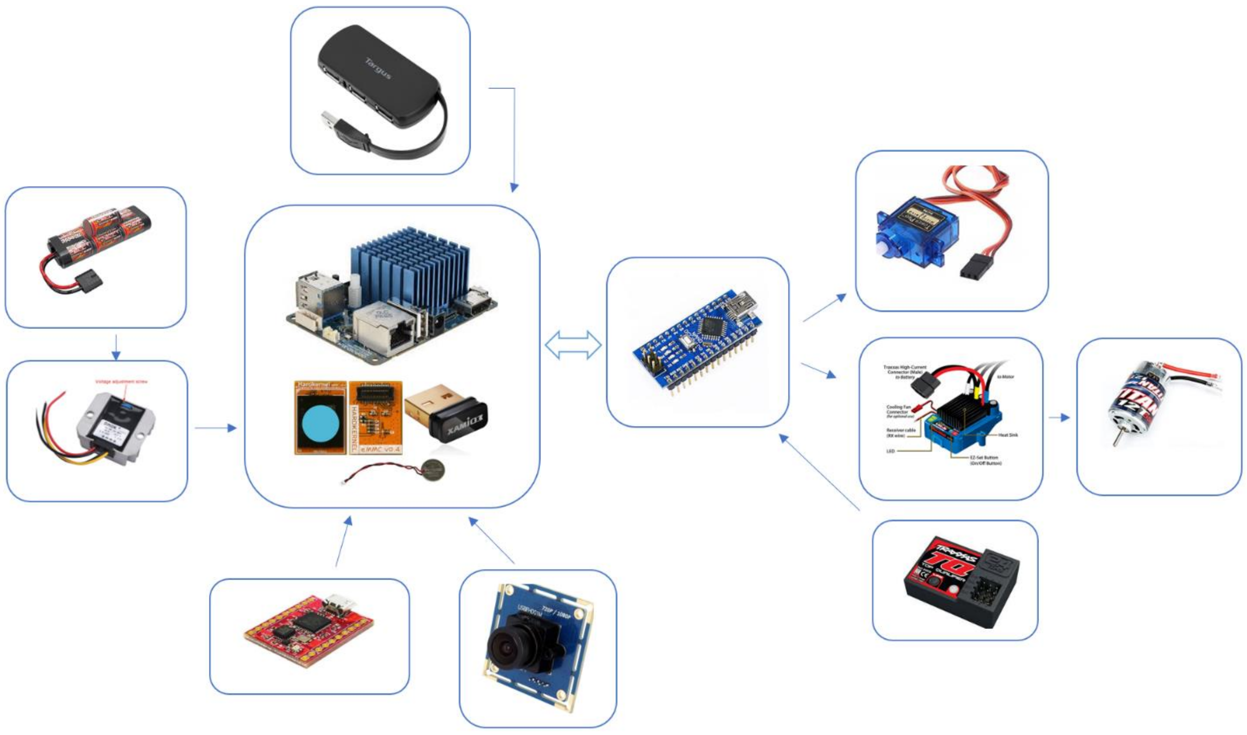


Figure 24: The electronic system consists of a main and sub computer, a camera and IMU as sensors and servo and DC motors as the system output, which need to be controlled.

Most of the electronic components of the system were installed in a deck on top of the rally car. The deck is made from a 6 mm acrylic material, which was cut by a laser machine. The 3D design (Figure 39) is a modified version of the BARC 3D design [17]. Figure 40 shows the final, assembled electronic components that go on the deck.

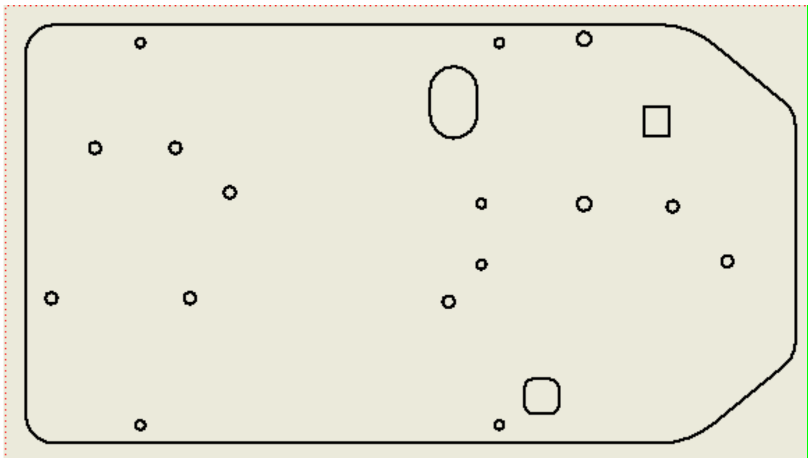


Figure 25: 3D design of the deck that holds most of the system's electronic components

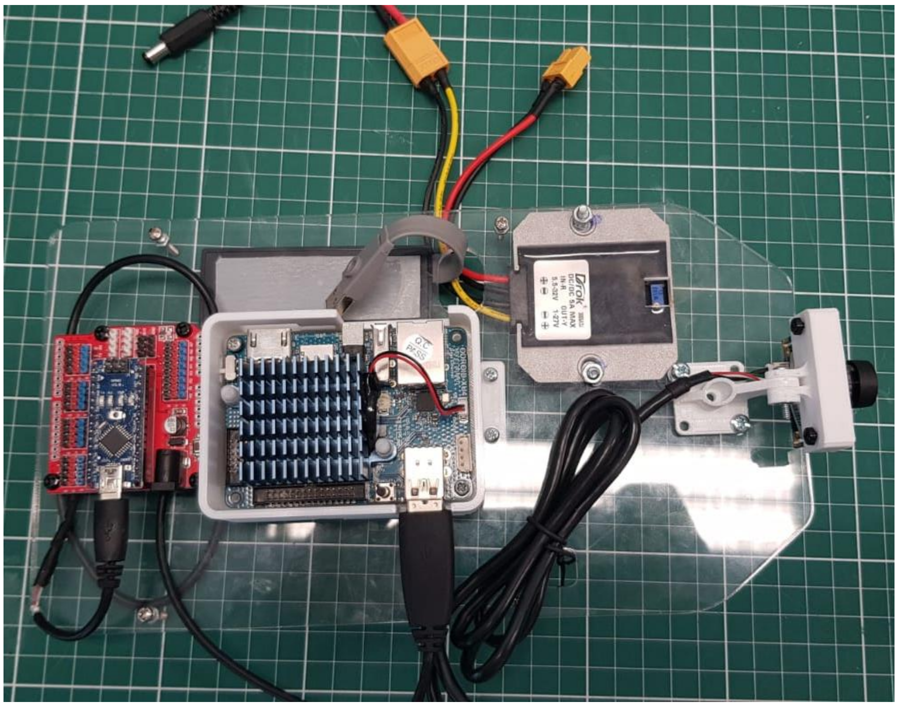


Figure 40: Final assembly of the electronic components for the deck

4.2.1.1 *Microcomputer*

The Odroid XU4Q from Hardkernel is the primary computing device (Figure 41). The Odroid XU4Q uses a Samsung Exynos5422 Cortex-A15 2 GHz and Cortex-A7 Octa core central processing units (CPUs). These CPUs offer a powerful computational process, compared with other developing boards, such as Raspberry Pi 3. The XU4Q has standard computer elements, for example 2 GB LPDDR3 RAM, a two USB 3.0 host, a one USB 2.0 host, Gigabit Ethernet port, HDMI 1.4 A for display and an array of general-purpose input/output (GPIO) pins. In addition, it comes with a Linux Kernel 4.14 LTS, which makes running the Ubuntu MATE operating system (OS) possible. The XU4Q has two types of storage. One is an eMMC (electronic multi-media controller) chip, the other is a microSD (secure digital) card. The eMMC chip offers a fast reading and writing speed, compared with the microSD card, and for that reason, a 125 GB eMMC chip was used to install the Ubuntu MATE OS and store data.

Image removed due to copyright restriction.

Figure 41: Odriod UX4Q board specifications. The eMMC Module connector can be found on back side of the board.

The Odroid XU4Q requires a power supply of 5 V (Volts) and current of 5 A (Amps). It is important to design the power system to meet the required power rating. Supplying a lower ampere, such as 2 A, will not launch the Odroid OS.

Odroid was chosen because it offers strong computing performance for a low price (A$72). It can carry out powerful processing for control techniques, such as MPC. However, it has a poor GPU (graphics processing unit), which cannot process a large number of images. This drawback affected the deep learning section of this project. Chapter 4 details this drawback and provides solutions for future study. A 3D-printed housing was created to firmly hold the Odroid board on the deck, as shown in Figure 39. According to Hardkernel, the Odroid board performs better than other miniature computers on the market, with the same level of processor (Table 4).

Table 4: Comparison between different miniature computers

Benchmarks (Index Score)	**Raspberry Pi 3**	**ODROID-C1+**	**ODROID-C2**	**ODROID-XU4**
Unixbench: Dhrystone-2	865.4	1571.6	2768.2	5941.4
Unixbench: Double-Precision Whetstone (x3)	1113	1887.3	3076.8	6186.3
Nbench 2.2.3: Integer (X40)	619.92	1173.6	1808.92	2430.52
Nbench 2.2.3: Floating-Point (X100)	781.8	1245.3	2300.3	3787.3
Benchmarks (Index Score)	Raspberry Pi 3	ODROID-C1+	ODROID-C2	ODROID-XU4

4.2.1.2 *Arduino Nano*

Arduino Nano was used as a sub controller for this project. The Arduino was programmed to work in two modes simultaneously. The first mode receives commands from the Odroid board and sends them to the servo and DC motors. The commands are for the throttle of the DC motor and the steering angles of the servo. In the second mode, the Arduino Nano captures the radio signals transmitted from the throttle and steering angles and directs them to the Odroid, rather than to the servo and DC motor. These recorded signals are used to gather data concerning the operator’s driving behaviour.

Image removed due to copyright restriction.

Figure 26: Sub controller Arduino Nano with its shield

The Odroid XU4Q comes with GPIO pins that are able to perform the same task given to the Arduino. However, because of the Odroid GPIO's complexity and lack of user manuals, the Arduino Nano was chosen. The Arduino Nano controller is based on the ATmega328P microcontroller. It comes with 22 digital input/output pins, six of which have a pulse with modulation (PWM) feature. The Arduino runs on 5 V with a 19 mA (milliamps) power consumption (Figure 42a).

The Funduino Nano V3 Shield was used to hold the Arduino Nano board. The Funduino Nano V3 Shield offers a practical way of connecting the servo and DC motors to the Arduino board (Figure 42b). The connection pin specifications for the Arduino, motors and radio signals can be found in Table 5.

Table 5: Pin-seven to ten of the Arduino pins that have a PWM feature

Device (components)	**Pin**
Throttle (from RC receiver channel 2)	D7
Steering (from RC receiver channel 1)	D8
Servo	D9
ESC (for 3-phase Motor)	D10

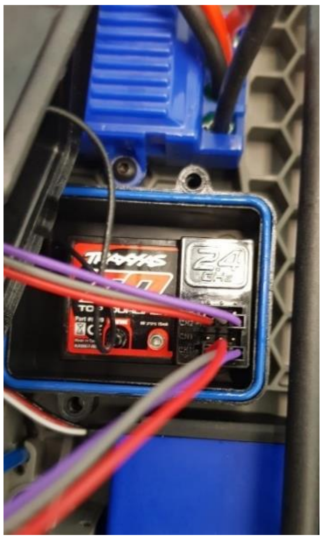


Figure 27: Motor cables are removed and replaced with the D7 and D8 Arduino pins

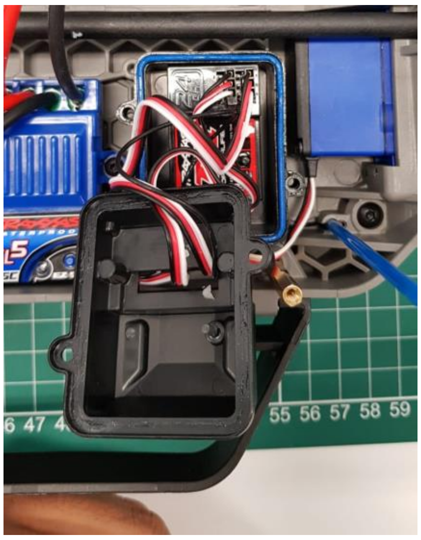


Figure 28: The servo and DC motor are directly connected to the radio receiver

By default, the servo and DC motor are connected to the radio receiver (channel 1 and 2) (Figure 44). These connections must be removed and replaced with pins D7 and D8 of the Arduino shield board, as suggested in Table 4 (Figure 43).

4.2.1.3 *Camera*

Self-driving cars come with various sensors, such as LiDAR, wheel encoders and cameras. However, cameras are the most useful and important source of data for a self-driving car. Because this project was developed based on Nvidia's end-to-end model, it uses only a camera as the source of data. Cameras come in three different types, monocular, stereo and 3D-RGBD (red, green, blue, depth). For simplicity, this project used a monocular ELP USB 1080 p (progressive scan) camera, with a 2.1 mm lens (Figure 45).

