ASHESI UNIVERSITY COLLEGE

AUTONOMOUS SELF-DRIVING VEHICLE:

PERCEPTION, SUPERVISED LEARNING AND CONTROL

APPLIED PROJECT

B.Sc. Computer Science

Benedict Quartey

2018

ASHESI UNIVERSITY COLLEGE

Autonomous Self-Driving Vehicle:

Perception, Supervised Learning, Control

APPLIED PROJECT

Applied project submitted to the Department of Computer Science, Ashesi University College in partial fulfilment of the requirements for the award of Bachelor of Science degree in Computer Science

Benedict Quartey

April 2018

DECLARATION

I hereby declare that this applied project is the result of my own original work and that no part of it has been presented for another degree in this university or elsewhere.

Candidate's Signature:

……

Candidate's Name:

……

Date: ……

I hereby declare that preparation and presentation of this applied project were supervised in accordance with the guidelines on supervision of Applied project laid down by Ashesi University College.

Supervisor's Signature:

……

Supervisor's Name:

……

Date: ……

Acknowledgement

This applied project would not have been possible without the help and input of several individuals. Firstly, I am grateful to God for giving me the knowledge and resilience to complete this project. I would also like to thank my supervisor Dr. Ayorkor Korsah, her guidance and constant feedback helped me implement this project successfully.

Secondly, I would like to thank my family, friends and fellow self-driving car enthusiasts from various self-driving and Deep Learning slack channels. They constantly helped me find answers to every question and shared their own ideas on my implementation.

Finally, I would like to acknowledge all the lecturers I have had the honour of studying under throughout my stay at Ashesi. I have learnt so much and the person I am now is as a result of their cumulative guidance.

Abstract

Road accidents are estimated to be the ninth leading cause of death across all age groups globally. 1.25 million people die annually from road accidents and Africa has the highest rate of road fatalities (WHO, 2015). Self-driving technology has the potential of saving over a million lives lost to preventable road accidents world-wide.

Africa accounts for the majority of road fatalities and as such would benefit immensely from this technology. However, financial constraints prevent viable experimentation and research into self-driving technology in Africa. In this applied project I designed and implemented RollE to bridge this gap. RollE is an affordable modular autonomous vehicle development platform. It is capable of road data collection and autonomous driving using a convolutional neural network. This system is aimed at providing students and researchers with an affordable autonomous vehicle to develop self-driving car technology.

Table of Content

List of Figures

Chapter 1: Introduction

1.1 Background and Significance

Road accidents are estimated to be the ninth leading cause of death across all age groups globally. Every year, the total estimated global tally of deaths as a result of road accidents hovers around 1.25 million people (WHO, 2015). These accidents are mostly due to preventable human driver error. Autonomous vehicles provide a prospective solution to this problem. Interest in the potential of autonomous vehicles has grown substantially in the past four years. As of June 2017, the research institution Brookings estimates the total investment in research and development of autonomous vehicles by industry leaders to have grown from under $1 Billion in late 2014 to about $80 billion (Kerry & Karsten, 2017).

As with other human technological achievements, self-driving technology might start a chain of innovations that improves human life in general. However, Africa seems to be lagging behind in this field. Investment data provided by Brookings reveals that the $80 billion research and development transactions and acquisitions stated earlier are situated in already developed economies. As an example, Uber donated $5.5 million to Carnegie Mellon in 2015 to solidify their robotics program and fund three graduate fellowships. This donation was made as part of a strategic partnership between the two entities in developing autonomous driverless-car technology (Kerry & Karsten, 2017). Toyota also invested $1 billion in launching a research institute solely for Artificial Intelligence (Kerry & Karsten, 2017). The report by Brookings also showed a number of transactions involving startup investments, partnerships and acquisitions all in the field of advanced perception systems, AI and autonomous vehicles. While these investments show the positive interest in autonomous vehicles and spell out an exciting future for

artificial intelligence, it also shows the unequal distribution of knowledge and resources in this field.

African universities and corporations are yet to make attempts to bridge this gap in unequal research into self-driving cars. Ironically, about 90% of the global death toll due to road accidents occur in low and middle-income countries; a category in which most African countries fall (WHO, 2015). This single statistic alone shows the importance of Africa joining the self-driving car research and development train. My project, which is an attempt at developing a modular autonomous self-driving vehicle platform, seeks help to bridge this gap. This platform provides students and researchers with a low cost autonomous vehicle to develop self-driving technology.

1.2 Related Work

Progressive developments in the pursuit of self-driving cars has led to technologies such as cruise control and Advanced Driver Assistance Systems(ADAS). These systems have been aimed at extending the sensory capabilities of human drivers to make the driving experience safer. This additional intelligent functionality however is described as level 1 autonomy. My project attempts level 4 autonomy, which involves the ability to act totally without human input in constrained or specific environments. Breakthroughs in the fields of computer vision, visual convolutional neural networks for image recognition and classification, as well as general advances in machine learning have made achieving such an autonomous system possible. The most important implementation of autonomous vehicles that my project closely models is Nvidia's end-to-end self-driving car experiment and Stanford University's Stanley–winner of the 2005 DARPA Grand Challenge (Thrun et al., 2006).

1.2.1 Autonomous Car Implementations

In 2016, Nvidia published the paper "End to End Learning for Self-Driving Cars" which details their own implementation of an end-to-end self-driving car system. In their implementation, they used a single front facing camera, which fed data into a convolutional neural network. This neural network then predicted steering commands. This system used supervised training, where data from human drivers was used to train the neural network. This particular end-to-end approach to self-driving is what my project employs.

Stanford University's self-driving car Stanley applied a more complex approach to self-driving. Unlike Nvidia's implementation that used a single camera, Stanley employed the use of multiple sensor systems for environment perception. These included radars, a camera, laser range finders and GPS antennas. This plethora of sensory data is integrated using an unscented Kalman filter to provide a much more accurate localization system (Thrun et al., 2006). This approach of complex sensor fusion would not be possible in the time frame of my applied project–such projects can span decades (Stavens, 2011). However, the insights and some techniques used by the Stanford team will improve my end-to-end implementation.

This project implements an autonomous vehicle using a modular architecture which the following components: perception, supervised learning and control. This approach necessitates building on work done by others under each module.

1.2.2 Perception

In the field of perception, Thorpe et al. (2001) of Carnegie Mellon university identify the shortcomings of existing efforts to improve perceptive reliability of mobile robots. They propose and illustrate the use of multiple sensor systems in developing reliable mobile robot perception. Professor of Smart Grids at the Institute of Networked and Embedded Systems, Elmenreich

(2002) shares the sentiments of the Carnegie Mellon roboticists on the importance of multiple sensory nodes in developing reliable perceptive robots. He expands on the concept of multiple sensing systems for robotics in the carefully developed art of sensor fusion. In his paper, "*An introduction to sensor fusion",* he gives an overview over the basic concepts of sensor fusion. He defines terminology in the field and discusses motivations and limitations of using sensor fusion. He also presents a detailed survey of the benefits and shortcomings of various sensor fusion architectures and algorithms including: Kalman Filters and inference methods. Thrope and Elmenreich's research provides information that will help in selecting the most efficient sensors to enable RollE observe its environment.

1.2.3 Supervised Learning

Pedro Domingos (2006) gives a simplistic yet informative and accurate overview of machine learning. He characterizes machine learning algorithms as algorithms with the potential to figure out how to perform tasks by generalizing from given examples. Supervised learning, unsupervised learning and reinforcement learning form the three main categories of machine learning algorithms. As stated by Domingos, most machine learning algorithms learn from examples. The nature of these examples is what differentiates supervised and unsupervised learning. Reinforcement learning on the other hand removes the need for examples. In supervised learning, these examples have labels, i.e. the computer is shown an example and also shown the expected behaviour for that example. From this example and correct behaviour pairs, the algorithm learns to make predictions that match the given label. Unsupervised learning, on the other hand, is given unlabelled data and learns the relationships and correlations that exist in the dataset. Reinforcement learning algorithms learn how to accomplish a task without examples. This is done using a trial and error approach with time delayed rewards associated to every

random decision the algorithm takes. In the end, learning is achieved by maximizing the rewards the algorithm accumulates.

Machine learning is essential in developing adaptable autonomous vehicles. It enables the system to capitalize on previous knowledge (training), to predict reasonable actions when given new problem sets. As an illustration, in Stanley, supervised machine learning techniques such as the Naïve Bayes algorithm were used to classify types of terrain and make speed selection predictions after training on previous terrain feature label combinations (Thrun et al., 2006). In my implementation, I use supervised learning techniques to enable RollE learn to drive itself using data gathered as a human agent drives the vehicle. This data is labelled with steering and throttle values associated with each data point. Domingos (2006) provides invaluable insights into the common pitfalls and best practices of developing practical successful machine learning applications, usually not stated in textbooks. His insights serve as a guideline I use to avoid common mistakes in designing my supervised learning system.

1.2.4 Control

Ronald Arkin and Douglas Makenzie, prolific researchers in Robotics and Artificial intelligence, discuss control architectures in their paper "Planning to Behave". They highlight the motivations and benefits of applying hybrid control architectures in mobile robot manipulation. They theorize that merging reactive and deliberative control architectures delivers an effective means of integrating world knowledge with reactive control (Arkin & Mackenzie, 1994). A case study in mobile manipulation in the context of the Autonomous Robot Architecture (AuRA), is used to solidify this theory. This article provides insight into effective control architectures for robots, such as a self-driving vehicle, that must be able to plan actions based on representation but also be able to adapt to changes and unforeseen circumstances. This desire to build a self-driving

vehicle able to adapt to unforeseen circumstances led to the design choice of using machine learning to implement the autonomous capability of RollE as opposed to rule-based programming.

Gat (1992) also presents a heterogeneous asynchronous architecture for controlling autonomous mobile robots, which can control a robot performing multiple tasks in real time, in noisy and unpredictable environments. Gat's findings confirm the benefits of hybridization in robot control as proposed by Arkin and Mackenzie (1994). The architecture proposed by Gat produces behaviour which is reliable, task-directed and reactive to contingencies–qualities which are essential to a self-driving car. The validity of the architecture detailed in this article can be seen from the experiments carried out on simulated and real-world robots. To merge deliberative and reactive control, both functions are implemented asynchronously using heterogeneous architectural elements. The central result of this work is to show that completely unmodified classical AI programming methodologies using centralized world models can be usefully incorporated into real-world embedded reactive systems (Gat, 1992).

1.3 Summary

Autonomous vehicles have the potential to significantly reduce the global death count due to road accidents. The purpose of this applied project is to apply breakthroughs in the fields of computer vision and artificial intelligence–specifically, machine learning–to implement an autonomous self-driving system. This system would help accelerate self-driving research in Africa by serving as a low cost modular autonomous vehicle development platform, aimed at providing students and researchers with a low cost autonomous vehicle to develop self-driving technology.

This project is divided into three modular components that communicate with each other yet are self-sufficient. This approach to the self-driving problem provides a skeleton that enables further developmental work on the independent components namely; perception, supervised learning and control. These independent modules fit into a larger self-driving vehicle ecosystem.

