Course No.: ME-GY 9966

Convolutional Neural Network-Based Autonomous Mobile Robot for Industrial Material Transfer

M. S. Project Report Submitted to the Department of Mechanical and Aerospace Engineering, Tandon School of Engineering, New York University, Brooklyn, NY

Submitted By
Mitra Varun Anand
N19790097

Abstract

This report presents the development of a mobile robotic system that integrates image processing, convolutional neural networks, and behavioral cloning with a computer in order to assist the transfer of materials from one location to another in an industrial environment. This project utilizes a RC vehicle platform fitted with a wide-angle camera module, an ultrasonic sensor, servo controller, and microcontroller, which wirelessly connects to a computer. Human users choose the object to be moved by the robot for the human through a computer interface, and the device can drive the object to the preset destination according to traffic signs that it detects along the way by making navigational choices. Image processing through the neural networks enables the microcontroller to make navigation decisions to the built platform through the servo controller. Additionally, the developed module will train the model to drive itself in a static environment to perform repetitive tasks, which will enable industries to increase productivity through automation in closing shifts. The developed prototype successfully performed self-navigation, image recognition of various traffic signs in the environment, and eventually the trained neural network performed autonomous navigation based on the commands provided by the user. An evaluation scheme, which was proposed to test the usability of the system, found that the developed product was user-friendly and neural networks were reliable.

Keywords: Autonomous mobile robot, convolutional neural network, autonomous navigation, machine learning, object recognition,traffic sign recognition, image processing, assistant robot, industrial mobile robot, tensorflow, behavioral cloning, user interface, motor control, artificial intelligence

Index:

1. Introduction

In this age of iterative advancements in technology, many industries continue to use user-controlled devices to transport objects from one part of the factory to another; this is an avoidable waste of time, which can be used to achieve a more efficient workflow and use only manpower for complex operations. Even industries that employ a robot to do this task do not heavily rely on it, since the industry's environment is continually changing [1]. This results in the need to program the robot to incorporate these changes, or to reduce the usage of such a robot, making it obsolete. This process needs a much smarter robot wherever possible, which forms the objective of this project. This smart mobile robot can navigate using six different number of traffic signs that will be strategically placed in the factory environment, and it uses these signs to make smart navigational choices, which is possible due to recent advancements in machine learning and artificial intelligence.

Nowadays, most of the robots work by following commands that humans set in the machine or use a pre-saved map of the environment that becomes unnecessary when the layout of factory operations constantly changes. Most industrial robotic systems also lack simple user interfaces. In addition, some of the robots need constant supervision, but the real condition is that workers cannot spend a lot of time to look after the robot while doing his/her own work, so they want a robot that can finish some tasks by itself [1]. The question is, how can the transporting of an object by a robot be made easier for dynamic environments? Hence, the operation of a robot needs to be that every time there is a change in the way a factory is set up, only minimal human intervention is needed to continue using the robot.

However, such operation methods are still not enormously available, and they are often too complex to be used by workers who have limited knowledge of these processes. Smart robots should combine a few important functions that make them convenient to use. The Convolutional Neural Network based robot will utilize a Raspberry Pi wide-angle lens camera to look for signs in the environment and to make decisions for navigation from one point to another. With the popularity and ease of use of computers, an implementation of such a process will lead to an increase in the usage of such robots. An added feature of this project is the use of behavioral cloning module, in which a user can manually drive the vehicle around the factory to train a convolutional neural network [2] to be able to drive the robot using the trained network in a static environment. In the project, the system consists of two main parts: one is a mobile base platform, and another is a micro-controller that controls how the base is going to move. Once the object is loaded and the destination is selected, the mobile base can drive the object to the drop-off point by using the traffic signs placed within the workspace of the industry; this is accomplished through image processing using the images frame taken by the camera. This report is organized in the following ways: Section 2 talks about the current industry leading robots and

their limitations; Section 3 presents the system components and introduces how the system works; Section 4 presents the experiment and results for system evaluation; Section 5 presents acknowledgment; and Section 6 references for this project.

2. Related Works

There are a wide range of mobile robots currently operating in a factory setting that are used in a variety of applications, including but not limited to payload operations, pick and place, sorting and security. In terms of the sensors used and the purpose, there are two robots that closely match this project. But none of them use deep neural networks for navigation, relying instead on LiDAR and computer vision.

2.1. OTTO 15OO

OTTO 15OO [3] navigates spaces just like a person does. It maintains a map of the space in its memory and uses visual reference points to always know its position. No guides or infrastructure are required.

2.1.1. Main Components

1. Safety Rated Lidar:
Drives around obstacles and find new routes without getting stuck.
2. 360º Indicator Lighting:
Displays familiar turn signals, brake lights, vehicle status lighting, and audible tones so you know where OTTO is going.
3. Configurable Payload Interface:
Carries payloads using a standard OTTO attachment, or has the flexibility to create your own using fixed mounting points and standard power and communication ports.
OTTO 15OO[3] can be loaded and unloaded manually or can use a standard attachment to automate the pick-up and delivery process.

2.2. MiR200

The MiR200 [4] is a safe, cost-effective mobile robot that quickly automates your internal transportation and logistics. The robot optimizes workflows, freeing staff resources so you can increase productivity and reduce costs.
MiR200 can be used in nearly any situation where employees are spending time pushing carts or making deliveries. These tasks can be automated, so employees can focus on higher value activities.

2.2.1. Sensors

1.SICK laser scanners S300 (front and back):
360° visual protection around robot
2. 3D camera Intel RealSense:
Detection of objects ahead 50-500 mm above floor
3. Ultrasonic scanners :
Detection of transparent objects ahead, e.g. glass doors

With built-in sensors and cameras and sophisticated software, the MiR200 can identify its surroundings and take the most efficient route to its destination, safely avoiding obstacles and people. Without the need to alter your facility with expensive, inflexible wires or sensors, the robot offers a fast return on investment, with payback in as little as a year[4].

3. Development of the System

3.1 The Overall system

The goal of project is to design a robotic system that contains a mobile robot base and a micro controller. This system can be operated remotely through a computer once the object for transportation is placed on it and can (1) complete autonomous navigation of an unknown map, (2) recognize traffic signs based on convolutional neural networks, and (3) reach the preset destination where the object is manually unloaded. The system consists of a computer, a raspberry pi 3 model-B, a RC racing truck as the build platform, a Sainsmart wide-angle camera module, a Sunfounder servo control board, a Parallax ultrasonic sensor, an Anker 10000mAH battery pack, and a 3D printed assembly to complete the set up. Figure 1 shows the overall robotic system.

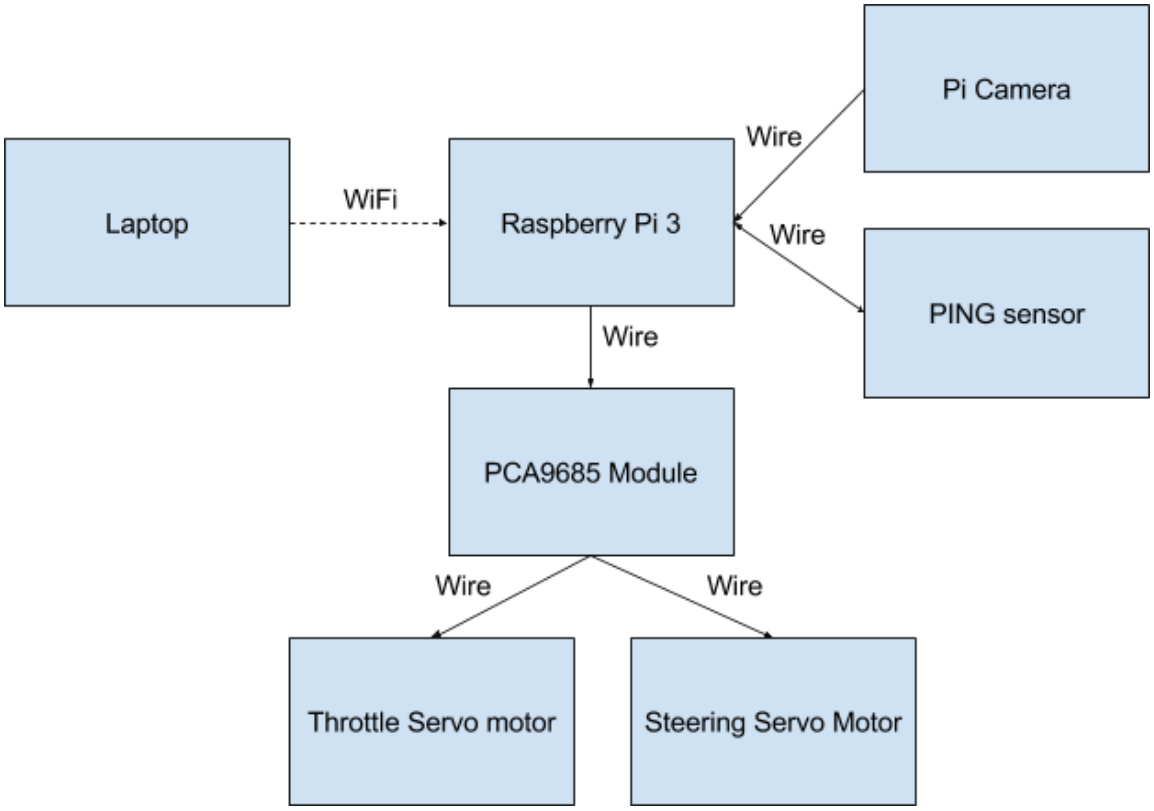


Figure 1. Components of the System

An open source design [4] from thingiverse was used to 3D print the Raspberry Pi camera holder assembly. The following figure shows the entire labelled assembly:

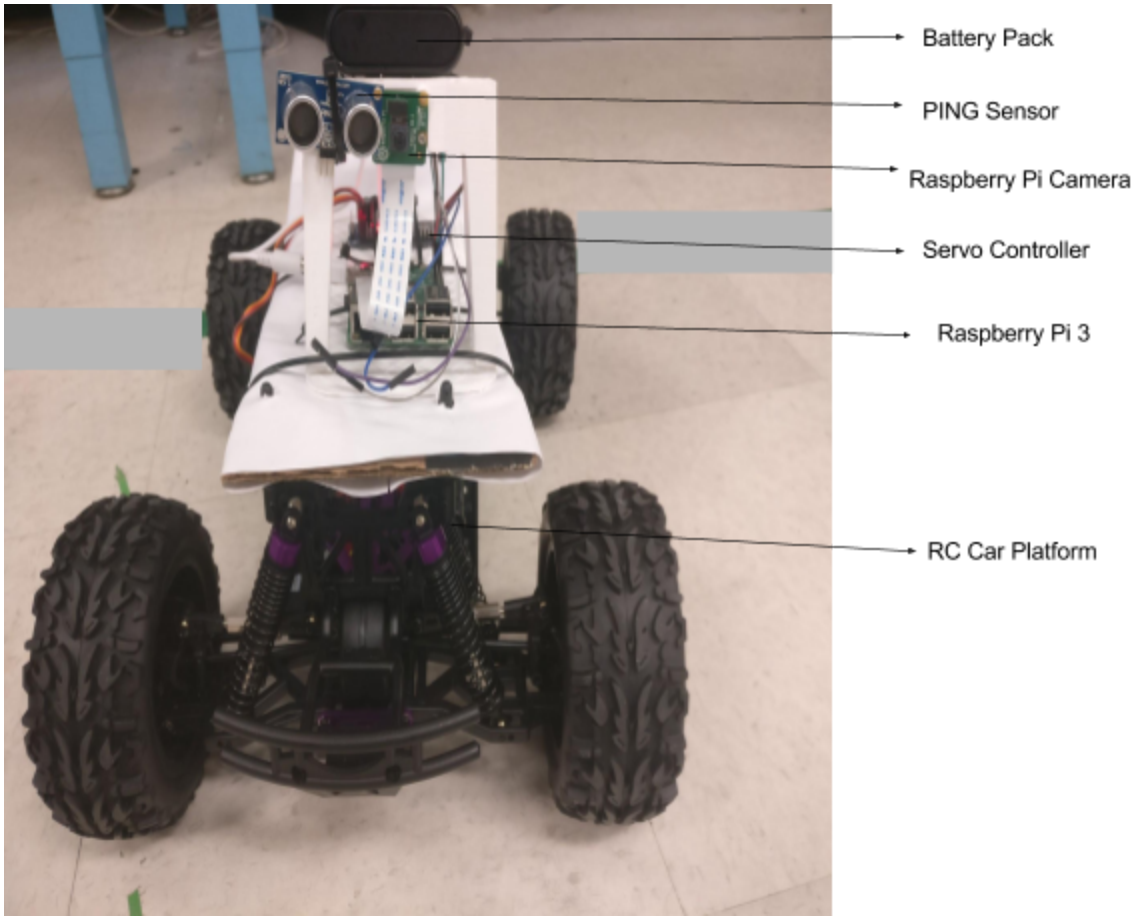


Figure 2. Robot Assembly

Figure 3. Front View of the Robot

Figure 4: Side View of the Robot

The following picture shows the alignment of the PING sensor with respect to x and y axes.

Figure 5: Alignment of the PING Sensor

3.2. Convolutional Neural Networks

The project utilizes two separate training of convolutional neural networks for each of the navigation and behavioral cloning modules. The following sections discuss how each of these networks are trained.

3.2.1. CNN for Traffic Sign Recognition

There are various aspects to consider when thinking about this problem:

- Neural network architecture
- Play around preprocessing techniques (normalization, RGB to grayscale, etc.)
- Number of examples per label (some have more than others)
- Generate fake data

The dataset preprocessing consisted of converting to grayscale - This worked well for Sermanet and LeCun [5] as described in their traffic sign classification article. It also helps to reduce training time, which was nice when a GPU wasn't available.

3.2.1.1. Data Augmentation:

Data augmentation is the single best method to increase accuracy of the model. Because several classes in the data have far fewer samples than others the model will tend to be biased toward those classes with more samples. Augmentation was done by creating copies of each sample for a class (sometimes several copies) in order to boost the number of samples for the class to 4 (since the class already had at least 800 samples). Each copy is fed into a "jitter" pipeline that randomly translates, scales, warps, and brightness adjusts the image. This was by far the most laborious part of the project, and it takes quite some time (more than 6 hours on a virtual machine through Amazon Web Services EC2 instance) to run the code.
The SciKit Learn train_test_split function was used to create a validation set out of the training set. 20% of the testing set was used to create the validation set.

3.2.1.2. Neural Network Architecture:

The modified LeNet architecture has been adapted from Sermanet/LeCunn traffic sign classification journal article [5] as shown below.