Figure 29: ELP USB 1080 p HD camera with 2.1 mm lens

This camera has a high frame rate (120 fps), a 180 degree wide-angle lens, 2 megapixel high pixel technology and accurate colour reproduction. It uses a UVC (USB video class) drive for the Ubuntu OS. These features come at an affordable price of around A$80, which was the reason for choosing this camera.

4.2.1.4 *Inertial Measurement Unit*

This project used the myAHRS+ IMU from Hardkernel (Figure 46) [36]. It is a low-cost, high performance AHRS (attitude heading reference system). It consists of a triple axis 16-bit gyroscope, accelerometer and 13-bit magnetometer. Moreover, it has three communication methods—a UART (universal asynchronous receiver/transmitter), an I2C (inter-integrated circuit) and a virtual (USB) COM (communication) port. The IMU is easily connected to the Odroid board using a micro-c cable. In addition, myAHRS+ IMU comes with on-board software, including an extended Kalman filter with a 100 Hz (Hertz) output rate and accurate reading [36]. The IMU is the second sensor for this project. Its purpose is to correct the steering angle (heading) of the car when the MPC is in use. However, the MPC will not be implemented until a future iteration of the project. So the IMU is not yet in use.

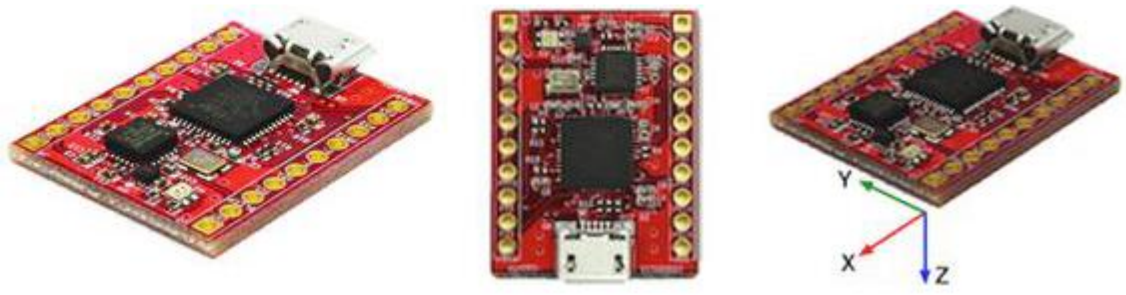


Figure 30: myAHRS+

4.2.1.5 *Networking Components*

Figure 31: WIFI Module 0

Figure 32: WIFI Module 5A

The WIFI module offers a way to connect the Odroid board to the internet. The Ubuntu OS desktop of the Odroid can be controlled and observed using Nomachine software. WIFI Module 0 (Figure 47) from Hardkernel was used first. It cost around A$5 and offered 150 Mbps (Megabits per second) data transfer speed. It was replaced with WIFI Module 5A (Figure 48), which offered 200 Mbps data transfer speed, dual-band and a high-speed USB.

4.3 Power System

The main source of power was an 8.7 V battery connected to the rally car electrical components. It was also connected to a DC-DC voltage converter (Figure 49). The converter supplies the control (electrical) system with the required power rating. The block diagram in Figure 50 shows the power distribution plan for all of the electrical components.

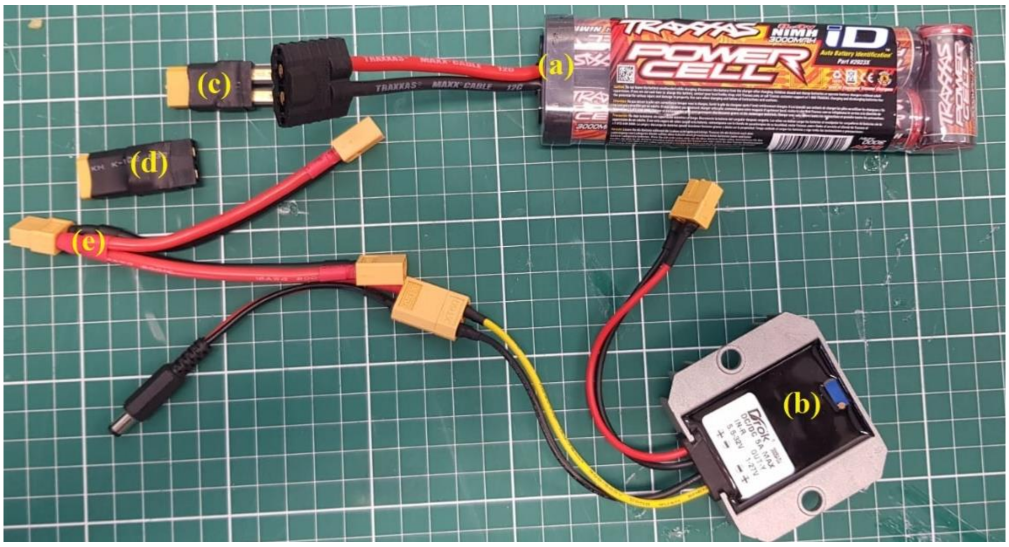


Figure 33: (a) 8.7 V battery (b) DC-DC converter (c) XT60 F-M Traxxas (d) XT60 F-F Traxxas (e) distribution cable

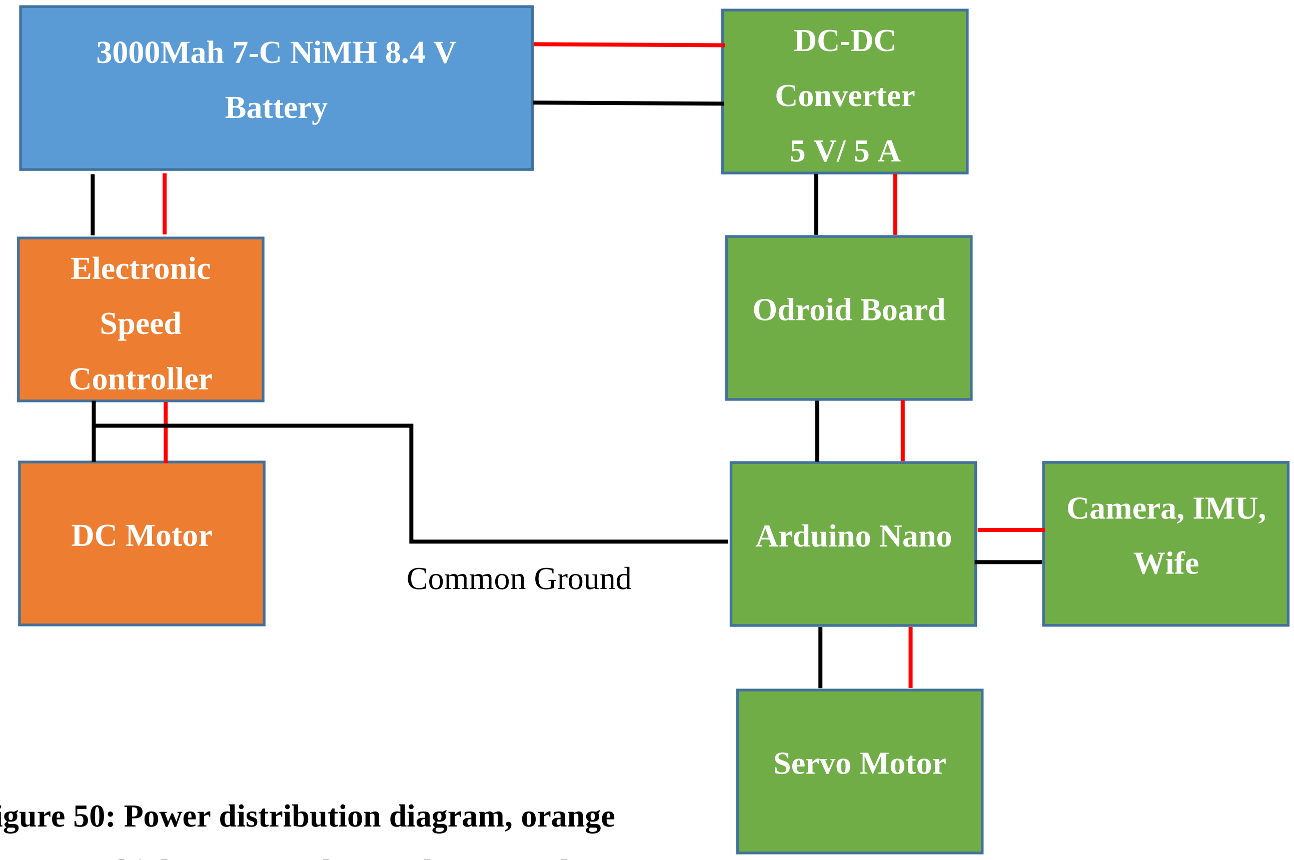


Figure 50: Power distribution diagram, orange boxes use high power and green boxes use low power

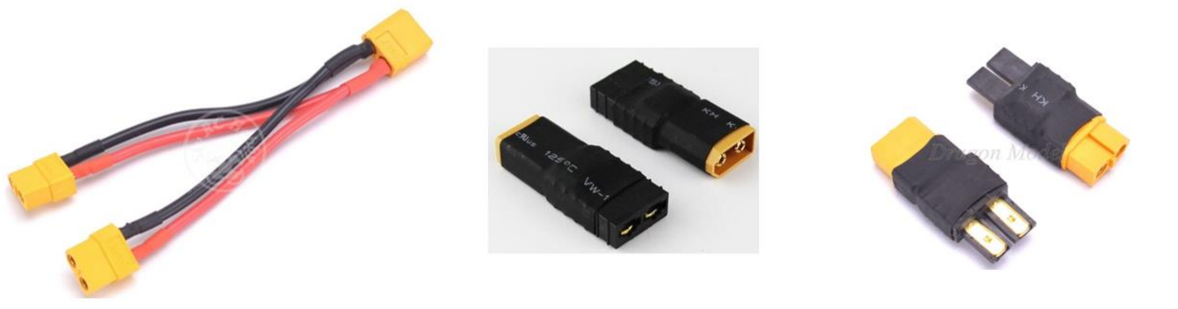
4.3.1.1 *Battery*

Image removed due to copyright restriction.

Figure 51: Power Cell battery , 3000 mAh (milliamp Hour) (NiMH [nickel–metal hydride battery], 7 C flat, 8.4 V)

The Traxxas 74064-1 Ford Fiesta ST Rally car comes with a 3000 mAh 7-C NiMH 8.4 V flat pack battery (Figure 51). This battery is capable of producing enough power to run the rally car's electrical components (ESC and radio receiver), the Odriod board, the Arudino, the sensors (camera and IMU) and the servo and DC motors. The rally car, including the control system, can run for ~1.5 hours on one fully-charged battery.

The battery specification 'C' refers to its charging/discharging rate [35]. For example, a battery with 1 C will fully charge or discharge 30 A in one hour. Upgrading to 2 C will reduce the charging/discharging time to 30 minutes. In our case, the chosen battery has 7 C and 30 A, which means that the battery will charge/discharge 30 A in one-seventh of an hour. This is a fast charging/discharging rate.



(a) cable power distributor (b) Traxxas Male to XT60 male (c) Traxxas Female to XT60 Male

Figure 34: Power distributor cable and port converters required for the battery connection

An XT60 power port cable distributor was used to split the battery power. However, the battery has a Traxxas female port, which does not fit with the distributor cable port. For that reason, a Traxxas female to XT male (and m–f) converter connecter was used (Figure 52).

4.3.1.2 *DC-DC Converter*

The main computer (Odroid) requires a 5 V/5 A power rating. However, the battery supplies an 8.4 voltage. For that reason, a DROK synchronous DC-DC adjustable converter was used (Figure 53). Its input voltage varies from 32 to 5.5 V and the output (step down voltage) varies from 1 to 27 V with maxima ampere of 5 A. In addition, DROK converters are waterproof and have an adjustable VDC (Volts DC) solid, durable power supply board.

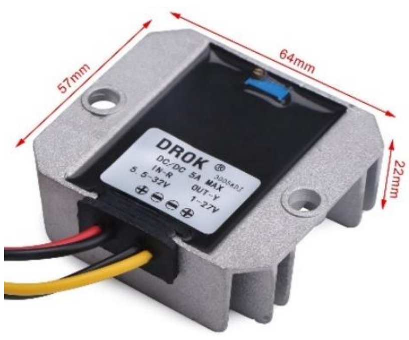


Figure 35: Dimension of DROK converter

The DC converter is located on the bottom side of the electronic deck, where it is secured with a 4 mm screw (Figure 55). A two XT60 female power port was soldered to the converter lead to provide a safe installation when plugging it into the battery (Figure 47). In addition, another XT60 male to power jack port cable was used to connect the voltage converter to the Odroid board (Figure 54).