Chapter 2: Requirement Specification

2.1 Project Overview

2.1.1 Objective

This applied project seeks to build a low cost modular autonomous vehicle development platform, aimed at providing students and researchers with an affordable autonomous vehicle to develop self-driving technology. This autonomous self-driving platform shall be called RollE MkII.

2.1.2 Approach

RollE MkII is a level 4 autonomous vehicle built using a 1/16 scale RC (remote control) vehicle as the mobile base. Despite the fact that RollE MkII is built on top of an RC car, similar technologies and machine learning frameworks used in full scale autonomous cars are used in developing this platform. This makes the software system scalable to larger car platforms with minimal changes to the code base. The approach this project uses is an end-to-end one, meaning that sensor data–in this use case a camera–is directly passed through a machine learning algorithm that would in turn output steering and throttle commands. With this approach, we do not explicitly teach the model to identify hand engineered features such as outlines of roads, however the system learns necessary representational features on its own directly from the training data provided.

RollE MkII acts as a development platform for possible future work on more advanced autonomous vehicle functionality, such as complex sensor fusion using Kalman filters and self-supervised learning.

2.2 Procedure for Requirement Gathering

Requirements for this project were developed from an analysis of requirement documents for similar robotics projects built primarily for the purpose of further research as opposed to consumer markets. The requirement documents primarily used in developing these requirements were DARPA's requirement specification for the 2005 Grand challenge (DARPA, 2004). The requirements for this project are applicable to RollE MK II because, much like the objectives for this project, a core interest area of DARPA is promoting the development of autonomous ground vehicles with the ability to navigate between points while avoiding obstacles. Also, according to DARPA, "The Grand Challenge serves as a field test for autonomous ground vehicles over realistic terrain and sets specific performance goals for distance and speed" (DARPA, 2004), therefore these requirements provide a realistic framework with which to measure success.

2.3 User Classes and Use Cases

The main users of this system are researchers and fellow students looking for a complete autonomous self-driving platform to build applications for self-driving cars, or to implement higher level functionality such as road sign recognition or communication protocols for other self-driving cars in a multi-robot system.

2.4 Functional Requirements

2.4.1 Core Features

The functional requirements for the system are classified under the following;

Mobility and Locomotion

- The system should have actuators and effectors that enable it move in its environment.

- The platform should mimic a full-size car hence it should utilize proportional steering mechanisms as opposed to differential drive mechanisms.
- The system should be able to autonomously drive itself on tracks similar to what it was trained on.

Processing Data

- The onboard computer should be capable of processing images and running computations simultaneously.
- The onboard computer should be capable of running a trained machine learning model to enable autonomous driving.
- The system should have a data pipeline infrastructure that enables training data to be collected and stored efficiently.

Sensing

- The system should be able to sense its internal components such as battery capacity and strength of communication signals (proprioception).
- The system should have a camera that enables it to sense its environment (exteroception).

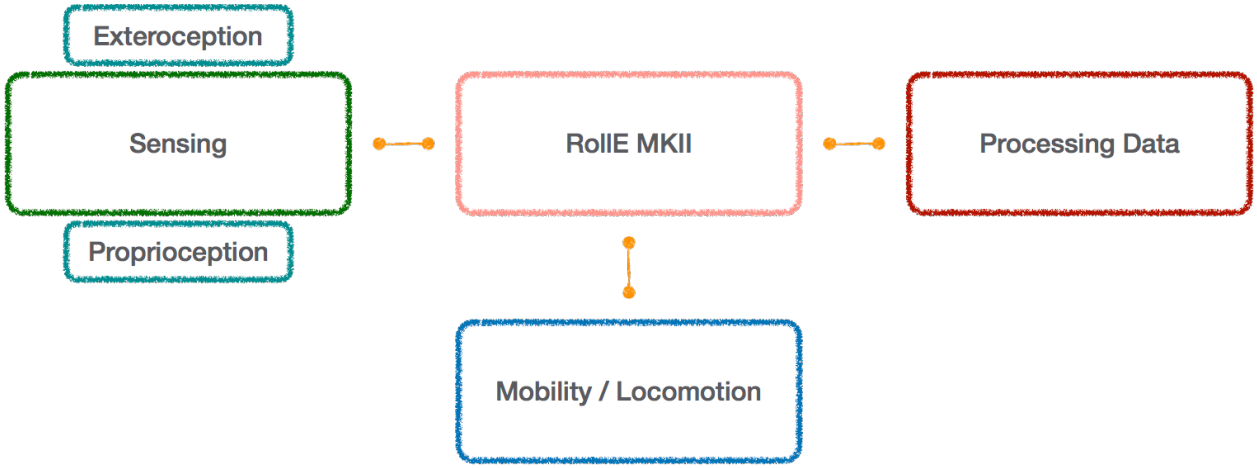


Figure 2.1 Diagram of core functionality classes of the Roll-E Mk II

2.5 Non-functional Requirements

In addition to the requirements that cover the functionality of RollE, there exists the following set of non-functional requirements:

- The system should be untethered from any station, either for power or computation. All necessary components should be onboard or communicated with via a wireless protocol.
- Batteries used to power the system should not pose a fire hazard to users.
- The system should include a visual interface to enable users view status of internal components and other processed information.

Chapter 3: Architecture and Design

3.1 System Overview

RollE MkII is a low-cost modular autonomous vehicle development platform aimed at providing students and researchers with an affordable autonomous vehicle to develop self-driving technology. This platform was developed primarily for researchers and fellow students looking for a self-driving platform to build applications for self-driving cars.

The system architecture of RollE provides a complete overview of its hardware and software infrastructure. RollE can be considered as having four layers: the physical layer which consists of hardware components, and a collection of logical layers which refers to conceptual layers representing the organization of related of software units.

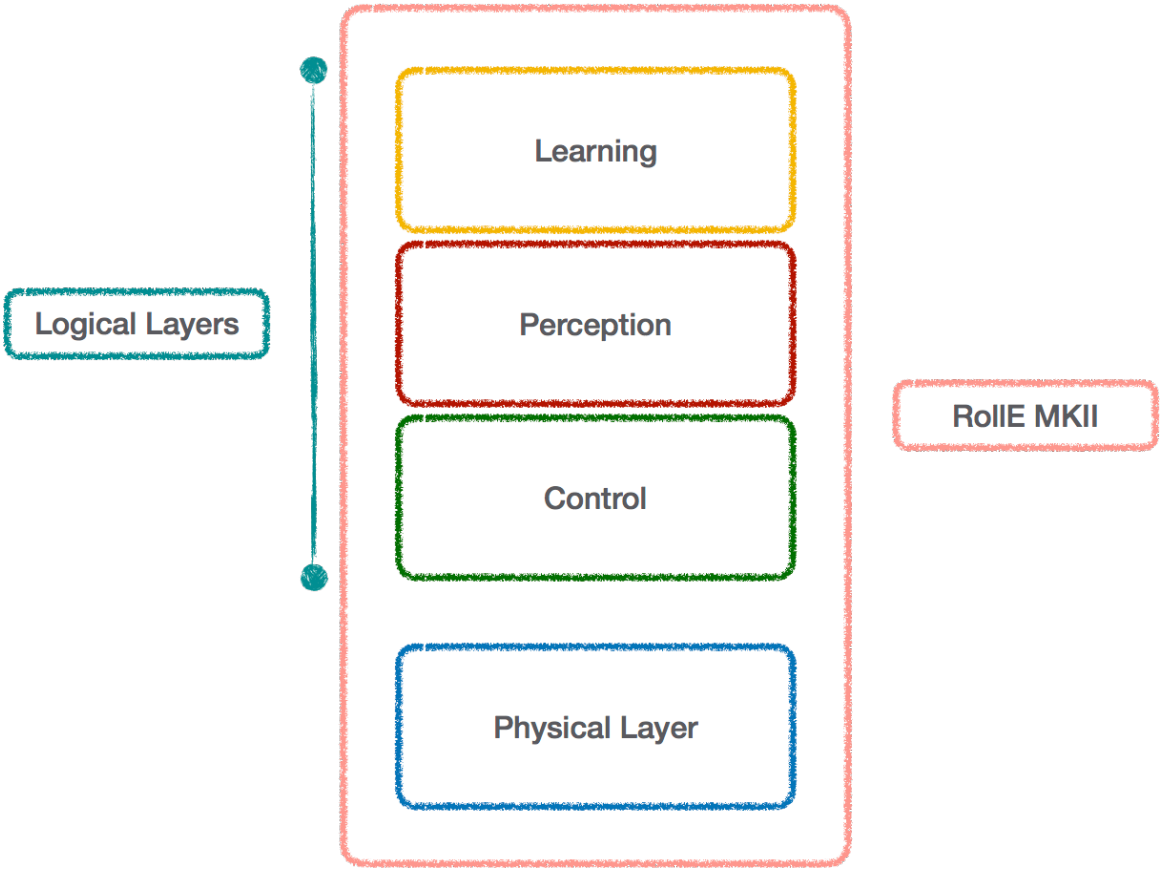


Figure 3.1 Layered architecture of RollE MKII

3.2 Physical Layer / Hardware Components

One of the objectives of RollE is to be modular; as such the system consists of a number of interacting physical components. The physical layer refers to the collection of hardware components carefully selected and assembled in a fashion that fulfils both the functional and non-functional requirements stated earlier.

The hardware components that make up the physical layer are:

- RC car: this is a 1/16 scale Exceed Blaze RC car that acts as the mobile base of RollE. Its 1/16 scale ratio means every inch on the replica vehicle corresponds to 16 inches on a real-life car. The Exceed Blaze was chosen for its build quality and proportional steering which, unlike the differential drive mechanism used in some mobile robots, closely imitates the steering mechanism of an actual car. It also has hobby-grade parts which makes upgrading easy. This component has actuators and effectors that enable RollE to move in its environment, thus fulfilling the "Mobility and Locomotion" functional requirement. The Exceed Blaze has the following electronic components;
    - 7.2V brushed DC motor with RPM <30000 which acts as the throttle motor
    - MG 996R servo motor which controls steering
    - WP-1040-Brushed Electronic Speed Controller (ESC)
    - 6 cell 1100mAH 7.2V Nickel Metal Hydride(Ni-MH) battery pack – chosen because this battery composition makes the battery pack less likely to explode during charge as compared to lithium polymer(Lipo) batteries.
    - 4-channel 2.4Ghz receiver and transmitter

- RollE Pilot: this is a remote controller that pairs with RollE and is designed to enable a user to manually drive the vehicle. It consists of the following components:
    - One Arduino Uno, an open source microcontroller designed to enable reading sensor inputs, performing some computation and effecting physical outputs through electronic components. It serves as the brain of the remote controller
    - A pair of XY joystick modules, each built from two potentiometers set up in a 2 - dimensional fashion that enables the movement of a central arm along the X and Y axes to be measured.
    - One liquid crystal display (LCD) module which displays information.
- Raspberry Pi: this is a single board computer that acts as the on-board processor of RollE. It is capable of processing images and running computations, hence fulfils the "processing data" functional requirement. The Raspberry Pi was chosen for its affordable price point, as well its plethora of I/O ports which gives the end users of RollE the freedom to add on various other sensors. The inbuilt general-purpose input/output (GPIO) pins provide an interface to programmatically communicate with low-level sensors and electronic devices such as the ESC and servo motor on the Exceed Blaze.
- Raspberry Pi Camera: this is a single board module fitted with a 5MP Omnivison 5647 fixed focus sensor. It is capable of taking high resolution images and gives RollE the ability to sense its environment, and thus fulfils the "Sensing (exteroception)" functional requirement.
- Router: this creates an on-board wireless network that enables a user to wirelessly communicate with RollE over considerable distances.