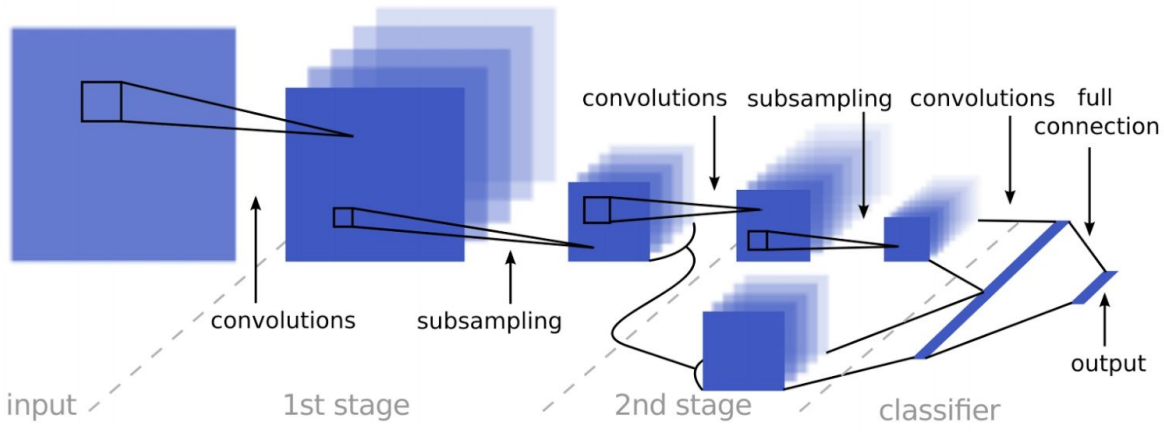


Figure 6: Modified LeNet Architecture [10]

The same architecture was implemented from the LeNet Lab [10], with no changes since the dataset is in grayscale. This model worked quite well to begin with (~96% validation accuracy), but implementation of the Sermanet/LeCun model from their traffic sign classifier paper and gave an immediate improvement. Although the paper doesn't go into detail describing exactly

how the model is implemented (particularly the depth of the layers) it was possible to make it work. The layers that used to train the model are set up like this:

- 5x5 convolution (32x32x1 in, 28x28x6 out)
- ReLU
- 2x2 max pool (28x28x6 in, 14x14x6 out)
- 5x5 convolution (14x14x6 in, 10x10x16 out)
- ReLU
- Flatten layers from numbers 8 (1x1x400 -> 400) and 6 (5x5x16 -> 400)
- Fully Connected Layer 2:Input = 145. Output = 84.
- Fully Connected Layer 3:Input = 84. Output = 4.

Test Set Accuracy = 0.927

3.2.1.3. Training the Model:

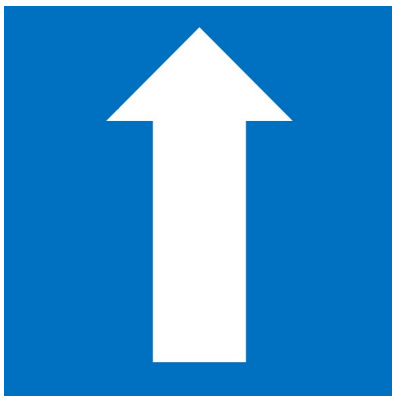
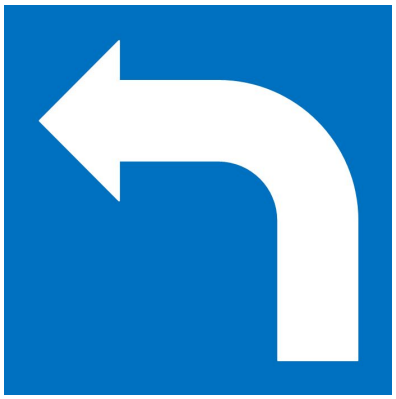
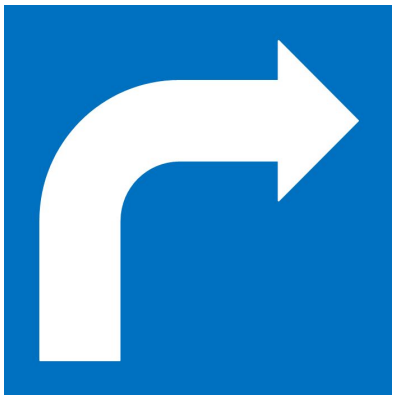
Adam optimizer was used to train the model [6] . The final settings used were:

- batch size: 100
- epochs: 60
- learning rate: 0.0009
- mu: 0
- sigma: 0.1
- dropout keep probability: 0.5

3.2.1.4. Test a Model on New Images:

Some of the images taken by me appear to be more easily distinguishable than quite a few images from the original dataset. I noticed that the images tend to be quite a bit brighter and might occupy a different range in the color space, possibly a range that the model was not trained on. In addition, the GTSRB dataset [13] states that the images "contain a border of 10 % around the actual traffic sign (at least 5 pixels) to allow for edge-based approaches" and the images that I used do not all include such a border. This could be another source of confusion for the model.
The model appears to have predicted the new signs perfectly for the four images to be used for testing in real time, with 100% accuracy. This is a good sign that the model performs well on real-world data. And while it's reasonable to assume that the accuracy would not remain so high given more data points, judging by the low fidelity of a number of images in the training dataset it's also reasonable to assume that if the real-world data were all as easily distinguishable as the four images chosen that the accuracy would remain very high, as shown below:

Input	Prediction	Probability	Action

		0.72	Go Straight
		0.82	Take Left
		0.77	Turn Right
		0.98	Stop

A video of the trained CNN used to detect different signs in real time is linked below.

Video Link: https://www.youtube.com/watch?v=5GonZgJXca0&feature=youtu.be

3.2.2. CNN for Behavioral Cloning

The objective of this project is to apply deep learning principles to effectively teach the robot to drive autonomously in a static environment to perform repetitive functions. The robot is trained by driving either manually or using hardcoding to collect input images that record the features in the environment. In training mode, user-generated driving data is collected in the form of simulated robot dashboard camera images and control data (steering angle, throttle, brake, speed). Using the Keras deep learning framework, a convolutional neural network (CNN) model is produced using the collected driving data and saved as model.json. Using the saved model, drive.py starts up a local server to control the robot in autonomous mode. The model weights are retrieved using the same name but with the extension .h5.

The challenge of this project involved not only developing a CNN model able to drive the robot around the test track without leaving the track boundary, but also feeding training data to the CNN in a way that allows the model to generalize well enough to drive in an environment it has not yet encountered,i.e with minor changes in the same environment.

3.2.2.1. Base Model and Adjustments

The diagram below is a depiction of the nVidia model architecture [6] which I have made use of, using a few modifications that are explained in the coming topics.

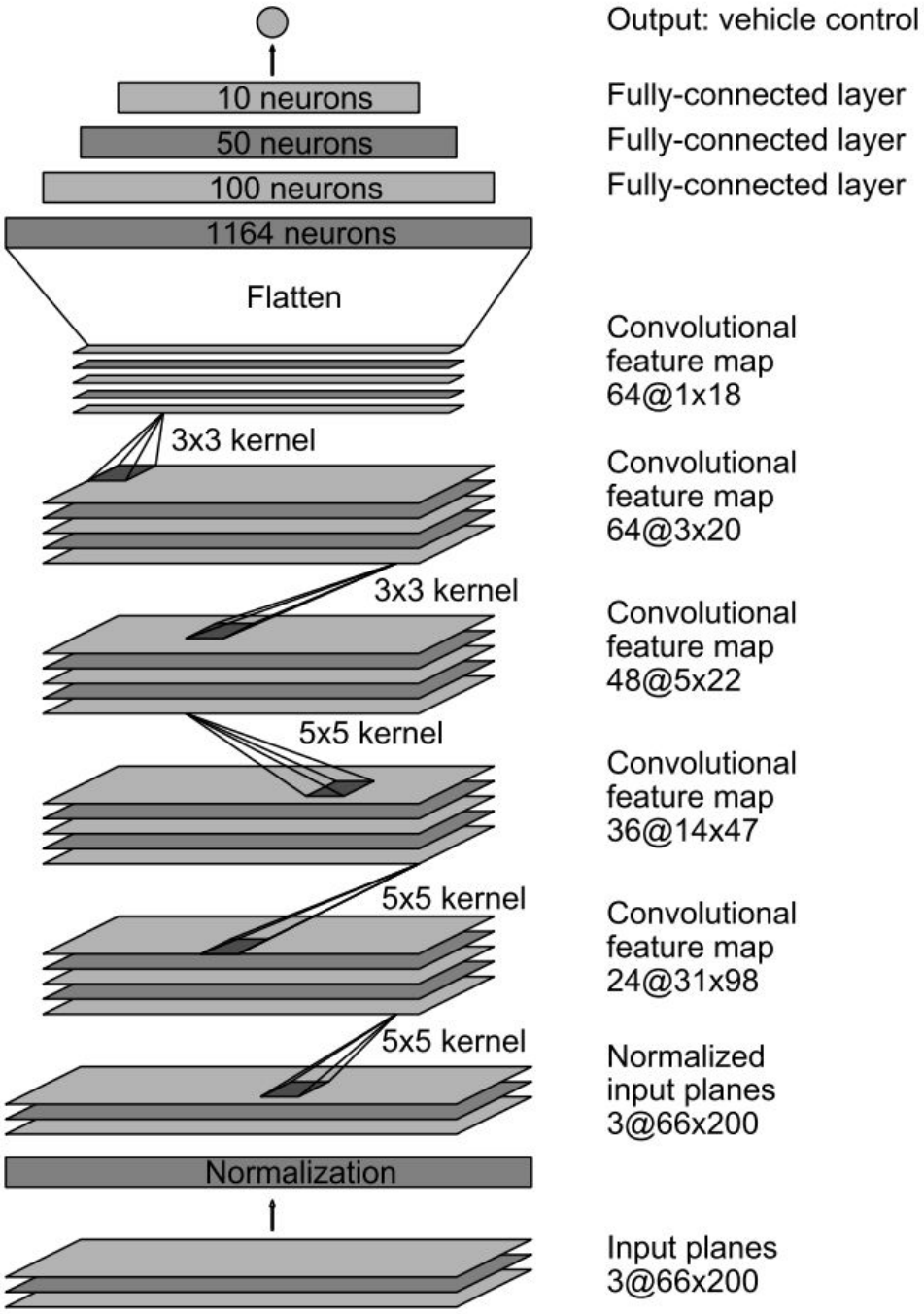


Figure 7: NVIDIA Architecture [11]

First I reproduced this model as depicted in the image - including image normalization using a Keras Lambda function, with three 5x5 convolution layers, two 3x3 convolution layers, and three fully-connected layers - and as described in the paper text - including converting from RGB to YUV color space, and 2x2 striding on the 5x5 convolutional layers. The paper [6] does not mention any sort of activation function or means of mitigating overfitting, so tanh activation functions was used on each fully-connected layer, and dropout (with a keep probability of 0.5) between the two sets of convolutional layers and after the first fully-connected layer. The Adam optimizer was chosen with default parameters and the chosen loss function was mean squared error (MSE). The final layer is a fully-connected layer with a single neuron.

3.2.2.2. Loading and Preprocessing

In training mode, the robot produces one images per frame while recording corresponding to center-mounted camera. This produces a csv file which includes file paths for each of these images for each frame. The algorithm loads the file paths for the camera views for each frame, along with the angle, into two numpy arrays image_paths and angles.

Images produced by the robot in training mode are 320x160, and therefore require preprocessing prior to being fed to the CNN because it expects input images to be size 200x66. To achieve this, the bottom 20 pixels and the top 35 pixels are cropped from the image and it is then resized to 200x66. A subtle Gaussian blur is also applied and the color space is converted from RGB to YUV. Because the code uses the same CNN model to predict steering angles in real time, it requires the same image preprocessing.

3.2.2.3. Further Model Adjustments

Some other strategies implemented to combat overfitting and otherwise attempt to get the robot to drive more smoothly are:

- Removing dropout layers and adding L2 regularization (lambda of 0.001) to all model layers - convolutional and fully-connected
- Removing tanh activations on fully-connected layers and adding RELU activations to all model layers - convolutional and fully-connected
- Adjust learning rate of Adam optimizer to 0.0001 (rather than the default of 0.001)

These strategies did, indeed, result in less bouncing back and forth between the sides of the track, particularly on the test track where the model was most likely to overfit to the recovery data.

3.2.2.6. Results

These strategies resulted in a model that performed well on both test and challenge tracks. The final dataset included a total of 59,664 data points. From these, only 17,350 remained after distribution flattening, and this set was further split into a training set of 16,482 (95%) data points and a test set of 868 (5%) data points. The validation data for the model is pulled from the training set, but doesn't undergo any jitter. The model architecture is described in the paragraphs above, but reiterated in the image below:

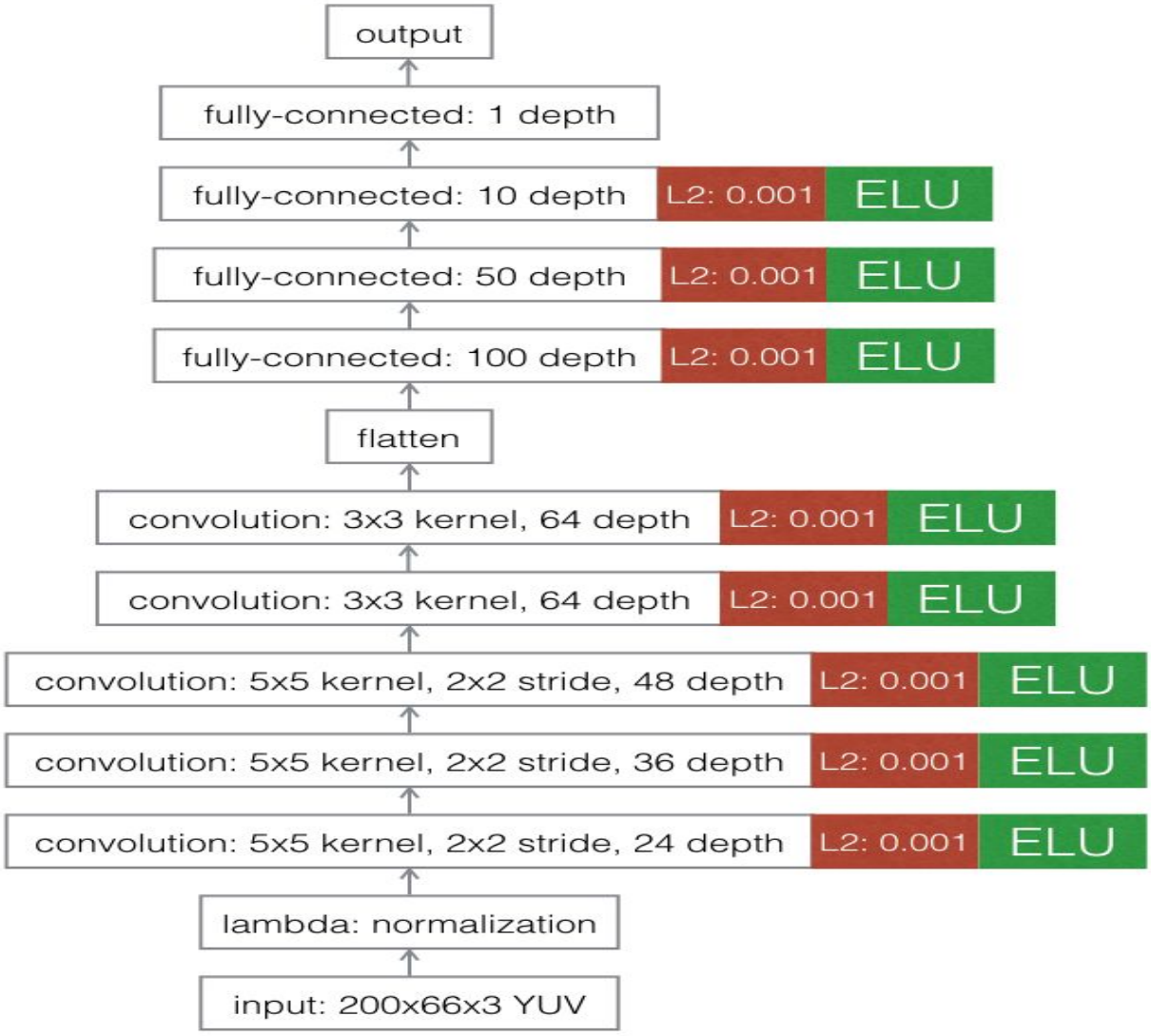


Figure 8: The Final Architecture

3.3. Packages Used:

The model depends on a number of packages that must be installed. They are:

- Pickle: Used to import and export pickled file that compresses the image database.
- Random: Used to generate random values for data augmentation.
- Os: Controls the operating system of the raspberry pi.
- Numpy: Used as an efficient multi-dimensional container of generic data to integrate with a wide variety of databases.