Figure 36: XT60 female to power jack cable

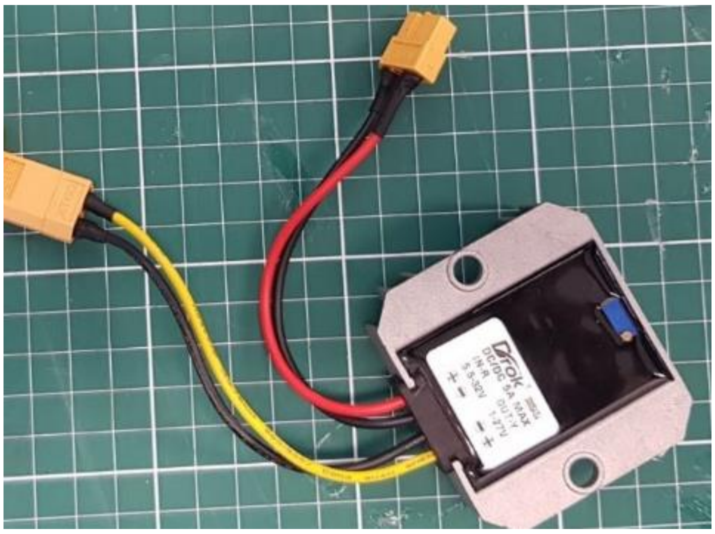


Figure 37: XT60 power port was soldered and covered with heat shrink tubes

4.4 Final Look of the Built Car

At this stage, the scaled RC car platform was completed (Figure 56). It has a camera and IMU as input sensors. The servo and DC motors are connected to the Arduino board, instead of directly connected to the ESC. In manual mode, the car runs by receiving transmitter signals for the throttle and steering angle, which are delivered via the radio receiver to the Arduino board, then via the Arduino board to the motors. This power distribution design was tested with a successful result, wherein the Odroid minicomputer and other electronic components were activated by sufficient power.

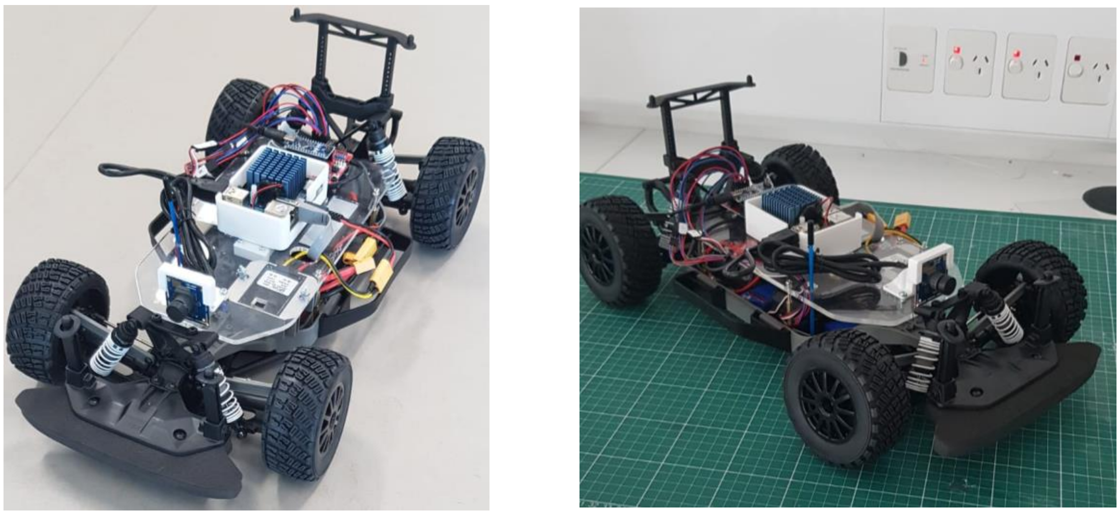


Figure 56: Final built of the scaled racing car with all of the components

Chapter 5: Running Practical Test Software

This chapter of the project will take the reader through installing the project OS, including essential software, such as Python and ROS. It also covers developing and running practical tests, the Nvidia model, lane detection with camera and Servo and DC motor control via Arduino Nano. Having detailed the finished RC car in the previous chapter, this chapter primarily concerns the work on the Odroid XU4Q board (main processor). More importantly, this chapter presents how the Nvidia model performed, compared with the simulation test result in chapter three.

5.1 Ubuntu MATE

Ubuntu is one of the largest Linux based desktops [38]. Other systems, such as Android (widely used by smartphone users), are also considered Linux based. Ubuntu comes in different versions and with its own desktop environment. Ubuntu MATE desktop OS was used for this project. Ubuntu MATE provides an intuitive and attractive desktop environment, using traditional metaphors [38]. Users who are familiar with Windows or Mac OS will have a familiar experience. Moreover, Ubuntu MATE is an open-source system. It is free of charge and developers have the right to contribute to and modify the system. In addition, Linux is supported by Intel, Red Hat, Samsung, IBM, Google, Canonical, Oracle, Microsoft and others. According to the Ubuntu MATE organisation, over 4000 developers contributed to Linux over the last 15 years [38].

5.1.1 Booting the Odroid with Ubuntu MATE

The Odroid XU4Q can be booted with the Ubuntu MATE system via two methods. The first method is to use and flash a MicroSD card. The second method is to flash an eMMC. The Odroid has an easy-access hardware switch to select either method for booting (Figure 41). The Odroid MicroSD interface supports a high performance UHS-1 module and a MicroSD (SD-class10) module. Conversely, the Odroid eMMC interface offers much faster reading and writing speed, compared with both MicroSD modules. The Odroid user manual [38] compares the performance of these booting methods with a 512 MB file (Figure 57). The results follow.

- The eMMC 5.0 storage is ~7x faster than the MicroSD Class-10 card in read tests.
- The MicroSD UHS-1 card is ~2x faster than the MicroSD Class-10 card in read tests.
- The MicroSD UHS-1 card provides a great, low-cost option for many applications.

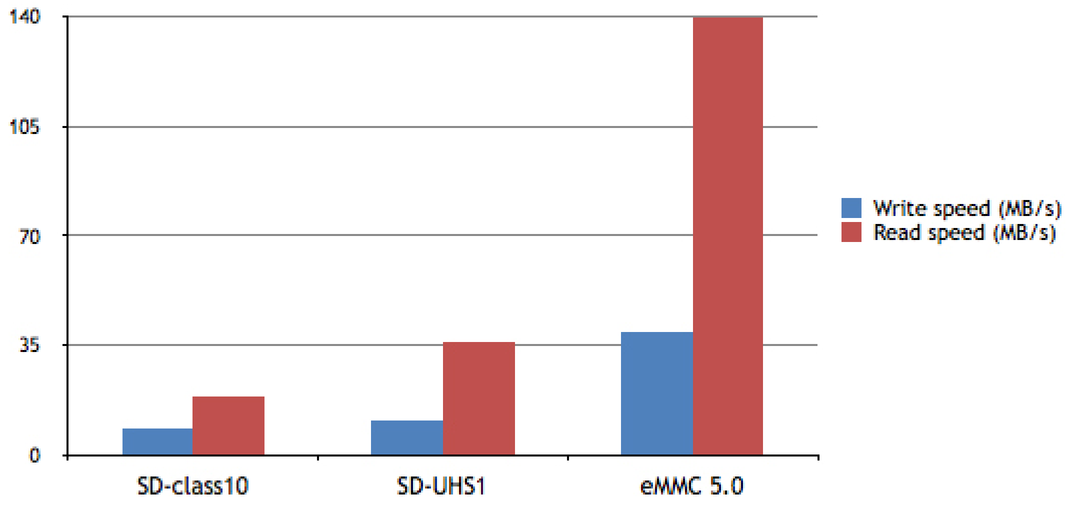


Figure 38: Odroid eMMC and MicroSD performance results

Table 6: Read/write performance of different storage modules

	SD-class10	**SD-UHS1**	**eMMC 5.0**
Write speed (MB/s)	8.5	10.8	39.3
Read speed (MB/s)	18.9	35.9	140

Table 6 and Figure 58 demonstrate that the eMMC provides the fastest reading and writing performance and, for that reason, a 128 GB eMMC was chosen for this project. The following section, explains the steps for flashing an Ubuntu MATE OS and then booting the Odroid computing board.

5.1.2 Flashing the eMMC

An image file of the Ubuntu MATE 18.04.3 can be found on the Odroid support page [39]. The Odroid's eMMC is then flashed with the downloaded image file. The image size is 8 GB, which requires a stable internet connection and adequate space. The eMMC is 128 GB, meaning that there is enough space to install other software and applications, including Python, ROS and other Python based libraries (TensorFlow, OpenCV, Keras). An eMMC-to-USB adaptor is required to establish a connection between the eMMC and a laptop. In this project, the USB3.0

eMMC module writer (Figure 58) from Hardkernel was used. It is produced by the same manufacturer as the eMMC and chosen to avoid and any compatibility errors.

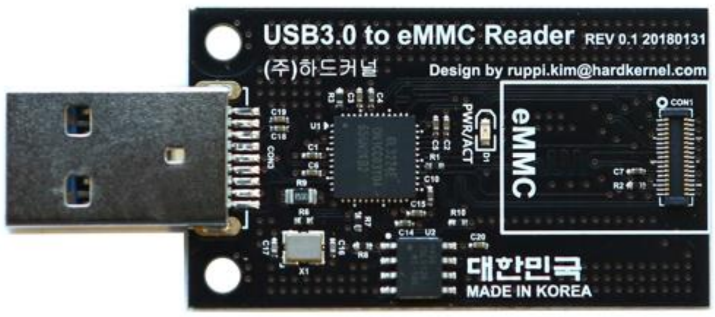
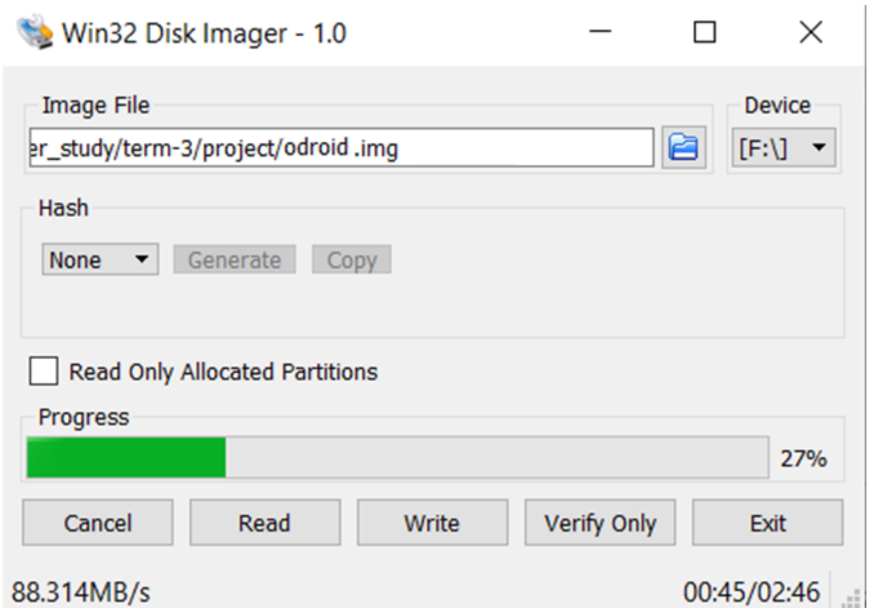

USB3.0 to eMMC Reader REV 0.1 20180131
(주)하드커널
Design by ruppl.kim@hardkernel.com
eMMC
PWR/ACT
대한민국
MADE IN KOREA

Figure 39: USB3.0 eMMC module writer

The following steps are required for flashing an eMMC (to be fulfilled sequentially).