- Adafruit PCA9685 16-Channel 12-bit PWM/Servo Driver: this module enables programmatic control of the steering servo motor and throttle DC motor on the Exceed Blaze through pulse width modulation signals.

The physical layer also consists of a collection of 3D modelled components designed to allow the other electronic components of the physical layer to be assembled in a functional and aesthetic fashion. The 3D components are also designed to enable end users attach more components, such as additional sensors like LIDARs, onto RollE.

3.3 Logical Layers

The logical layers of the system each contain a collection of related software units or processes that work together to accomplish some goal. Processes within a layer can communicate with each other as well as communicate with processes from other layers. Inter-process communication is achieved through a publish/subscribe-based communication system. This allows processes to subscribe to specific topics of interest in order to receive broadcast messages published on those topics by other processes. This communication architecture is implemented using the lightweight Message Queuing Telemetry Transport(MQTT) protocol built for connecting devices on networks with minimal bandwidth.

RollE consists of the following interacting logical layers:

- Control: the software units in this layer serve as an interface between the user and the hardware components of the physical layer. They are also responsible for converting steering outputs from the processes in the learning layer into specific pulse values the PCA9685 PWM module can understand. This layer also contains a command-line software implementation of the RollE Pilot (remote controller) called Soft Pilot. This

terminal application gives a user discrete control of the throttle and steering values that drives RollE.

- Perception: the software units in this layer mainly implement computer vision procedures for capturing, formatting and either transmitting or locally storing images obtained from the on-board camera sensor.
- Learning: the software units in this layer deal with machine learning. They specify the architecture for machine learning models and contain code that manage data processing, training the machine learning model and transmitting predictions from trained models to other processes. The machine learning model architecture of choice for this problem domain is a convolutional neural network. Convolutional neural networks are multilayer perceptron machine learning algorithms optimised for analysing visual images and feature extraction by incrementally applying convolutional operations to images at certain layers of the network.

Chapter 4: Implementation

This chapter contains a detailed description of the implementation of this applied project. As stated in the earlier chapter, the architecture of RollE MkII consists of a physical layer and a collection of logical layers, each comprising various components and modules. In this chapter, I give an overview of the functionality of the complete prototype and a detailed description of how each of these modules are implemented to satisfy the functional requirements. I also detail the software tools and libraries I used in my implementation.

4.1 Implementation Overview

A front-facing camera connected to the on-board computer (Raspberry Pi) acts as the single exteroception sensory node of RollE. RollE has two modes of operation:

- Data collection mode
- Autonomous mode

When in the data collection mode, RollE is controlled by a human agent using either the RollE Pilot or the terminal remote-control application (Soft Pilot). In this mode, image frames are captured from the camera at a resolution of 200x66 at a frame rate of 32 fps. Each frame is stored and its filename recorded with a timestamp and the corresponding throttle and steering values sent from the remote controller at the time of capture. At the end of a data collection run, the images are stored in a folder and the records of steering and throttle commands compiled and saved in a csv file.

The data from a data collection run is used to train an end-to-end convolutional neural network implemented using Keras, a neural network application programming interface, and based on the architecture used by Nvidia in their self-driving car experiment.

In autonomous mode, RollE is controlled by an autopilot. The camera repeatedly captures frames of its environment and the autopilot software, running locally on RollE, uses the trained convolutional neural network model to predict steering angles for each frame. The throttle value for speed control is set to a constant value.

The captured image frames and steering predictions are transmitted via a socket connection to a user's computer for visualization.

4.2 Software Libraries / Tools

The implementation of this applied project depended on a suite of existing libraries and software tools. The following libraries and tools were used in the development of RollE MkII;

- Keras: This is a high-level application programming interface for designing neural networks. In my implementation, I used Tensorflow, which is a software library for numerical computation using data-flow graphs, as a backend for Keras. Keras allows neural network architectures to be represented as a sequential stack of layers and is aimed at allowing fast experimentation with neural networks.
- OpenCV: This is a library of pre-implemented functions aimed at easing the development of computer vision applications.
- Paho-mqtt: This library provides an implementation of the MQTT protocol wrapped as a collection of objects and functions.
- Pyserial: This is a python application programming interface that allows a user to access serial ports.
- Autodesk Fusion 360: This is computer-aided design (CAD), computer-aided manufacturing(CAM) and computer-aided engineering (CAE) software that enables robust design, testing and fabrication of complex 3D components.

- Pandas: This is a data analysis library that provides data structures and data analysis tools.
- Numpy: Numpy is a python computational library optimized for performing mathematical operations on multi-dimensional arrays and matrices.
- PiCamera Library: This is a library with a collection of functions that simplifies interfacing with a Raspberry Pi camera module.
- Adafruit_PCA9685 Library: This is a library with a range of intuitively designed objects and functions that makes interfacing with the Adafruit PCA9685 16-Channel 12-bit PWM/Servo Driver easy.

4.3 Physical Layer

The physical layer consists of a collection of hardware components carefully selected and assembled in a fashion that fulfils both the functional and non-functional requirements stated in Chapter 2. These hardware components are put together to form the complete RollE MkII system which consists of a complete RollE Rover and RollE Pilot.

4.3.1 RollE Rover

The RollE Rover uses a 1/16 scale Exceed Blaze RC car as its mobile platform. This RC car comes with a 7.2V brushed motor, MG 996R steering servo motor, electronic speed controller, a 6 cell 1100mAH 7.2V Nickel Metal Hydride battery pack and a 4 channel 2.4 Ghz receiver. The default setup of the RC car wired the battery pack and throttle motor to the electronic speed control. The steering servo and ESC were connected to separate channels on the 2.4Ghz receiver, enabling the car to be controlled via a radio transmitter.

For my purposes, I kept the connection between the battery pack, ESC and throttle motor. However, I rewired the steering servo and ESC from the 2.4 Ghz receiver to individual channels on an Adafruit PCA9685 PWM driver. I then connected the PWM driver to a Raspberry Pi via the on-board GND, VCC, SCL and SDA GPIO pins of the Raspberry Pi. This connection enables programmatic control of the servo and throttle motors by sending pulse width modulation signals to these components. Enabling otherwise discrete 1/0 electric signals to become a range of values. This is achieved through the repeated pulsing of 1/0 states in a specified duty cycle. The technique of pulse width modulation enables a user to programmatically vary the speed of the throttle motor or specify the desired angle of turn of the steering servo motor. I also connected a single Raspberry Pi camera, which acts as RollE's sensory node, to the Raspberry Pi.

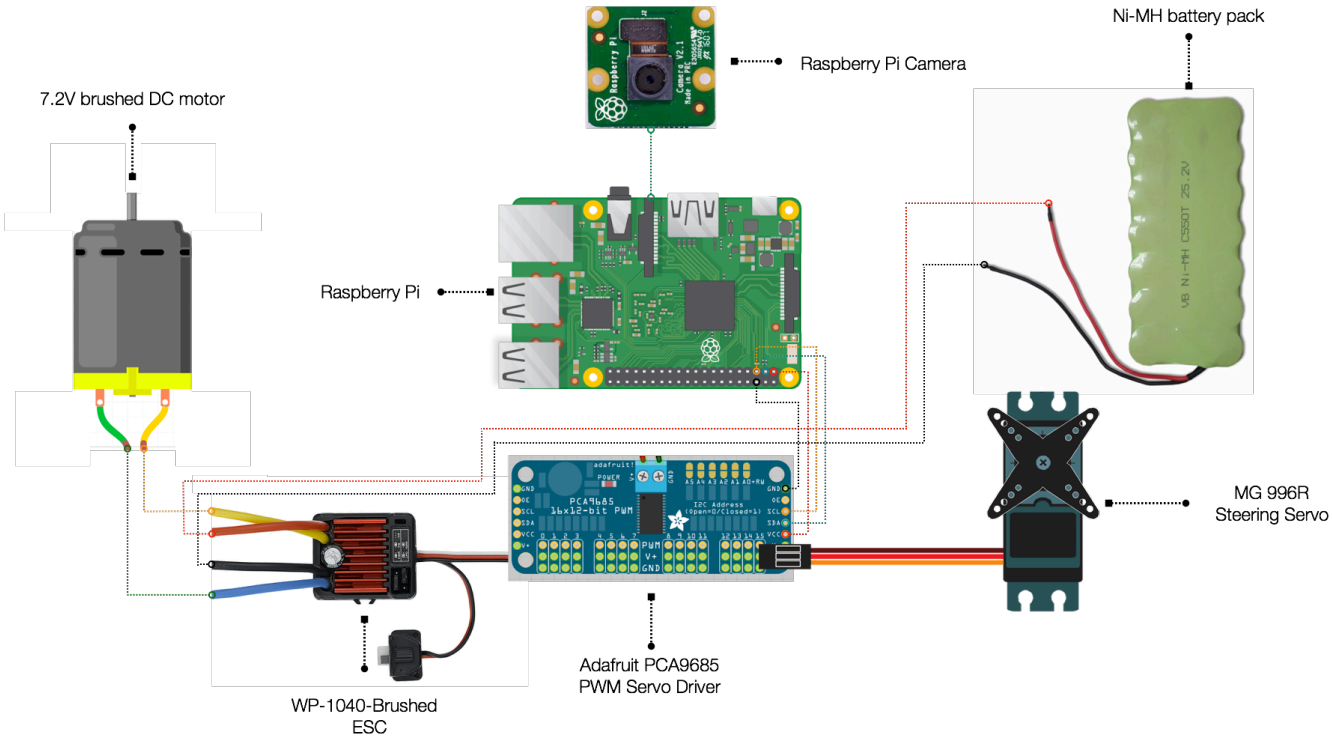


Figure 4.1 Diagram of RollE Rover's component connections

A collection of modular 3D printable components designed in Autodesk Fusion 360 provide a functional and aesthetic way to attach all the components of the RollE Rover to the mobile RC car platform. These components are designed to vertically stack on top of a base structure called the spine. This enables end users to vertically stack as many additional components and sensors as desired.