- TensorFlow: An open-source software library for building and training neural networks to detect and decipher patterns and correlations.
- Time: Used for giving appropriate delays in the program.
- Csv: Used to import .csv type files in the database.
- Donkey: Gives server and control access to the raspberry pi from the host computer.
- Scipy: A scientific python library used to train the neural networks efficiently.
- Adafruit_PWM: Used to control the servo motors of the robot using PWM values from the raspberry pi.

3.4. Components:

3.4.1. Raspberry Pi 3 Model B:

The Raspberry Pi 3 is a pocket size micro computer that has functions for manipulating the input and output pins.

Figure 9: The Raspberry Pi 3 B

The specifications of the model are:

- Quad Core 1.2GHz Broadcom BCM2837 64 bit CPU
- 1GB RAM
- 40-pin extended GPIO
- CSI camera port for connecting a Raspberry Pi camera
- Micro SD port for loading your operating system and storing data
- Micro USB power source up to 2.5A

3.4.2. Raspberry Pi Camera Module V2:

The Camera Module can be used to take high-definition video, as well as stills photographs. The v2 Camera Module has a Sony IMX219 8-megapixel sensor (compared to the 5-megapixel OmniVision OV5647 sensor of the original camera).

It’s a leap forward in image quality, colour fidelity, and low-light performance. It supports 1080p30, 720p60 and VGA90 video modes, as well as still capture. It attaches via a 15 cm ribbon cable to the CSI port on the Raspberry Pi. The project utilizes the camera as an input as well as a feedback device, to train the model as well as during real time testing.

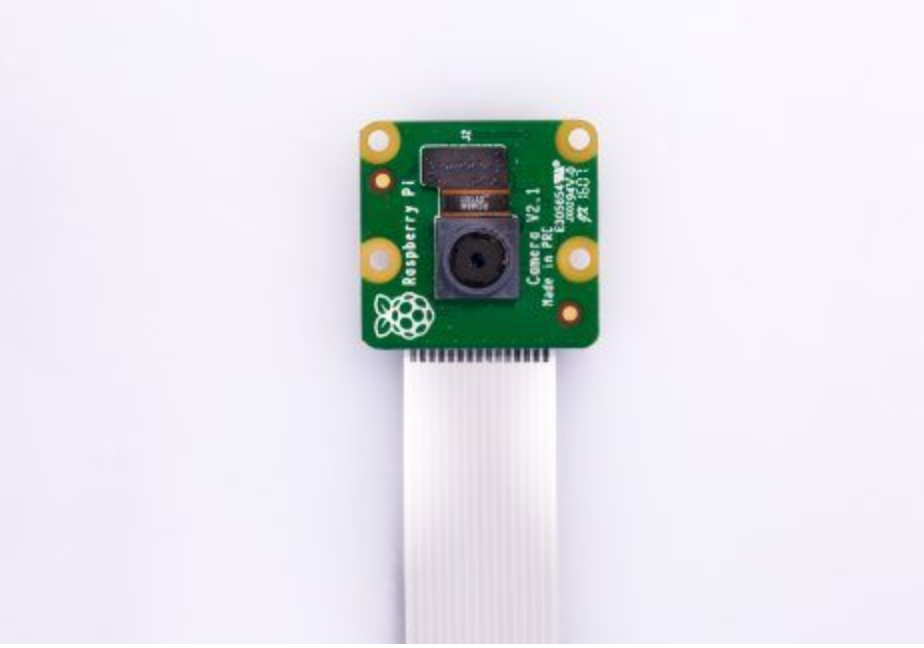


Figure 10: The Raspberry Pi Camera V2

3.4.3. Redcat Racing RC Car:

The mobile robot platform consists of the Redcat Racing RC Car that can be hacked using a simple PWM controller connected from the raspberry pi. It has :

- Electric Brushed 27T 540 Motor, Four Wheel Drive, High Torque Servo
- Polycarbonate Body, Aluminum Capped Oil Filled Shocks
- Transmission Forward and Reverse. 7.2v 2000mAh NiMh Battery

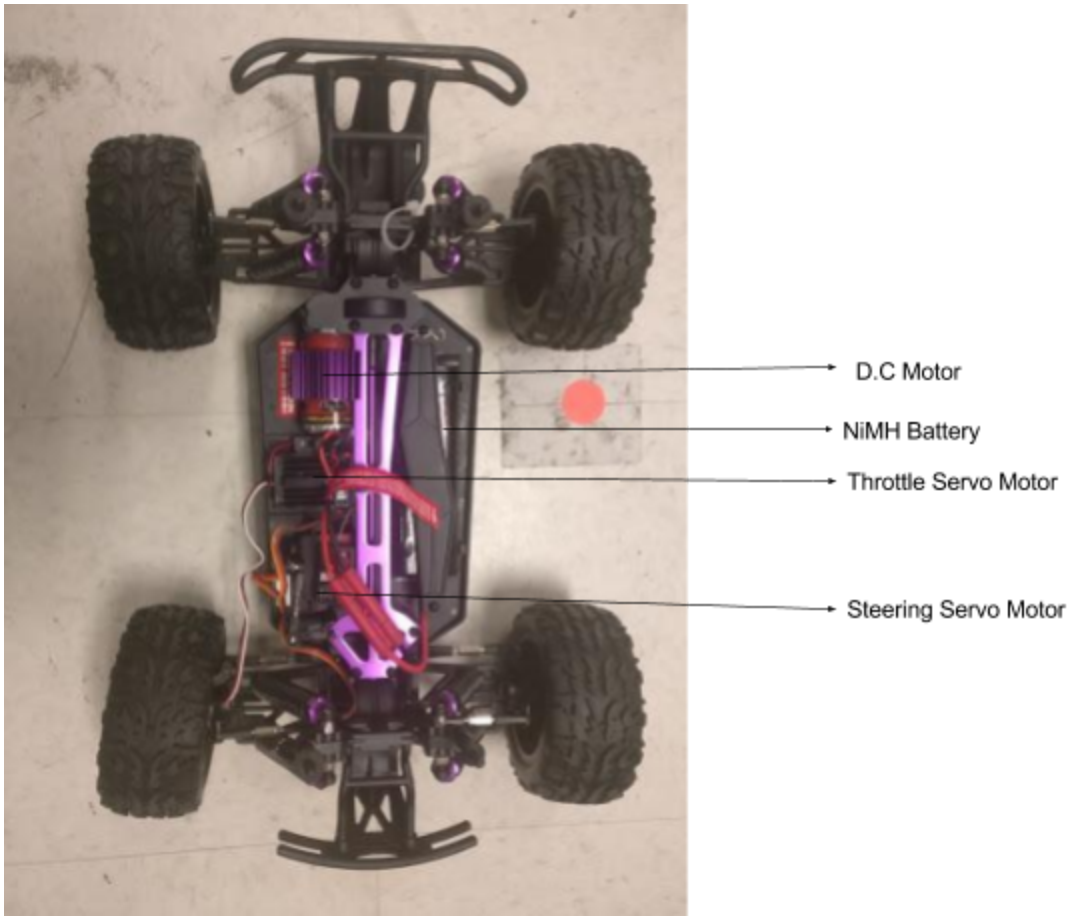


Figure 11: Redcat RC Car

This gives an opportunity to use an affordable platform as a test bed for evaluating the results of the project, at the same time, not compromising on performance.

3.4.4. Sunfounder PCA 9685 Servo Controller:

The controller communicates with the raspberry pi through an I2C interface and directly controls the servo motors on the platform through Pulse Width Modulation.

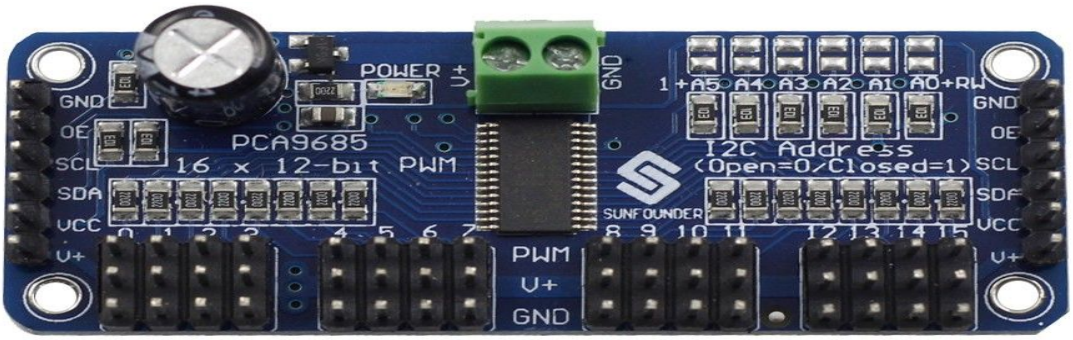


Figure 12: PCA 9685 Controller

- Contains an I2C-controlled PWM driver with a built-in clock. It means, unlike the TLC5940, you do not need to continuously send it signals tying up your microcontroller; it's completely free running.
- Support using only two pins to control 16 free-running PWM outputs.
- 12-bit resolution for each output - for servos, that means about 4us resolution at 60Hz update rate.

3.4.5. Anker Battery Pack:

The battery pack is used to give power to the raspberry pi, the camera module as well as the servo controller.

Figure 13: Anker Power Bank

It has a total capacity of 10,000mAh, making it capable of delivering the required power to the unit for more than 20 hours with a single charge.

3.4.6. Alienware 15 Laptop:

The computer is used for training the neural networks as well as for wireless communication to the raspberry pi through a SSH connection established through the command line of the computer. The specifications of the laptop are:

- CPU: 2.9 GHz Intel Core i5-4210 CPU
- Operating System: Ubuntu 16.04
- RAM: 16GB
- Hard Drive Size: 128GB SSD
- Hard Drive Type: M.2 SSD
- Display Size: 15.6
- Native Resolution: 1920x1080
- Graphics Card: Nvidia GeForce GTX 970M with 3GB GDDR5
- Video Memory: 3GB
- Wi-Fi Model: Killer 1525 802.11ac
- Bluetooth: Bluetooth 4.1

3.5. Electrical Connection:

Fritzing [12] wiring software is used to visually illustrate the circuit connections between the raspberry pi, the camera module, the servo controller and the servo motors of the platform vehicle is shown below:

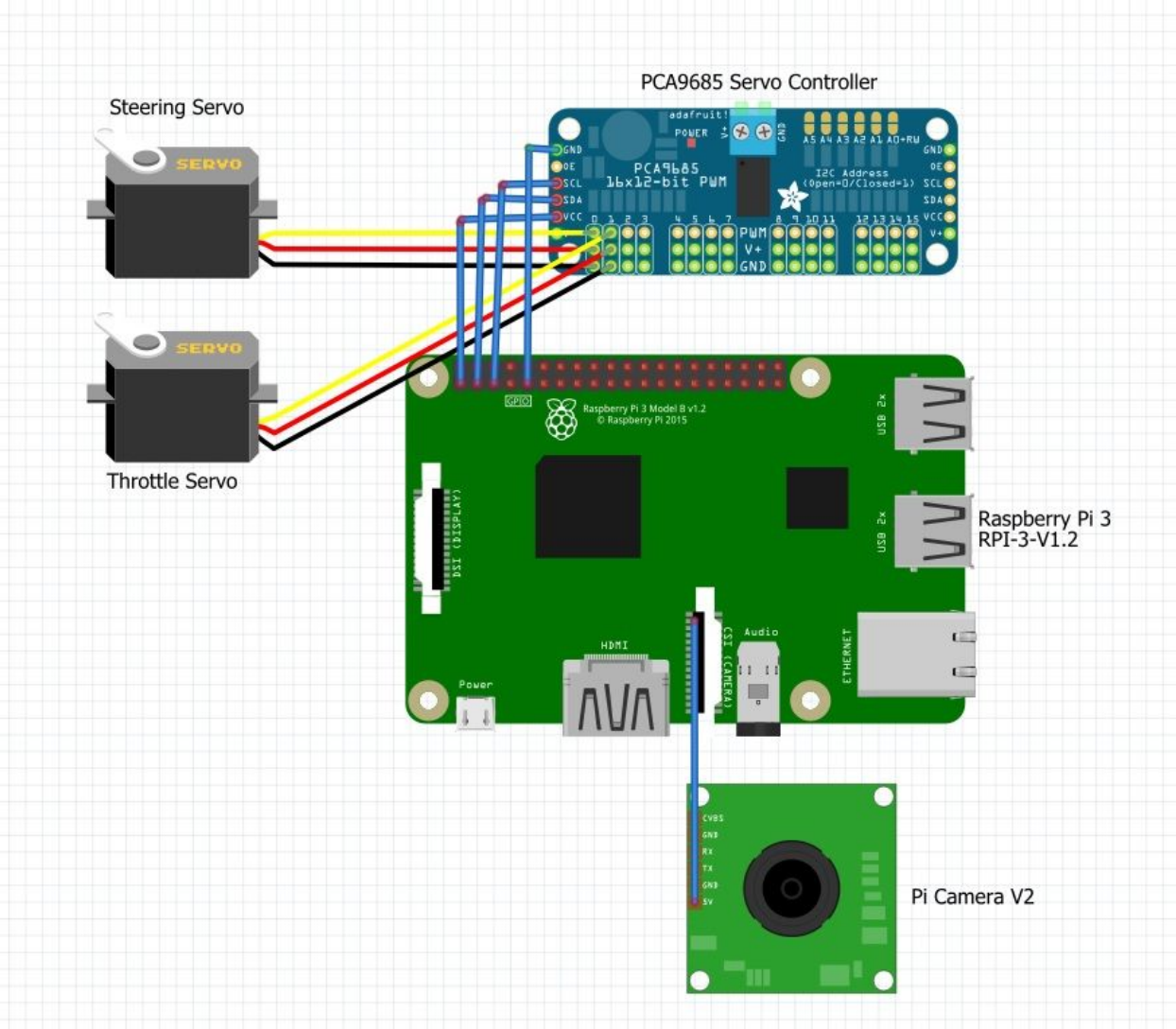


Figure 14. The Electrical Connection for Behavioral Cloning

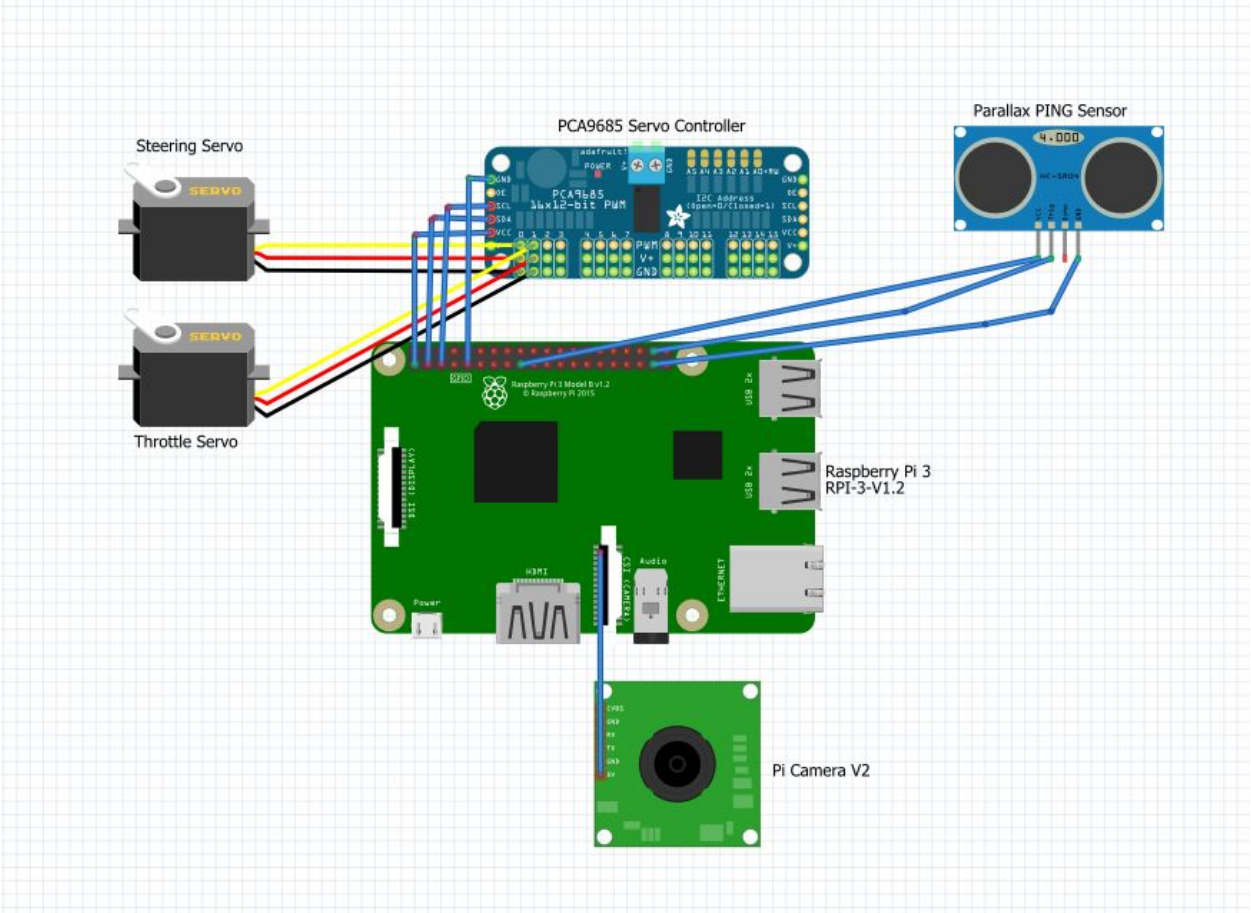


Figure 15: The Electrical Connection for Sign Based Navigation

The raspberry pi receives input images(or videos) from the camera module and distance measurements from Ping sensor. It then does the computation required and sends the required PWM values to the steering and throttle servo motors through the PCA9685 module. The various sorts of communication happening within the modules are shown below:

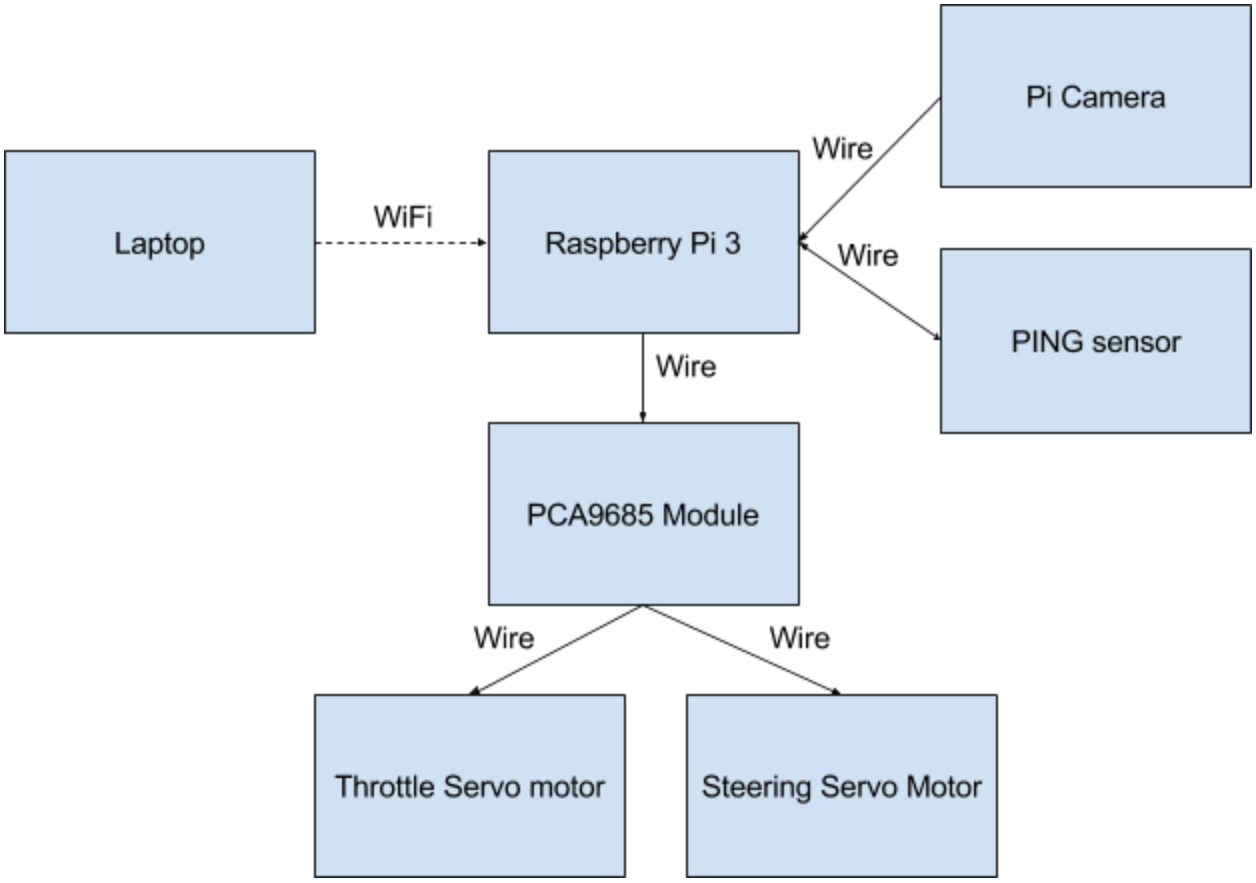


Figure 16: Schematic for Communication

4. Evaluation

4.1. Behavioral Cloning Module

A test layout was created to manually drive the robot around and capture around 15,000 images. A simple circular path was followed by the robot in order to facilitate training. The model was tested by adding a green line boundary to simulate a change in the environment to test for the accuracy of the model when there are minor changes to the environment. The test layout is shown below:

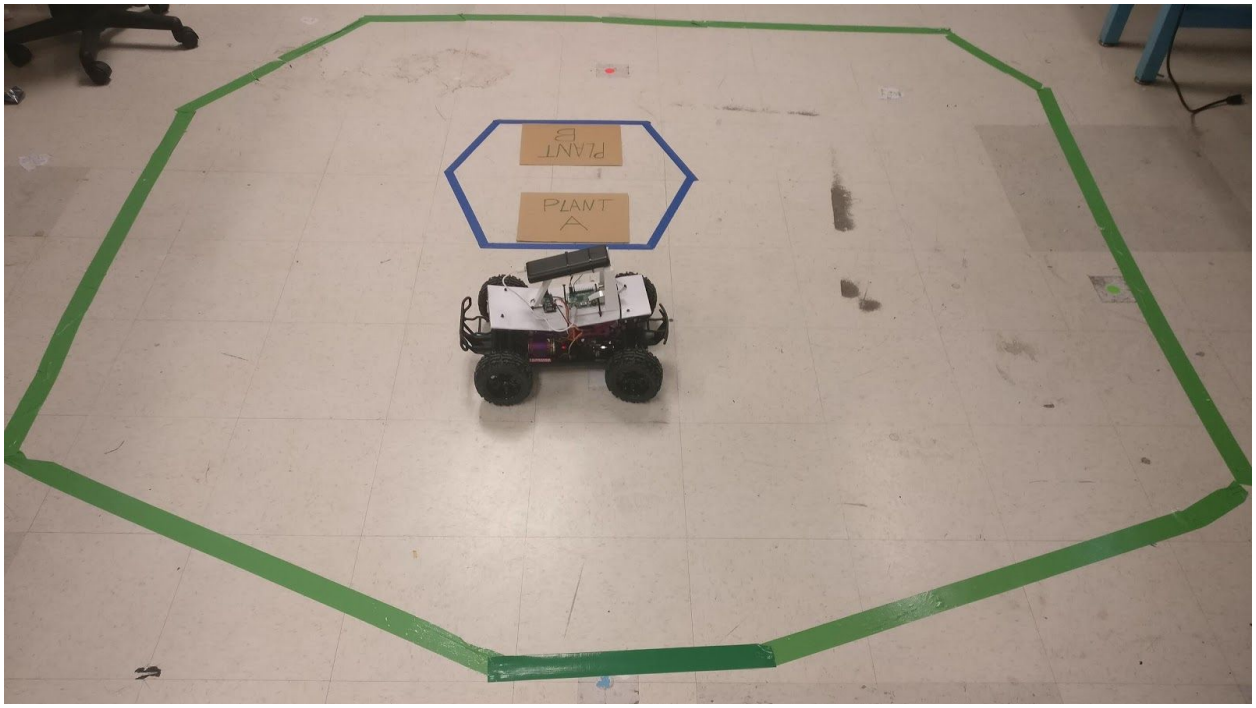


Figure 17. Testing Layout 1

The aim for evaluating the module was as follows:

- Load an object at “Plant A” (see next figure).
- Start the module and test if it can reach “Plant B” where the object is unloaded and a new object is loaded.
- The robot then brings the new object back to “Plant A.”

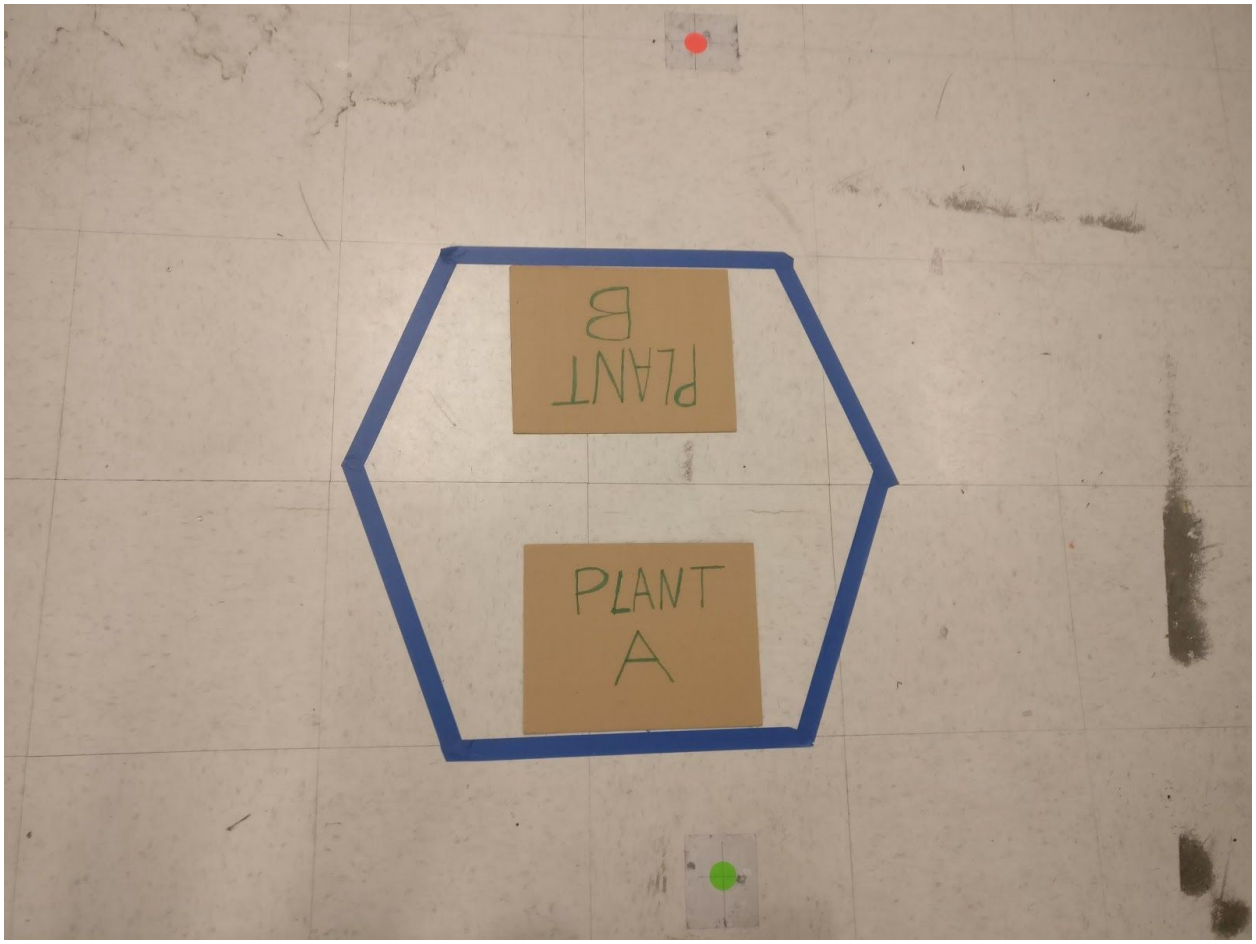


Figure 18. Training Layout 1

This loop is repeated for a sufficient number of start positions to (1) verify if it can stay within the boundary (marked by green and blue tapes) and (2) match the steering values that it was expected to give, which is evaluated in the coming sections.

A more complex track was built to train and test the same module to test for repeatability. This module was trained in a rectangular track for 20 laps, and then the layout was modified for the last 8 laps to see if the model was able to learn and make the robot drive in the new track. A diagram of how these two layouts looks is shown below.