1. Connect eMMC to USB3.0 eMMC Module Writer
2. Connect eMMC adapter (with eMMC) to laptop
3. Run the Win32DiskImager application (Windows)
4. Select the Ubuntu MATE image file
5. Select the eMMC storage directory
6. Hit start to flash the eMMC (Figure 59)
7. Safe-remove the eMMC



Win32 Disk Imager - 1.0

Image File: er_study/term-3/project/odroid .img

Device: [F:\]

Hash: None

☐ Read Only Allocated Partitions

Progress: 27%

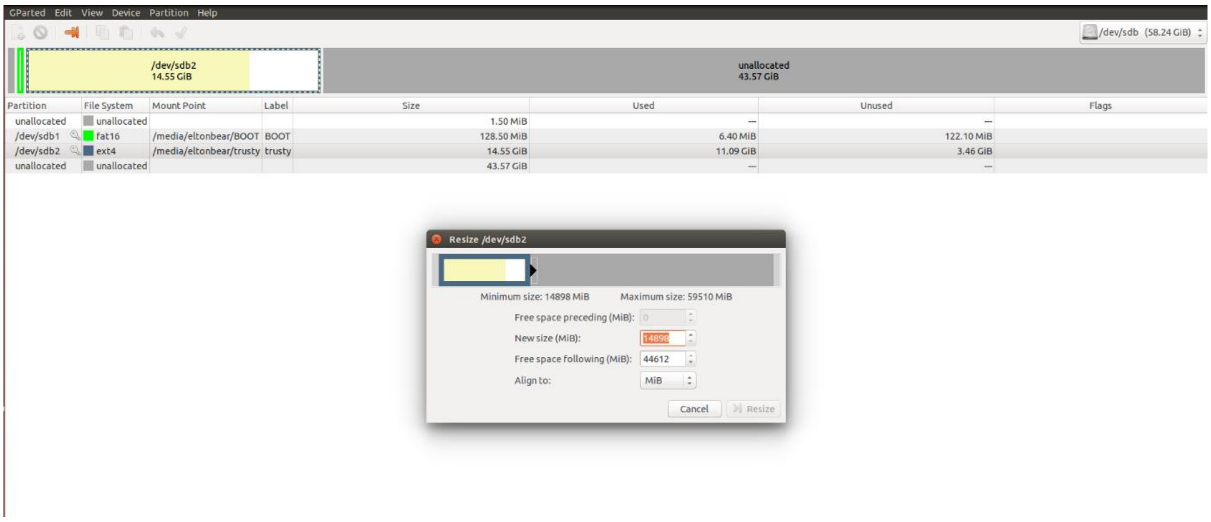
88.314MB/s 00:45/02:46

Figure 40: Flashing the eMMC with Ubuntu MATE OS

5.1.3 Resizing the eMMC

Flashing the 128 GB eMMC will shrink it to the size of the image file. To fix this issue the eMMC space needs to be resized using a special software, Gparted. Gparted is a free partition editor used for creating, deleting, resizing, moving, checking and copying disk partitions and their file systems [40]. It runs on Ubuntu (Linux) OS, so a laptop running Ubuntu is required for this fix. The following steps detail installing Gparted to fix the eMMC space issue.

1. Run Terminal to install Gparted by typing '$ sudo apt-get install gparted'
2. Plug the eMMC with the USB-to-eMMC adapter into the USB port
3. Run the Gparted software
4. Select the correct USB drive
5. Click to resize and drag the limit bar to the end to maximise the size of the eMMC (Figures 60 to 62)
6. Click save and remove



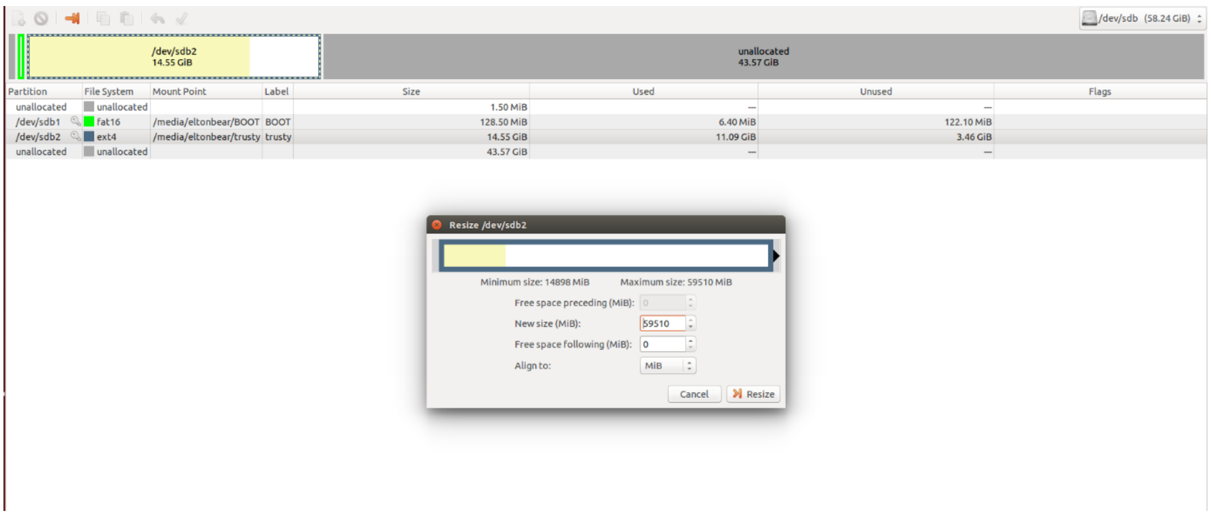
Partition	File System	Mount Point	Label	Size	Used	Unused	Flags
unallocated	unallocated			1.50 MiB	—	—	
/dev/sdb1	🟩 fat16	/media/eltonbear/BOOT	BOOT	128.50 MiB	6.40 MiB	122.10 MiB	
/dev/sdb2	🟦 ext4	/media/eltonbear/trusty	trusty	14.55 GiB	11.09 GiB	3.46 GiB	
unallocated	unallocated			43.57 GiB	—	—	

Resize /dev/sdb2

Minimum size: 14898 MiB   Maximum size: 59510 MiB

Free space preceding (MiB): 0
New size (MiB): 14898
Free space following (MiB): 44612
Align to: MiB

Figure 60: Selecting the desired storage drive and editing space



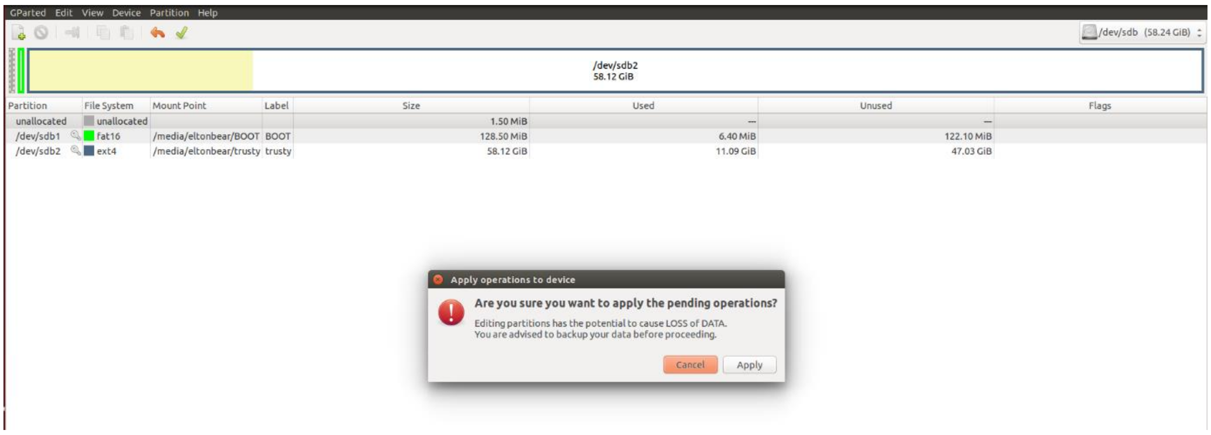
Partition	File System	Mount Point	Label	Size	Used	Unused	Flags
unallocated	unallocated			1.50 MiB	—	—	
/dev/sdb1	🟩 fat16	/media/eltonbear/BOOT	BOOT	128.50 MiB	6.40 MiB	122.10 MiB	
/dev/sdb2	🟦 ext4	/media/eltonbear/trusty	trusty	14.55 GiB	11.09 GiB	3.46 GiB	
unallocated	unallocated			43.57 GiB	—	—	

Resize /dev/sdb2

Minimum size: 14898 MiB Maximum size: 59510 MiB

Free space preceding (MiB): 0
New size (MiB): 59510
Free space following (MiB): 0
Align to: MiB

Figure 61: Moving the limit bare to the end to maximise the space



Partition	File System	Mount Point	Label	Size	Used	Unused	Flags
unallocated	unallocated			1.50 MiB	—	—	
/dev/sdb1	🟩 fat16	/media/eltonbear/BOOT	BOOT	128.50 MiB	6.40 MiB	122.10 MiB	
/dev/sdb2	🟦 ext4	/media/eltonbear/trusty	trusty	58.12 GiB	11.09 GiB	47.03 GiB	

Apply operations to device

Are you sure you want to apply the pending operations?

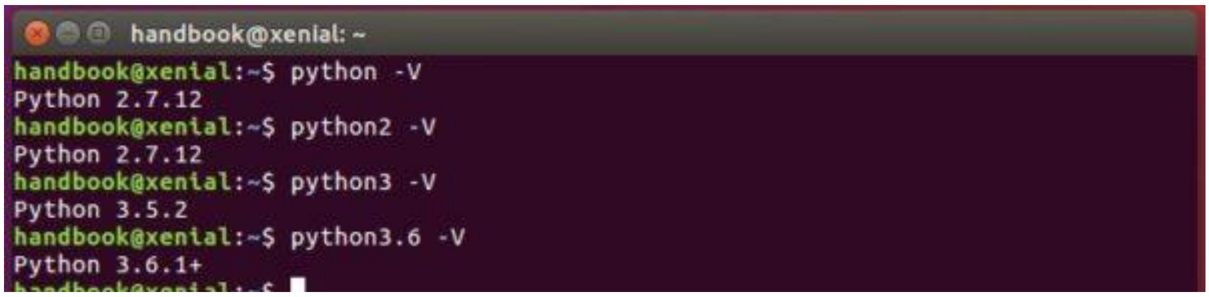
Editing partitions has the potential to cause LOSS of DATA.
You are advised to backup your data before proceeding.

Figure 62: Save and remove the storage drive

5.2 Installing Software (Python3 and ROS)

5.2.1 Installing Python

At this point, the Odroid XU4Q can be booted with a fresh Ubuntu MATE system, which needs Python3 and its libraries to run the codes developed for this project. Python is a high-level, interactive, interpreted and object-oriented scripting language [41]. C++ is a desirable programming language for self-driving cars because of its fast response time'when the machine code is implemented in the car [42]. In contrast, Python is considered a preferable testing language to test prototypes before implementing them. For that reason, and from a learning perspective, this project used Python as testing platform to develop and test the prototype as fast as possible. Both languages are essential to this project and to any autonomous vehicle engineer. To install Python in Ubuntu, the system must first be updated (sudo apt-get update). Python can then be installed by entering the command 'sudo apt-get install Python3.6' in the terminal. Finally, to check that Python was successfully installed, run ‘Python version check’ (Figure 63).



```
handbook@xenial: ~
handbook@xenial:~$ python -V
Python 2.7.12
handbook@xenial:~$ python2 -V
Python 2.7.12
handbook@xenial:~$ python3 -V
Python 3.5.2
handbook@xenial:~$ python3.6 -V
Python 3.6.1+
```

Figure 41: Checking each Python version in Ubuntu terminal

After installing Python, it is easy to install its libraries by using the pip command followed by the library name. Table 7 shows all of the libraries used for this project.

Table 7: prerequisite python libraries

Library	**Description**
Numpy	For large, multi-dimensional arrays and matrices, etc.
Matplotlib	A plotting library for the Python programming language and its numerical mathematics extension
Keras	An open-source neural network library
Sklearn	Machine learning library featuring various classification, regression and clustering algorithms
Imgaug	Library for image augmentation in machine learning experiments
OpenCV	Library with > 2500 algorithms
Pandas	For data manipulation and analysis

5.2.2 Installing ROS

ROS is a powerful, open-source meta-OS for writing robot software [43]. It makes a hard task like building a robust robot system easy. ROS includes hardware abstraction layers, similar to operating systems’However, unlike more common operating systems, it can include numerous combinations of hardware implementation [43]. ROS preferences C++ and Python language as its high-level languages. There are various ROS distributions; this project uses ROS Kinetic. For a full installation of ROS kinetic, the following instruction should be entered in the terminal.

```
sudo apt-get install ros-kinetic-desktop-full
```

With this instruction, all required libraries, including Python2, will be installed. ROS can be tested by running the IMU sensor and reading its data. But first, the IMU ‘myahrs_driver’ package must be installed, so that the ROS is able to establish a connection with the IMU.

```
sudo apt-get install ros-indigo-kinetic-driver
```

It is not required to create a C++ or Python application to test the IMU. That is because the myahrs_driver package comes with a test application. The test application can be launched using the following ROS launch command [44].

```
rosrun myahrs_driver myahrs_driver _port:=/dev/ttyACM0
```

This test runs the Rviz application and provides a live, visual display of the IMU orientation in the x y z axes.

The raw data of the IMU is what matters to this project. It can be acquired by subscribing to the topic of the ‘myahrs’ node. The data collected by the IMU were intended to be used for the MPC, to correct the estimated state. However, this project did not include the MPC, so the IMU data will only be collected in future iterations.

5.3 Arduino Servo DC Motor Test

There are two goals to using the Arduino. The first is to enable the Arduino Nano to control the servo and DC motor signals. This is the autonomous mode, where the Odroid sends commands to the Arduino (Figure 38). Before the Arduino was added, the motors were controlled directly by a transmitter controller, which sent the signals to the radio receiver device, which sent them to the motors. This feature still exists, but in a modified state—the modification of which is the second goal of using the Arduino. This goal involves establishing the manual mode, where the radio receiver still receives the signals, which then pass through the Arduino, which directs them to the motors and keeps a record of them in the Odroid. The aim of the manual mode is to record the driver’s behaviour. This allows for applying the behavioural cloning technique to the manual drive.

The DC motor is attached to an ESC. The ESC sends modulated signals (PWM) to control the speed of the DC motor. The ESC was previously connected to channel-2 of the radio receiver.