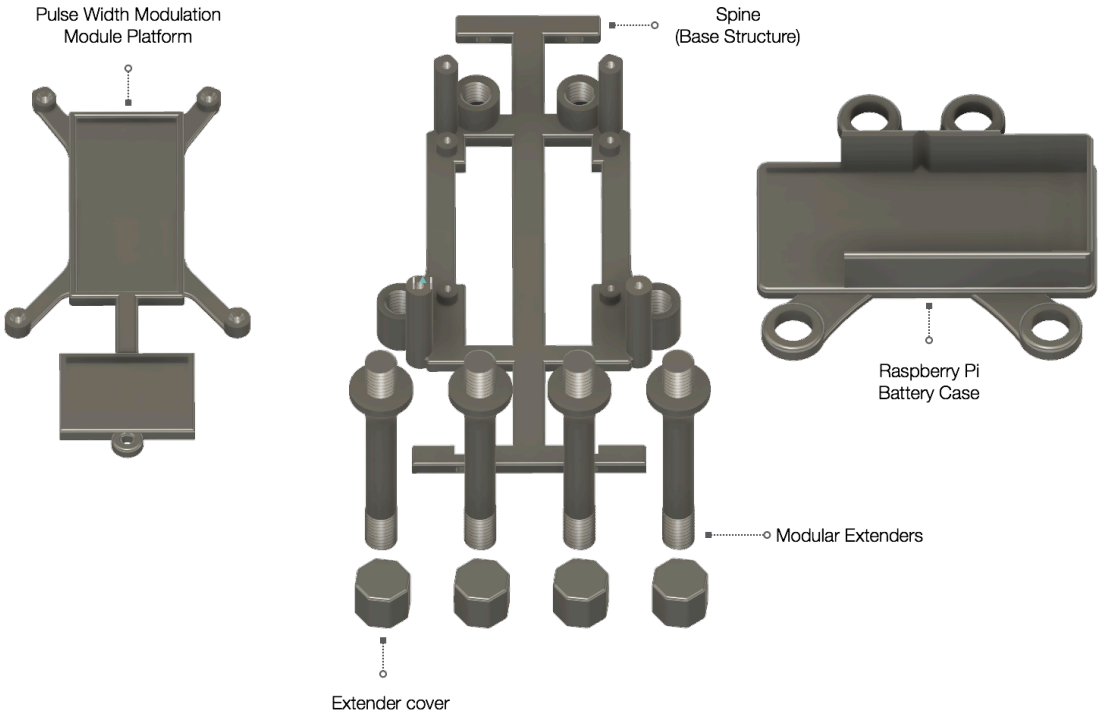


Figure 4.2 3D printable components of RollE Rover

Below is an image of what the RollE Rover looks like after the complete circuit has been built and 3D printable components have been assembled.

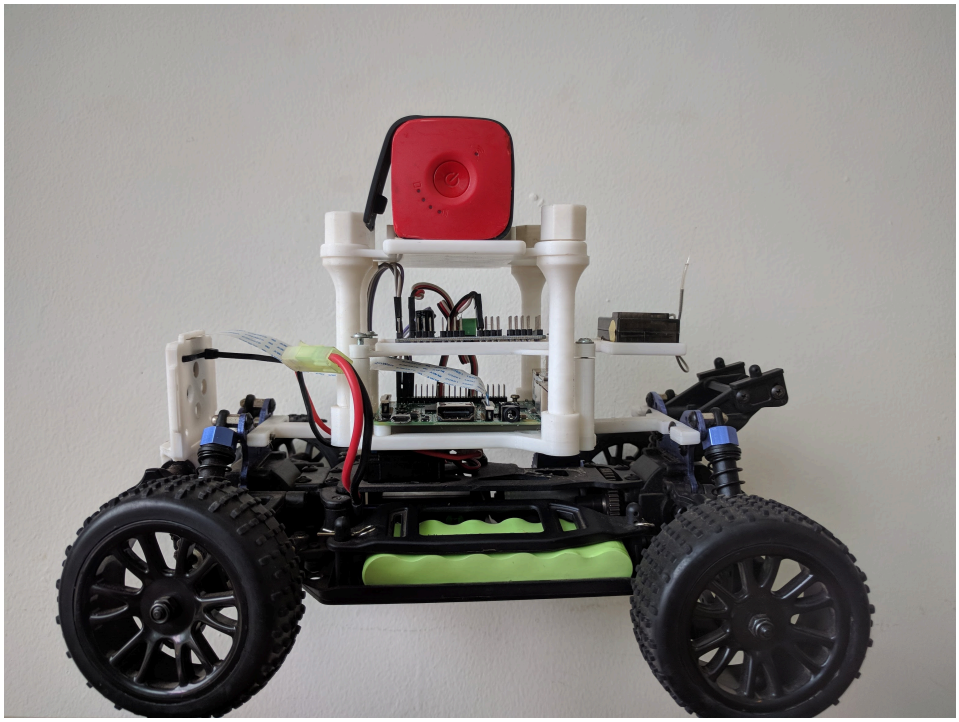


Figure 4.3 Completely assembled RollE Rover

4.3.2 RollE Pilot

The RollE Pilot (remote controller) is built from an Arduino Uno, a pair of XY joystick modules and a liquid crystal display (LCD). It is designed to allow end users to add additional functionality; it is designed to be attachable to a standard breadboard providing room for experimentation. The Arduino acts as the brain of the remote controller; it takes sensor readings, performs computation and outputs information. A pair of XY joysticks acts as the sensors in this system; each joystick is connected to the Arduino in a fashion that restricts readings to one axis per joystick. With this configuration, the joystick that takes readings from the X-axis controls throttle while the other takes readings from the Y-axis and controls steering. An LCD module displays the real-time readings from each joystick and prints out error messages. Sensor readings from joysticks are analog and fall in a range of values from 0 to 1023. I remap these values to a range between -1 to 1. These readings are then transmitted via serial communication from the

Arduino to a python script that separates each joystick reading and broadcasts the values to two MQTT topics (RollE_MKII/throttle and RollE_MKII/steering). The description of the control logical layer expands on this python script.

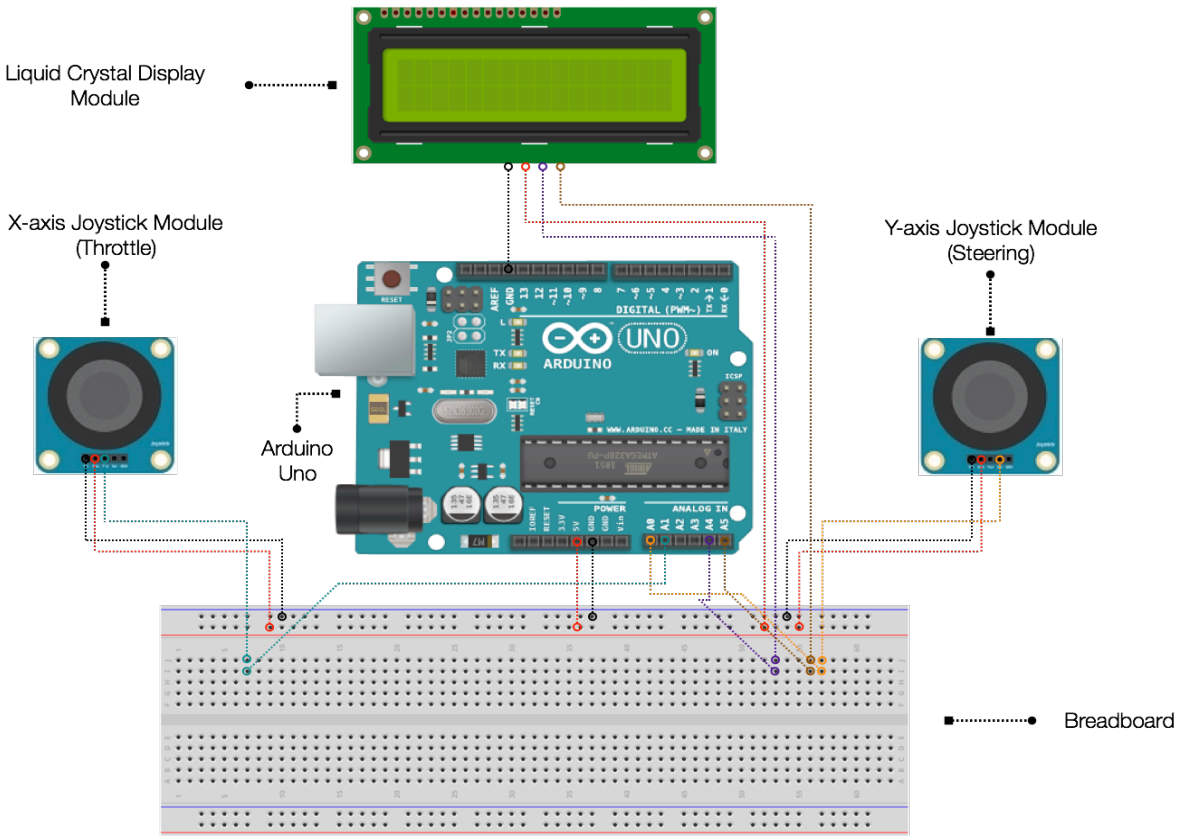


Figure 4.4 Diagram of RollE Pilot component connection.

Much like the RollE Rover, the remote controller also has a collection of 3D printable components designed to hold all components in a functional and aesthetic fashion. The base 3D component of the remote controller is what allows it to be attached to a standard breadboard, enabling end users to experiment with the remote and attach more components.

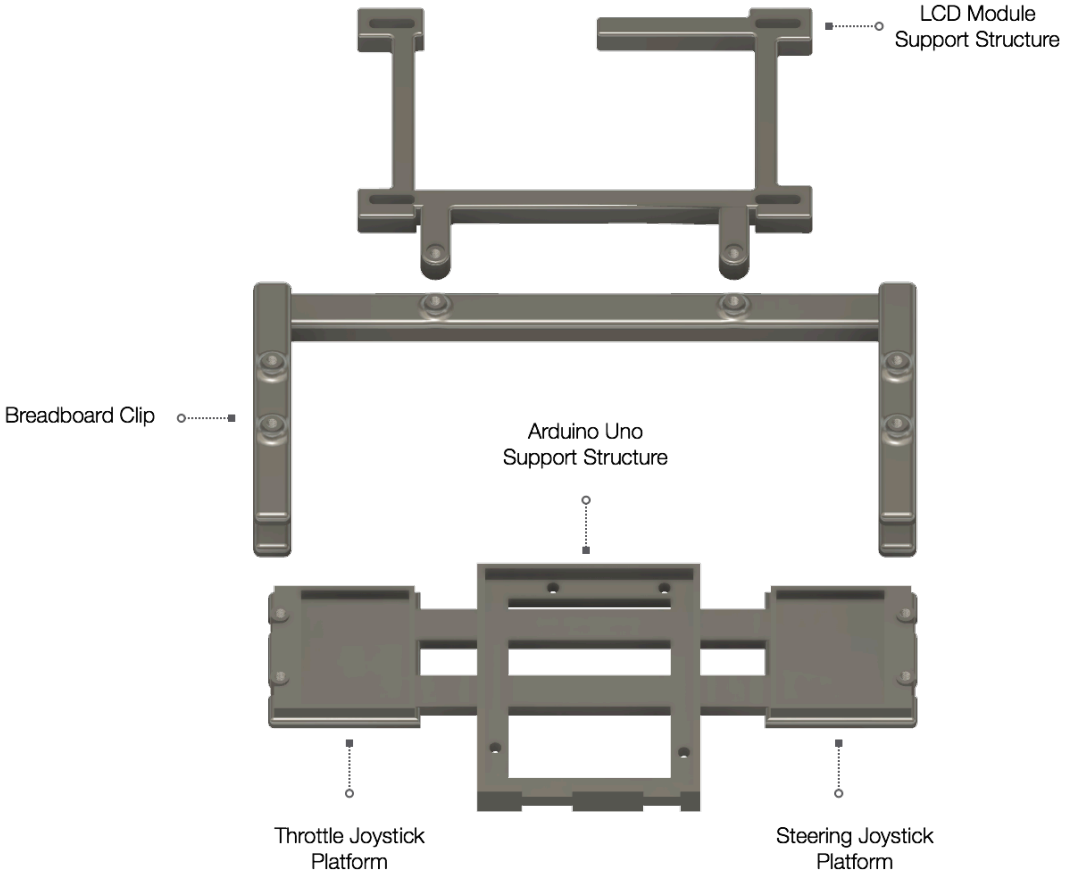


Figure 4.5 3D printable components of RollE Pilot

Below is an image of what the RollE Rover looks like after the complete circuit has been built and 3D printable components have been assembled.