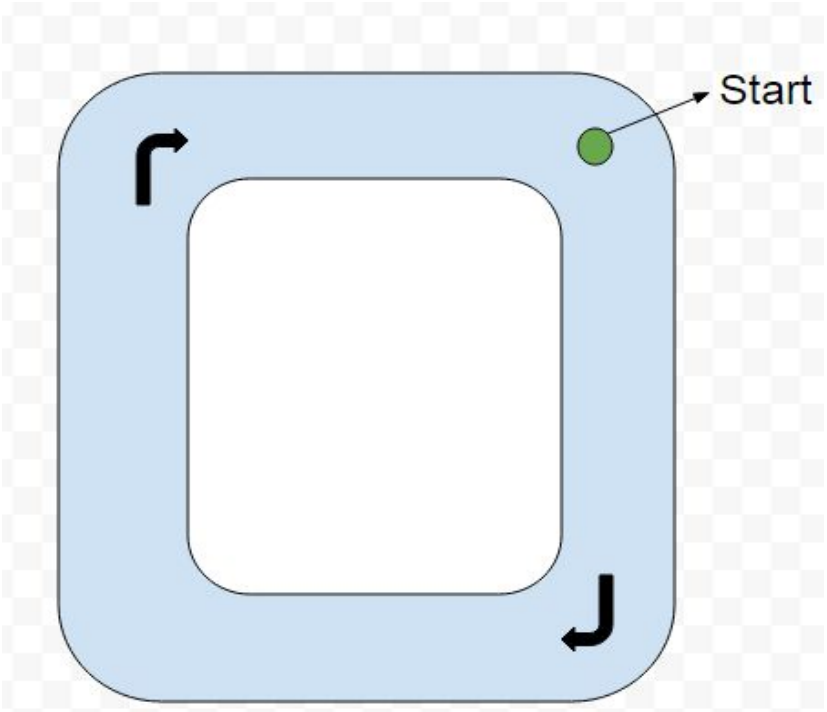


Figure 19. Training Layout 2

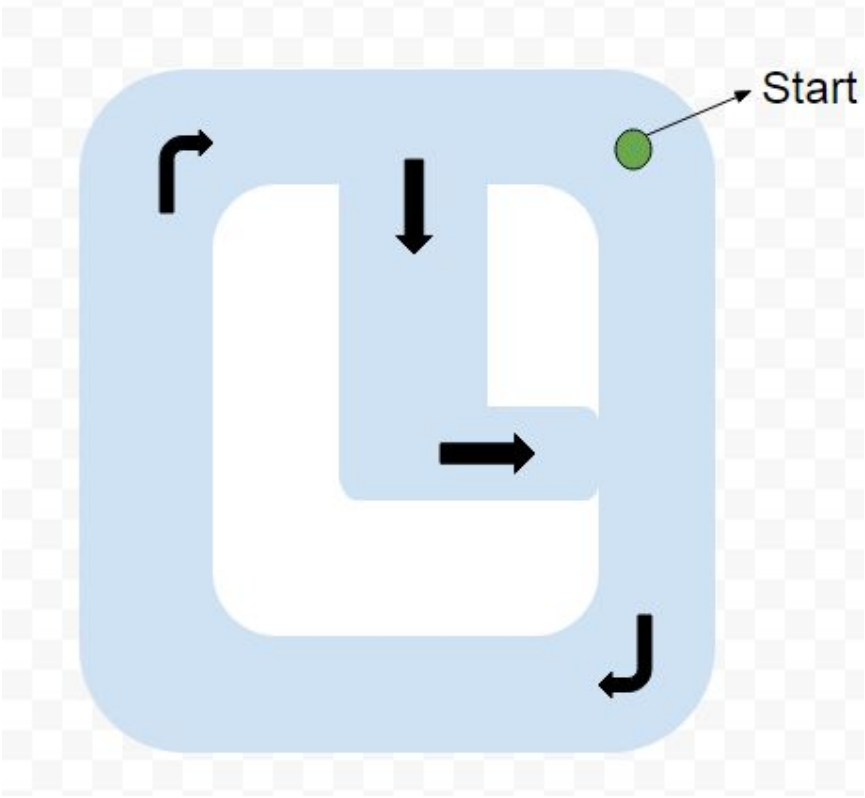


Figure 20. Testing Layout 2

The pictures taken during testing are shown below:

Figure 21. Testing Track 2- Image 1

Figure 22. Testing Track 2- Image 2

Figure 23. Testing Track 2- Image 3

4.1.1. Comparison of Steering Angles:

Since the model involved training by driving around the track in a circular path, the PWM values are expected to be constant for the steering motor. However, in a practical world, there will be a few discrepancies, owing to change in lighting conditions, minor changes in the environment, etc. The following table plots expected and actual PWM values for 20 trials:

Trail	**Expected Steering PWM Value**	**Actual Steering PWM Value**
1	550	551
2	550	550
3	550	550
4	550	550
5	550	550
6	550	550
7	550	551

8	550	551
9	550	551
10	550	550
11	550	550
12	550	550
13	550	550
14	550	550
15	550	552
16	550	551
17	550	550
18	550	550
19	550	549
20	550	550

As can be observed from the table, the model's predictions are almost equal to the expected values and varies only in the error range of 2, which is perfectly acceptable, since it does not cause drastic changes in the path of the robot.

Video Link: https://www.youtube.com/watch?v=euSkEnNLKf8&feature=youtu.be
Video Link 2: https://www.youtube.com/watch?v=0V-GQOhfABw&feature=youtu.be

For module 2, the results were different and are shown in the form of line graphs for three different batch sizes below:

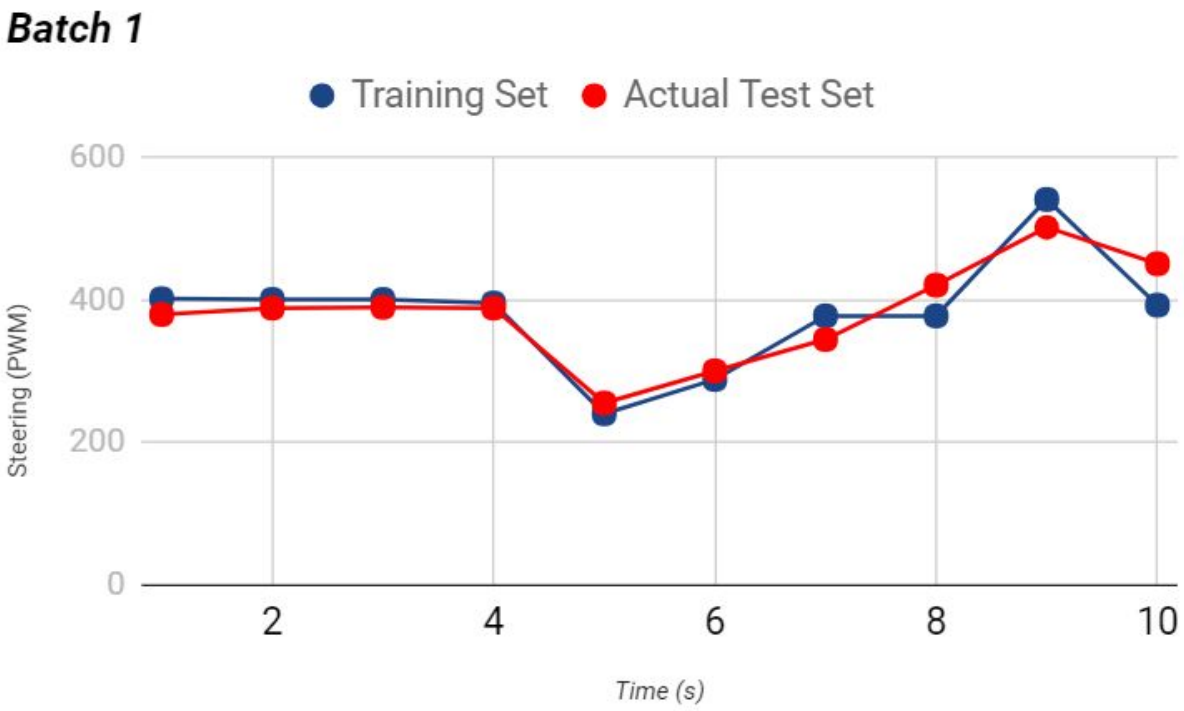


Figure 24. Steering Comparison

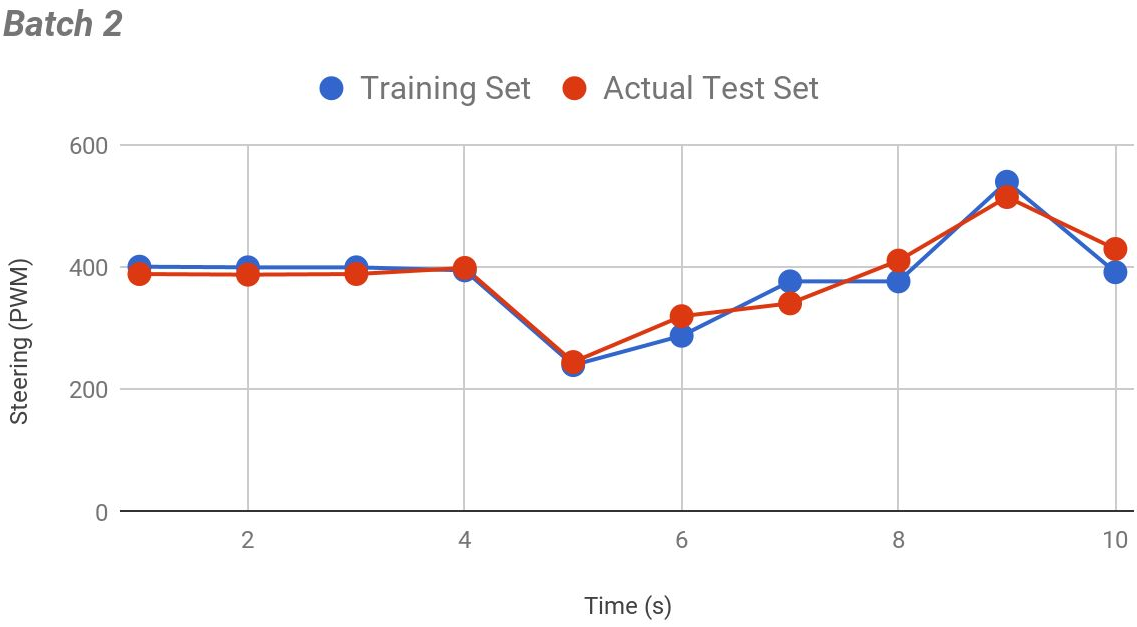


Figure 25. Steering Comparison

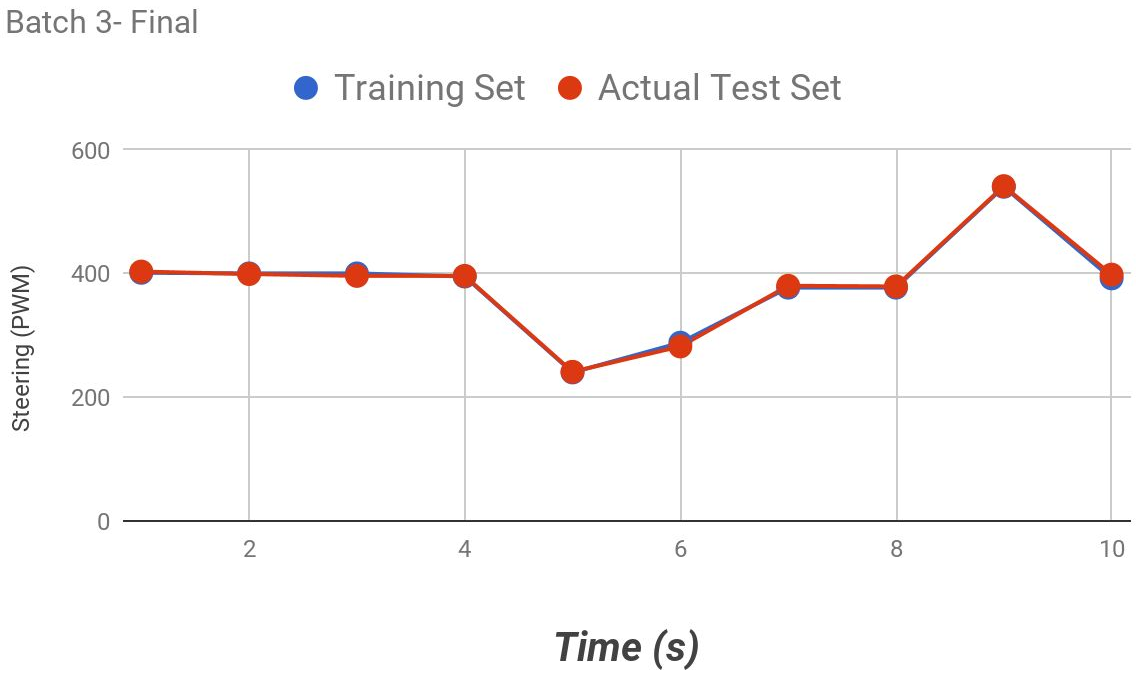


Figure 26. Steering Comparison

4.1.2. Boundary limits

The next part of evaluation is to observe if the model can keep the robot within the track at all times for different starting positions (Plant A and Plant B) when performed for 5 trials each. These results are tabulated below:

Start Point	**Trial**	**Touch Outer Boundary?**	**Touch Inner Boundary?**
Plant A	1	No	No
	2	No	No
	3	No	No
	4	Yes	No
	5	No	No

Plant B	1	No	No
	2	No	No
	3	No	No
	4	No	No
	5	No	No

As can be observed from the table, the model performed exceptionally well, only touching the outer boundary once out of 10 trials.

For the second track, the results are tabulated as follows.

Start Point	**Trial**	**Touch Outer Boundary?**	**Touch Inner Boundary?**
Start A	1	No	No
	2	No	No
	3	Yes	No
	4	Yes	No
	5	No	No
	6	No	No
	7	No	Yes
	8	No	No
	9	No	No
	10	No	No

The model performed well considering the complex setup of the track and the hardware limitations of the robot, only touching the boundary four times out of 10 trials.

4.2. Traffic Sign Module

The goal of the traffic sign module is for the robot to be able to navigate based on only the signs it encounters in the environment. The general working of this module is as follows:

- Wait for material loading and start program by the user.
- Go straight until encountering a sign (sliding window search and template matching) and avoid obstacles.
- Preprocess the images and feed it to the trained network – outputting one of the four classes.
- Navigate based on the obtained output and continue straight.
- Repeat the process until the stop sign is detected.