However, this was changed by connecting the ESC to the PWM Arduino pin-ten and connecting Channel-2 of the radio receiver to Arduino pin-seven (Table 4).

The servo motor PWM signal cable was connected to channel-1 of the radio receiver. However, this connection was modified so that the servo connected to pin-9 of the Arduino and channel-1 connected to pin-8 (Table 4). It is important to assign the Arduino pins properly when writing the code.

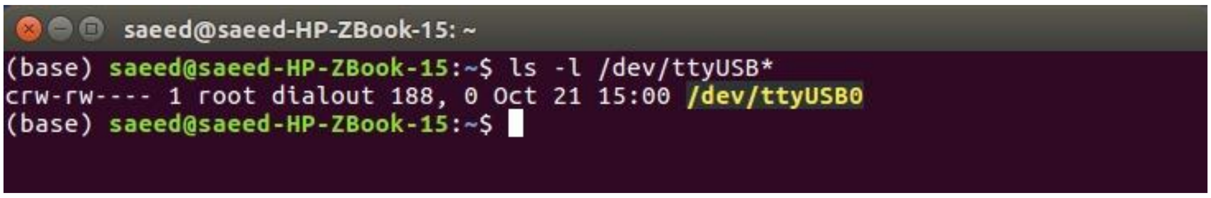
After connecting the motors to the Arduino Nano, the operating characteristics of the motors and the ESC must be found. This was achieved in the current project by using a multi-function instrument from Digilent, AnalogDiscovery [45]. It comes with measurement tools, such as Oscilloscope, which was used to measure the power and PWM of the LaTraX receiver, LaTrax ESC and the servo motor. These measurements can be found in Table 8.

Table 8: Receiver, ESC and servo power rating and control signal specification

Receiver, LaTrax micro, 2.4 GHz (3-channel)	**LaTrax ESC**	**Servo steering signal**

Control both servo and ESC
100 hz PWM, 3 V
	Supply 6 V for the system
	PWM signal supply = 3 V
	stop = 1.502 ms
	Lowest movement = 1.565 ms
	Max forward speed = 1.998 ms
	Max reverse speed = 997 us
		Stop = 1.528 ms
		Right = 1.986 ms
		Left = 1.016 ms
		Supply 6 V
		PWM supply 3 V

5.3.1 Autonomous Mode



```
saeed@saeed-HP-ZBook-15: ~
(base) saeed@saeed-HP-ZBook-15:~$ ls -l /dev/ttyUSB*
crw-rw---- 1 root dialout 188, 0 Oct 21 15:00 /dev/ttyUSB0
(base) saeed@saeed-HP-ZBook-15:~$
```

Figure 42: Terminal USB port number check

By using ROS, the Odroid board is able to connect to and command the Arduino to deliver the required signal for each motor, via a USB port. When the Arduino is connected to Ubuntu MATE, it is assigned the port ‘ttyUSB*’. To check the exact port number, the following command must be entered in the terminal ‘ls -l /dev/ttyUSB*’ (Figure 64).

The autonomous mode was created by using the ROS 'rosserial_Python' package to establish a connection with the USB port of the Arduino.

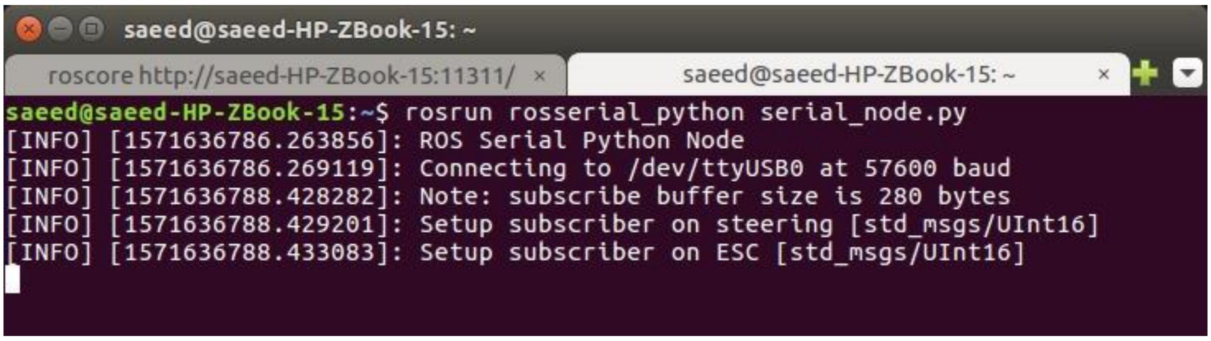
5.3.2 Installing the package

```
sudo apt-get install ros-kinetic-rosserial-Python
```

The code developed for the Arduino Nano is a C++ script, created by Arduino IDE. The IDE does not have the rosserial as a default package. Rosserial must be added to the Arduino library file to be recognised. The following command adds the library to Arduino IDE.

```
sudo apt-get install ros-kinetic-rosserial-arduino
```

After meeting the previous requirements, a C++ script was created (appendix 3) to successfully establish a connection between the Odroid board and Arduino Nano. The Arduino was programmed to subscribe to two ROS topics (steering and ESC). The Odroid board runs the ROS master and sends the PWM values (e.g. '1900 ESC topic') (Table 6). This makes the DC motor run at maximum speed in a forward direction. Running rosserial shows the Arduino's subscribed topics (Figure 65).


saeed@saeed-HP-ZBook-15: ~
roscore http://saeed-HP-ZBook-15:11311/
saeed@saeed-HP-ZBook-15: ~

```
saeed@saeed-HP-ZBook-15:~$ rosrun rosserial_python serial_node.py
[INFO] [1571636786.263856]: ROS Serial Python Node
[INFO] [1571636786.269119]: Connecting to /dev/ttyUSB0 at 57600 baud
[INFO] [1571636788.428282]: Note: subscribe buffer size is 280 bytes
[INFO] [1571636788.429201]: Setup subscriber on steering [std_msgs/UInt16]
[INFO] [1571636788.433083]: Setup subscriber on ESC [std_msgs/UInt16]
```

Figure 43: Steering topic and ESC for DC motors. Arduino waiting for commands from the Odroid board

5.4 Lane Detection Using Camera

This project used machine learning to apply the behavioural cloning technique. For that reason, the camera is considered the main source of data in this project. As mentioned in chapter 4, the chosen camera has a high frame rate of 120 fps, which requires a fast USB connection with the Odroid board. The Odroid board, XU4Q, has two standard-size USB 3.0 SuperSpeed host ports. According to the Odroid datasheet [38], USB 3.0 access speed is ~10x faster than USB 2.0, in the XU4Q (Figure 66), so it is important to connect the camera to USB3.

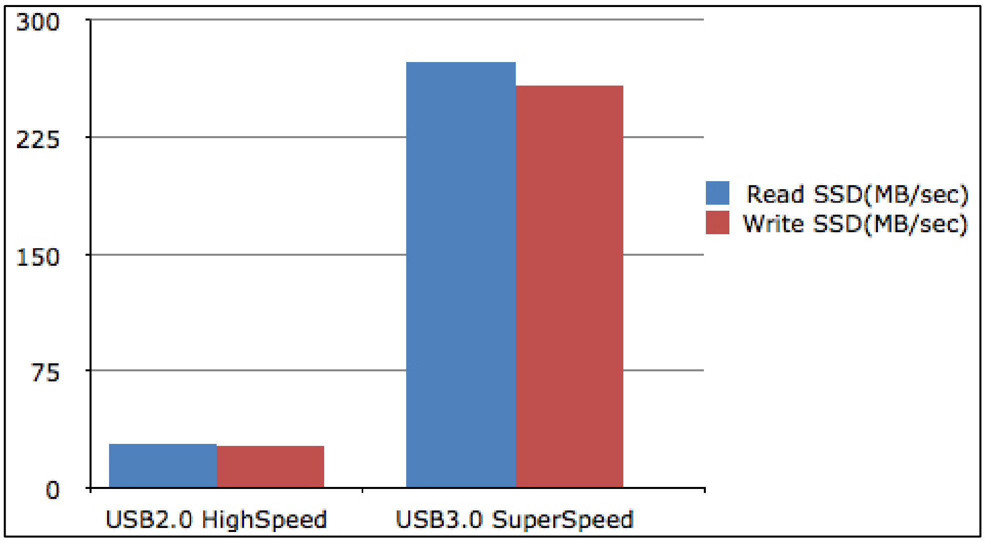


Figure 44: USB3 performance, compared with USB2

ROS uses the ‘usb_cam’ package to run the camera and establish a ROS master connection with its raw image data. However, the camera first needs to be calibrated using the ‘camera_calibration’ package, which uses OpenCV camera calibration [46]. Real lenses usually have some distortion—mostly radial distortion and slight tangential distortion. Calibration is required to remove this distortion. An 8x6 chessboard with 108 mm squares was used to calibrate the camera (Figure 67).

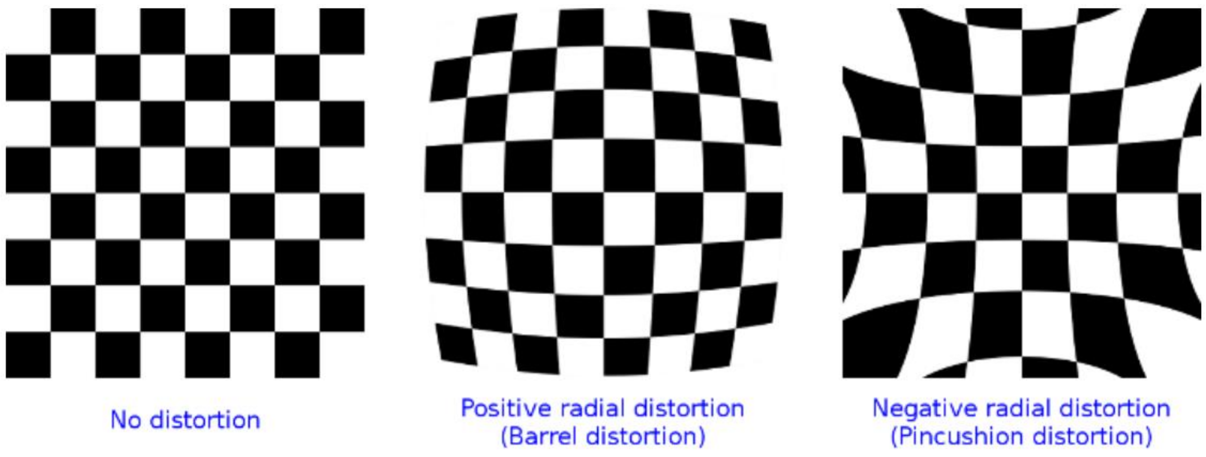


Figure 45: Types of camera distortion

After calibrating the camera, a lane detection Python script was created (Appendix 1 [2. Land Detection]). The script was created to detect road lanes in a recorded video (test.mp4) (Figure 68). The OpenCV Python library was used for image processing. First, the colour code was converted from RBG (red, green, blue) to greyscale. Then, a Gaussian blur filter was applied to smooth the image and remove noise. After that, the image edges were detected using canny (Figure 69).

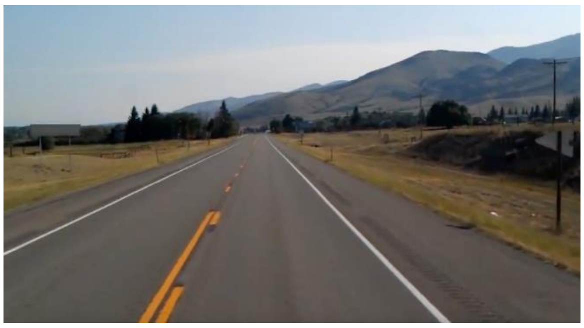


Figure 46: Single image of test video

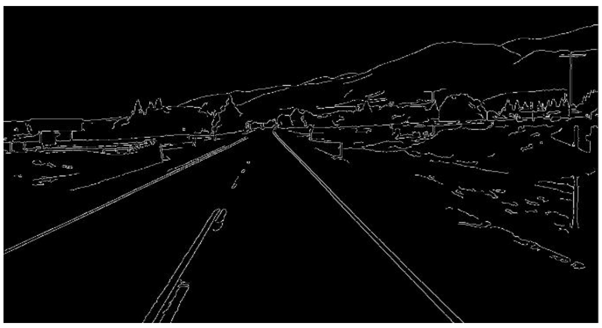


Figure 47: Canny result image

After processing, the image was cropped, leaving only the region of interest (Figure 70). Finally, the Hough transform was used to extract the image features which define the traffic lane (Figure 71).