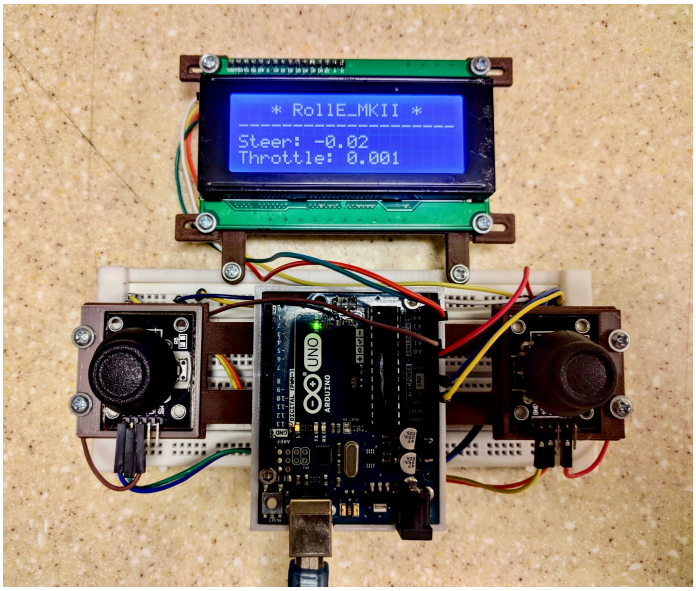


Figure 4.6 Completely assembled RollE Pilot

4.4 Control Layer

This layer contains a collection of software units that directly interface with the components of the physical layer. The software in this layer has three primary functions:

- provide programmatic control of the steering and throttle motors of the RollE rover
- take joystick position readings and drive the LCD display on RollE Pilot
- transmit steering and throttle values to the rover

Three software units “Actuation.py”, “RollE_Pilot.ino” and “Pilot_Transmitter.py / soft_Pilot.py” respectively provide this functionality.

4.4.1 Actuation.py

This is a python script that runs on the Raspberry Pi on the RollE rover and uses the Adafruit_PCA9685 library to specify which signals the PWM module sends to steering and throttle motors. I remap input values from a range of -1 to 1 into pulse signals that correspond to maximum and minimum duty cycles. This technique associates the maximum and minimum steering servo angles and throttle motor speed with a value of -1 and 1 respectively. All values between this range then represents different steering positions and throttle speeds. As an example, a steering value of -1 would turn the steering servo to its maximum position on the right while a steering value of 0 would turn the steering servo to its middle position; similar logic applies to throttle speeds.

Limiting motor control inputs to the range -1 to 1 simplifies controlling RollE for end users as they can create higher level programs and only have to think of steering and throttle outputs in terms of proportionality as opposed to raw duty cycles. This also allows them to make simple modifications to global maximum and minimum duty cycle values in actuation.py to change the range of responsiveness or sharpness of steering turns without changing the outputs from their

higher-level programs. An example of such higher-level programs is the software implementation of the RollE pilot. This program takes keyboard inputs from users and outputs steering and throttle to values in the range -1 to 1 to control RollE.

4.4.2 RollE_Pilot.ino

This is an Arduino sketch that controls the RollE Pilot controller. It takes readings from the 2-dimensional potentiometers that make up each joystick module and remaps these analog values from a range of 0 to 1024 to values in the range -1 to 1. It also drives a 20x4 LCD module which displays the real-time remapped steering and throttle values from the joysticks. This code is flashed and runs on the onboard microcontroller of the RollE Pilot.

4.4.3 Pilot_Transmitter.py / soft_Pilot.py

The pilot_Transmitter.py script communicates with the RollE Pilot via a serial interface and transmits steering and throttle values obtained from the controller to the RollE rover via MQTT, a wireless lightweight messaging protocol. Soft_Pilot.py is also a command-line software implementation of the physical remote controller; it accepts user keyboard inputs and outputs steering and throttle values in the -1 to 1 range to RollE also via MQTT. MQTT is a publish/subscribe based messaging protocol, the messaging network is facilitated by a host (broker) and enables clients to subscribe to topics and receive messages published to that topic by publisher clients. In RollE, the onboard Raspberry Pi acts as the broker for the network. A user connects the RollE Pilot to their computer and the Pilot_Transmitter.py program, which also runs on the user's computer, transmits real-time steering and throttle values to the RollE_MKII/throttle and RollE_MKII/steering topics respectively. Actuation.py, which subscribes to these topics, receives the transmitted steering and throttle values and converts them to PWM duty cycles to drive RollE.

4.5 Perception Layer

The software units in this layer implement computer vision procedures for capturing, formatting and either transmitting or locally storing images obtained from the on-board camera sensor of the RollE rover. In the data collection mode, images are stored locally on the Raspberry Pi. In autonomous mode, images are served as inputs to a trained neural network model running locally on the Raspberry Pi for steering predictions. These images can also be transmitted via a socket connection to the user's computer for visualization.

The computer vision pre-processing operations performed on each image frame are:

- performing a 180-degree rotation on captured images. This is done because restrictions in the length of ribbon cable that connects the Pi camera to the Raspberry Pi forced the camera to be placed upside down during assembly of the Rover.
- cropping image to remove unwanted parts of images. Unwanted parts are classified as parts of the image above the track.
- resizing image to fit the shape that the convolutional neural network accepts. The acceptable shape must have an image height of 66 pixels, width of 200 pixels and 3 colour channels (3@66x200).
- converting the colour model of the image frame from RGB to the YUV colour space. The YUV colour space represents images with one luma component (Y) and two chrominance components(UV). The input image is split into these individual YUV planes before being passed into the convolutional network for feature extraction.

Below is a series of images that show the transition of an image from its original state to its state after pre-processing:

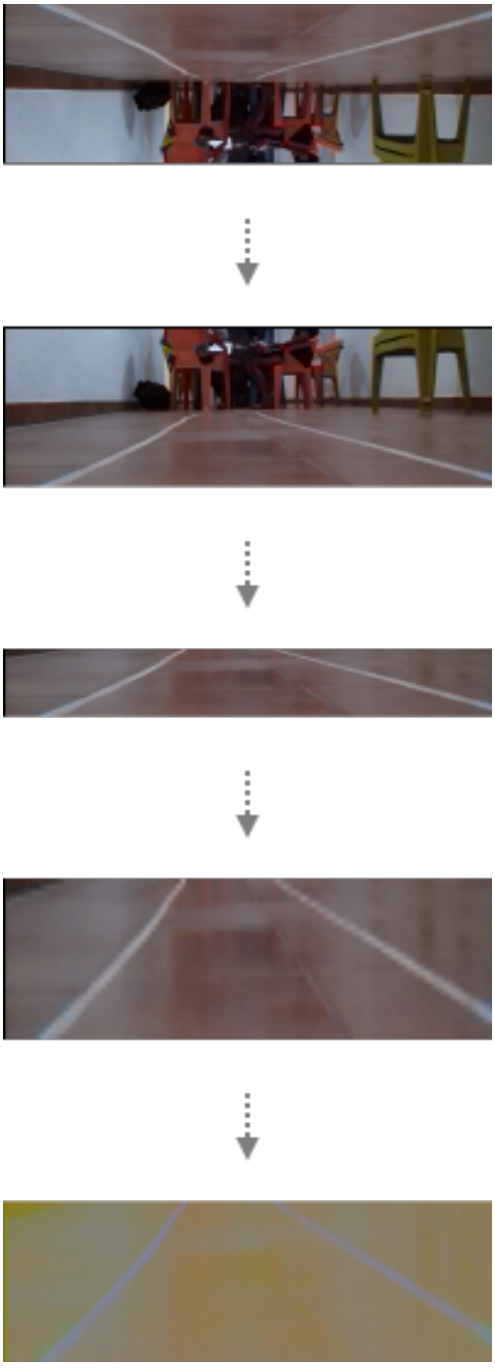


Figure 4.7 Image pre-processing results

4.6 Learning Layer

The software units in this layer deal with machine learning. They specify the architecture of machine learning models and contain code that manage data processing and augmentation, training the model and transmitting predictions from trained models to other processes. The model used for this system is a convolutional neural network based on the end-to-end architecture used by Nvidia in their self-driving car experiment.