4.2.1. Individual Testing

To test the functioning of model in real time, each traffic sign was tested individually. The following table shows how the model reacted to the signs for each of the 5 trials:

Traffic Sign	Trial	Success of Detection
Right turn	1	Yes
	2	Yes
	3	No
	4	Yes
	5	Yes
Left turn	1	Yes
	2	Yes
	3	Yes
	4	No
	5	Yes
Stop	1	Yes
	2	Yes

	3	Yes
	4	No
	5	Yes
Straight	1	Yes
	2	Yes
	3	Yes
	4	Yes
	5	Yes

Video Link 1: https://www.youtube.com/watch?v=IXC2TxQnW8o
Video Link 2: https://www.youtube.com/watch?v=pFbWb_RIWvY

4.2.2. Testing Entire System:

To test the detection of signs in real time, all of the traffic signs being used was tested together in a track layout as shown below:

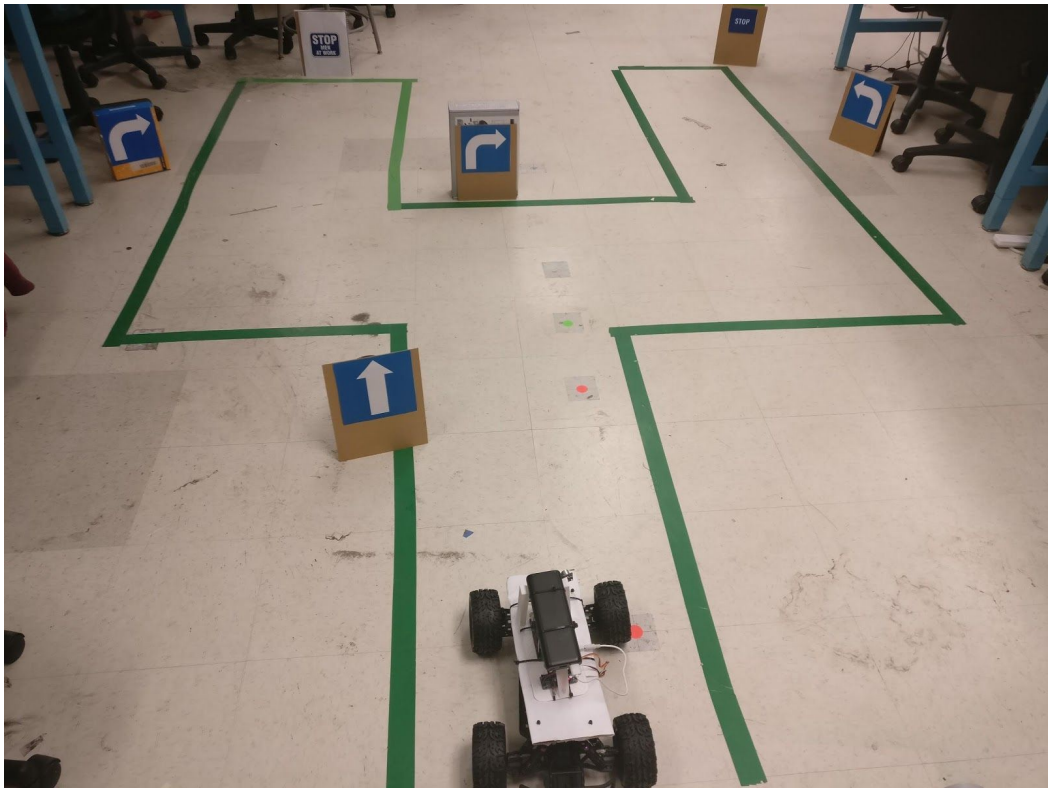


Figure 27. Testing Layout

The following table shows how the model reacted to the signs done for 10 trials:

Trail	Detection	Total Time (s)
1	All	32
2	Right, Left, Stop	32
3	All	33
4	All	32
5	Right, Stop	32
6	All	33
7	All	32
8	All	34
9	All	35

10	All	34
Average		**32.9**

4.2.3. Delay Time:

The time taken for detecting each of the signs for 5 trials is documented below:

Traffic Sign	Trial	Time for Detection (s)
Right turn	1	2
	2	2.1
	3	2.1
	4	2
	5	2.1
	Average	**2.06**
Left turn	1	2.2
	2	2.2
	3	2.2
	4	2.1
	5	2.1
	Average	**2.16**
Stop	1	2.2
	2	2.1
	3	2.1
	4	2.0
	5	2.0

	Average	**2.125**
Straight	1	2.1
	2	2.1
	3	2.1
	4	2.0
	5	2.0
	Average	**2.06**

4.2.4. Boundary limit:

The next part of the evaluation is to observe if the model can keep the robot within the track at all times for different starting positions for 5 trials each. These results are tabulated below:

Trial	**Boundary exceeded?**
1	Yes
2	No
3	Yes
4	No
5	No

It can be said that the system needs to be more accurate to not go beyond the lines, which might be a major hassle in an industrial setup, but this issue mainly arises because the robot platform has limited steering capabilities.

Subjective Evaluation

An experiment was designed to test the effectiveness of the system in a simulated factory environment. It involved one person loading the material to be transferred and start the robot, while the other person waits for the material and unloads it off of the robot as shown in the pictures below:

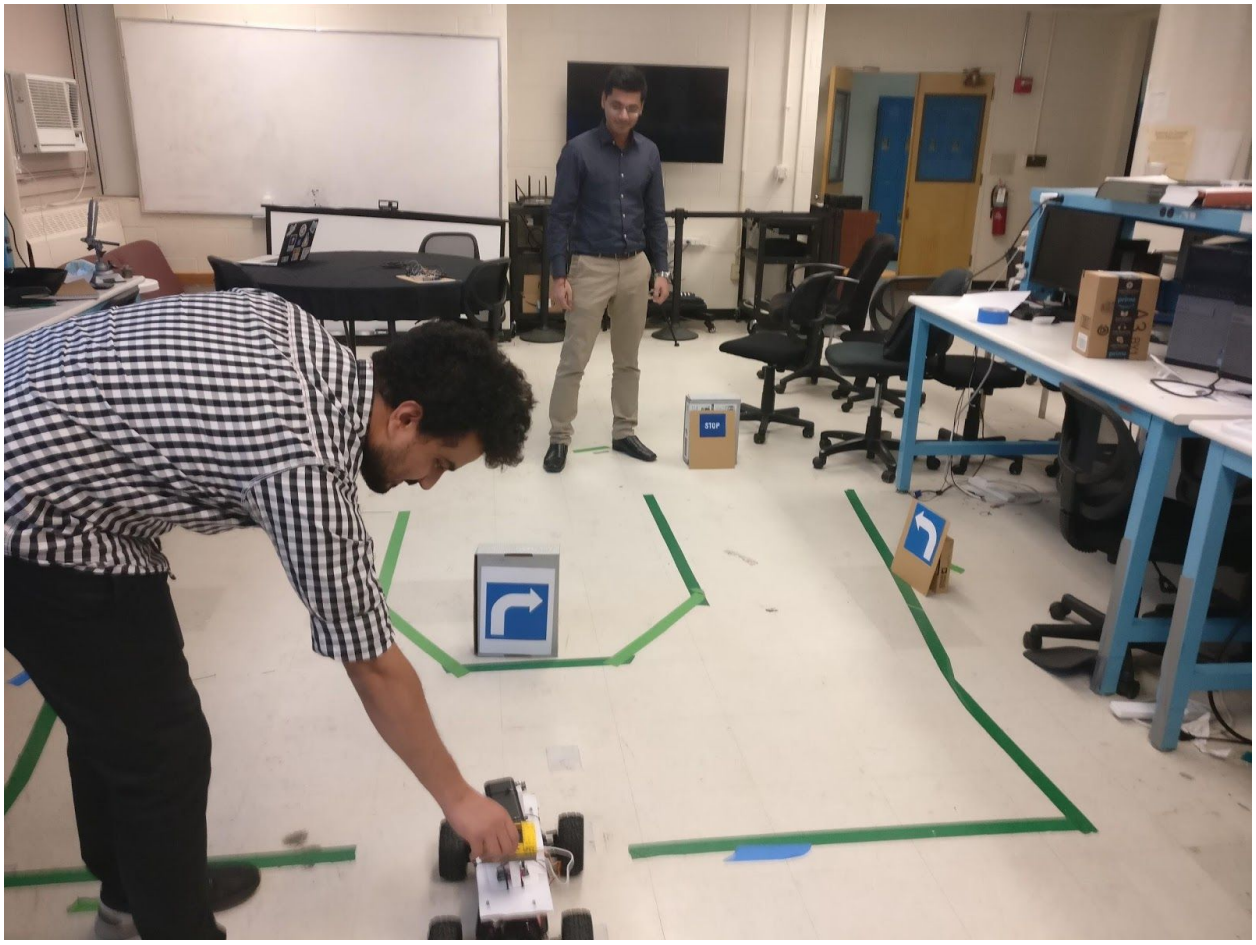


Figure 28: Loading the material

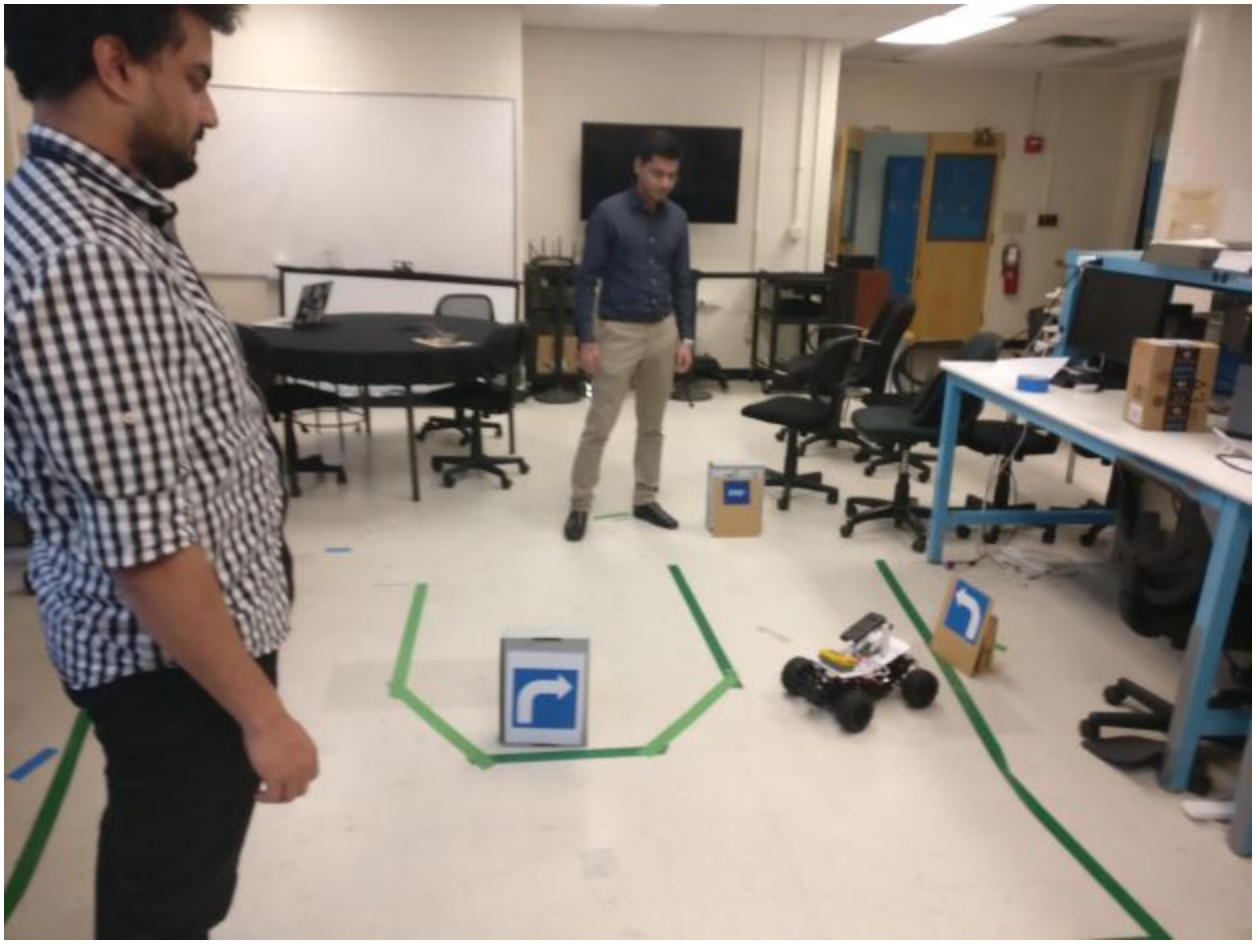


Figure 29: Robot Navigating to the location

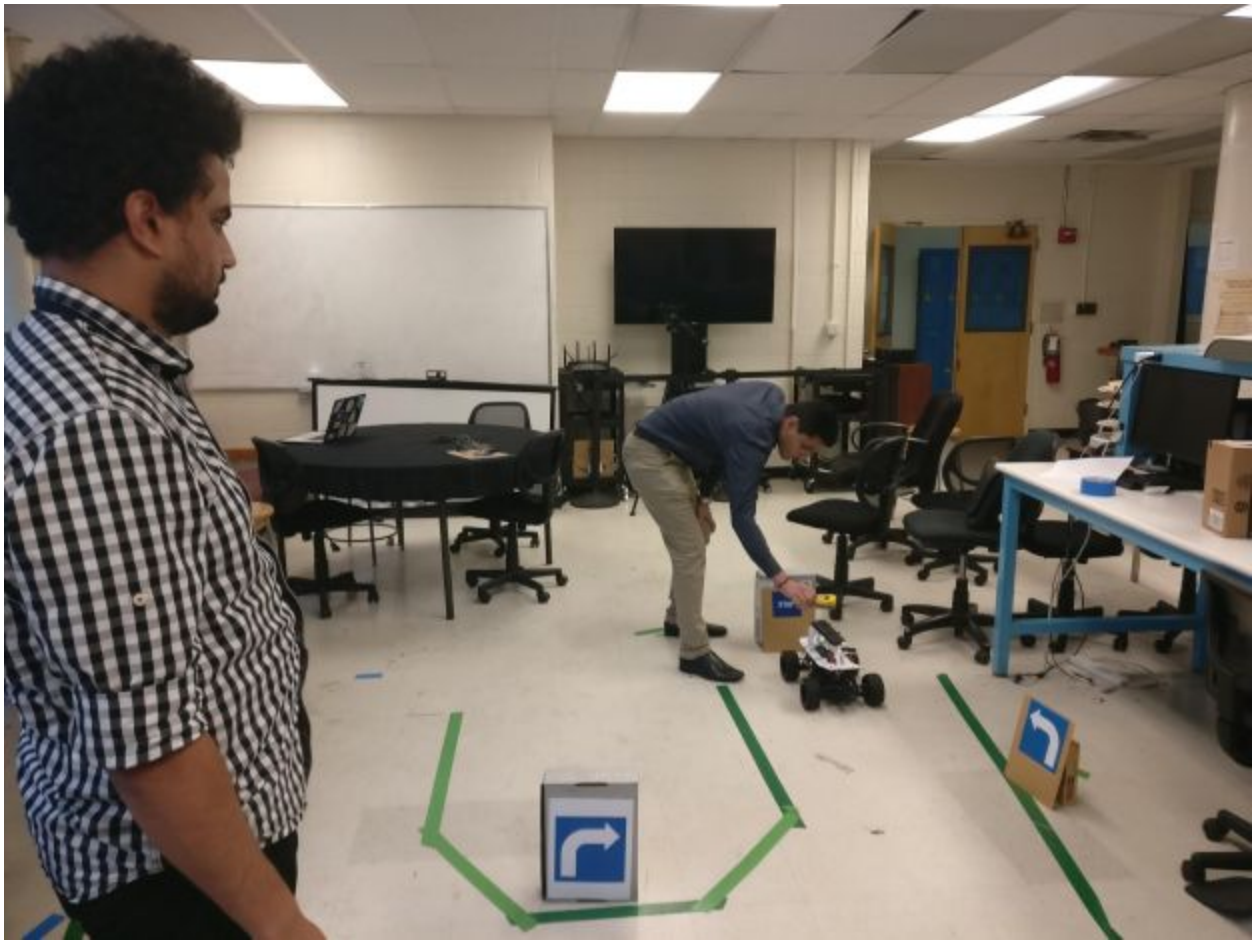


Figure 30: Unloading the material

This was repeated for 22 test subjects. A survey was conducted with questions concerning safety, effectiveness, time taken, etc., and the results were summarized below:

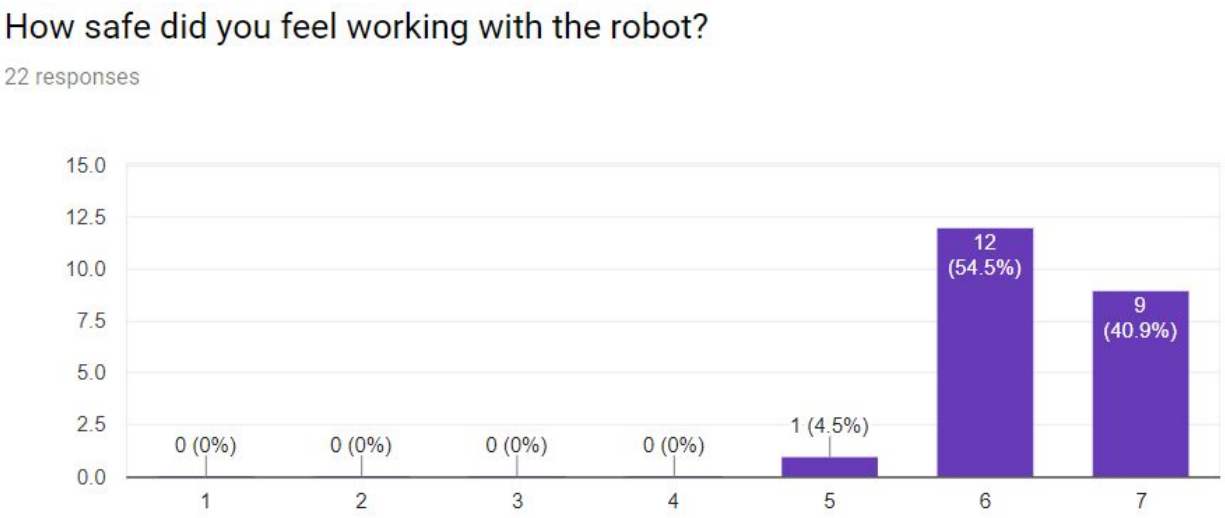


Figure 31: Subjective Evaluation 1

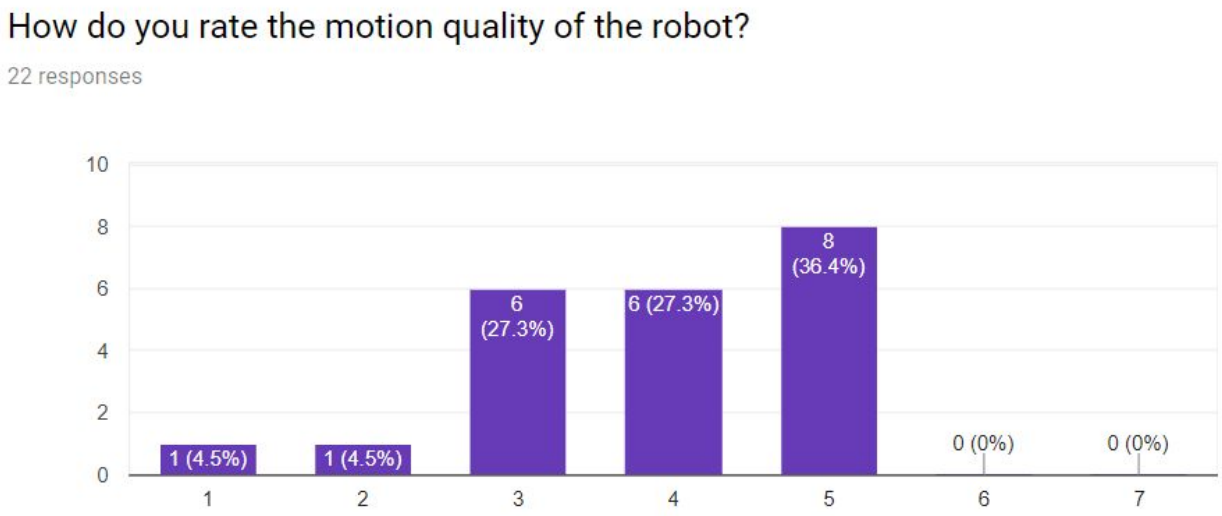


Figure 32: Subjective Evaluation 2

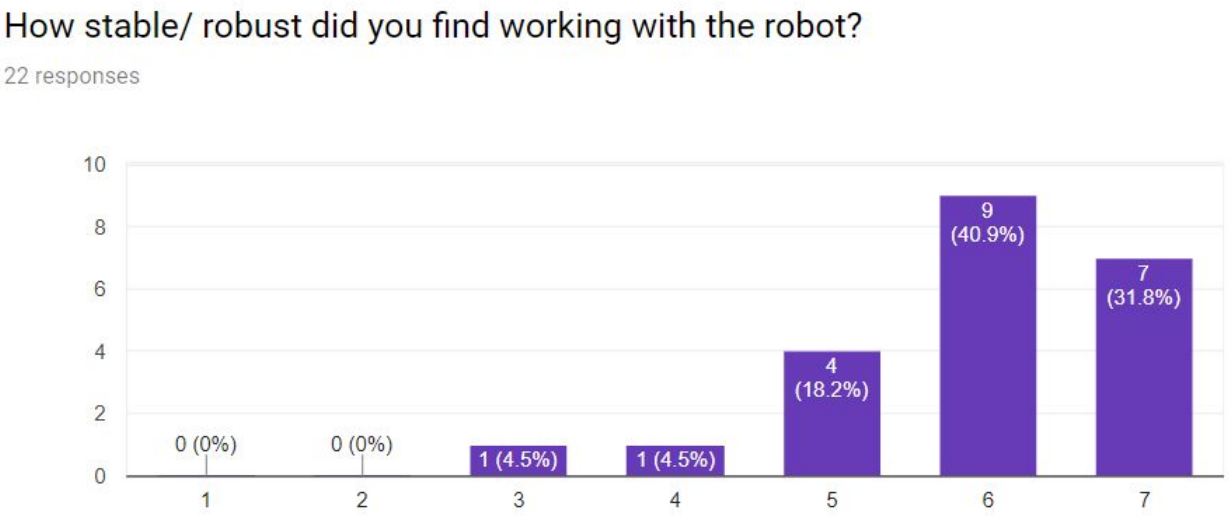


Figure 33: Subjective Evaluation 3

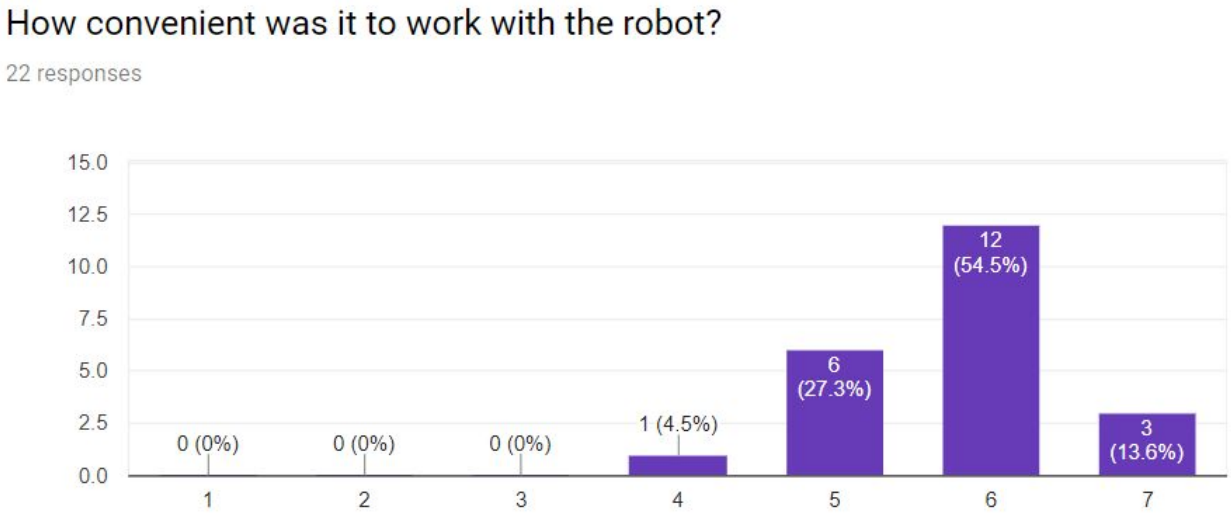


Figure 34: Subjective Evaluation 4

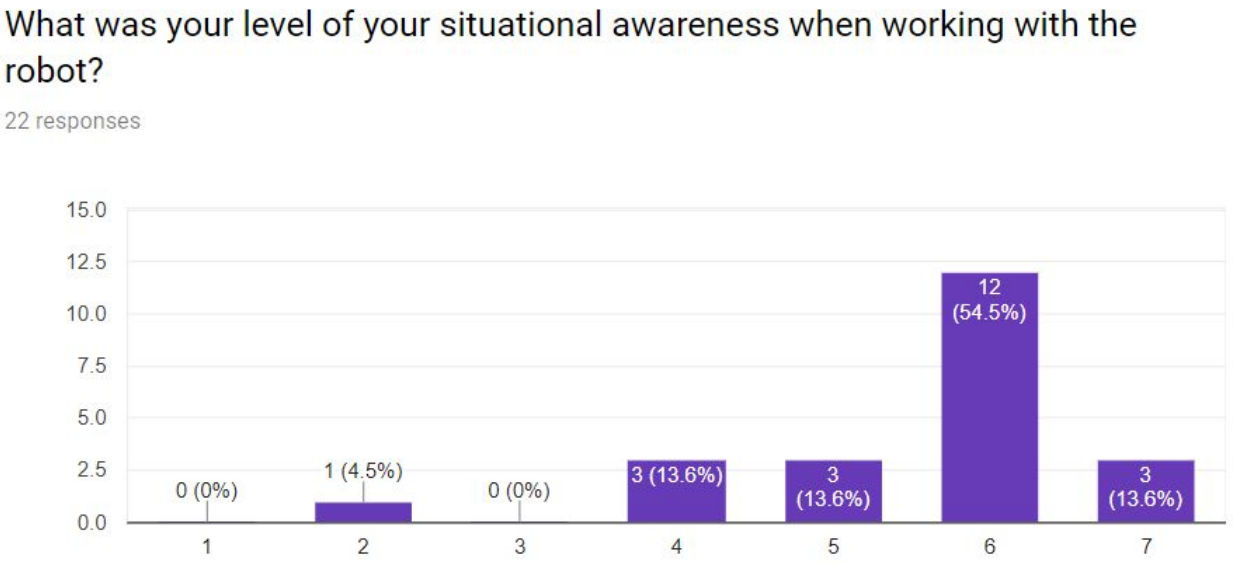


Figure 35: Subjective Evaluation 5

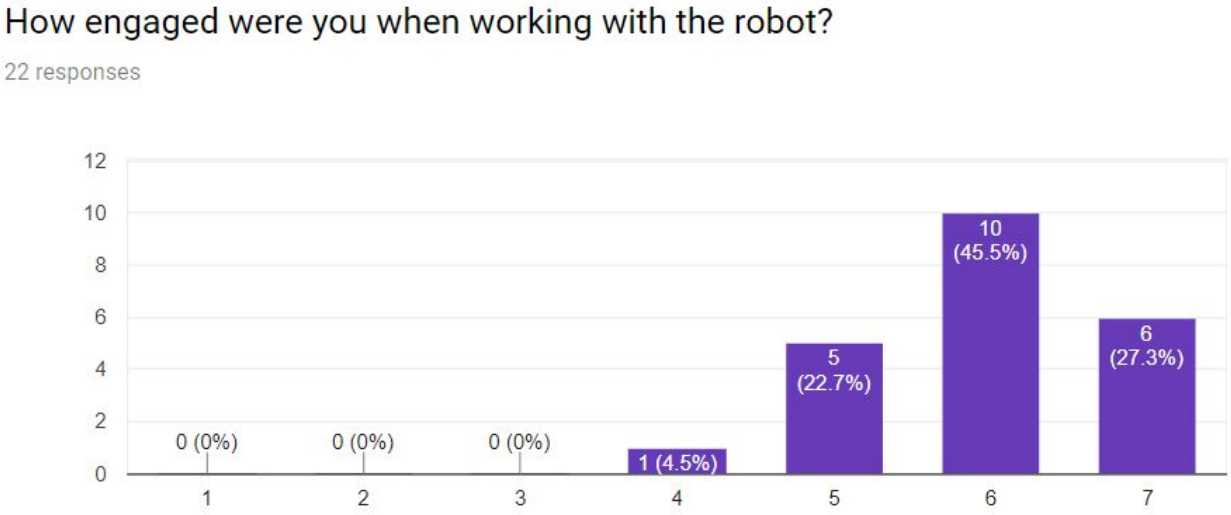


Figure 36: Subjective Evaluation 6

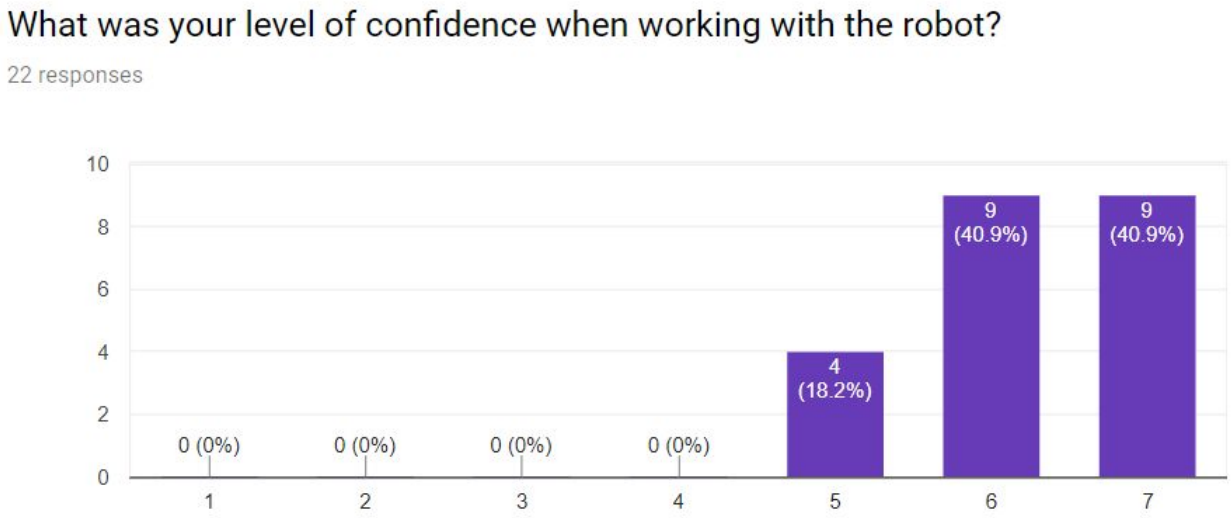


Figure 37: Subjective Evaluation 7

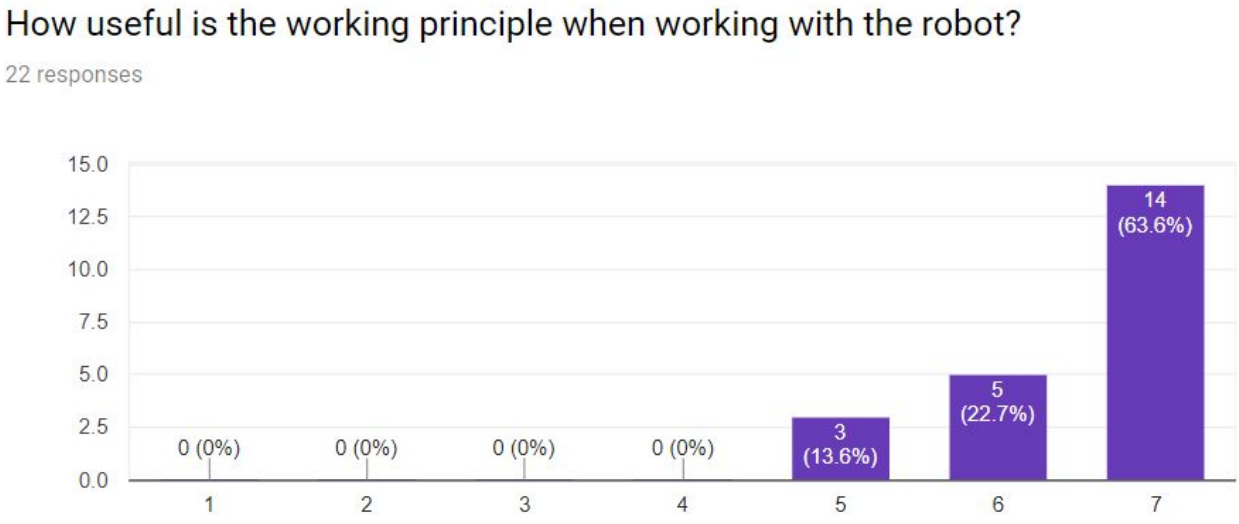


Figure 38: Subjective Evaluation 8

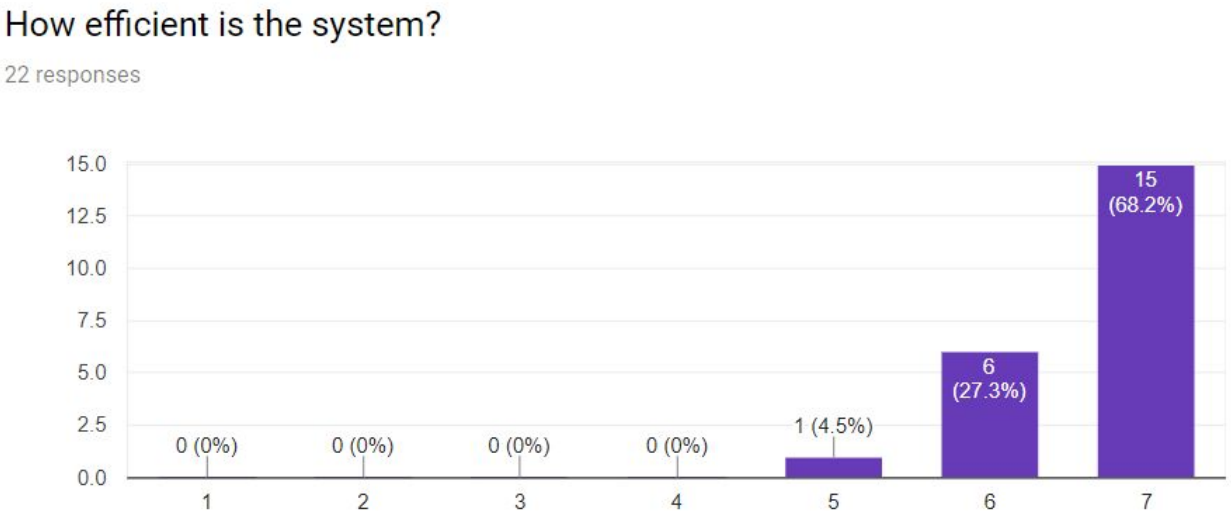


Figure 39: Subjective Evaluation 9

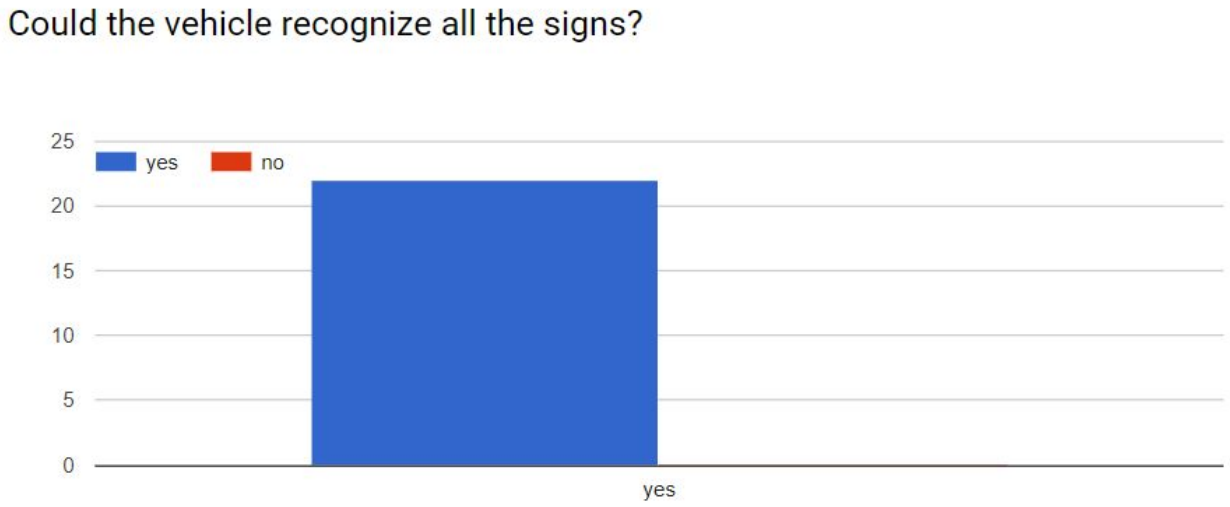


Figure 40 :Subjective Evaluation 10

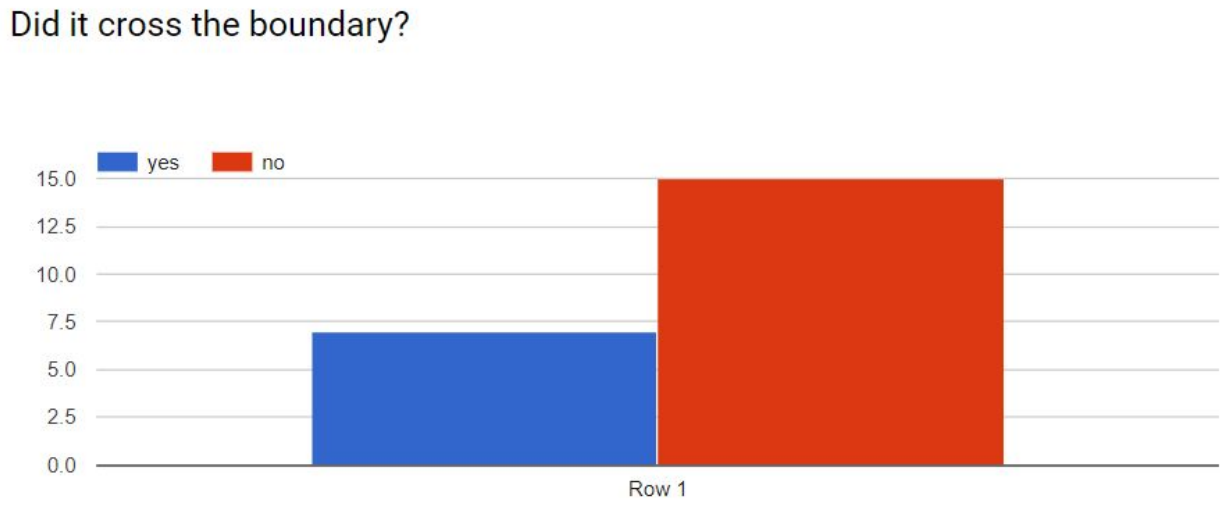


Figure 41 Subjective Evaluation 41

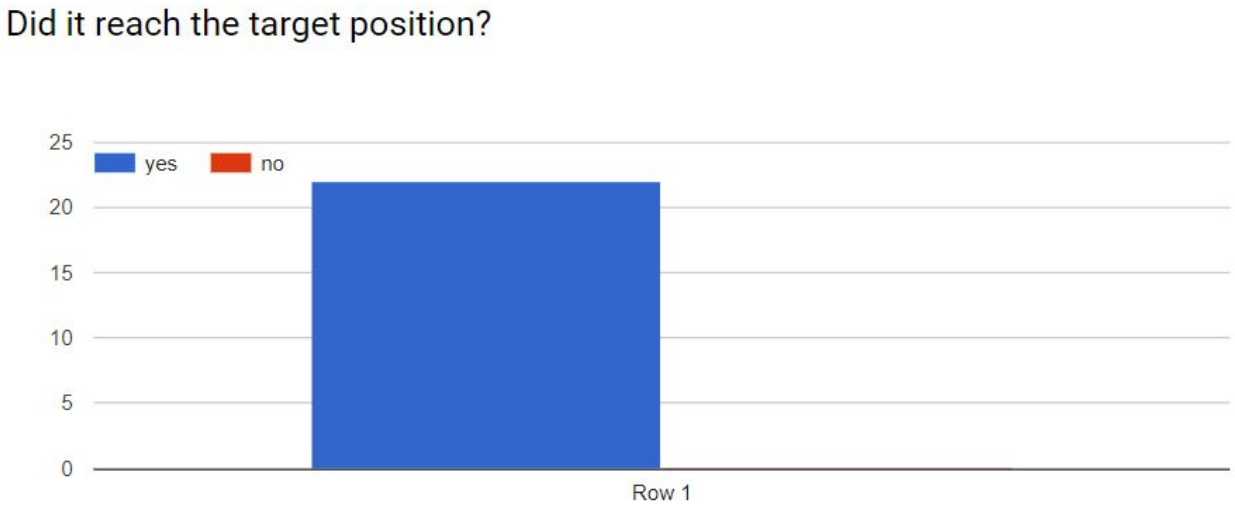


Figure 42: Subjective Evaluation 42

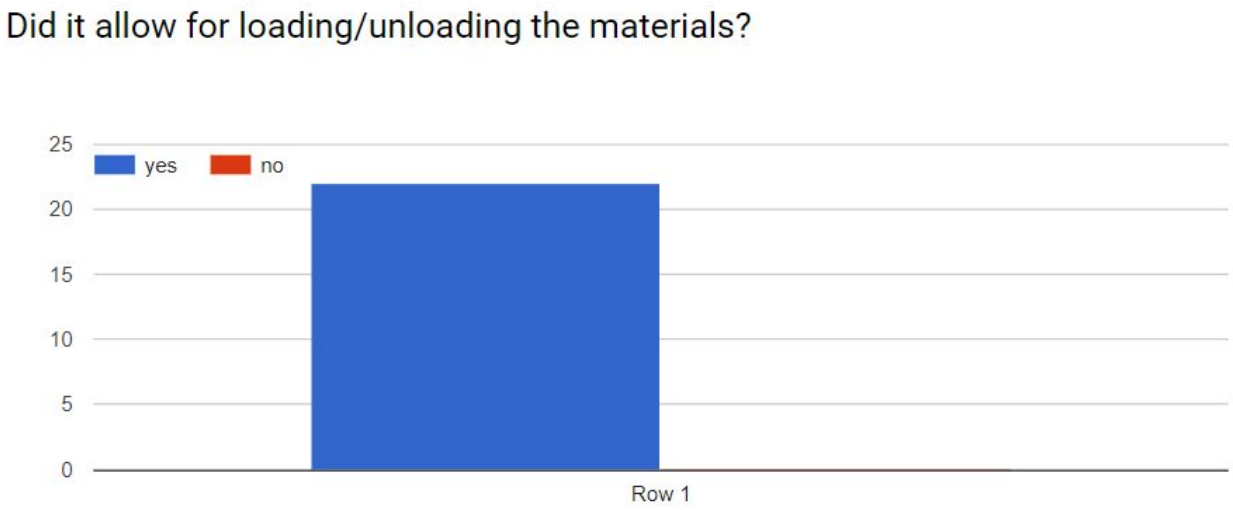


Figure 43: Subjective Evaluation 43

As can be seen from the results above, the robot performs exceedingly well in all of the important measures, except the motion's quality. This was an expected attribute since the robot has now been programmed to go one step at a time to make sure that each sign is detected.

5. Conclusion and Discussion

This project presents two novel approaches for navigating an industrial material transfer robot/vehicle – namely, behavioral cloning and traffic sign based, autonomous navigation. There were numerous challenges faced during the development of the robot, including the following: creating an elaborate and efficient dataset to train the CNN model, performing various model adjustments and data augmentation, and making the system work in real time using a low cost computing platform like raspberry pi. Even though there was an average of 2.1 seconds delay in recognizing the sign, it can be considered negligible, considering the entire cost of the project was less than $350.

An opportunity to improve the overall efficiency of the system exists, and future work on this project may take into account a few of the following improvements in order to create an even better industrial material transfer robot:

- A different build platform to control the robot more efficiently. This is due to the fact that the Redcat Racing RC platform cannot be efficiently controlled to steer at right angles (maximum steering angle is 55 degrees) [9] and has an inefficient and flawed battery system.

- Make use of a dedicated controller where neural networks can be trained locally. This eliminates the use of a stand-alone computer, making the system more portable, as well as giving extra computational power to the microcontroller.