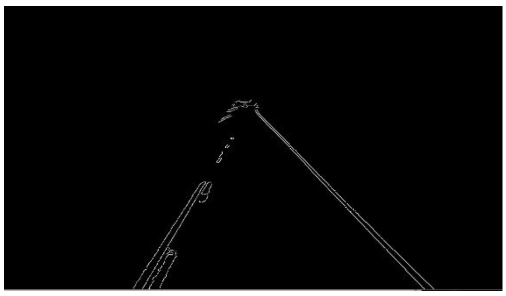


Figure 48: Region of interest after canny

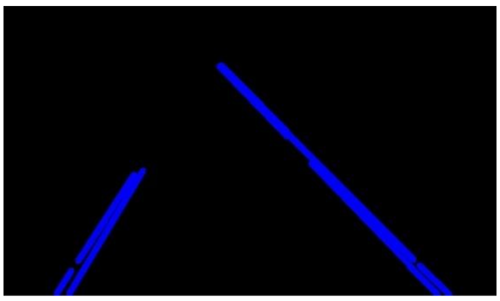


Figure 71: Hough transform to find lane

The resulting Python script was tested on a recorded video and it successfully extracted the lane (Figure 72).

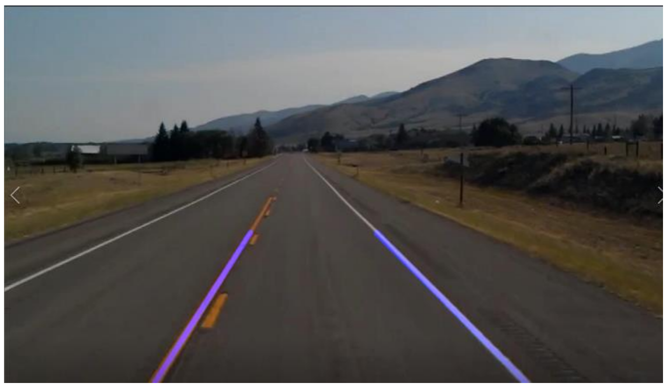


Figure 49: Resulting image

Another lane detection tool, developed for the BARC project at sBerkeley University, was used in this project with minor modification (Appendix 5). It creates a green zone indicating the width of the track and how well the car is adhering to the centre of the road (Figure 73).

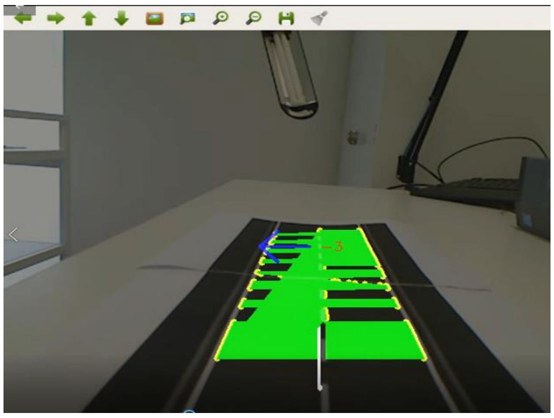


Figure 50: BARC lane detection result tested in lab

Chapter 6: Conclusion

This project was designed to use the Nvidia CNN model to build an autonomous racing car model. The used methodology of creating the CNN model was developed as part of the Udacity self-driving car course, as discussed in chapter 3. Up to this stage, the contribution of this thesis was training the developed CNN model. This training focused on a fast driving behaviour and successfully running the car autonomously in simulation.

The practical build of the car was successfully completed however, the model car could not run successfully. The minicomputer (Odroid) was not computationally powerful enough to quickly return a processed image to guide the car. There is a delay of around three seconds to generate a processed image of the actual view. Moreover, tests have shown that a small dataset (10000 images) is not enough to train the CNN. In addition, collecting a reliable dataset requires time-consuming work.

A proposed solution for implanting the Nvidia model is to source the machine learning tasks to a laptop capable of processing the images and the machine learning. A WIFI would use to establish a connection between the laptop and the scaled racing car’s computer. Unfortunately, there was not enough time to implement this solution in this project. Another solution was to use a more powerful minicomputer like the Jetson AGX Xavier from Nvidia.

If time permitted, the project would take a different approach. It would rely heavily on the MPC, rather than the machine learning approach.

Appendices

Lane detection

```python
import cv2
import numpy as np

def make_points(image, line):
    slope, intercept = line
    y1 = int(image.shape[0])# bottom of the image
    y2 = int(y1*3/5)         # slightly lower than the middle
    x1 = int((y1 - intercept)/slope)
    x2 = int((y2 - intercept)/slope)
    return [[x1, y1, x2, y2]]

def average_slope_intercept(image, lines):
    left_fit    = []
    right_fit   = []
    if lines is None:
        return None
    for line in lines:
        for x1, y1, x2, y2 in line:
            fit = np.polyfit((x1,x2), (y1,y2), 1)
            slope = fit[0]
            intercept = fit[1]
            if slope < 0: # y is reversed in image
                left_fit.append((slope, intercept))
            else:
                right_fit.append((slope, intercept))
    # add more weight to longer lines
    left_fit_average  = np.average(left_fit, axis=0)
    right_fit_average = np.average(right_fit, axis=0)
    left_line  = make_points(image, left_fit_average)
    right_line = make_points(image, right_fit_average)
    averaged_lines = [left_line, right_line]
    return averaged_lines

def canny(img):
    gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    kernel = 5
    blur = cv2.GaussianBlur(gray,(kernel, kernel),0)
    canny = cv2.Canny(gray, 50, 150)
    return canny

def display_lines(img,lines):
    line_image = np.zeros_like(img)
    if lines is not None:
        for line in lines:
            for x1, y1, x2, y2 in line:
                cv2.line(line_image,(x1,y1),(x2,y2),(255,0,0),10)
    return line_image

def region_of_interest(canny):
    height = canny.shape[0]
    width = canny.shape[1]
    mask = np.zeros_like(canny)

    triangle = np.array([[
    (200, height),
    (550, 250),
    (1100, height),]], np.int32)

    cv2.fillPoly(mask, triangle, 255)
    masked_image = cv2.bitwise_and(canny, mask)
    return masked_image



cap = cv2.VideoCapture("test2.mp4") # a test videos is used to resemble a camera
 function to test the code
while(cap.isOpened()):
    _, frame = cap.read()
    canny_image = canny(frame)
    cropped_canny = region_of_interest(canny_image)
    lines = cv2.HoughLinesP(cropped_canny, 2, np.pi/180, 100, np.array([]), minL
ineLength=40,maxLineGap=5)
    averaged_lines = average_slope_intercept(frame, lines)
    line_image = display_lines(frame, averaged_lines)
    combo_image = cv2.addWeighted(frame, 0.8, line_image, 1, 1)
    cv2.imshow("result", combo_image)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

Nvidia end-to-end model (Python code)

```
# -*- coding: utf-8 -*-
"""Copy of Behavioral Cloning -
this  code was developed by Saeed as part of the learning process
in the course of intro-to self driving car at udemy
and it is used as part of the thesis project

Original file is located at
    https://colab.research.google.com/drive/1gnAtHSwYJu_LxSNKcqwecQFAXjZI
QyyB

**Getting the relevant data from the github.**
"""


!git clone https://github.com/saeedn1/ARCAR

"""**Installing the imgaug library.**"""

!pip3 install imgaug

"""**Importing all the relevant libraries.**"""

import os
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import keras
from keras.models import Sequential
from keras.optimizers import Adam
from keras.layers import Convolution2D, Dropout, Flatten, Dense
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
from imgaug import augmenters as iaa
import cv2
import pandas as pd
import ntpath
import random

"""**Cleaning up the data to make it more easy to work with.**"""

datadir = 'Behavioral-Cloning/Data'
columns = ['center', 'left', 'right', 'steering', 'throttle', 'reverse',
'speed']
data = pd.read_csv(os.path.join(datadir, 'driving_log.csv'), names = colu
mns)
pd.set_option('display.max_colwidth', -1)
data.head()
```

```
def path_leaf(path):
  head, tail = ntpath.split(path)
  return tail
data['center'] = data['center'].apply(path_leaf)
data['left'] = data['left'].apply(path_leaf)
data['right'] = data['right'].apply(path_leaf)
data.head()

"""**Trimming the data to reduce bias towards an angle of zero degrees.**
"""


num_bins = 25
samples_per_bin = 400
hist, bins = np.histogram(data['steering'], num_bins)
center = (bins[:-1]+ bins[1:]) * 0.5
plt.bar(center, hist, width=0.05)
plt.plot((np.min(data['steering']), np.max(data['steering'])), (samples_p
er_bin, samples_per_bin))

print('total data:', len(data))
remove_list = []
for j in range(num_bins):
  list_ = []
  for i in range(len(data['steering'])):
    if data['steering'][i] >= bins[j] and data['steering'][i] <= bins[j+1
]:
      list_.append(i)
  list_ = shuffle(list_)
  list_ = list_[samples_per_bin:]
  remove_list.extend(list_)

print('removed:', len(remove_list))
data.drop(data.index[remove_list], inplace=True)
print('remaining:', len(data))

hist, _ = np.histogram(data['steering'], (num_bins))
plt.bar(center, hist, width=0.05)
plt.plot((np.min(data['steering']), np.max(data['steering'])), (samples_p
er_bin, samples_per_bin))

print(data.iloc[1])
def load_img_steering(datadir, df):
  image_path = []
  steering = []
  for i in range(len(data)):
    indexed_data = data.iloc[i]
    center, left, right = indexed_data[0], indexed_data[1], indexed_data[
2]
    image_path.append(os.path.join(datadir, center.strip()))
    steering.append(float(indexed_data[3]))
  image_paths = np.asarray(image_path)
  steerings = np.asarray(steering)
  return image_paths, steerings

image_paths, steerings = load_img_steering(datadir + '/IMG', data)

"""**Splitting the data into training data and validation data for our ne
ural network.**"""

X_train, X_valid, y_train, y_valid = train_test_split(image_paths, steeri
ngs, test_size=0.2, random_state=6)
print('Training Samples: {}\nValid Samples: {}'.format(len(X_train), len(
X_valid)))

fig, axes = plt.subplots(1, 2, figsize=(12, 4))
axes[0].hist(y_train, bins=num_bins, width=0.05, color='blue')
axes[0].set_title('Training set')
axes[1].hist(y_valid, bins=num_bins, width=0.05, color='red')
axes[1].set_title('Validation set')

"""**Defining a function to zoom into an image for data augmentation.**""
"


def zoom(image):
  zoom = iaa.Affine(scale=(1, 1.3))
  image = zoom.augment_image(image)
  return image

image = image_paths[random.randint(0, 1000)]
original_image = mpimg.imread(image)
zoomed_image = zoom(original_image)

fig, axs = plt.subplots(1, 2, figsize=(15, 10))
fig.tight_layout()

axs[0].imshow(original_image)
axs[0].set_title('Original Image')

axs[1].imshow(zoomed_image)
axs[1].set_title('Zoomed Image')

"""**Defining a function to pan around an image for data augmentation.**"
""
```

```
def pan(image):
  pan = iaa.Affine(translate_percent= {"x" : (-0.1, 0.1), "y": (-
0.1, 0.1)})
  image = pan.augment_image(image)
  return image

image = image_paths[random.randint(0, 1000)]
original_image = mpimg.imread(image)
panned_image = pan(original_image)

fig, axs = plt.subplots(1, 2, figsize=(15, 10))
fig.tight_layout()

axs[0].imshow(original_image)
axs[0].set_title('Original Image')

axs[1].imshow(panned_image)
axs[1].set_title('Panned Image')

"""**Defining a function to adjust the brightness of an image for data au
gmentation.**"""

def img_random_brightness(image):
    brightness = iaa.Multiply((0.2, 1.2))
    image = brightness.augment_image(image)
    return image

image = image_paths[random.randint(0, 1000)]
original_image = mpimg.imread(image)
brightness_altered_image = img_random_brightness(original_image)

fig, axs = plt.subplots(1, 2, figsize=(15, 10))
fig.tight_layout()

axs[0].imshow(original_image)
axs[0].set_title('Original Image')

axs[1].imshow(brightness_altered_image)
axs[1].set_title('Brightness altered image ')

"""**Defining a function to flip an image and the respective steering ang
le for data augmentation.**"""

def img_random_flip(image, steering_angle):
    image = cv2.flip(image,1)
    steering_angle = -steering_angle
    return image, steering_angle
```

```
random_index = random.randint(0, 1000)
image = image_paths[random_index]
steering_angle = steerings[random_index]


original_image = mpimg.imread(image)
flipped_image, flipped_steering_angle = img_random_flip(original_image, s
teering_angle)

fig, axs = plt.subplots(1, 2, figsize=(15, 10))
fig.tight_layout()

axs[0].imshow(original_image)
axs[0].set_title('Original Image - ' + 'Steering Angle:' + str(steering_a
ngle))

axs[1].imshow(flipped_image)
axs[1].set_title('Flipped Image - ' + 'Steering Angle:' + str(flipped_ste
ering_angle))