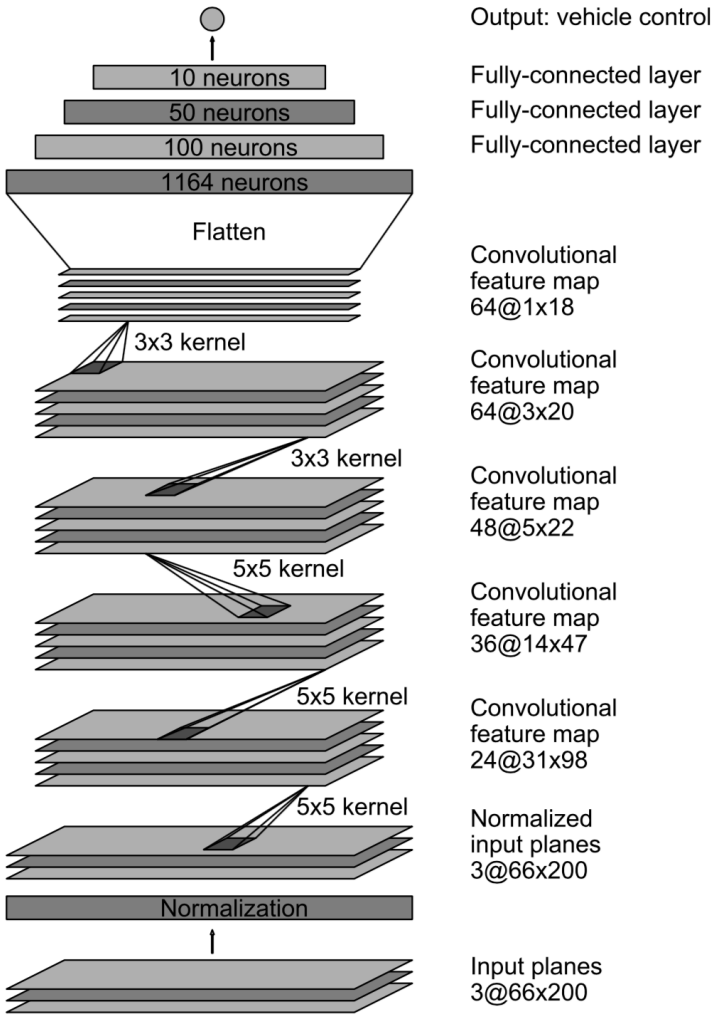


Figure 4.8 Nvidia's Convolutional Neural Network Architecture

Below is a snapshot of Nvidia's model implemented with Keras:

```
**********************************************************************
def model():
    model = Sequential()
    # normalize inputs to range -1 to 1
    model.add(Lambda(lambda x: x/127.5-1.0, input_shape=(imageHeight,
imageWidth, imageChannels)))
    #convolutional layers for feature extraction
    model.add(Conv2D(24, 5, 5, activation='elu', subsample=(2, 2)))
    model.add(Conv2D(36, 5, 5, activation='elu', subsample=(2, 2)))
    model.add(Conv2D(48, 5, 5, activation='elu', subsample=(2, 2)))
    model.add(Conv2D(64, 3, 3, activation='elu'))
    model.add(Conv2D(64, 3, 3, activation='elu'))
    #dropout with probability of 0.5 to reduce overfitting
    model.add(Dropout(0.5))
    model.add(Flatten())
    # fully connected layers with elu activation
    model.add(Dense(100, activation='elu'))
    model.add(Dense(50, activation='elu'))
    model.add(Dense(10, activation='elu'))
    # this is the output layer
    model.add(Dense(1))
  **********************************************************************
```

After data is collected during a data collection run, the recorded collection of image frames and steering labels are used to train this convolutional neural network. Data augmentation is performed on the training examples to produce variations that help the model learn more from

existing examples. Two key data augmentation operations, namely, random horizontal flips and random shadowing, are performed on training examples in real-time before the actual training is done. I randomly flip some training examples and negate the steering angle to create new data points. I also create random shadows on some training examples to help the model learn how to handle real-world scenarios where captured images are dark due to shadows from objects.

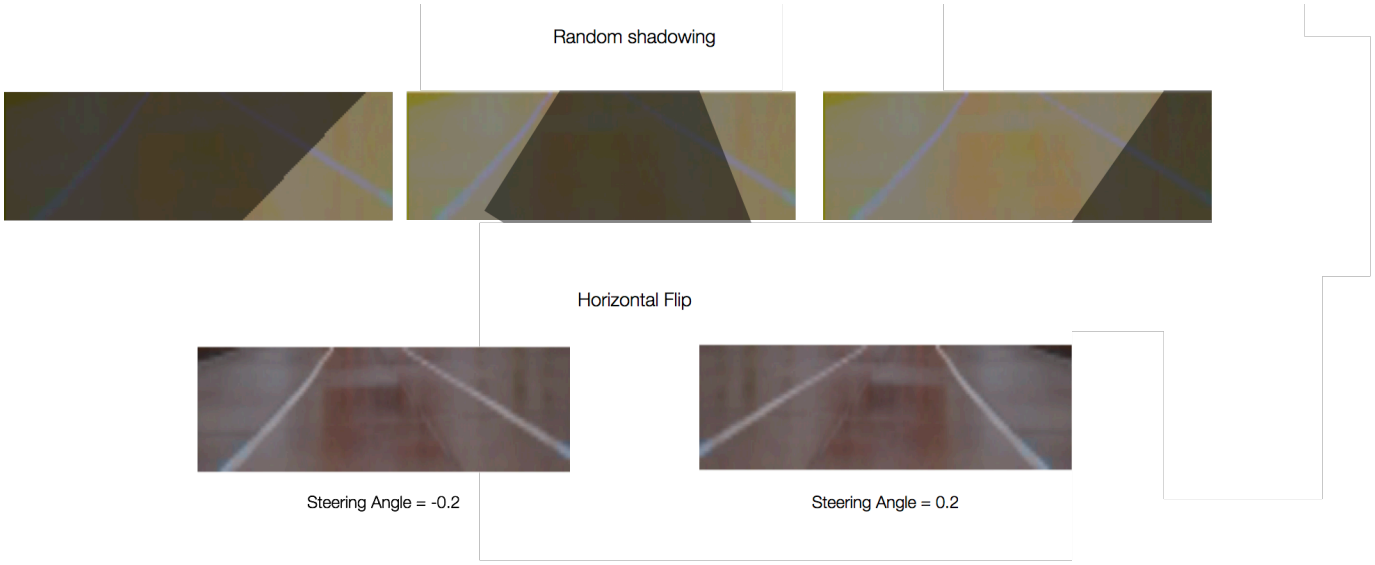


Figure 4.9 Data augmentation results

During autonomous mode, a constant throttle value is set, and new image frames are inputted into the trained neural network which predicts a steering angle. This steering value is published via MQTT to *actuation.py*, effectively steering the car as it moves.

Chapter 5: Testing and Results

5.1 Approach

In implementing this applied project, various component and system-wide tests were conducted to confirm that all software units functioned optimally. The collection of tests conducted can be grouped into two sub groups, namely, component testing and system testing. This chapter discusses the details of various tests and their observed results.

5.2 Component Testing

Component testing encapsulates all tests conducted on individual software units and components that collectively make up the RollE system. As stated earlier, the architecture of RollE comprises four interacting layers: the physical layer and the control, perception and learning logical layers. Tests were conducted for each of these layers.

5.2.1 Physical Layer Tests

Testing circuit connections of RollE Pilot and RollE Rover.

This test was performed to verify the correct circuit connection of all electronic components of the RollE Rover and Pilot according to the circuit diagrams described in Chapter 4.3. It was also performed to identify possible defects in components or jumper wires.

RollE Rover circuit testing

Circuit connections between the Adafruit PWM module, GPIO pins of Raspberry Pi, electronic speed controller and motors were confirmed using the schematic 4.1 and the system was powered on. The following results were expected:

- an indicating LED on the PWM module was expected to turn on

- the electronic speed controller was expected to blink a red LED and give out a single short beep
- steering servo motor was expected to freeze its position and resist manual repositioning

Results

After conducting the test all the expected behaviours highlighted above were demonstrated by the RollE Rover.

RollE Pilot circuit testing

Circuit connections between the Arduino, XY joysticks and LCD module were confirmed using the schematic provided in Figure 4.4 and the system was powered on. The following results were expected:

- LCD backlight was expected to turn on
- Green power LED near reset button of Arduino was expected to turn on.

Results

After performing this test, the following observation was made;

- The expected LCD backlight turned on for a few seconds and turned off again
- The power indicative LED on the Arduino turned on for a few seconds and turned back off

Resolution

I used a hardware variation of step debugging to isolate the cause of this behaviour. By incrementally reassembling and testing individual components of the system, I identified the problem to be a fault in the soldering of the header pins of one XY joystick module. This mistake caused the setup to short-circuit. A safety feature built into the Arduino microcontroller prevents the flow of excessive current by immediately shutting down the microcontroller. Hence, the

system shuts down seconds after turning on. I replaced the faulty joystick and the expected behavior was exhibited.

5.2.2 Control Layer Tests and Results

Testing throttle and steering values computation on RollE Pilot and displaying results on LCD module

This unit test was performed to verify that throttle and steering values could be computed from positions of the XY joysticks to fit in a range from of –1 to 1. The test was also performed to confirm that real-time throttle and steering values could be displayed on the LCD screen of the remote controller. In this test, I used the inbuilt serial plotter tool in the Arduino IDE to graph the computed throttle and steering values.

The following results were expected:

- Joystick positions converted to throttle and steering values in the range -1 to 1
- The computed throttle and steering values should be displayed and updated on the LCD module in real-time

Results

During the testing, the following behaviours were observed:

- The serial plotter tool revealed that indeed computed throttle and steering values were in the range -1 to 1.

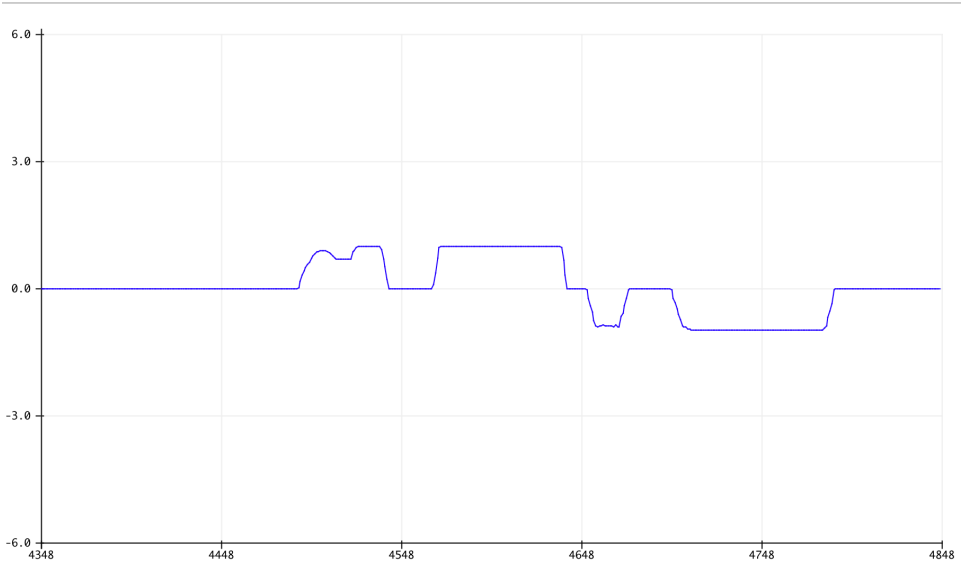


Figure 5.1 Serial plotter output

The plot above shows throttle values mapped out over time, it can be noticed that values peak and flatten at -1 and 1. The same behaviour is exhibited by steering values.

- Another behaviour noticed was that the LCD failed to display the computed throttle and steering values. The bulk of text displayed on the LCD was an unintelligible sequence of characters and symbols.

Resolution

Isolating the display system and incrementally reconnecting and testing its circuit revealed the problem to be a faulty jumper cable. A faulty jumper cable connecting one of the signal pins of the LCD module to the Arduino introduced noise which was interpreted as valid signal by the LCD module. Replacing this cable rectified the problem.