- Incorporate mapping and localization techniques. This is to give the user a choice of using a pre-built map for navigating the industrial environment in a static environment which is faster, or to use the navigation module based on traffic sign recognition and behavioral cloning.
- A better sliding window search algorithm. This will detect traffic signs and match them to a virtual bounding box, which may increase the accuracy of the CNN in real-time detection.
- Use a LiDAR. To replace the unreliable ultrasonic sensor in a dynamic environment to implement collision avoidance – making it safer, as well as facilitating the implementation of SLAM or other localization techniques.

6. Acknowledgement

First, I acknowledge Prof. Vikram Kapila of the Department of Mechanical and Aerospace Engineering, Tandon School of Engineering, New York University, Brooklyn, NY for kindly accepting me to work in his laboratory and providing me with all the necessary resources for this project. Next, I thank Dr. Mizanoor Rahman of the Department of Mechanical and Aerospace Engineering, Tandon School of Engineering, New York University, Brooklyn, NY for his instructions and guidelines for carrying out the project activities and preparing this report. I also appreciate Mr. Ashwin Raj Kumar and Mr. Saiprasanth Krishnamoorthy of the Department of Mechanical and Aerospace Engineering, Tandon School of Engineering, New York University, Brooklyn, NY for their technical support. I would also like to acknowledge Angad Boralkar, Tanaya Bhave and Abhideeptha for their valuable input and for recording videos. Finally, I am grateful the IGNITE Fellowship grant for AWS, which helped me train the networks in a small amount of time.

7. References:

[1] Issues with Industrial Robots: https://www.iso.org/obp/ui/#iso:std:iso:8373:ed-2:v1:en

[2] Otto Robot: https://www.ottomotors.com/otto1500

[3] MIR Robot: http://www.mobile-industrial-robots.com/

[4] 3D Design files: https://www.thingiverse.com/thing:2260575

[5] "Traffic Sign Recognition with Multi-Scale Convolutional Networks." Pierre Sermanet and Yann LeCun, Courant Institute of Mathematical Sciences, New York University.

[6] "End to End Learning for Self-Driving Cars." 2016. Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, Karol Zieba.

[7] What is the best multi-stage architecture for object recognition, In International Conference on Computer Vision,*K. Jarrett, K. Kavukcuoglu, M. A. Ranzato, and Y. LeCun*, pages 2146–2153. IEEE, 2009

[8] "CNN Features off-the-shelf: An Astounding Baseline for Recognition." *Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, Stefan Carlsson"*

[9] Issues with hardware: http://www.redcatracing.com/FAQ

[10] Image Reference: https://medium.com/udacity/udacity-self-driving-car-nanodegree-project-3-behavioral-cloning-446461b7c7f9

[11 Image Reference: http://jokla.me/robotics/traffic-signs/

[12]Fritzing Wiring Library Software, http://fritzing.org/home/

[13] German Traffic Sign Database, http://benchmark.ini.rub.de/