"""**Defining a function to apply a 50% chance of applying a specific aug
mentation to an image.**"""

def random_augment(image, steering_angle):
    image = mpimg.imread(image)
    if np.random.rand() < 0.5:
      image = pan(image)
    if np.random.rand() < 0.5:
      image = zoom(image)
    if np.random.rand() < 0.5:
      image = img_random_brightness(image)
    if np.random.rand() < 0.5:
      image, steering_angle = img_random_flip(image, steering_angle)

    return image, steering_angle

"""**Comparing the original image side by side its augmented counter part
.**"""

ncol = 2
nrow = 10

fig, axs = plt.subplots(nrow, ncol, figsize=(15, 50))
fig.tight_layout()

for i in range(10):
  randnum = random.randint(0, len(image_paths) - 1)
  random_image = image_paths[randnum]
  random_steering = steerings[randnum]
  original_image = mpimg.imread(random_image)
  augmented_image, steering = random_augment(random_image, random_steerin
g)

  axs[i][0].imshow(original_image)
  axs[i][0].set_title("Original Image")

  axs[i][1].imshow(augmented_image)
  axs[i][1].set_title("Augmented Image")

"""**Applying preprocessing techniques to our images.**"""

def img_preprocess(img):
    img = img[60:135,:,:]
    img = cv2.cvtColor(img, cv2.COLOR_RGB2YUV)
    img = cv2.GaussianBlur(img,  (3, 3), 0)
    img = cv2.resize(img, (200, 66))
    img = img/255
    return img

image = image_paths[100]
original_image = mpimg.imread(image)
preprocessed_image = img_preprocess(original_image)

fig, axs = plt.subplots(1, 2, figsize=(15, 10))
fig.tight_layout()
axs[0].imshow(original_image)
axs[0].set_title('Original Image')
axs[1].imshow(preprocessed_image)
axs[1].set_title('Preprocessed Image')

"""**Defining an image generator that will produce our augmented data set
.**"""

def batch_generator(image_paths, steering_ang, batch_size, istraining):

  while True:
    batch_img = []
    batch_steering = []

    for i in range(batch_size):
      random_index = random.randint(0, len(image_paths) - 1)

      if istraining:
        im, steering = random_augment(image_paths[random_index], steering
_ang[random_index])
```

```
        else:
          im = mpimg.imread(image_paths[random_index])
          steering = steering_ang[random_index]

        im = img_preprocess(im)
        batch_img.append(im)
        batch_steering.append(steering)
      yield (np.asarray(batch_img), np.asarray(batch_steering))

x_train_gen, y_train_gen = next(batch_generator(X_train, y_train, 1, 1))
x_valid_gen, y_valid_gen = next(batch_generator(X_valid, y_valid, 1, 0))

fig, axs = plt.subplots(1, 2, figsize=(15, 10))
fig.tight_layout()

axs[0].imshow(x_train_gen[0])
axs[0].set_title('Training Image')

axs[1].imshow(x_valid_gen[0])
axs[1].set_title('Validation Image')

"""**Defining our convolutional neural network.**"""

def model():
  model = Sequential()
  model.add(Convolution2D(24, 5, 5, subsample=(2, 2), input_shape=(66, 20
0, 3), activation='elu'))
  model.add(Convolution2D(36, 5, 5, subsample=(2, 2), activation='elu'))
  model.add(Convolution2D(48, 5, 5, subsample=(2, 2), activation='elu'))
  model.add(Convolution2D(64, 3, 3, activation='elu'))

  model.add(Convolution2D(64, 3, 3, activation='elu'))

  model.add(Flatten())

  model.add(Dense(100, activation = 'elu'))

  model.add(Dense(50, activation = 'elu'))

  model.add(Dense(10, activation = 'elu'))

  model.add(Dense(1))

  optimizer = Adam(lr=1e-4)
  model.compile(loss='mse', optimizer=optimizer)
  return model

model = model()
```

```
print(model.summary())

"""**Running the training process using an augmented dataset.**"""

history = model.fit_generator(batch_generator(X_train, y_train, 100, 1),
                                  steps_per_epoch=340,
                                  epochs=10,
                                  validation_data=batch_generator(X_valid
, y_valid, 100, 0),
                                  validation_steps=200,
                                  verbose=1,
                                  shuffle = 1)

"""**Plotting the loss observed during the training process.**"""

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(['training', 'validation'])
plt.title('Loss')
plt.xlabel('Epoch')

"""**Saving our trained model for use with the self driving car simulator
.**"""

model.save('model.h5')

from google.colab import file
s
files.download('model.h5')
```
1.

Arduino Code – Autonomous mode

Servo & DC motors control for autunmous mode using rosserial

```
/*
 * rosserial Servo Control modified version of  the ROS.org Tutorial
 * create topics to of the DC motor (ESC) and the servo (steering)
 */

#if defined(ARDUINO) && ARDUINO >= 100
  #include "Arduino.h"
#else
  #include <WProgram.h>
#endif

#include <Servo.h>
#include <ros.h>
#include <std_msgs/UInt16.h>

ros::NodeHandle  nh;

Servo steering;
Servo ESC;

void servo_steer( const std_msgs::UInt16& cmd_msg){
  steering.writeMicroseconds(cmd_msg.data); //set servo angle, should be from 0-180
  digitalWrite(13, HIGH-digitalRead(13));  //toggle led
}

void servo_ESC( const std_msgs::UInt16& cmd_msg){
  ESC.writeMicroseconds(cmd_msg.data); //set servo angle, should be from 0-180
  digitalWrite(13, HIGH-digitalRead(13));  //toggle led
}

ros::Subscriber<std_msgs::UInt16> sub_steering("steering", servo_steer);
ros::Subscriber<std_msgs::UInt16> sub_ESC("ESC", servo_ESC);

void setup(){
  pinMode(13, OUTPUT);

  nh.initNode();
  nh.subscribe(sub_steering);
  nh.subscribe(sub_ESC);
  steering.attach(9,1000,2000); //attach it to pin 9
  ESC.attach(10,1000,2000);
}
void loop(){
  nh.spinOnce();
  delay(1);
}
```

BRAC– Autonomous mode

Reduced version of the Berkeley **BARC** Arduino servo & DC motor control.

```
/* ---------------------------------------------------------------------------
# Licensing Information: You are free to use or extend these projects for
# education or reserach purposes provided that (1) you retain this notice
# and (2) you provide clear attribution to UC Berkeley, including a link
# to http://barc-project.com
#
# Attibution Information: The barc project ROS code-base was developed
# at UC Berkeley in the Model Predictive Control (MPC) lab by Jon Gonzales
# (jon.gonzales@berkeley.edu)  Development of the web-server app Dator was
# based on an open source project by Bruce Wootton, with contributions from
# Kiet Lam (kiet.lam@berkeley.edu). The RC Input code was based on sample code
# from http://rcarduino.blogspot.com/2012/04/how-to-read-multiple-rc-channels-draft.html
# --------------------------------------------------------------------------- */



// include libraries
#include <ros.h>
#include <barc/Encoder.h>
#include <barc/ECU.h>
#include <Servo.h>
#include <EnableInterrupt.h>

/**************************************************************************************
CAR CLASS DEFINITION (would like to refactor into car.cpp and car.h but can't figure out arduino build
process so far)
**************************************************************************************/
class Car {
  public:
    void initEncoders();
    void initRCInput();
    void initActuators();
    void armActuators();
    void writeToActuators(const barc::ECU& ecu);
    // Used for copying variables shared with interrupts to avoid read/write
    // conflicts later
    void readAndCopyInputs();
    // Getters
    uint16_t getRCThrottle();
    uint16_t getRCSteering();

    float getVelEstFL();
    float getVelEstFR();
    float getVelEstBL();
    float getVelEstBR();

    // Interrupt service routines
    void incFR();
    void incFL();
    void incBR();
    void incBL();
    void calcThrottle();
    void calcSteering();
    void calcVelocityEstimate();
    void killMotor();
  private:
    // Pin assignments

    const int THROTTLE_PIN = 7;
    const int STEERING_PIN = 8;
    const int MOTOR_PIN = 10;
    const int SERVO_PIN= 9;




    // Motor limits
    // TODO  fix limits?
    const int MOTOR_MAX = 2000;
    const int MOTOR_MIN = 800;
    const int MOTOR_NEUTRAL = 1500;
    const int THETA_CENTER = 1500;
    const int THETA_MAX = 1900;
    const int THETA_MIN = 1100;

    // Interfaces to motor and steering actuators
    Servo motor;
    Servo steering;
```

```
    // RC joystick control variables
    uint32_t throttleStart;
    uint32_t steeringStart;
    volatile uint16_t throttleInShared;
    volatile uint16_t steeringInShared;
    uint16_t throttleIn = 1500;
    uint16_t steeringIn = 1500;

    // motor / servo neutral state (milliseconds)
    float throttle_neutral_ms = 1500.0;
    float servo_neutral_ms = 1500.0;



    // Utility functions
    uint16_t microseconds2PWM(uint16_t microseconds);
    float saturateMotor(float x);
    float saturateServo(float x);
};

// Boolean keeping track of whether the Arduino has received a signal from the ECU recently
int received_ecu_signal = 0;
float pi                = 3.141593;
float R                 = 0.051;        // radius of the wheel

// Initialize an instance of the Car class as car
Car car;

// Callback Functions
// These are really sad solutions to the fact that using class member functions
// as callbacks is complicated in C++ and I haven't figured it out. If you can
// figure it out, please atone for my sins.
void ecuCallback(const barc::ECU& ecu) {
  car.writeToActuators(ecu);
  received_ecu_signal = 1;
}
void incFLCallback() {
  car.incFL();
```

```
}
void incFRCallback() {
  car.incFR();
}
void incBLCallback() {
  car.incBL();
}
void incBRCallback() {
  car.incBR();
}
void calcSteeringCallback() {
  car.calcSteering();
}
void calcThrottleCallback() {
  car.calcThrottle();
}

// Variables for time step
volatile unsigned long dt;
volatile unsigned long t0;
volatile unsigned long ecu_t0;

// Global message variables
// Encoder, RC Inputs, Electronic Control Unit, Ultrasound
barc::ECU ecu;
barc::ECU rc_inputs;
barc::Encoder encoder;
barc::Encoder vel_est;

ros::NodeHandle nh;

ros::Publisher pub_encoder("encoder", &encoder);
ros::Publisher pub_vel_est("vel_est", &vel_est);
ros::Publisher pub_rc_inputs("rc_inputs", &rc_inputs);
ros::Subscriber<barc::ECU> sub_ecu("ecu_pwm", ecuCallback);

/**************************************************************************
ARDUINO INITIALIZATION
**************************************************************************/
void setup()
```

```
{
  // Set up encoders, rc input, and actuators
  car.initEncoders();
  car.initRCInput();
  car.initActuators();

  // Start ROS node
  nh.initNode();

  // Publish and subscribe to topics
  nh.advertise(pub_encoder);
  nh.advertise(pub_rc_inputs);
  nh.advertise(pub_vel_est);
  nh.subscribe(sub_ecu);

  // Arming ESC, 1 sec delay for arming and ROS
  car.armActuators();
  t0 = millis();
  ecu_t0 = millis();

}


/**************************************************************************
ARDUINO MAIN lOOP
**************************************************************************/
void loop() {
  // compute time elapsed (in ms)
  dt = millis() - t0;

  // kill the motor if there is no ECU signal within the last 1s
  if( (millis() - ecu_t0) >= 1000){
    if(!received_ecu_signal){
        car.killMotor();
    } else{
        received_ecu_signal = 0;
    }
    ecu_t0 = millis();
  }
  if (dt > 50) {
    car.readAndCopyInputs();

    // publish velocity estimate
    car.calcVelocityEstimate();
    vel_est.FL  = car.getVelEstFL();
    vel_est.FR  = car.getVelEstFR();
    vel_est.BL  = car.getVelEstBL();
    vel_est.BR  = car.getVelEstBR();
    pub_vel_est.publish(&vel_est);

    // publish encoder ticks
    encoder.FL = car.getEncoderFL();
    encoder.FR = car.getEncoderFR();
    encoder.BL = car.getEncoderBL();
    encoder.BR = car.getEncoderBR();
    pub_encoder.publish(&encoder);

    rc_inputs.motor = car.getRCThrottle();
    rc_inputs.servo = car.getRCSteering();
    pub_rc_inputs.publish(&rc_inputs);

    t0 = millis();
  }

  nh.spinOnce();
}

/**************************************************************************
CAR CLASS IMPLEMENTATION
**************************************************************************/
float Car::saturateMotor(float x) {
  if (x > MOTOR_MAX) { x = MOTOR_MAX; }
  if (x < MOTOR_MIN) { x = MOTOR_MIN; }
  return x;
}

float Car::saturateServo(float x) {
  if (x > THETA_MAX) { x = THETA_MAX; }
  if (x < THETA_MIN) { x = THETA_MIN; }
  return x;
}



void Car::initRCInput() {
  pinMode(THROTTLE_PIN, INPUT_PULLUP);
  pinMode(STEERING_PIN, INPUT_PULLUP);
  enableInterrupt(THROTTLE_PIN, calcThrottleCallback, CHANGE);
  enableInterrupt(STEERING_PIN, calcSteeringCallback, CHANGE);
}

void Car::initActuators() {
  motor.attach(MOTOR_PIN);
  steering.attach(SERVO_PIN);
}

void Car::armActuators() {
  motor.writeMicroseconds( (uint16_t) throttle_neutral_ms );
  steering.writeMicroseconds( (uint16_t) servo_neutral_ms );
  delay(1000);
}

void Car::writeToActuators(const barc::ECU& ecu) {
  motor.writeMicroseconds( (uint16_t) saturateMotor( ecu.motor ) );
  steering.writeMicroseconds( (uint16_t) saturateServo( ecu.servo ) );
}

uint16_t Car::microseconds2PWM(uint16_t microseconds) {
  // Scales RC pulses from 1000 - 2000 microseconds to 0 - 180 PWM angles
  // Mapping from microseconds to pwm angle
  // 0 deg -> 1000 us , 90 deg -> 1500 us , 180 deg -> 2000 us
  // ref: camelsoftware.com/2015/12/25/reading-pwm-signals-from-an-rc-receiver-with-arduino