Testing Serial communication between RollE Pilot and Pilot_Transmitter.py

This unit test was performed to verify that steering and throttle values could be transmitted from the RollE Pilot to a user's computer running "Pilot_Transmitter.py" via serial communication. In this test, the Pilot was connected via a USB cable to a laptop. A serial connection with a baud

rate of 9600 (9600 bits of data transmitted per second) was established between the two devices through a test python script. The throttle and steering joysticks were randomly moved to various positions during this test.

The expected result was for the throttle and steering values computed by the remote controller to be logged on the screen of the laptop in real-time.

Results

The expected behavior was observed during the test. Below is a snapshot of the logged values transmitted from the remote controller.

```
Benedicts-MacBook-Pro:Control benedictquartey$ python pilot_transmitter.py
b'Throttle:  0.10:\r\n'
b'Steering: -0.03:\r\n'
b'Throttle:  0.03:\r\n'
b'Steering: -0.03:\r\n'
b'Throttle:  -0.01:\r\n'
b'Steering: -0.03:\r\n'
b'Throttle:  -0.01:\r\n'
b'Steering: -0.03:\r\n'
b'Throttle:  -0.01:\r\n'
b'Steering: -0.03:\r\n'
b'Throttle:  -0.01:\r\n'
b'Steering: -0.03:\r\n'
b'Throttle:  -0.01:\r\n'
b'Steering: -0.03:\r\n'
b'Throttle:  -0.01:\r\n'
b'Steering: -0.03:\r\n'
b'Throttle:  0.07:\r\n'
b'Steering: -0.03:\r\n'
b'Throttle:  0.90:\r\n'
b'Steering: -0.03:\r\n'
b'Throttle:  0.98:\r\n'
b'Steering: -0.03:\r\n'
b'Throttle:  0.98:\r\n'
b'Steering: -0.03:\r\n'
```

Figure 5.2 Snapshot of the RollE Pilot serial transmission

Testing communication between PWM module and motors

This unit test was to ensure that the Adafruit PWM module correctly interfaced with the motors and electronic speed control. The essence of the test was also to verify that pulse width modulation signals could be used to control the throttle and steering motors. In this test, I wrote a

python script to instantiate a connection between the Raspberry Pi and the throttle and steering motors through the Adafruit PWM module. This script then sent test PWM signals to the motors. The following behaviour was expected:

- A blinking LED on the ESC was expected to stop blinking and stay on, the ESC was also expected to give out a beep that lasted for a second
- The servo motor was expected to reset its position to its default and resist manual repositioning
- When PWM signals were sent to steering motor it was expected to turn to its leftmost position, wait a second and turn to its rightmost position
- When PWM signals were sent to the throttle motor, it was expected to drive forward at it maximum speed, brake after two seconds and reverse at its maximum speed

Results

After conducting this test, all of the expected behaviour outlined above was exhibited by the system.

5.2.3 Learning Layer Tests and Results

Training convolutional neural network model for different hyper parameters

This unit test was designed to verify that my implementation of Nvidia's neural network works and yields acceptable results. It was also used to give an idea of which range of hyper parameter values of the network would produce a good model. In this test, I gathered data by driving RollE and trained 3 models each with some changes to hyper parameters. I kept the number of epochs and learning rate hyper parameters constant and only varied the number of samples per epoch for each model.

Results

I graphed the mean squared error of both training and validation sets over all epochs to evaluate the performance of each model.

Model 1

Hyper parameters:

Number of Epochs = 10, number of samples per Epoch = 200, learning rate = $1.0 * e^{-4}$

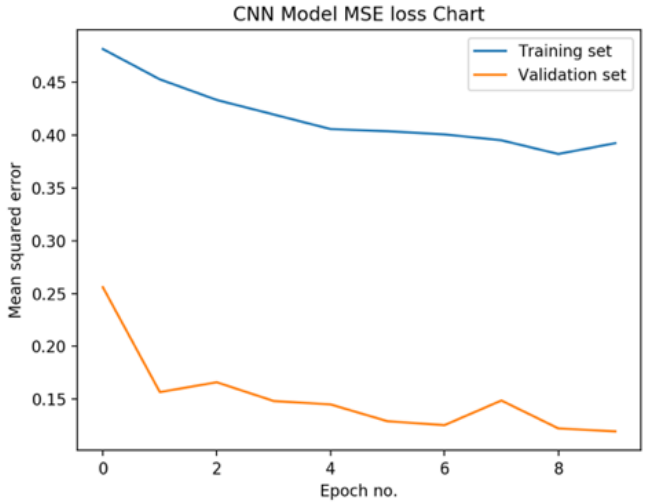
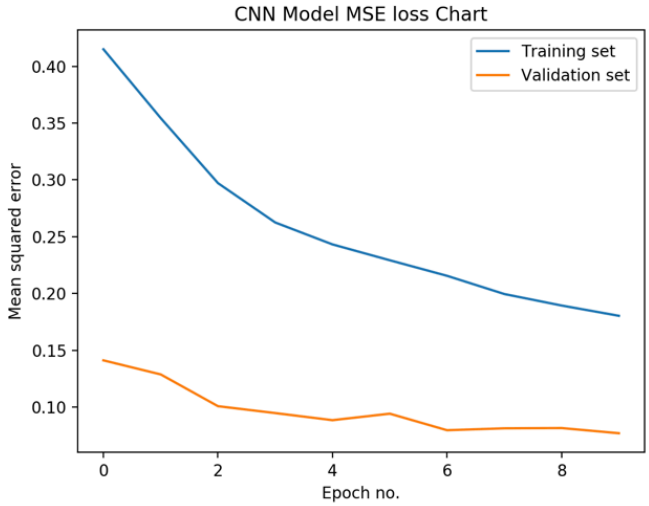


Figure 5.3 Model 1 mean squared error graph

Model 2

Hyper parameters:

Number of Epochs = 10, number of samples per Epoch = 2000, learning rate = $1.0 * e^{-4}$



Model 3 Figure 5.5 Model 2 mean squared error graph

Hyper parameters:

Number of Epochs = 10, number of samples per Epoch = 20000, learning rate = $1.0 * e^{-4}$

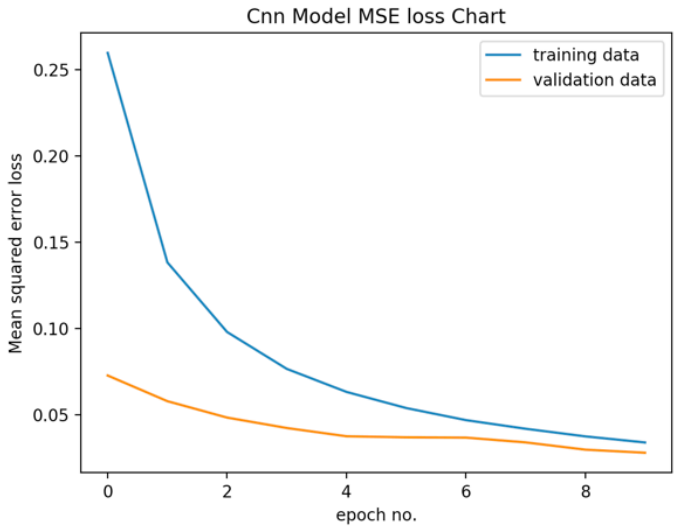


Figure 5.6 Model 3 mean squared error graph

It took approximately 20 minutes, 2.5 hours and 20 hours to train models 1, 2 and 3 respectively on a 2.3GHz dual-core Intel Core i5 processor. It can be seen that Model 3 produces the least validation mean squared error (below 0.03) and thus would produce the best steering predictions.

5.2.3 Perception Layer Tests and Results

Testing image recording on Raspberry Pi and transmission to user computer

This test was designed to verify that multiple images could be captured with the Raspberry Pi camera and also to ensure that images can be transmitted in real-time from the Raspberry Pi to a user's computer. In this test, I wrote a python script to take a continuous stream of image frames and display these frames on the Raspberry Pi. The script also transmits these images from the Raspberry Pi to a laptop via a socket connection.

The following behaviour was expected:

- Image frames display and update on screen of Raspberry Pi with negligible latency
- Image frames successfully transmit to test laptop
- Transmitted image frames display on laptop screen

Results

After conducting the test all expected behaviours was exhibited. However, images displayed on the laptop screen updated with a considerable latency. This is due to the time it takes to transmit the image over the wireless network. This latency is not a problem since the images displayed on a user's computer are just for visualization.

5.3 System Testing

System testing refers to tests carried out on the complete prototype. They test and evaluate overall performance and discover potential component integration problems. Three primary system tests were conducted on RollE: an inter module communication test, data collection testing and autonomous driving testing.

5.3.1 Testing MQTT communication between all modules

This test was designed to ensure that all modules could subscribe and publish to desired topics. In this test, I powered on the RollE Rover, connected its remote controller to the test laptop and started the Pilot_transmitter.py program. I moved the throttle and steering joysticks to various positions. This was to verify that various software units in the Control Layer could communicate with each other via MQTT to control the Physical layer. To help debug possible errors, I opened a terminal session and subscribed to the topic the remote controller was publishing on.
The following behaviour was expected:

- A stream of throttle and steering values should be displayed from the terminal session subscribed to Pilot's control topic
- RollE was supposed to move around as the steering and throttle joysticks were moved.

Results

During the test all expected results were exhibited by the system.

I also verified that the Learning layer could communicate with the Control layer by running the Autopilot.py program which predicted steering angles and transmitted it to Actuation.py in the Control layer via MQTT. I logged out predicted values from Autopilot.py on screen.

The following behaviour was expected:

- RollE should continually update its steering position to the values predicted by Autopilot.py

Results

During the test the system functioned as expected and exhibited the behaviour described above.

5.3.2 Data collection test

This test was conducted to ensure that the entire pipeline of data collection processes functioned optimally. The test verified that images were correctly captured and stored on-board the Raspberry with a timestamp. It also verified that a csv file with image paths labelled with steering and throttle values was created at the end of a data collection run. In this test, I drove RollE in data collection mode on a tiled path bordered on both sides by lawns.

The following behaviour was expected at the end of the run:

- Data/IMG folder created on the Raspberry Pi with multiple images stored with timestamps as filenames

- The Data folder should also contain a CSV file with three fields: a filename column, steering value column and a throttle value column

Results

After conducting this test, the expected behaviour was achieved. A total of 3144 images were collected and stored on the Raspberry Pi. A CSV file displaying the steering and throttle values and filename of corresponding images was created.

data/IMG/2018-04-07 02.35.09.851221.png	-0.01	-0.94
data/IMG/2018-04-07 02.35.09.886229.png	-0.01	-0.94
data/IMG/2018-04-07 02.35.09.922546.png	-0.01	-0.93
data/IMG/2018-04-07 02.35.10.030299.png	-0.01	-0.93
data/IMG/2018-04-07 02.35.10.119079.png	-0.01	-0.97
data/IMG/2018-04-07 02.35.10.199482.png	-0.01	-0.97
data/IMG/2018-04-07 02.35.10.223215.png	-0.01	-0.97

Figure 5.7 Snapshot of data collection CSV file

5.3.3 Autonomous driving test

This test was conducted to ensure that once a model has been trained RollE can successfully drive itself on the tracks similar to what it was trained on. In this test, I trained a convolutional neural network using the data collected in the Data collection test. The model had the following hyper-parameters:

- Number of Epochs = 10
- Number of samples per Epoch = 20000
- Learning Rate = 1.0e-4

The validation loss of the trained model was 0.028 which was very good. After training I set RollE on the tiled path and ran the Autopilot program.