  // saturate signal
  if(microseconds > 2000 ){ microseconds = 2000; }
  if(microseconds < 1000 ){ microseconds = 1000; }

  // map signal from microseconds to pwm angle
  uint16_t pwm = (microseconds - 1000.0)/1000.0*180;
  return static_cast<uint8_t>(pwm);
}

void Car::calcThrottle() {
  if(digitalRead(THROTTLE_PIN) == HIGH) {
    // rising edge of the signal pulse, start timing
    throttleStart = micros();
  } else {
    // falling edge, calculate duration of throttle pulse
    throttleInShared = (uint16_t)(micros() - throttleStart);
    // set the throttle flag to indicate that a new signal has been received
    updateFlagsShared |= THROTTLE_FLAG;
  }
}

void Car::calcSteering() {
  if(digitalRead(STEERING_PIN) == HIGH) {
    steeringStart = micros();
  } else {
    steeringInShared = (uint16_t)(micros() - steeringStart);
    updateFlagsShared |= STEERING_FLAG;
  }
}

void Car::killMotor(){
  motor.writeMicroseconds( (uint16_t) throttle_neutral_ms );
  steering.writeMicroseconds( (uint16_t) servo_neutral_ms );
}

void Car::incFL() {
  FL_count_shared++;
  FL_old_time = FL_new_time;
  FL_new_time = micros();
  updateFlagsShared |= FL_FLAG;
}

void Car::incFR() {
  FR_count_shared++;
  FR_old_time = FR_new_time;
  FR_new_time = micros();
  updateFlagsShared |= FR_FLAG;
}

void Car::incBL() {
  BL_count_shared++;
  BL_old_time = BL_new_time;
  BL_new_time = micros();
  updateFlagsShared |= BL_FLAG;
}

void Car::incBR() {
  BR_count_shared++;
  BR_old_time = BR_new_time;
  BR_new_time = micros();
  updateFlagsShared |= BR_FLAG;
}

void Car::readAndCopyInputs() {
  // check shared update flags to see if any channels have a new signal
  if (updateFlagsShared) {
    // Turn off interrupts, make local copies of variables set by interrupts,
    // then turn interrupts back on. Without doing this, an interrupt could
    // update a shared multibyte variable while the loop is in the middle of
    // reading it
    noInterrupts();
    // make local copies
    updateFlags = updateFlagsShared;
    if(updateFlags & THROTTLE_FLAG) {
      throttleIn = throttleInShared;
    }
    if(updateFlags & STEERING_FLAG) {
      steeringIn = steeringInShared;
    }
    if(updateFlags & FL_FLAG) {
      FL_count = FL_count_shared;
      FL_DeltaTime = FL_new_time - FL_old_time;
    }
    if(updateFlags & FR_FLAG) {
      FR_count = FR_count_shared;
      FR_DeltaTime = FR_new_time - FR_old_time;
    }
    if(updateFlags & BL_FLAG) {
      BL_count = BL_count_shared;
      BL_DeltaTime = BL_new_time - BL_old_time;
    }
    if(updateFlags & BR_FLAG) {
      BR_count = BR_count_shared;
      BR_DeltaTime = BR_new_time - BR_old_time;
    }
    // clear shared update flags and turn interrupts back on
    updateFlagsShared = 0;
    interrupts();
  }
}

uint16_t Car::getRCThrottle() {
  return throttleIn;
}
uint16_t Car::getRCSteering() {
  return steeringIn;
}



float Car::getVelEstFL() {
  return vel_FL;
}
float Car::getVelEstFR() {
  return vel_FR;
}
float Car::getVelEstBL() {
  return vel_BL;
}
float Car::getVelEstBR() {
  return vel_BR;
}
```

Bibliography

[1] B. Goldfain, P. Drews, C. You, M. Barulic, O. Velev, P. Tsiotras and J. M. Rehg, "AutoRally: An Open Platform for Aggressive Autonomous Driving," *IEEE Control Systems Magazine*, vol. 39, no. 1, pp. 26–55, Feb. 2019.

[2] Roborace.com. (2019). *Global championship of driverless cars* [Online]. Available at: https://roborace.com

[3] NHTSA. (2019). *Automated Vehicles for Safety* [Online]. Available at: https://www.nhtsa.gov/technology-innovation/automated-vehicles-

[4] D. Agency and A. Us(2019), "The Grand Challenge for Autonomous Vehicles", *Darpa.mil*. [Online]. Available: https://www.darpa.mil/about-us/timeline/-grand-challenge-for-autonomous-vehicles.

[5] S. Thrun et al., "Stanley: The robot that won the DARPA Grand Challenge." *Journal of Field Robotics*, vol. 23, no. 9, pp., Sept. 2006.

[6] Self Racing Cars. (2019). *About - Self Racing Cars* [Online]. Available at: http://selfracingcars.com/about

[7] Ddl.stanford.edu. (2019). *Vehicle Dynamics and Control At The Limits of Handling | Dynamic Design Lab* [Online]. Available at: https://ddl.stanford.edu/content/vehicle-dynamics-and-control-limits-handling

[8] Carsim.com. (2019). *CarSim Overview*. [Online]. Available at: https://www.carsim.com/products/carsim/index.php

[9] GitHub. (2019). *udacity/self-driving-car-sim* [Online]. Available at: https://github.com/udacity/self-driving-car-sim

[10] Mpc.berkeley.edu. (2019). *MPC Lab @ UC-Berkeley* [Online]. Available at: http://www.mpc.berkeley.edu/

[11] G. Williams, B. Goldfain, P. Drews, K. Saigol, J. M. Rehg and E. A. Theodorou, "Robust Sampling Based Model Predictive Control with Sparse Objective Information." *Robotics: Science and Systems*, 2018. doi 10.15607/rss.2018.xiv.042

[12] Stanford.edu. (2013). *CS221* [Online]. Available at: https://stanford.edu/~cpiech/cs221/apps/driverlessCar.html

[13] Mit-racecar.github.io. (2019). *RACECAR* [Online] Available at: https://mit-racecar.github.io/

[14] F1tenth.org. (2019). *F1tenth* [Online] Available at: http://f1tenth.org/

[15] GitHub. (2019). *Autorope* [Online]. Available at: https://github.com/autorope

[16] Hamster-Robot. (2018). *Hamster V7, Smart ROS Autonomous Ground Vehicles for Industry and Aacdemic R&D* [Online]. Available at: https://www.hamster-robot.com/

[17] BARC. (2017). *Berkeley Autonomous Race Car* [Online]. Available at: http://www.barc-project.com/

[18] MathWorks (2019, May 25). *Model Predictive Control Toolbox* [Online]. Available at: https://www.mathworks.com/products/mpc.html?s_eid=PSM_15028

[19] U. Rosolia, A. Carvalho, F. Borrelli and Cornell University (2017, Nov. 9). *Autonomous Racing using Learning Model Predictive Control* [Online]. Available: https://arxiv.org/abs/1610.06534

[20] M. Bojarski, B.Firner, B. Flepp, L. Jackel, U. Muller, K. Zieba, D. Testa, M. Bojarski, L. Jackel, B. Firner, U. Muller, A., Dixit, N. Dasan, Y. Cheng, L. An, J. Park, D. Testa, J. Gu, X. Yang, S. Mello, J. Kautz, Nvidia Developer Blog (2016, Aug. 17). *End-to-End Deep Learning for Self-Driving Cars* [Blog]. Available at: https://devblogs.nvidia.com/deep-learning-self-driving-cars/

[21] Raschka, Sebastian, *Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow*, 2nd ed. Birmingham, United Kingdom: Packt Publishing, Kindle Edition, 2017, p. 378.

[22] S. Saha and Towards Data Science (2018, Dec. 16). *A Comprehensive Guide to Convolutional Neural Networks—the ELI5 way* [Online]. Available: barhttps://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[23] K. Kritayakirana, "Autonomous Vehicle Control as the Limits of Handling," *International Journal of Vehicle Autonomous Systems*, vol. 10, no. 4, pp. 271-296, 2012.

[24] Y. Gao, A. Gray, J. V. Frasch, T. Lin, E. Tseng, J. K. Hedrick, and F. Borrelli, "Spatial predictive control for agile semi-autonomous ground vehicles," in 11th International Symposium on Advanced Vehicle Control, 2012.

[25] J. V. Frasch, A. Gray, M. Zanon, H. J. Ferreau, S. Sager, F. Borrelli, and M. Diehl, "An auto-generated nonlinear mpc algorithm for real- time obstacle avoidance of ground vehicles," in Control Conference (ECC), 2013 European. IEEE, 2013, pp. 4136–4141.

[26] R. Verschueren, S. De Bruyne, M. Zanon, J. V. Frasch, and M. Diehl, "Towards time-optimal race car driving using nonlinear mpc in real-time," in 53rd IEEE Conference on Decision and Control. IEEE, 2014, pp. 2505–2510

[27] Chollet, F. (2019). *Home - Keras Documentation*. [online] Keras.io. Available at: https://keras.io/

[28] Colab.research.google.com. (2019). *Google Colaboratory*. [online] Available at: https://colab.research.google.com/notebooks/welcome.ipynb#scrollTo=5fCEDCU_qrC0

[29] Scikit-learn.org. (2019). *sklearn.model_selection.train_test_split — scikit-learn 0.21.3 documentation*. [online] Available at: https://scikit learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html.

[30] Imgaug.readthedocs.io. (2019). *imgaug — imgaug 0.3.0 documentation*. [online] Available at: https://imgaug.readthedocs.io/en/latest/

[31] GitHub. (2019). *pallets/flask*. [online] Available at: https://github.com/pallets/flask [32] GitHub. (2019). *Sarmyt/Behavioral-Cloning*. [online] Available at: https://github.com/Sarmyt/Behavioral-Cloning/blob/master/Code/Drive.py

[33] Keras.io. (2019). *Sequential - Keras Documentation*. [online] Available at: https://keras.io/models/sequential/ [Accessed 3 Oct. 2019].

[34] Traxxas.com. (2019). *Traxxas Ford Fiesta VR46 | Electric RC Rally Car*. [online] Available at: https://traxxas.com/products/models/electric/ford-fiesta-st-vr46.

[35] Web.mit.edu. (2019). *A Guide to Understanding Battery Specifications*. [online] Available at: http://web.mit.edu/evt/summary_battery_specifications.pdf

[36] Anon, 2019. *withrobot/myAHRS_plus*. [online] GitHub. Available at: https://github.com/withrobot/myAHRS_plus/tree/master/tutorial.

[37] Team, U., 2019. What is Ubuntu MATE?. [online] Ubuntu MATE. Available at: https://ubuntu-mate.org/what-is-ubuntu-mate/.

[38] Anon, 2019. [online] Magazine.odroid.com. Available at: <https://magazine.odroid.com/wp-content/uploads/odroid-xu4-user-manual.pdf> [Accessed 20 Oct. 2019].

[39] Anon, 2019. *odroid-xu4:getting_started:os_installation_guide [ODROID Wiki]*. [online] Wiki.odroid.com. Available at: <https://wiki.odroid.com/odroid-xu4/getting_started/os_installation_guide#tab__odroid-xu4>

[40] Gedak, C., 2019. *GParted -- A free application for graphically managing disk device partitions*. [online] Gparted.org. Available at: <https://gparted.org/>

[41] Anon, 2019. *Welcome to Python.org*. [online] Python.org. Available at: <https://www.python.org>

[42] Anon, 2019. *C++ vs. Python for Automotive Software*. [online] Medium. Available at: <https://medium.com/self-driving-cars/c-vs-python-for-automotive-software-40211536a4ad>

[43] Anon, 2019. *ROS.org | About ROS*. [online] Ros.org. Available at: <https://www.ros.org/about-ros/>

[44] Anon, 2019. *myahrs_driver - ROS Wiki*. [online] Wiki.ros.org. Available at: <http://wiki.ros.org/myahrs_driver>

[45] Anon, 2019. *Starting with the Analog Discovery [Reference.Digilentinc]*. [online] Reference.digilentinc.com. Available at: <https://reference.digilentinc.com/waveforms3/analogdiscovery>

[46] Anon, 2019. *Camera Calibration and 3D Reconstruction — OpenCV 2.4.13.7 documentation*. [online] Docs.opencv.org. Available at: <https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html>