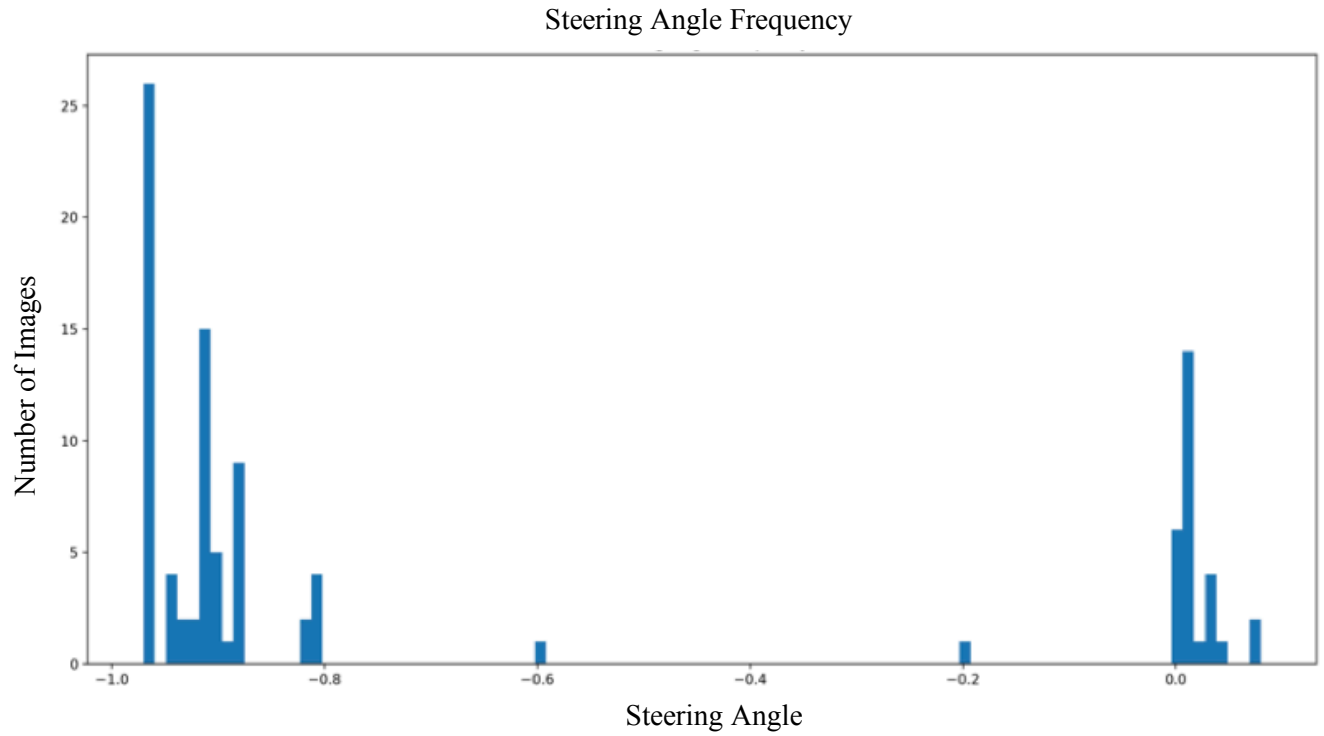
The following behaviour was expected:

- RollE should successfully drive itself in a fashion that shows intelligence and intentionality. For instance, RollE should avoid the lawns and stick to the tiled path.

Results and Observations

During the testing RollE exhibited intelligence and intentionality in its driving. It successfully made corrective turns to avoid the lawns and stick to the tiled path. RollE seemed to favour driving towards the left and after a couple of meters, it completely swayed to the left and ended up on the lawn. This behaviour was because of the distribution of the training data; while training I favoured steering towards the left. The horizontal flipping data augmentation operation helped balance out this data. However, the magnitude of examples with steering towards the left forced this behaviour to be incorporated in the model.

The graphs below show the distribution of steering values for a random sample of 100 training examples. It can be seen that, most of my steering was either towards the left (-1) or in the resting position (0) in the original data. After performing data augmentation, the data distribution became more balanced. However, it is still clear that left steering values would be favoured.



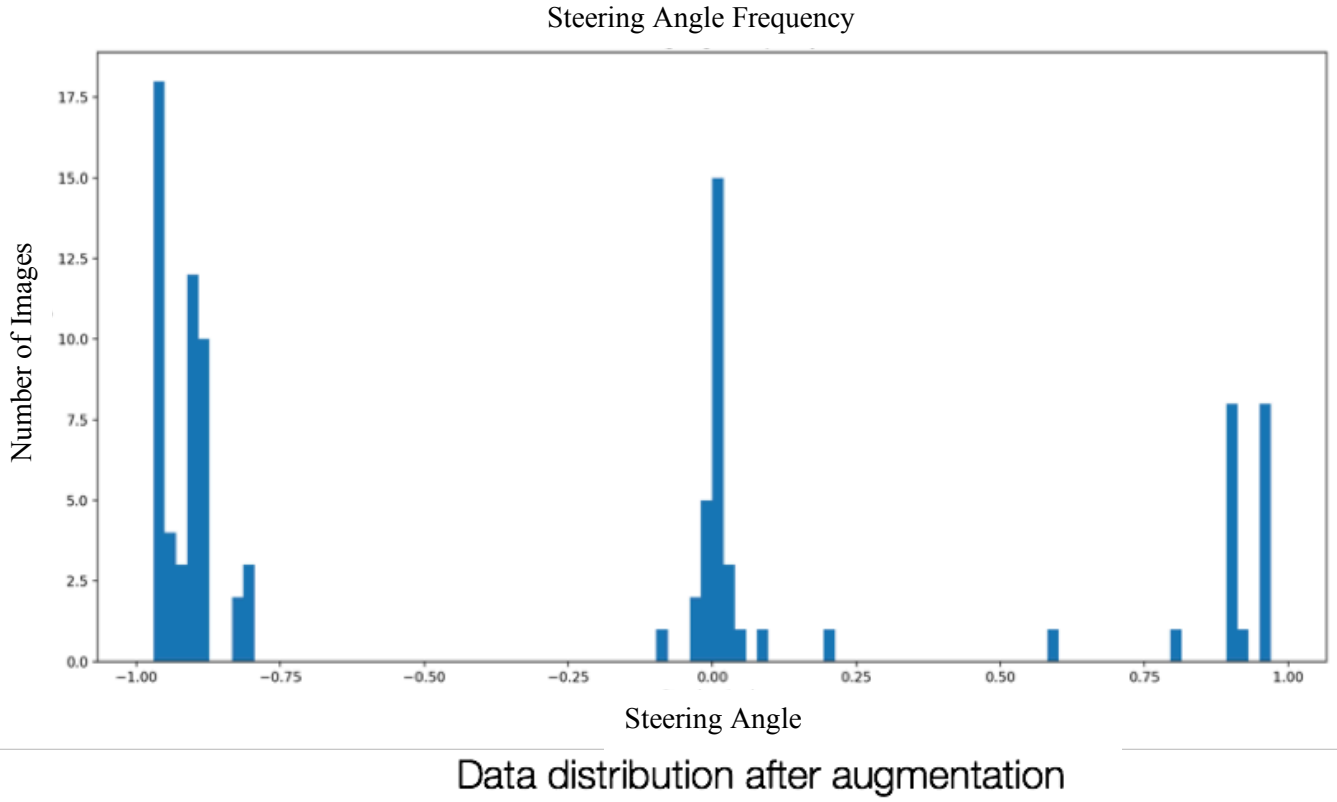


Figure 5.8 Data distribution charts

Chapter 6: Conclusion and Recommendations

This chapter presents a summary of this applied project. It reiterates the goal of the project and provides a summary of what was implemented. This chapter also discusses limitations and challenges faced during implementation and states some recommendations for future work.

6.1 Summary

Since its inception in the DARPA Grand Challenge in 2004, steady progress has been made towards developing truly self-driving cars. This technology has the potential of saving over a million lives lost to preventable road accidents world-wide. Road fatalities in Africa is the highest in the entire world and as such we would benefit immensely from this technology. However, financial constraints prevent viable experimentation and research into self-driving technology in Africa. The goal of this applied project was to help bridge the gap by developing an affordable self-driving development platform.

This project lead to the design and implementation of RollE MKII. RollE is an affordable modular self-driving development platform aimed at providing students and researchers with an autonomous vehicle to develop self-driving technology. RollE is a complete ecosystem of interacting software and hardware units that together create an invaluable self-driving research tool.

6.2 Limitations and Challenges

A couple of challenges were faced while developing and testing the RollE system. The most important of these limitations and challenges is due to the end-to-end behavioural cloning approach used to develop RollE's autonomous behaviour. In behavioural cloning, the machine learning model learns to drive solely from examples provided by a human agent. As such the human agent needs to have considerable skill in driving RollE, in order to produce fairly

consistent data. Also, the smoothness of RollE's driving would depend on the skill of the human agent.

6.3 Future Work

This project accomplished its goal of implementing a complete development platform. The system comes with a stock end-to-end machine learning model for autonomous driving. While this implementation works quite well, further work can be done using the tools provided by RollE to develop and test other machine learning techniques. RollE would also benefit from the following additions to its hardware components:

- a GPS module for precise localization
- a Light Detection and Ranging(LIDAR) sensor for advanced perception

References

Arkin, R. C., & Mackenzie, D. C. (1994). Planning to Behave: A Hybrid Deliberative/Reactive Robot Control Architecture for Mobile Manipulation. Atlanta, G: Georgia Institute of Technology.

Domingos, P. (2006, October). A Few Useful Things to Know about Machine Learning. Communications of the ACM, 55 (10), 78-87 . doi:10.1145/2347736.2347755

Elmenreich, W. (2002). An Introduction to Sensor Fusion. Austria: Institut fu¨r Technische Informatik Vienna University of Technology.

Gat, E. (1992). Integrating Planning and Reacting in a Heterogeneous Asynchronous Architecture for Controlling Real-World Mobile Robots. Tenth national conference on Artificial intelligence, (pp. 809-815). San Jose, California.

Kerry, C. F., & Karsten, J. (2017). Gauging investment in self-driving cars. Brookings.

Thorpe, C., Clatz, O., Duggins, D., MacLachlan, R., Miller, R. J., Mertz, C., . . . Yata, T. (2001). Dependable Perception for Robots. International Advanced Robotics Programme. IEEE.

Thrun Sebastian, M. M. (2006). Stanley: The Robot that Won the DARPA Grand Challenge. Journal of Field Robotics, 23.

World Health Organization. (2015). GLOBAL STATUS REPORT ON ROAD SAFETY 2015. Geneva, Switzerland: World Health Organization.